Amazon EKS

User Guide

Amazon EKS: User Guide

Table of Contents

What Is Amazon EKS? 1
Amazon EKS Control Plane Architecture 1
How Does Amazon EKS Work? 2
Getting Started with Amazon EKS 3
Getting Started with `eksctl` 3
Prerequisites 3
Create Your Amazon EKS Cluster and Worker Nodes 6
Next Steps 11
Getting Started with the Console 11
Amazon EKS Prerequisites 11
Step 1: Create Your Amazon EKS Cluster 14
Step 2: Create a `kubeconfig` File 16
Step 3: Launch and Configure Amazon EKS Worker Nodes 17
Next Steps 22
Clusters 24
Creating a Cluster 24
Updating Kubernetes Version 33
Cluster Endpoint Access 42
Modifying Cluster Endpoint Access 43
Accessing the API Server from within the VPC 45
Control Plane Logging 46
Enabling and Disabling Control Plane Logs 46
Viewing Cluster Control Plane Logs 48
Deleting a Cluster 48
Kubernetes Versions 51
Available Amazon EKS Kubernetes Versions 51
Kubernetes 1.14 51
Kubernetes 1.13 52
Amazon EKS Version Deprecation 53
Platform Versions 53
Kubernetes version 1.14 54
Kubernetes version 1.13 54
Kubernetes version 1.12 55
Kubernetes version 1.11 56
Kubernetes version 1.10 57
Windows Support 58
Considerations 58
Enabling Windows Support 59
Deploy a Windows Sample Application 62
Worker Nodes 64
Amazon EKS-Optimized Linux AMI 64
Amazon EKS-Optimized AMI Build Scripts 68
Amazon EKS-Optimized AMI with GPU Support 68
Retrieving Amazon EKS-Optimized AMI IDs 72
Amazon EKS-Optimized Windows AMI 73
Retrieving Amazon EKS-Optimized Windows AMI IDs 74
Partner AMIs 75
Launching Amazon EKS Linux Worker Nodes 76
Launching Amazon EKS Windows Worker Nodes 83
Worker Node Updates 89
Migrating to a New Worker Node Group 89
Updating an Existing Worker Node Group 94
Storage 97
Storage Classes 97

Amazon EBS CSI Driver ... 98
Amazon EFS CSI Driver ... 101
Autoscaling ... 106
Cluster Autoscaler ... 106
Create an Amazon EKS Cluster ... 106
Create Node Groups for your Cluster ... 107
Deploy the Cluster Autoscaler ... 108
View your Cluster Autoscaler Logs ... 109
Horizontal Pod Autoscaler ... 109
Install the Metrics Server ... 110
Run a Horizontal Pod Autoscaler Test Application ... 111
Vertical Pod Autoscaler ... 113
Install the Metrics Server ... 113
Deploy the Vertical Pod Autoscaler ... 114
Test your Vertical Pod Autoscaler Installation ... 115
Load Balancing and Ingress ... 119
Load Balancing ... 119
Subnet Tagging for Load Balancers ... 119
ALB Ingress Controller on Amazon EKS ... 120
Networking ... 124
Creating a VPC for Amazon EKS ... 124
Next Steps ... 126
Cluster VPC Considerations ... 126
VPC IP Addressing ... 127
VPC Tagging Requirement ... 127
Subnet Tagging Requirement ... 127
Cluster Security Group Considerations ... 128
Pod Networking (CNI) ... 130
CNI Configuration Variables ... 131
External SNAT ... 132
CNI Custom Networking ... 133
CNI Metrics Helper ... 136
CNI Upgrades ... 139
Installing CoreDNS ... 140
Installing Calico on Amazon EKS ... 141
Stars Policy Demo ... 142
Managing Cluster Authentication ... 146
Installing `kubectl` ... 146
Installing `aws-iam-authenticator` ... 151
Create a `kubeconfig` for Amazon EKS ... 154
Managing Users or IAM Roles for your Cluster ... 157
`eksctl` ... 161
Installing or Upgrading `eksctl` ... 161
Guest Book ... 163
Metrics Server ... 166
Prometheus Metrics ... 168
Viewing the Raw Metrics ... 168
Deploying Prometheus ... 168
Using Helm ... 172
Tutorial: Deploy Kubernetes Dashboard ... 175
Prerequisites ... 175
Step 1: Deploy the Kubernetes Metrics Server ... 175
Step 2: Deploy the Dashboard ... 177
Step 3: Create an `eks-admin` Service Account and Cluster Role Binding ... 177
Step 4: Connect to the Dashboard ... 178
Step 5: Next Steps ... 179
Getting Started with App Mesh and Kubernetes ... 180

Prerequisites 180
Step 1: Create Your Service Mesh 180
Step 2: Create Your Virtual Nodes 180
Step 3: Create Your Virtual Routers 181
Step 4: Create Your Routes 182
Step 5: Create Your Virtual Services 182
Step 6: Updating Your Microservice Pod Specifications 183
Tutorial: Configure App Mesh Integration with Kubernetes 185
Prerequisites 185
Step 1: Install the Controller and Custom Resources 185
Step 2: Install the Sidecar Injector 186
.... 186
Step 3: Configure App Mesh 187
Create Kubernetes Custom Resources 187
Sidecar Injection 188
Step 4: Remove Integration Components (Optional) 189
Deploy a Mesh Connected Service 189
Prerequisites 189
Deploy a Sample Application 189
Run Application 191
Change Configuration 191
Remove Application 192
Deep Learning Containers 193
Security 194
Identity and Access Management 194
Audience 195
Authenticating With Identities 195
Managing Access Using Policies 197
How Amazon EKS Works with IAM 198
Identity-Based Policy Examples 201
Service IAM Role 203
Worker Node IAM Role 205
IAM Roles for Service Accounts 208
Troubleshooting 219
Logging and Monitoring 220
Compliance Validation 220
Resilience 220
Infrastructure Security 221
Configuration and Vulnerability Analysis 222
Pod Security Policy 222
Amazon EKS Default Pod Security Policy 222
Tagging Your Resources 226
Tag Basics 226
Tagging Your Resources 226
Tag Restrictions 227
Working with Tags Using the Console 227
Adding Tags on an Individual Resource On Creation 227
Adding and Deleting Tags on an Individual Resource 228
Working with Tags Using the CLI or API 228
CloudTrail 230
Amazon EKS Information in CloudTrail 230
Understanding Amazon EKS Log File Entries 231
Related Projects 232
Management Tools 232
eksctl 232
AWS Service Operator 232
Networking 232

Amazon VPC CNI plugin for Kubernetes 232
AWS Application Load Balancer (ALB) Ingress Controller for Kubernetes 232
ExternalDNS 233
Security 233
AWS IAM Authenticator 233
Machine Learning 233
Kubeflow 233
Auto Scaling 233
Cluster Autoscaler 233
Escalator 234
Monitoring 234
Prometheus 234
Continuous Integration / Continuous Deployment 234
Jenkins X 234
Troubleshooting 235
Insufficient Capacity 235
`aws-iam-authenticator` Not Found 235
Worker Nodes Fail to Join Cluster 235
Unauthorized or Access Denied (`kubectl`) 235
`hostname doesn't match` 236
`getsockopt: no route to host` 236
CNI Log Collection Tool 236
IAM 237
AccessDeniedException 237
I Am Not Authorized to Perform iam:PassRole 237
I Want to View My Access Keys 238
I'm an Administrator and Want to Allow Others to Access Amazon EKS 238
I Want to Allow People Outside of My AWS Account to Access My Amazon EKS Resources 238
Service Limits 240
Document History 241
AWS Glossary 247

What Is Amazon EKS?

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

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

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

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

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

Amazon EKS Control Plane Architecture

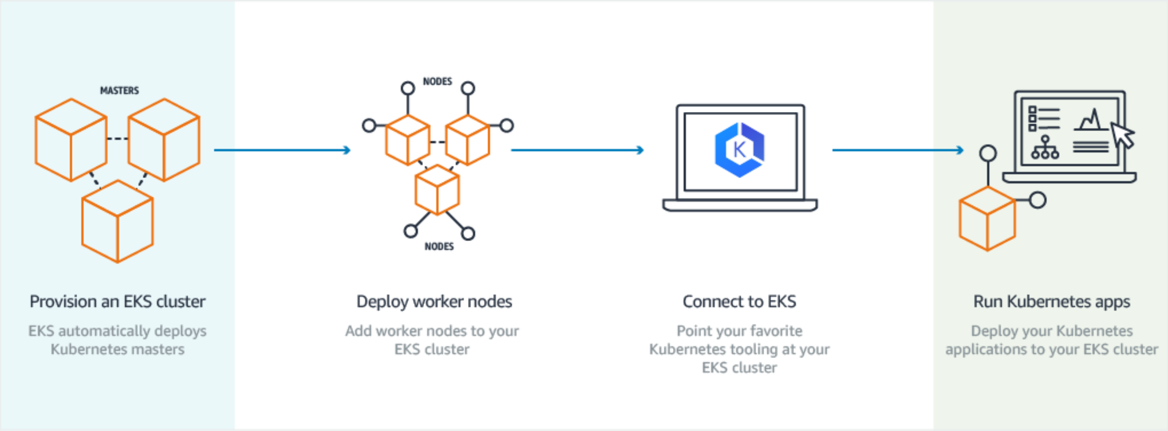
Amazon EKS runs a single tenant Kubernetes control plane for each cluster, and control plane infrastructure is not shared across clusters or AWS accounts.

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

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

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

How Does Amazon EKS Work?


MASTERS
Provision an EKS cluster
EKS automatically deploys Kubernetes masters
NODES
NODES
Deploy worker nodes
Add worker nodes to your EKS cluster
K
Connect to EKS
Point your favorite Kubernetes tooling at your EKS cluster
Run Kubernetes apps
Deploy your Kubernetes applications to your EKS cluster

Getting started with Amazon EKS is easy:

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

For more information about creating your required resources and your first Amazon EKS cluster, see Getting Started with Amazon EKS (p. 3).

Getting Started with Amazon EKS

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

- Getting Started with `eksctl` (p. 3): This getting started guide helps you to install all of the required resources to get started with Amazon EKS using `eksctl`, a simple command line utility for creating and managing Kubernetes clusters on Amazon EKS. At the end of this tutorial, you will have a running Amazon EKS cluster with worker nodes, and the `kubectl` command line utility will be configured to use your new cluster. This is the fastest and simplest way to get started with Amazon EKS.
- Getting Started with the AWS Management Console (p. 11): This getting started guide helps you to create all of the required resources to get started with Amazon EKS in the AWS Management Console. In this guide, you manually create each resource in the Amazon EKS or AWS CloudFormation consoles, and the workflow described here gives you complete visibility into how each resource is created and how they interact with each other; however, this is a more complicated and time consuming way to get started with Amazon EKS.

Getting Started with `eksctl`

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

Prerequisites

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

Install the Latest AWS CLI

To use `kubectl` with your Amazon EKS clusters, you must install a binary that can create the required client security token for cluster API server communication. The **aws eks get-token** command, available in version 1.16.232 or greater of the AWS CLI, supports client security token creation. To install or upgrade the AWS CLI, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

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

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

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

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

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

If you are unable to install version 1.16.232 or greater of the AWS CLI on your system, you must ensure that the AWS IAM Authenticator for Kubernetes is installed on your system. For more information, see Installing `aws-iam-authenticator` (p. 151).

Configure Your AWS CLI Credentials

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

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

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

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

Install `eksctl`

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

Choose the tab below that best represents your client setup.

macOS

To install or upgrade `eksctl` on macOS using Homebrew

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

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

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
master/install)"
```

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 `0.6.0`. If not, check your terminal output for any installation or upgrade errors.

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/download/
   latest_release/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 `0.6.0`. If not, check your terminal output for any installation or upgrade errors.

Windows

To install or upgrade `eksctl` on Windows using Chocolatey

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

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

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

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

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

   ```
   eksctl version
   ```

Note
The GitTag version should be at least 0.7.0. If not, check your terminal output for any installation or upgrade errors.

Install and Configure **kubectl** for Amazon EKS

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

Note
If you used the preceding Homebrew instructions to install eksctl on macOS, then kubectl and the aws-iam-authenticator have already been installed on your system. You can skip to Create Your Amazon EKS Cluster and Worker Nodes (p. 6).

To install kubectl for Amazon EKS

- You have multiple options to download and install **kubectl** for your operating system.
  - The kubectl binary is available in many operating system package managers, and this option is often much easier than a manual download and install process. You can follow the instructions for your specific operating system or package manager in the Kubernetes documentation to install.
  - Amazon EKS also vends **kubectl** binaries that you can use that are identical to the upstream **kubectl** binaries with the same version. To install the Amazon EKS-vended binary for your operating system, see Installing kubectl (p. 146).

Create Your Amazon EKS Cluster and Worker Nodes

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

To create your cluster and worker nodes with eksctl

1. Choose a tab below that matches your workload requirements. If you only intend to run Linux workloads on your cluster, choose **Linux**. If you want to run Linux and Windows workloads on your cluster, choose **Windows**.

   Linux

   This procedure assumes that you have installed eksctl, and that your eksctl version is at least 0.6.0. 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. 161).

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

   Important
   Amazon EKS will deprecate Kubernetes version 1.11 on November 4th, 2019. On this day, you will no longer be able to create new 1.11 clusters, and all Amazon EKS clusters running Kubernetes version 1.11 will be updated to the latest available platform version of Kubernetes version 1.12. For more information, see Amazon EKS Version Deprecation (p. 53).

Kubernetes version 1.10 is no longer supported on Amazon EKS. You can no longer create new 1.10 clusters, and all existing Amazon EKS clusters running Kubernetes version 1.10 will eventually be automatically updated to the latest available platform version of Kubernetes version 1.11. For more information, see Amazon EKS Version Deprecation (p. 53).
Please update any 1.10 clusters to version 1.11 or higher in order to avoid service interruption. For more information, see Updating an Amazon EKS Cluster Kubernetes Version (p. 33).

```
eksctl create cluster \
--name prod \
--version 1.14 \
--nodegroup-name standard-workers \
--node-type t3.medium \
--nodes 3 \
--nodes-min 1 \
--nodes-max 4 \
--node-ami auto
```

Note

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

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

Output:

```
[#]  using region us-west-2
[#]  setting availability zones to [us-west-2b us-west-2c us-west-2d]
[#]  subnets for us-west-2b - public:192.168.0.0/19 private:192.168.96.0/19
[#]  subnets for us-west-2c - public:192.168.32.0/19 private:192.168.128.0/19
[#]  subnets for us-west-2d - public:192.168.64.0/19 private:192.168.160.0/19
[#]  nodegroup "standard-workers" will use
 "ami-0923e4b35a30a5f53" [AmazonLinux2/1.12]
[#]  creating EKS cluster "prod" in "us-west-2" region
[#]  will create 2 separate CloudFormation stacks for cluster itself and the
 initial nodegroup
[#]  if you encounter any issues, check CloudFormation console or try 'eksctl utils
 describe-stacks --region=us-west-2 --name=prod'
[#]  building cluster stack "eksctl-prod-cluster"
[#]  creating nodegroup stack "eksctl-prod-nodegroup-standard-workers"
[#]  all EKS cluster resource for "prod" had been created
[#]  saved kubeconfig as "/Users/username/.kube/config"
[#]  adding role "arn:aws:iam::111122223333:role/eksctl-prod-nodegroup-standard-wo-
NodeInstanceRole-IJP4S12W3020" to auth ConfigMap
[#]  nodegroup "standard-workers" has 0 node(s)
[#]  waiting for at least 1 node(s) to become ready in "standard-workers"
[#]  nodegroup "standard-workers" has 2 node(s)
[#]  node "ip-192-168-22-17.us-west-2.compute.internal" is not ready
[#]  node "ip-192-168-32-184.us-west-2.compute.internal" is ready
[#]  kubectl command should work with "/Users/username/.kube/config", try 'kubectl
 get nodes'
[#]  EKS cluster "prod" in "us-west-2" region is ready
```

Windows

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.7.0`. 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. 161).

Retrieve the Windows worker node AMI ID that you want to use with your cluster.

Kubernetes version 1.14.6

Region	Amazon EKS-optimized Windows Server 2019 Full	Amazon EKS-optimized Windows Server 2019 Core
US East (Ohio) (`us-east-2`)	View AMI ID	View AMI ID
US East (N. Virginia) (`us-east-1`)	View AMI ID	View AMI ID
US West (Oregon) (`us-west-2`)	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (`ap-east-1`)	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (`ap-south-1`)	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (`ap-northeast-1`)	View AMI ID	View AMI ID
Asia Pacific (Seoul) (`ap-northeast-2`)	View AMI ID	View AMI ID
Asia Pacific (Singapore) (`ap-southeast-1`)	View AMI ID	View AMI ID
Asia Pacific (Sydney) (`ap-southeast-2`)	View AMI ID	View AMI ID
EU (Frankfurt) (`eu-central-1`)	View AMI ID	View AMI ID
EU (Ireland) (`eu-west-1`)	View AMI ID	View AMI ID
EU (London) (`eu-west-2`)	View AMI ID	View AMI ID
EU (Paris) (`eu-west-3`)	View AMI ID	View AMI ID
EU (Stockholm) (`eu-north-1`)	View AMI ID	View AMI ID
Middle East (Bahrain) (`me-south-1`)	View AMI ID	View AMI ID

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

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

metadata:
  name: windows-prod
  region: us-west-2
  version: '1.14'

nodeGroups:
  - name: linux-ng
    instanceType: t2.large
    minSize: 2
  - name: windows-ng
    instanceType: m5.large
    minSize: 2
    volumeSize: 100
    ami: ami-0c7f1b5f1bebccac2
    amiFamily: WindowsServer2019FullContainer
```

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

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

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

Output:

```
[#]  using region us-west-2
[#]  setting availability zones to [us-west-2a us-west-2d us-west-2c]
[#]  subnets for us-west-2a - public:192.168.0.0/19 private:192.168.96.0/19
[#]  subnets for us-west-2d - public:192.168.32.0/19 private:192.168.128.0/19
[#]  subnets for us-west-2c - public:192.168.64.0/19 private:192.168.160.0/19
[#]  nodegroup "linux-ng" will use "ami-076c743acc3ec4159" [AmazonLinux2/1.14]
[#]  nodegroup "windows-ng" will use
 "ami-0c7f1b5f1bebccac2" [WindowsServer2019FullContainer/1.14]
[#]  using Kubernetes version 1.14
[#]  creating EKS cluster "windows-cluster" in "us-west-2" region
[#]  2 nodegroups (linux-ng, windows-ng) were included (based on the include/
exclude rules)
[#]  will create a CloudFormation stack for cluster itself and 2 nodegroup stack(s)
[#]  if you encounter any issues, check CloudFormation console or try 'eksctl utils
 describe-stacks --region=us-west-2 --name=windows-cluster'
[#]  CloudWatch logging will not be enabled for cluster "windows-cluster" in "us-
west-2"
[#]  you can enable it with 'eksctl utils update-cluster-logging --region=us-west-2
 --name=windows-cluster'
[#]  3 sequential tasks: { create cluster control plane "windows-cluster", 2
 parallel sub-tasks: { create nodegroup "linux-ng", create nodegroup "windows-
ng" }, install Windows VPC controller }
[#]  building cluster stack "eksctl-windows-cluster-cluster"
[#]  deploying stack "eksctl-windows-cluster-cluster"
```

```
[#]  building nodegroup stack "eksctl-windows-cluster-nodegroup-linux-ng"
[#]  building nodegroup stack "eksctl-windows-cluster-nodegroup-linux-ng"
0m[#]  --nodes-max=2 was set automatically for nodegroup windows-ng
[#]  --nodes-max=2 was set automatically for nodegroup linux-ng
[#]  deploying stack "eksctl-windows-cluster-nodegroup-windows-ng"
[#]  deploying stack "eksctl-windows-cluster-nodegroup-linux-ng"
[#]  created "ClusterRole.rbac.authorization.k8s.io/vpc-resource-controller"
[#]  created "ClusterRoleBinding.rbac.authorization.k8s.io/vpc-resource-controller"
[#]  created "kube-system:ServiceAccount/vpc-resource-controller"
[#]  created "kube-system:Deployment.apps/vpc-resource-controller"
[#]  created "CertificateSigningRequest.certificates.k8s.io/vpc-admission-
webhook.kube-system"
[#]  created "kube-system:secret/vpc-admission-webhook-certs"
[#]  created "kube-system:Service/vpc-admission-webhook"
[#]  created "kube-system:Deployment.apps/vpc-admission-webhook"
[#]  created "kube-
system:MutatingWebhookConfiguration.admissionregistration.k8s.io/vpc-admission-
webhook-cfg"
[#]  all EKS cluster resources for "windows-cluster" have been created
[#]  saved kubeconfig as "C:\\Users\\username/.kube/config"
[#]  adding role "arn:aws:iam::123456789012:role/eksctl-windows-cluster-nodegroup-
NodeInstanceRole-ZR93IIUZSYPR" to auth ConfigMap
[#]  nodegroup "linux-ng" has 0 node(s)
[#]  waiting for at least 2 node(s) to become ready in "linux-ng"
[#]  nodegroup "linux-ng" has 2 node(s)
[#]  node "ip-192-168-8-247.us-west-2.compute.internal" is ready
[#]  node "ip-192-168-80-253.us-west-2.compute.internal" is ready
[#]  adding role "arn:aws:iam::123456789012:role/eksctl-windows-cluster-nodegroup-
NodeInstanceRole-XM9UZN3NXBOB" to auth ConfigMap
[#]  nodegroup "windows-ng" has 0 node(s)
[#]  waiting for at least 2 node(s) to become ready in "windows-ng"
[#]  nodegroup "windows-ng" has 2 node(s)
[#]  node "ip-192-168-4-192.us-west-2.compute.internal" is ready
[#]  node "ip-192-168-63-224.us-west-2.compute.internal" is ready
[#]  kubectl command should work with "C:\\Users\\username/.kube/config", try
 'kubectl get nodes'
[#]  EKS cluster "windows-cluster" in "us-west-2" region is ready
```

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

```
kubectl get svc
```

Note

If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 151).
If you receive any other authorization or resource type errors, see Unauthorized or Access Denied (`kubectl`) (p. 235) in the troubleshooting section.

Output:

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

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

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

Next Steps

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

- Launch a Guest Book Application (p. 163) — Create a sample guest book application to test your cluster and Linux worker nodes.
- ??? (p. 62) — Deploy a sample application to test your cluster and Windows worker nodes.
- Tutorial: Deploy the Kubernetes Web UI (Dashboard) (p. 175) — This tutorial guides you through deploying the Kubernetes dashboard to your cluster.
- Using Helm with Amazon EKS (p. 172) — The `helm` package manager for Kubernetes helps you install and manage applications on your cluster.
- Installing the Kubernetes Metrics Server (p. 166) — The Kubernetes metrics server is an aggregator of resource usage data in your cluster.
- Control Plane Metrics with Prometheus (p. 168) — This topic helps you deploy Prometheus into your cluster with `helm`.

Getting Started with the AWS Management Console

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

For a simpler and more automated getting started experience, see Getting Started with `eksctl` (p. 3).

Amazon EKS Prerequisites

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

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

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

Create your Amazon EKS Service Role

To create your Amazon EKS service 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 **Allows Amazon EKS to manage your clusters on your behalf** for your use case, 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 `eksServiceRole`, then choose **Create role**.

Create your Amazon EKS Cluster VPC

This section guides you through creating a VPC for your cluster with either 3 public subnets, or two public subnets and two private subnets, which are provided with internet access through a NAT gateway. We recommend a network architecture that uses private subnets for your worker nodes, and public subnets for Kubernetes to create public load balancers within.

Choose the tab below that represents your desired VPC configuration.

Only public subnets

To create your cluster VPC with only public subnets

1. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. From the navigation bar, select a Region that supports Amazon EKS.
3. Choose **Create stack**.
4. For **Choose a template**, select **Specify an Amazon S3 template URL**.
5. Paste the following URL into the text area and choose **Next**:

```
https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2019-10-08/amazon-eks-
vpc-sample.yaml
```

6. On the **Specify Details** page, fill out the parameters accordingly, and then choose **Next**.
   - **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it **eks-vpc**.
   - **VpcBlock**: Choose a CIDR range for your VPC. You can keep the default value.
   - **Subnet01Block**: Specify a CIDR range for subnet 1. We recommend that you keep the default value so that you have plenty of IP addresses for pods to use.
   - **Subnet02Block**: Specify a CIDR range for subnet 2. We recommend that you keep the default value so that you have plenty of IP addresses for pods to use.
   - **Subnet03Block**: Specify a CIDR range for subnet 3. We recommend that you keep the default value so that you have plenty of IP addresses for pods to use.
7. (Optional) On the **Options** page, tag your stack resources. Choose **Next**.
8. On the **Review** page, choose **Create**.
9. When your stack is created, select it in the console and choose **Outputs**.
10. Record the **SecurityGroups** value for the security group that was created. You need this when you create your EKS cluster; this security group is applied to the cross-account elastic network interfaces that are created in your subnets that allow the Amazon EKS control plane to communicate with your worker nodes.
11. Record the **VpcId** for the VPC that was created. You need this when you launch your worker node group template.
12. Record the **SubnetIds** for the subnets that were created. You need this when you create your EKS cluster; these are the subnets that your worker nodes are launched into.

Public and private subnets

To create your cluster VPC with public and private subnets

1. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. From the navigation bar, select a Region that supports Amazon EKS.
3. Choose **Create stack**.
4. For **Choose a template**, select **Specify an Amazon S3 template URL**.
5. Paste the following URL into the text area and choose **Next**:

   ```
   https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2019-10-08/amazon-eks-
   vpc-private-subnets.yaml
   ```
6. On the **Specify Details** page, fill out the parameters accordingly, and then choose **Next**.
   - **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it **eks-vpc**.
   - **VpcBlock**: Choose a CIDR range for your VPC. You can keep the default value.
   - **PublicSubnet01Block**: Specify a CIDR range for public subnet 1. We recommend that you keep the default value so that you have plenty of IP addresses for pods to use.
   - **PublicSubnet02Block**: Specify a CIDR range for public subnet 2. We recommend that you keep the default value so that you have plenty of IP addresses for pods to use.
   - **PrivateSubnet01Block**: Specify a CIDR range for private subnet 1. We recommend that you keep the default value so that you have plenty of IP addresses for pods to use.
   - **PrivateSubnet02Block**: Specify a CIDR range for private subnet 2. We recommend that you keep the default value so that you have plenty of IP addresses for pods to use.
7. (Optional) On the **Options** page, tag your stack resources. Choose **Next**.
8. On the **Review** page, choose **Create**.
9. When your stack is created, select it in the console and choose **Outputs**.
10. Record the **SecurityGroups** value for the security group that was created. You need this when you create your EKS cluster; this security group is applied to the cross-account elastic network interfaces that are created in your subnets that allow the Amazon EKS control plane to communicate with your worker nodes.
11. Record the **VpcId** for the VPC that was created. You need this when you launch your worker node group template.
12. Record the **SubnetIds** for the subnets that were created. You need this when you create your EKS cluster; these are the subnets that your worker nodes are launched into.
13. Tag your private subnets so that Kubernetes knows that it can use them for internal load balancers.

    a. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.

    b. Choose **Subnets** in the left navigation.

    c. Select one of the private subnets for your Amazon EKS cluster's VPC (you can filter them with the string `PrivateSubnet`), and choose the **Tags** tab, and then **Add/Edit Tags**.

    d. Choose **Create Tag** and add the following key and value, and then choose **Save**.

Key	**Value**
`kubernetes.io/role/internal-elb`	`1`

    e. Repeat these substeps for each private subnet in your VPC.

Install and Configure **kubectl** for Amazon EKS

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

To install kubectl for Amazon EKS

- You have multiple options to download and install **kubectl** for your operating system.
  - The `kubectl` binary is available in many operating system package managers, and this option is often much easier than a manual download and install process. You can follow the instructions for your specific operating system or package manager in the Kubernetes documentation to install.
  - Amazon EKS also vends **kubectl** binaries that you can use that are identical to the upstream **kubectl** binaries with the same version. To install the Amazon EKS-vended binary for your operating system, see Installing `kubectl` (p. 146).

Install the Latest AWS CLI

To use `kubectl` with your Amazon EKS clusters, you must install a binary that can create the required client security token for cluster API server communication. The **aws eks get-token** command, available in version 1.16.232 or greater of the AWS CLI, supports client security token creation. To install or upgrade the AWS CLI, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

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

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

```
aws --version
```

> **Note**
> Your system's Python version must be 2.7.9 or greater. 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.

If you are unable to install version 1.16.232 or greater of the AWS CLI on your system, you must ensure that the AWS IAM Authenticator for Kubernetes is installed on your system. For more information, see Installing `aws-iam-authenticator` (p. 151).

Step 1: Create Your Amazon EKS Cluster

Now you can create your Amazon EKS cluster.

> **Important**
> When an Amazon EKS cluster is created, the IAM entity (user or role) that creates the cluster is added to the Kubernetes RBAC authorization table as the administrator (with `system:master` permissions. Initially, only 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. 157). If you use the console to create the cluster, you must ensure that the same IAM user credentials are in the AWS SDK credential chain when you are running **kubectl** commands on your cluster. If you install and configure the AWS CLI, you can configure the IAM credentials for your user. If the AWS CLI is configured properly for your user, then `eksctl` and the AWS IAM Authenticator

for Kubernetes can find those credentials as well. For more information, see Configuring the AWS CLI in the *AWS Command Line Interface User Guide*.

To create your cluster with the console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose **Create cluster**.

   Note
   If your IAM user does not 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. 201).
3. On the **Create cluster** page, fill in the following fields and then choose **Create**:

   - **Cluster name**: A unique name for your cluster.
   - **Kubernetes version**: The version of Kubernetes to use for your cluster. By default, the latest available version is selected.
   - **Role ARN**: Select the IAM role that you created with Create your Amazon EKS Service Role (p. 11).
   - **VPC**: The VPC you created with Create your Amazon EKS Cluster VPC (p. 12). You can find the name of your VPC in the drop-down list.
   - **Subnets**: The **SubnetIds** values (comma-separated) from the AWS CloudFormation output that you generated with Create your Amazon EKS Cluster VPC (p. 12). Specify all subnets that will host resources for your cluster (such as private subnets for worker nodes and public subnets for load balancers). By default, the available subnets in the VPC specified in the previous field are preselected.
   - **Security Groups**: The **SecurityGroups** value from the AWS CloudFormation output that you generated with Create your Amazon EKS Cluster VPC (p. 12). This security group has **ControlPlaneSecurityGroup** in the drop-down name.

     Important
     The worker node AWS CloudFormation template modifies the security group that you specify here, so **Amazon EKS strongly recommends that you use a dedicated security group for each cluster control plane (one per cluster)**. If this security group is shared with other resources, you might block or disrupt connections to those resources.
   - **Endpoint 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 will use the private VPC endpoint. For more information, see Amazon EKS Cluster Endpoint Access Control (p. 42).
   - **Endpoint 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. For more information, see Amazon EKS Cluster Endpoint Access Control (p. 42).
   - **Logging** – For each individual log type, choose whether the log type should be **Enabled** or **Disabled**. By default, each log type is **Disabled**. For more information, see Amazon EKS Control Plane Logging (p. 46)
   - **Tags** – (Optional) Add any tags to your cluster. For more information, see Tagging Your Amazon EKS Resources (p. 226).

   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. 235).

4. On the **Clusters** page, choose the name of your newly created cluster to view the cluster information.
5. The **Status** field shows **CREATING** until the cluster provisioning process completes. Cluster provisioning usually takes between 10 and 15 minutes.

Step 2: Create a kubeconfig File

In this section, you create a kubeconfig file for your cluster with the AWS CLI **update-kubeconfig** command. If you do not want to install the AWS CLI, or if you would prefer to create or update your kubeconfig manually, see Create a kubeconfig for Amazon EKS (p. 154).

To create your kubeconfig file with the AWS CLI

1. Ensure that you have at least version 1.16.232 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 greater. 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.

   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*.

   ```
   aws eks --region region update-kubeconfig --name cluster_name
   ```
3. Test your configuration.

   ```
   kubectl get svc
   ```

   Note
   If you receive the error "aws-iam-authenticator": executable file not found in $PATH, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing aws-iam-authenticator (p. 151).
   If you receive any other authorization or resource type errors, see Unauthorized or Access Denied (kubectl) (p. 235) in the troubleshooting section.

Output:

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

Step 3: Launch and Configure Amazon EKS Worker Nodes

Now that your VPC and Kubernetes control plane are created, you can launch and configure your worker nodes.

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

To launch your worker nodes

1. Wait for your cluster status to show as `ACTIVE`. If you launch your worker nodes before the cluster is active, the worker nodes will fail to register with the cluster and you will have to relaunch them.
2. Choose the tab below that corresponds to your cluster's Kubernetes version, then choose a **Launch workers** link that corresponds to your region and AMI type. This opens the AWS CloudFormation console and pre-populates several fields for you.

   Kubernetes version 1.14.7

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

Region	Amazon EKS-optimized AMI	with GPU support
EU (Ireland) (`eu-west-1`)	Launch workers	Launch workers
EU (London) (`eu-west-2`)	Launch workers	Launch workers
EU (Paris) (`eu-west-3`)	Launch workers	Launch workers
EU (Stockholm) (`eu-north-1`)	Launch workers	Launch workers
Middle East (Bahrain) (`me-south-1`)	Launch workers	Launch workers

Kubernetes version 1.13.11

Region	Amazon EKS-optimized AMI	with GPU support
US East (Ohio) (`us-east-2`)	Launch workers	Launch workers
US East (N. Virginia) (`us-east-1`)	Launch workers	Launch workers
US West (Oregon) (`us-west-2`)	Launch workers	Launch workers
Asia Pacific (Hong Kong) (`ap-east-1`)	Launch workers	Launch workers
Asia Pacific (Mumbai) (`ap-south-1`)	Launch workers	Launch workers
Asia Pacific (Tokyo) (`ap-northeast-1`)	Launch workers	Launch workers
Asia Pacific (Seoul) (`ap-northeast-2`)	Launch workers	Launch workers
Asia Pacific (Singapore) (`ap-southeast-1`)	Launch workers	Launch workers
Asia Pacific (Sydney) (`ap-southeast-2`)	Launch workers	Launch workers
EU (Frankfurt) (`eu-central-1`)	Launch workers	Launch workers
EU (Ireland) (`eu-west-1`)	Launch workers	Launch workers
EU (London) (`eu-west-2`)	Launch workers	Launch workers
EU (Paris) (`eu-west-3`)	Launch workers	Launch workers
EU (Stockholm) (`eu-north-1`)	Launch workers	Launch workers
Middle East (Bahrain) (`me-south-1`)	Launch workers	Launch workers

Kubernetes version 1.12.10

Region	Amazon EKS-optimized AMI	with GPU support
US East (Ohio) (`us-east-2`)	Launch workers	Launch workers
US East (N. Virginia) (`us-east-1`)	Launch workers	Launch workers
US West (Oregon) (`us-west-2`)	Launch workers	Launch workers
Asia Pacific (Hong Kong) (`ap-east-1`)	Launch workers	Launch workers
Asia Pacific (Mumbai) (`ap-south-1`)	Launch workers	Launch workers
Asia Pacific (Tokyo) (`ap-northeast-1`)	Launch workers	Launch workers
Asia Pacific (Seoul) (`ap-northeast-2`)	Launch workers	Launch workers
Asia Pacific (Singapore) (`ap-southeast-1`)	Launch workers	Launch workers
Asia Pacific (Sydney) (`ap-southeast-2`)	Launch workers	Launch workers
EU (Frankfurt) (`eu-central-1`)	Launch workers	Launch workers
EU (Ireland) (`eu-west-1`)	Launch workers	Launch workers
EU (London) (`eu-west-2`)	Launch workers	Launch workers
EU (Paris) (`eu-west-3`)	Launch workers	Launch workers
EU (Stockholm) (`eu-north-1`)	Launch workers	Launch workers
Middle East (Bahrain) (`me-south-1`)	Launch workers	Launch workers

Kubernetes version 1.11.10

Region	Amazon EKS-optimized AMI	with GPU support
US East (Ohio) (`us-east-2`)	Launch workers	Launch workers
US East (N. Virginia) (`us-east-1`)	Launch workers	Launch workers
US West (Oregon) (`us-west-2`)	Launch workers	Launch workers

Region	Amazon EKS-optimized AMI	with GPU support
Asia Pacific (Hong Kong) (`ap-east-1`)	Launch workers	Launch workers
Asia Pacific (Mumbai) (`ap-south-1`)	Launch workers	Launch workers
Asia Pacific (Tokyo) (`ap-northeast-1`)	Launch workers	Launch workers
Asia Pacific (Seoul) (`ap-northeast-2`)	Launch workers	Launch workers
Asia Pacific (Singapore) (`ap-southeast-1`)	Launch workers	Launch workers
Asia Pacific (Sydney) (`ap-southeast-2`)	Launch workers	Launch workers
EU (Frankfurt) (`eu-central-1`)	Launch workers	Launch workers
EU (Ireland) (`eu-west-1`)	Launch workers	Launch workers
EU (London) (`eu-west-2`)	Launch workers	Launch workers
EU (Paris) (`eu-west-3`)	Launch workers	Launch workers
EU (Stockholm) (`eu-north-1`)	Launch workers	Launch workers
Middle East (Bahrain) (`me-south-1`)	Launch workers	Launch workers

Note
If you intend to only deploy worker nodes to private subnets, you should edit this template in the AWS CloudFormation designer and modify the `AssociatePublicIpAddress` parameter in the `NodeLaunchConfig` to be `false`.

```
AssociatePublicIpAddress: 'false'
```

3. On the **Quick create stack** page, fill out the following parameters accordingly.

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

     Important
     This name must exactly match the name you used in Step 1: Create Your Amazon EKS Cluster (p. 14); otherwise, your worker nodes cannot join the cluster.
   - **ClusterControlPlaneSecurityGroup**: Choose the **SecurityGroups** value from the AWS CloudFormation output that you generated with Create your Amazon EKS Cluster VPC (p. 12).
   - **NodeGroupName**: Enter a name for your node group. This name can be used later to identify the Auto Scaling node group that is created for your worker nodes.
   - **NodeAutoScalingGroupMinSize**: Enter the minimum number of nodes that your worker node Auto Scaling group can scale in to.

- **NodeAutoScalingGroupDesiredCapacity**: Enter the desired number of nodes to scale to when your stack is created.
- **NodeAutoScalingGroupMaxSize**: Enter the maximum number of nodes that your worker node Auto Scaling group can scale out to.
- **NodeInstanceType**: Choose an instance type for your worker nodes.

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

- **NodeImageIdSSMParam**: Pre-populated based on the version that you launched your worker nodes with in step 2. This value is the Amazon EC2 Systems Manager Parameter Store parameter to use for your worker node AMI ID. For example, the `/aws/service/eks/optimized-ami/1.14/amazon-linux-2/recommended/image_id` parameter is for the latest recommended Kubernetes version 1.14 Amazon EKS-optimized AMI.

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

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

  > **Note**
  >
  > If you do not provide a keypair here, the AWS CloudFormation stack creation fails.

- **BootstrapArguments**: Specify any optional arguments to pass to the worker node bootstrap script, such as extra **kubelet** arguments. For more information, view the bootstrap script usage information at https://github.com/awslabs/amazon-eks-ami/blob/master/files/bootstrap.sh
- **VpcId**: Enter the ID for the VPC that you created in Create your Amazon EKS Cluster VPC (p. 12).
- **Subnets**: Choose the subnets that you created in Create your Amazon EKS Cluster VPC (p. 12). If you created your VPC using the steps described at Creating a VPC for Your Amazon EKS Cluster (p. 124), then specify only the private subnets within the VPC for your worker nodes to launch into.

4. Acknowledge that the stack might create IAM resources, and then choose **Create stack**.
5. When your stack has finished creating, select it in the console and choose the **Outputs** tab.
6. Record the **NodeInstanceRole** for the node group that was created. You need this when you configure your Amazon EKS worker nodes.

To enable worker nodes to join your cluster

1. Download, edit, and apply the AWS authenticator configuration map:

   a. Download the configuration map with the following command:

   ```
   curl -o aws-auth-cm.yaml https://amazon-eks.s3-us-west-2.amazonaws.com/
   cloudformation/2019-10-08/aws-auth-cm.yaml
   ```

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

   Important
   Do not modify any other lines in this file.

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

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

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

   Note
   If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 151).
   If you receive any other authorization or resource type errors, see Unauthorized or Access Denied (`kubectl`) (p. 235) in the troubleshooting section.

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

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

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

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

(Optional) To launch Windows worker nodes

Add Windows support to your cluster and launch Windows worker nodes. For more information, see Windows Support (p. 58). All Amazon EKS clusters must contain at least one Linux worker node, even if you only want to run Windows workloads in your cluster.

Next Steps

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

- Launch a Guest Book Application (p. 163) — Create a sample guest book application to test your cluster and Linux worker nodes.
- ??? (p. 62) — Deploy a sample application to test your cluster and Windows worker nodes.

- Tutorial: Deploy the Kubernetes Web UI (Dashboard) (p. 175) — This tutorial guides you through deploying the Kubernetes dashboard to your cluster.
- Using Helm with Amazon EKS (p. 172) — The `helm` package manager for Kubernetes helps you install and manage applications on your cluster.
- Installing the Kubernetes Metrics Server (p. 166) — The Kubernetes metrics server is an aggregator of resource usage data in your cluster.
- Control Plane Metrics with Prometheus (p. 168) — This topic helps you deploy Prometheus into your cluster with `helm`.

Amazon EKS Clusters

An Amazon EKS cluster consists of two primary components:

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

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

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

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

Topics

- Creating an Amazon EKS Cluster (p. 24)
- Updating an Amazon EKS Cluster Kubernetes Version (p. 33)
- Amazon EKS Cluster Endpoint Access Control (p. 42)
- Amazon EKS Control Plane Logging (p. 46)
- Deleting a Cluster (p. 48)
- Amazon EKS Kubernetes Versions (p. 51)
- Platform Versions (p. 53)
- Windows Support (p. 58)

Creating an Amazon EKS Cluster

This topic walks you through creating an Amazon EKS cluster.

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

> **Important**
> When an Amazon EKS cluster is created, the IAM entity (user or role) that creates the cluster is added to the Kubernetes RBAC authorization table as the administrator (with `system:master` permissions. Initially, only 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. 157). If you use the console to create the cluster, you must ensure that the same IAM user credentials are in the AWS SDK credential chain when you are running **kubectl** commands on your cluster. If you install and configure the AWS CLI, you can configure the IAM credentials for your user. If the AWS CLI is configured properly for your user, then `eksctl` and the AWS IAM Authenticator for Kubernetes can find those credentials as well. For more information, see Configuring the AWS CLI in the *AWS Command Line Interface User Guide*.

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

eksctl

To create your cluster and worker nodes with `eksctl`

1. Choose a tab below that matches your workload requirements. If you only intend to run Linux workloads on your cluster, choose **Linux**. If you want to run Linux and Windows workloads on your cluster, choose **Windows**.

   Linux

   This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.6.0`. 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. 161).

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

   Important

   Amazon EKS will deprecate Kubernetes version 1.11 on November 4th, 2019. On this day, you will no longer be able to create new 1.11 clusters, and all Amazon EKS clusters running Kubernetes version 1.11 will be updated to the latest available platform version of Kubernetes version 1.12. For more information, see Amazon EKS Version Deprecation (p. 53).
   Kubernetes version 1.10 is no longer supported on Amazon EKS. You can no longer create new 1.10 clusters, and all existing Amazon EKS clusters running Kubernetes version 1.10 will eventually be automatically updated to the latest available platform version of Kubernetes version 1.11. For more information, see Amazon EKS Version Deprecation (p. 53).
   Please update any 1.10 clusters to version 1.11 or higher in order to avoid service interruption. For more information, see Updating an Amazon EKS Cluster Kubernetes Version (p. 33).

   ```
   eksctl create cluster \
   --name prod \
   --version 1.14 \
   --nodegroup-name standard-workers \
   --node-type t3.medium \
   --nodes 3 \
   --nodes-min 1 \
   --nodes-max 4 \
   --node-ami auto
   ```

   Note

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

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

   Output:

   ```
   [#]  using region us-west-2
   [#]  setting availability zones to [us-west-2b us-west-2c us-west-2d]
   [#]  subnets for us-west-2b - public:192.168.0.0/19 private:192.168.96.0/19
   [#]  subnets for us-west-2c - public:192.168.32.0/19 private:192.168.128.0/19
   ```

```
[#]  subnets for us-west-2d - public:192.168.64.0/19 private:192.168.160.0/19
[#]  nodegroup "standard-workers" will use
 "ami-0923e4b35a30a5f53" [AmazonLinux2/1.12]
[#]  creating EKS cluster "prod" in "us-west-2" region
[#]  will create 2 separate CloudFormation stacks for cluster itself and the
 initial nodegroup
[#]  if you encounter any issues, check CloudFormation console or try 'eksctl
 utils describe-stacks --region=us-west-2 --name=prod'
[#]  building cluster stack "eksctl-prod-cluster"
[#]  creating nodegroup stack "eksctl-prod-nodegroup-standard-workers"
[#]  all EKS cluster resource for "prod" had been created
[#]  saved kubeconfig as "/Users/username/.kube/config"
[#]  adding role "arn:aws:iam::111122223333:role/eksctl-prod-nodegroup-standard-
wo-NodeInstanceRole-IJP4S12W3020" to auth ConfigMap
[#]  nodegroup "standard-workers" has 0 node(s)
[#]  waiting for at least 1 node(s) to become ready in "standard-workers"
[#]  nodegroup "standard-workers" has 2 node(s)
[#]  node "ip-192-168-22-17.us-west-2.compute.internal" is not ready
[#]  node "ip-192-168-32-184.us-west-2.compute.internal" is ready
[#]  kubectl command should work with "/Users/username/.kube/config", try
 'kubectl get nodes'
[#]  EKS cluster "prod" in "us-west-2" region is ready
```

Windows

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.7.0`. 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. 161).

Retrieve the Windows worker node AMI ID that you want to use with your cluster.

Kubernetes version 1.14.6

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

Region	Amazon EKS-optimized Windows Server 2019 Full	Amazon EKS-optimized Windows Server 2019 Core
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
EU (Frankfurt) (`eu-central-1`)	View AMI ID	View AMI ID
EU (Ireland) (`eu-west-1`)	View AMI ID	View AMI ID
EU (London) (`eu-west-2`)	View AMI ID	View AMI ID
EU (Paris) (`eu-west-3`)	View AMI ID	View AMI ID
EU (Stockholm) (`eu-north-1`)	View AMI ID	View AMI ID
Middle East (Bahrain) (`me-south-1`)	View AMI ID	View AMI ID

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

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

metadata:
  name: windows-prod
  region: us-west-2
  version: '1.14'

nodeGroups:
  - name: linux-ng
    instanceType: t2.large
    minSize: 2
  - name: windows-ng
    instanceType: m5.large
    minSize: 2
    volumeSize: 100
    ami: ami-0c7f1b5f1bebccac2
    amiFamily: WindowsServer2019FullContainer
```

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

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

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

Output:

```
[#]  using region us-west-2
[#]  setting availability zones to [us-west-2a us-west-2d us-west-2c]
[#]  subnets for us-west-2a - public:192.168.0.0/19 private:192.168.96.0/19
[#]  subnets for us-west-2d - public:192.168.32.0/19 private:192.168.128.0/19
[#]  subnets for us-west-2c - public:192.168.64.0/19 private:192.168.160.0/19
[#]  nodegroup "linux-ng" will use "ami-076c743acc3ec4159" [AmazonLinux2/1.14]
[#]  nodegroup "windows-ng" will use
 "ami-0c7f1b5f1bebccac2" [WindowsServer2019FullContainer/1.14]
[#]  using Kubernetes version 1.14
[#]  creating EKS cluster "windows-cluster" in "us-west-2" region
[#]  2 nodegroups (linux-ng, windows-ng) were included (based on the include/
exclude rules)
[#]  will create a CloudFormation stack for cluster itself and 2 nodegroup
 stack(s)
[#]  if you encounter any issues, check CloudFormation console or try 'eksctl
 utils describe-stacks --region=us-west-2 --name=windows-cluster'
[#]  CloudWatch logging will not be enabled for cluster "windows-cluster" in
 "us-west-2"
[#]  you can enable it with 'eksctl utils update-cluster-logging --region=us-
west-2 --name=windows-cluster'
[#]  3 sequential tasks: { create cluster control plane "windows-cluster", 2
 parallel sub-tasks: { create nodegroup "linux-ng", create nodegroup "windows-
ng" }, install Windows VPC controller }
[#]  building cluster stack "eksctl-windows-cluster-cluster"
[#]  deploying stack "eksctl-windows-cluster-cluster"
[#]  building nodegroup stack "eksctl-windows-cluster-nodegroup-linux-ng"
[#]  building nodegroup stack "eksctl-windows-cluster-nodegroup-linux-ng"
0m[#]  --nodes-max=2 was set automatically for nodegroup windows-ng
[#]  --nodes-max=2 was set automatically for nodegroup linux-ng
[#]  deploying stack "eksctl-windows-cluster-nodegroup-windows-ng"
[#]  deploying stack "eksctl-windows-cluster-nodegroup-linux-ng"
[#]  created "ClusterRole.rbac.authorization.k8s.io/vpc-resource-controller"
[#]  created "ClusterRoleBinding.rbac.authorization.k8s.io/vpc-resource-
controller"
[#]  created "kube-system:ServiceAccount/vpc-resource-controller"
[#]  created "kube-system:Deployment.apps/vpc-resource-controller"
[#]  created "CertificateSigningRequest.certificates.k8s.io/vpc-admission-
webhook.kube-system"
[#]  created "kube-system:secret/vpc-admission-webhook-certs"
[#]  created "kube-system:Service/vpc-admission-webhook"
[#]  created "kube-system:Deployment.apps/vpc-admission-webhook"
[#]  created "kube-
system:MutatingWebhookConfiguration.admissionregistration.k8s.io/vpc-admission-
webhook-cfg"
[#]  all EKS cluster resources for "windows-cluster" have been created
[#]  saved kubeconfig as "C:\\Users\\username/.kube/config"
[#]  adding role "arn:aws:iam::123456789012:role/eksctl-windows-cluster-
nodegroup-NodeInstanceRole-ZR93IIUZSYPR" to auth ConfigMap
[#]  nodegroup "linux-ng" has 0 node(s)
[#]  waiting for at least 2 node(s) to become ready in "linux-ng"
[#]  nodegroup "linux-ng" has 2 node(s)
```

```
[#]  node "ip-192-168-8-247.us-west-2.compute.internal" is ready
[#]  node "ip-192-168-80-253.us-west-2.compute.internal" is ready
[#]  adding role "arn:aws:iam::123456789012:role/eksctl-windows-cluster-
nodegroup-NodeInstanceRole-XM9UZN3NXBOB" to auth ConfigMap
[#]  nodegroup "windows-ng" has 0 node(s)
[#]  waiting for at least 2 node(s) to become ready in "windows-ng"
[#]  nodegroup "windows-ng" has 2 node(s)
[#]  node "ip-192-168-4-192.us-west-2.compute.internal" is ready
[#]  node "ip-192-168-63-224.us-west-2.compute.internal" is ready
[#]  kubectl command should work with "C:\\Users\\username/.kube/config", try
 'kubectl get nodes'
[#]  EKS cluster "windows-cluster" in "us-west-2" region is ready
```

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

   ```
   kubectl get svc
   ```

   Note
   If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 151).
   If you receive any other authorization or resource type errors, see Unauthorized or Access Denied (`kubectl`) (p. 235) in the troubleshooting section.

   Output:

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

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

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

AWS Management Console

To create your cluster with the console

This procedure has the following prerequisites:

- You have created a VPC and a dedicated security group that meet the requirements for an Amazon EKS cluster. For more information, see Cluster VPC Considerations (p. 126) and Cluster Security Group Considerations (p. 128). The Getting Started with the AWS Management Console (p. 11) guide creates a VPC that meets the requirements, or you can also follow Creating a VPC for Your Amazon EKS Cluster (p. 124) to create one.
- You have created an Amazon EKS service role to apply to your cluster. The Getting Started with Amazon EKS (p. 3) guide creates a service role for you, or you can also follow Amazon EKS IAM Roles (p. 200) to create one manually.

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose **Create cluster**.

> **Note**
> If your IAM user doesn't have administrative privileges, you must explicitly add permissions for that user to call the Amazon EKS API operations. For more information, see Amazon EKS Identity-Based Policy Examples (p. 201).

3. On the **Create cluster** page, fill in the following fields and then choose **Create**:

   - **Cluster name** – A unique name for your cluster.
   - **Kubernetes version** – The version of Kubernetes to use for your cluster. Unless you require a specific Kubernetes version for your application, we recommend that you use the latest version available in Amazon EKS.

     > **Important**
     > Amazon EKS will deprecate Kubernetes version 1.11 on November 4th, 2019. On this day, you will no longer be able to create new 1.11 clusters, and all Amazon EKS clusters running Kubernetes version 1.11 will be updated to the latest available platform version of Kubernetes version 1.12. For more information, see Amazon EKS Version Deprecation (p. 53).
     > Kubernetes version 1.10 is no longer supported on Amazon EKS. You can no longer create new 1.10 clusters, and all existing Amazon EKS clusters running Kubernetes version 1.10 will eventually be automatically updated to the latest available platform version of Kubernetes version 1.11. For more information, see Amazon EKS Version Deprecation (p. 53).
     > Please update any 1.10 clusters to version 1.11 or higher in order to avoid service interruption. For more information, see Updating an Amazon EKS Cluster Kubernetes Version (p. 33).
   - **Role name** – Choose the Amazon EKS service role to allow Amazon EKS and the Kubernetes control plane to manage AWS resources on your behalf. For more information, see Amazon EKS IAM Roles (p. 200).
   - **VPC** – The VPC to use for your cluster.
   - **Subnets** – The subnets within the preceding VPC to use for your cluster. By default, the available subnets in the VPC are preselected. Specify all subnets that will host resources for your cluster (such as private subnets for worker nodes and public subnets for load balancers). Your subnets must meet the requirements for an Amazon EKS cluster. For more information, see Cluster VPC Considerations (p. 126).
   - **Security Groups** – Specify one or more (up to a limit of five) security groups within the preceding VPC to apply to the cross-account elastic network interfaces for your cluster. Your cluster and worker node security groups must meet the requirements for an Amazon EKS cluster. For more information, see Cluster Security Group Considerations (p. 128).

     > **Important**
     > The worker node AWS CloudFormation template modifies the security group that you specify here, so **Amazon EKS strongly recommends that you use a dedicated security group for each cluster control plane (one per cluster)**. If this security group is shared with other resources, you might block or disrupt connections to those resources.
   - **Endpoint 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. For more information, see Amazon EKS Cluster Endpoint Access Control (p. 42).
   - **Endpoint 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 receive only requests from within the cluster VPC. For more information, see Amazon EKS Cluster Endpoint Access Control (p. 42).

- **Logging** – For each individual log type, choose whether the log type should be **Enabled** or **Disabled**. By default, each log type is **Disabled**. For more information, see Amazon EKS Control Plane Logging (p. 46).
- **Tags** – (Optional) Add any tags to your cluster. For more information, see Tagging Your Amazon EKS Resources (p. 226).

> **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. 235).

4. On the **Clusters** page, choose the name of your new cluster to view the cluster information.
5. The **Status** field shows **CREATING** until the cluster provisioning process completes. When your cluster provisioning is complete (usually between 10 and 15 minutes), note the **API server endpoint** and **Certificate authority** values. These are used in your **kubectl** configuration.
6. Now that you have created your cluster, follow the procedures in Installing `aws-iam-authenticator` (p. 151) and Create a `kubeconfig` for Amazon EKS (p. 154) to enable communication with your new cluster.
7. After you enable communication, follow the procedures in Launching Amazon EKS Linux Worker Nodes (p. 76) to add Linux worker nodes to your cluster to support your workloads.
8. (Optional) After you add Linux worker nodes to your cluster, follow the procedures in Windows Support (p. 58) to add Windows support to your cluster and to add Windows worker nodes. All Amazon EKS clusters must contain at least one Linux worker node, even if you only want to run Windows workloads in your cluster.

AWS CLI

To create your cluster with the AWS CLI

This procedure has the following prerequisites:

- You have created a VPC and a dedicated security group that meets the requirements for an Amazon EKS cluster. For more information, see Cluster VPC Considerations (p. 126) and Cluster Security Group Considerations (p. 128). The Getting Started with the AWS Management Console (p. 11) guide creates a VPC that meets the requirements, or you can also follow Creating a VPC for Your Amazon EKS Cluster (p. 124) to create one.
- You have created an Amazon EKS service role to apply to your cluster. The Getting Started with Amazon EKS (p. 3) guide creates a service role for you, or you can also follow Amazon EKS IAM Roles (p. 200) to create one manually.

1. Create your cluster with the following command. Substitute your cluster name, the Amazon Resource Name (ARN) of your Amazon EKS service role that you created in Create your Amazon EKS Service Role (p. 11), and the subnet and security group IDs for the VPC that you created in Create your Amazon EKS Cluster VPC (p. 12).

> **Important**
> Amazon EKS will deprecate Kubernetes version 1.11 on November 4th, 2019. On this day, you will no longer be able to create new 1.11 clusters, and all Amazon EKS clusters running Kubernetes version 1.11 will be updated to the latest available platform version of Kubernetes version 1.12. For more information, see Amazon EKS Version Deprecation (p. 53).
> Kubernetes version 1.10 is no longer supported on Amazon EKS. You can no longer create new 1.10 clusters, and all existing Amazon EKS clusters running Kubernetes

version 1.10 will eventually be automatically updated to the latest available platform version of Kubernetes version 1.11. For more information, see Amazon EKS Version Deprecation (p. 53).
Please update any 1.10 clusters to version 1.11 or higher in order to avoid service interruption. For more information, see Updating an Amazon EKS Cluster Kubernetes Version (p. 33).

```
aws eks --region region create-cluster --name devel --kubernetes-version 1.14 \
--role-arn arn:aws:iam::111122223333:role/eks-service-role-
AWSServiceRoleForAmazonEKS-EXAMPLEBKZRQR \
--resources-vpc-config subnetIds=subnet-
a9189fe2,subnet-50432629,securityGroupIds=sg-f5c54184
```

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

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

Note
If your IAM user doesn't have administrative privileges, you must explicitly add permissions for that user to call the Amazon EKS API operations. For more information, see Amazon EKS Identity-Based Policy Examples (p. 201).

Output:

```
{
    "cluster": {
        "name": "devel",
        "arn": "arn:aws:eks:us-west-2:111122223333:cluster/devel",
        "createdAt": 1527785885.159,
        "version": "1.14",
        "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. 235).

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

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

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

   a. Retrieve the `endpoint`.

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

   b. Retrieve the `certificateAuthority.data`.

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

4. Now that you have created your cluster, follow the procedures in Installing `aws-iam-authenticator` (p. 151) and Create a `kubeconfig` for Amazon EKS (p. 154) to enable communication with your new cluster.
5. After you enable communication, follow the procedures in Launching Amazon EKS Linux Worker Nodes (p. 76) to add worker nodes to your cluster to support your workloads.
6. (Optional) After you add Linux worker nodes to your cluster, follow the procedures in Windows Support (p. 58) to add Windows support to your cluster and to add Windows worker nodes. All Amazon EKS clusters must contain at least one Linux worker node, even if you only want to run Windows workloads in your cluster.

Updating an Amazon EKS Cluster Kubernetes Version

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

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

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

> **Note**
> Although Amazon EKS runs a highly available control plane, you might experience minor service interruptions during an update. For example, if you attempt to connect to an API server just

before or just after it's terminated and replaced by a new API server running the new version of Kubernetes, you might experience API call errors or connectivity issues. If this happens, retry your API operations until they succeed.

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

Kubernetes Version	**1.14**	**1.13**	**1.12**	**1.11**
Amazon VPC CNI plug-in	We recommend the latest available CNI version (1.5.4)			
DNS	CoreDNS 1.3.1	CoreDNS 1.2.6	CoreDNS 1.2.2	CoreDNS 1.1.3
KubeProxy	1.14.6	1.13.10	1.12.6	1.11.8

Important
Amazon EKS will deprecate Kubernetes version 1.11 on November 4th, 2019. On this day, you will no longer be able to create new 1.11 clusters, and all Amazon EKS clusters running Kubernetes version 1.11 will be updated to the latest available platform version of Kubernetes version 1.12. For more information, see Amazon EKS Version Deprecation (p. 53).
Kubernetes version 1.10 is no longer supported on Amazon EKS. You can no longer create new 1.10 clusters, and all existing Amazon EKS clusters running Kubernetes version 1.10 will eventually be automatically updated to the latest available platform version of Kubernetes version 1.11. For more information, see Amazon EKS Version Deprecation (p. 53).
Please update any 1.10 clusters to version 1.11 or higher in order to avoid service interruption. For more information, see Updating an Amazon EKS Cluster Kubernetes Version (p. 33).

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.

Choose the tab below that corresponds to your desired cluster update method:

eksctl

To update an existing cluster with `eksctl`

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.6.0`. 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. 161).

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

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

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

   ```
   kubectl version --short
   ```

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

  ```
  kubectl get nodes
  ```

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

  We recommend that you update your worker nodes to your cluster's current pre-update Kubernetes minor version prior to your cluster update. Your worker nodes must not run a newer Kubernetes version than your control plane. For example, if your control plane is running version 1.13 and your workers are running version 1.11, update your worker nodes to version 1.12 or 1.13 (recommended) before you update your cluster's Kubernetes version to 1.14. For more information, see Worker Node Updates (p. 89).

2. Update your Amazon EKS cluster Kubernetes version with the following command, replacing *dev* with your cluster name:

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

   This process takes several minutes to complete.

3. Patch the `kube-proxy` daemonset to use the image that corresponds to your current cluster Kubernetes version (in this example, *1.14.6*).

Kubernetes Version	1.14	1.13	1.12	1.11
KubeProxy	1.14.6	1.13.10	1.12.6	1.11.8

   ```
   kubectl set image daemonset.apps/kube-proxy \
   -n kube-system \
   kube-proxy=602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/kube-proxy:v1.14.6
   ```

4. 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, 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 CoreDNS (p. 140) to install CoreDNS on your cluster and then return here.

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

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

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

- **Kubernetes 1.14:** `1.3.1`
- **Kubernetes 1.13:** `1.2.6`
- **Kubernetes 1.12:** `1.2.2`
- **Kubernetes 1.11:** `1.1.3`

If your current `coredns` version doesn't match the recommendation for your cluster version, update the `coredns` deployment to use the recommended image.

```
kubectl set image --namespace kube-system deployment.apps/coredns \
coredns=602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/coredns:v1.3.1
```

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

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

Output:

```
amazon-k8s-cni:1.5.3
```

If your CNI version is earlier than 1.5.4, use the following command to upgrade your CNI version to the latest version:

- For Kubernetes 1.10 clusters:

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
release-1.5/config/v1.5/aws-k8s-cni-1.10.yaml
```

- For all other Kubernetes versions:

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
release-1.5/config/v1.5/aws-k8s-cni.yaml
```

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

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

8. After your cluster update is complete, update your worker nodes to the same Kubernetes version of your updated cluster. For more information, see Worker Node Updates (p. 89).

AWS Management Console

To update an existing cluster with the console

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

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

```
kubectl version --short
```

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

```
kubectl get nodes
```

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

We recommend that you update your worker nodes to your cluster's current pre-update Kubernetes minor version prior to your cluster update. Your worker nodes must not run a newer Kubernetes version than your control plane. For example, if your control plane is running version 1.13 and your workers are running version 1.11, update your worker nodes to version 1.12 or 1.13 (recommended) before you update your cluster’s Kubernetes version to 1.14. For more information, see Worker Node Updates (p. 89).

2. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
3. Choose the name of the cluster to update and choose **Update cluster version**.
4. For **Kubernetes version**, select the version to update your cluster to and choose **Update**.

   Important
   Amazon EKS will deprecate Kubernetes version 1.11 on November 4th, 2019. On this day, you will no longer be able to create new 1.11 clusters, and all Amazon EKS clusters running Kubernetes version 1.11 will be updated to the latest available platform version of Kubernetes version 1.12. For more information, see Amazon EKS Version Deprecation (p. 53).
   Kubernetes version 1.10 is no longer supported on Amazon EKS. You can no longer create new 1.10 clusters, and all existing Amazon EKS clusters running Kubernetes version 1.10 will eventually be automatically updated to the latest available platform version of Kubernetes version 1.11. For more information, see Amazon EKS Version Deprecation (p. 53).
   Please update any 1.10 clusters to version 1.11 or higher in order to avoid service interruption. For more information, see Updating an Amazon EKS Cluster Kubernetes Version (p. 33).

   Important
   Because Amazon EKS runs a highly available control plane, you must 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 1.12 and you want to upgrade to 1.14, you must first upgrade your cluster to 1.13 and then upgrade it from 1.13 to 1.14. If you try to update directly from 1.12 to 1.14, the update version command throws an error.
5. For **Cluster name**, type the name of your cluster and choose **Confirm**.

   Note
   The cluster update should finish in a few minutes.
6. Patch the `kube-proxy` daemonset to use the image that corresponds to your current cluster Kubernetes version (in this example, *`1.14.6`*).

Kubernetes Version	1.14	1.13	1.12	1.11
KubeProxy	1.14.6	1.13.10	1.12.6	1.11.8

```
kubectl set image daemonset.apps/kube-proxy \
-n kube-system \
kube-proxy=602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/kube-proxy:v1.14.6
```

7. 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, 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 CoreDNS (p. 140) to install CoreDNS on your cluster and then return here.

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

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

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

   - **Kubernetes 1.14:** `1.3.1`
   - **Kubernetes 1.13:** `1.2.6`
   - **Kubernetes 1.12:** `1.2.2`
   - **Kubernetes 1.11:** `1.1.3`

   If your current `coredns` version doesn't match the recommendation for your cluster version, update the `coredns` deployment to use the recommended image.

   ```
   kubectl set image --namespace kube-system deployment.apps/coredns \
   coredns=602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/coredns:v1.3.1
   ```

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

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

   Output:

   ```
   amazon-k8s-cni:1.5.3
   ```

   If your CNI version is earlier than 1.5.4, use the following command to upgrade your CNI version to the latest version.

   - For Kubernetes 1.10 clusters:

     ```
     kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
     release-1.5/config/v1.5/aws-k8s-cni-1.10.yaml
     ```

- For all other Kubernetes versions:

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
release-1.5/config/v1.5/aws-k8s-cni.yaml
```

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

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

11. After your cluster update is complete, update your worker nodes to the same Kubernetes version of your updated cluster. For more information, see Worker Node Updates (p. 89).

AWS CLI

To update an existing cluster with the AWS CLI

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

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

   ```
   kubectl version --short
   ```

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

   ```
   kubectl get nodes
   ```

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

   We recommend that you update your worker nodes to your cluster's current pre-update Kubernetes minor version prior to your cluster update. Your worker nodes must not run a newer Kubernetes version than your control plane. For example, if your control plane is running version 1.13 and your workers are running version 1.11, update your worker nodes to version 1.12 or 1.13 (recommended) before you update your cluster’s Kubernetes version to 1.14. For more information, see Worker Node Updates (p. 89).

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

   Important
   Amazon EKS will deprecate Kubernetes version 1.11 on November 4th, 2019. On this day, you will no longer be able to create new 1.11 clusters, and all Amazon EKS clusters running Kubernetes version 1.11 will be updated to the latest available platform version of Kubernetes version 1.12. For more information, see Amazon EKS Version Deprecation (p. 53).
   Kubernetes version 1.10 is no longer supported on Amazon EKS. You can no longer create new 1.10 clusters, and all existing Amazon EKS clusters running Kubernetes version 1.10 will eventually be automatically updated to the latest available platform version of Kubernetes version 1.11. For more information, see Amazon EKS Version Deprecation (p. 53).

Please update any 1.10 clusters to version 1.11 or higher in order to avoid service interruption. For more information, see Updating an Amazon EKS Cluster Kubernetes Version (p. 33).

Important
Because Amazon EKS runs a highly available control plane, you must 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 1.12 and you want to upgrade to 1.14, you must first upgrade your cluster to 1.13 and then upgrade it from 1.13 to 1.14. If you try to update directly from 1.12 to 1.14, the update version command throws an error.

```
aws eks --region region update-cluster-version --name prod --kubernetes-
version 1.14
```

Output:

```
{
    "update": {
        "id": "b5f0ba18-9a87-4450-b5a0-825e6e84496f",
        "status": "InProgress",
        "type": "VersionUpdate",
        "params": [
            {
                "type": "Version",
                "value": "1.14"
            },
            {
                "type": "PlatformVersion",
                "value": "eks.1"
            }
        ],
        "createdAt": 1544051347.305,
        "errors": []
    }
}
```

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

   Note
   The cluster update should finish in a few minutes.

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

Output:

```
{
    "update": {
        "id": "b5f0ba18-9a87-4450-b5a0-825e6e84496f",
        "status": "Successful",
        "type": "VersionUpdate",
        "params": [
            {
                "type": "Version",
                "value": "1.14"
            },
            {
                "type": "PlatformVersion",
                "value": "eks.1"
            }
        ],
        "createdAt": 1544051347.305,
        "errors": []
    }
}
```

4. Patch the `kube-proxy` daemonset to use the image that corresponds to your current cluster Kubernetes version (in this example, *1.14.6*).

Kubernetes Version	1.14	1.13	1.12	1.11
KubeProxy	1.14.6	1.13.10	1.12.6	1.11.8

```
kubectl set image daemonset.apps/kube-proxy \
-n kube-system \
kube-proxy=602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/kube-proxy:v1.14.6
```

5. Check your cluster's DNS provider. Clusters that were created with Kubernetes version 1.10 shipped with `kube-dns` as the default DNS and service discovery provider. If you have updated a 1.10 cluster to a newer version and you want to use CoreDNS for DNS and service discovery, 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 CoreDNS (p. 140) to install CoreDNS on your cluster and then return here.

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

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

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

   - **Kubernetes 1.14:** `1.3.1`
   - **Kubernetes 1.13:** `1.2.6`
   - **Kubernetes 1.12:** `1.2.2`
   - **Kubernetes 1.11:** `1.1.3`

   If your current `coredns` version doesn't match the recommendation for your cluster version, update the `coredns` deployment to use the recommended image.

```
kubectl set image --namespace kube-system deployment.apps/coredns \
coredns=602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/coredns:v1.3.1
```

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

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

Output:

```
amazon-k8s-cni:1.5.3
```

If your CNI version is earlier than 1.5.4, use the following command to upgrade your CNI version to the latest version.

- For Kubernetes 1.10 clusters:

  ```
  kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
  release-1.5/config/v1.5/aws-k8s-cni-1.10.yaml
  ```
- For all other Kubernetes versions:

  ```
  kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
  release-1.5/config/v1.5/aws-k8s-cni.yaml
  ```

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

   ```
   kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/1.0.0-
   beta/nvidia-device-plugin.yml
   ```
9. After your cluster update is complete, update your worker nodes to the same Kubernetes version of your updated cluster. For more information, see Worker Node Updates (p. 89).

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 completely disable public access so that it's not accessible from the internet.

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

You can enable private access to the Kubernetes API server so that all communication between your worker nodes and the API server stays within your VPC. You can also completely disable public access to your API server so that it's not accessible from the internet.

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

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

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

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

Modifying Cluster Endpoint Access

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

API server endpoint access options

Endpoint Public Access	Endpoint Private Access	Behavior
Enabled	Disabled	• This is the default behavior for new Amazon EKS clusters. • Kubernetes API requests that originate from within your cluster's VPC (such as worker node to control plane communication) leave the VPC but not Amazon's network. • Your cluster API server is accessible from the internet.
Enabled	Enabled	• Kubernetes API requests within your cluster's VPC (such as worker node to control plane communication) use the private VPC endpoint. • Your cluster API server is accessible from the internet.
Disabled	Enabled	• All traffic to your cluster API server must come from within your cluster's VPC. • There is no public access to your API server from the internet. Any kubectl commands must come from within the VPC as well. For connectivity options, see Accessing the API Server from within the VPC (p. 45).

To modify your cluster API server endpoint access with the console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose the name of the cluster to display your cluster information.

3. Under **Networking**, choose **Update**.
4. For **Endpoint 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 **Endpoint 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. Choose **Update** to finish.

To modify your cluster API server endpoint access with 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.

   Note
   The following command enables private access for the API server endpoint and completely disables public access.

   ```
   aws eks --region region update-cluster-config --name dev --resources-vpc-config
    endpointPublicAccess=false,endpointPrivateAccess=true
   ```

   Output:

   ```
   {
       "update": {
           "id": "70e7ad6d-8de4-4ed3-9040-1ced27f8c332",
           "status": "InProgress",
           "type": "EndpointAccessUpdate",
           "params": [
               {
                   "type": "EndpointPublicAccess",
                   "value": "false"
               },
               {
                   "type": "EndpointPrivateAccess",
                   "value": "true"
               }
           ],
           "createdAt": 1551817408.563,
           "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 --region region describe-update --name dev --update-
   id 70e7ad6d-8de4-4ed3-9040-1ced27f8c332
   ```

   Output:

   ```
   {
       "update": {
           "id": "70e7ad6d-8de4-4ed3-9040-1ced27f8c332",
           "status": "Successful",
           "type": "EndpointAccessUpdate",
   ```

```
        "params": [
            {
                "type": "EndpointPublicAccess",
                "value": "false"
            },
            {
                "type": "EndpointPrivateAccess",
                "value": "true"
            }
        ],
        "createdAt": 1551817408.563,
        "errors": []
    }
}
```

Accessing the API Server from within the VPC

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. Here are a few possible ways to access the Kubernetes API server endpoint from within the VPC:

> **Note**
> You must ensure that your Amazon EKS control plane security group contains rules to allow ingress traffic for the following solutions. For example, if you are using an Amazon EC2 bastion host or AWS Cloud9 IDE to communicate with your cluster, then your control plane security group must allow ingress traffic on port 443 from your bastion host or IDE security group. For more information, see Cluster Security Group Considerations (p. 128).
> The DNS name of the Kubernetes cluster endpoint is only resolvable from the worker node VPC, for the following reasons:
>
> - The Route 53 private hosted zone that is created for the endpoint is only associated with the worker node VPC.
> - The private hosted zone is created in a separate AWS managed account and cannot be altered.
>
> If you want to reach the cluster endpoint from a peered VPC or your on premises network through AWS Direct Connect or a transit gateway, you must enable DNS resolution for the cluster endpoint to work outside of the worker node VPC. For more information, see Enabling DNS resolution for Amazon EKS cluster endpoints.

- **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.

  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. 157) and Unauthorized or Access Denied (`kubectl`) (p. 235).
- **Transit Gateway:** A transit gateway is a network transit hub that you can use to interconnect your VPCs and on-premises networks. For more information, see What is a Transit Gateway? in the *Amazon VPC Transit Gateways* documentation.
- **Amazon VPC connectivity options:** Amazon VPC provides multiple network connectivity options for you to leverage depending on your current network designs and requirements. These connectivity options include leveraging either the internet or an AWS Direct Connect connection as the network backbone and terminating the connection into either AWS or user-managed network endpoints. For more information, see Amazon Virtual Private Cloud Connectivity Options.

- **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.

  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. 157) and Unauthorized or Access Denied (`kubectl`) (p. 235).

Amazon EKS Control Plane Logging

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

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

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

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

- **Kubernetes API server component logs (`api`)** – Your cluster's API server is the control plane component that exposes the Kubernetes API. For more information, see kube-apiserver in the Kubernetes documentation.
- **Audit (`audit`)** – Kubernetes audit logs provide a record of the individual users, administrators, or system components that have affected your cluster. For more information, see Auditing in the Kubernetes documentation.
- **Authenticator (`authenticator`)** – Authenticator logs are unique to Amazon EKS. These logs represent the control plane component that Amazon EKS uses for Kubernetes Role Based Access Control (RBAC) authentication using IAM credentials. For more information, see Managing Cluster Authentication (p. 146).
- **Controller manager (`controllerManager`)** – The controller manager manages the core control loops that are shipped with Kubernetes. For more information, see kube-controller-manager in the Kubernetes documentation.
- **Scheduler (`scheduler`)** – The scheduler component manages when and where to run pods in your cluster. For more information, see kube-scheduler in the Kubernetes documentation.

Enabling and Disabling Control Plane Logs

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

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

To enable or disable control plane logs with the console

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

To enable or disable control plane logs with the AWS CLI

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

   ```
   aws --version
   ```

   If your AWS CLI version is below 1.16.139, you must first update to the latest version. To install or upgrade the AWS CLI, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.
2. Update your cluster's control plane log export configuration with the following AWS CLI command. Substitute your cluster name and desired endpoint access values.

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

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

Output:

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

Viewing Cluster Control Plane Logs

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

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

To view your cluster control plane logs on the CloudWatch console

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

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

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

Deleting a Cluster

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

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

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

eksctl

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

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.6.0`. 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. 161).

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

1. List all services running in your cluster.

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

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

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

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

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

   Output:

   ```
   [#]  using region us-west-2
   [#]  deleting EKS cluster "prod"
   [#]  will delete stack "eksctl-prod-nodegroup-standard-workers"
   [#]  waiting for stack "eksctl-prod-nodegroup-standard-workers" to get deleted
   [#]  will delete stack "eksctl-prod-cluster"
   [#]  the following EKS cluster resource(s) for "prod" will be deleted: cluster. If
    in doubt, check CloudFormation console
   ```

AWS Management Console

To delete an Amazon EKS cluster with the AWS Management Console

1. List all services running in your cluster.

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

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

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

3. Delete the worker node AWS CloudFormation stack.

   a. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
   b. Select the worker node stack to delete and then choose **Actions**, **Delete Stack**.
   c. On the **Delete Stack** confirmation screen, choose **Yes, Delete**.

4. Delete the cluster.

   a. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
   b. Select the cluster to delete and choose **Delete**.
   c. On the delete cluster confirmation screen, choose **Delete**.

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

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

AWS CLI

To delete an Amazon EKS cluster with the AWS CLI

1. List all services running in your cluster.

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

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

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

3. Delete the worker node AWS CloudFormation stack.

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

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

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

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

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

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

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

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

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

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

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

Amazon EKS Kubernetes Versions

The Kubernetes project is rapidly evolving with new features, design updates, and bug fixes. The community releases new Kubernetes minor versions, such as 1.14, as generally available approximately every three months, and each minor version is supported for approximately one year after it is first released.

Available Amazon EKS Kubernetes Versions

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

- 1.14.6
- 1.13.10
- 1.12.10
- 1.11.10

Important
Amazon EKS will deprecate Kubernetes version 1.11 on November 4th, 2019. On this day, you will no longer be able to create new 1.11 clusters, and all Amazon EKS clusters running Kubernetes version 1.11 will be updated to the latest available platform version of Kubernetes version 1.12. For more information, see Amazon EKS Version Deprecation (p. 53).
Kubernetes version 1.10 is no longer supported on Amazon EKS. You can no longer create new 1.10 clusters, and all existing Amazon EKS clusters running Kubernetes version 1.10 will eventually be automatically updated to the latest available platform version of Kubernetes version 1.11. For more information, see Amazon EKS Version Deprecation (p. 53).
Please update any 1.10 clusters to version 1.11 or higher in order to avoid service interruption. For more information, see Updating an Amazon EKS Cluster Kubernetes Version (p. 33).

Unless your application requires a specific version of Kubernetes, we recommend that you choose the latest available Kubernetes version supported by Amazon EKS for your clusters. As new Kubernetes versions become available in Amazon EKS, we recommend that you proactively update your clusters to use the latest available version. For more information, see Updating an Amazon EKS Cluster Kubernetes Version (p. 33).

Kubernetes 1.14

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

Important
The `--allow-privileged` flag has been removed from `kubelet` on Amazon EKS 1.14 worker nodes. If you have modified or restricted the Amazon EKS Default Pod Security Policy (p. 222) on your cluster, you should verify that your applications have the permissions they need on 1.14 worker nodes.

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

- Container Storage Interface Topology is in beta for Kubernetes version 1.14 clusters. For more information, see CSI Topology Feature in the Kubernetes CSI Developer Documentation. The following CSI drivers provide a CSI interface for container orchestrators like Kubernetes to manage the lifecycle of Amazon EBS volumes, Amazon EFS file systems, and Amazon FSx for Lustre file systems:
  - Amazon Elastic Block Store (EBS) CSI driver
  - Amazon EFS CSI Driver
  - Amazon FSx for Lustre CSI Driver
- Process ID (PID) limiting is in beta for Kubernetes version 1.14 clusters. This feature allows you to set quotas for how many processes a pods can create, which can prevent resource starvation for other applications on a cluster. For more information, see Process ID Limiting for Stability Improvements in Kubernetes 1.14.
- Persistent Local Volumes are now GA and make locally attached storage available as a persistent volume source. For more information, see Kubernetes 1.14: Local Persistent Volumes GA.
- Pod Priority and Preemption is now GA and allows pods to be assigned a scheduling priority level. For more information, see Pod Priority and Preemption in the Kubernetes documentation.
- Windows worker node support is GA with Kubernetes 1.14. Amazon EKS currently supports running Windows nodes and containers as part of a public preview. Official support is coming soon.

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

Kubernetes 1.13

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

- The `PodSecurityPolicy` admission controller is now enabled. This admission controller allows fine-grained control over pod creation and updates. For more information, see Pod Security Policy (p. 222).
- Amazon ECR interface VPC endpoints (AWS PrivateLink) are supported. When you enable these endpoints in your VPC, all network traffic between your VPC and Amazon ECR is restricted to the Amazon network. For more information, see Amazon ECR Interface VPC Endpoints (AWS PrivateLink) in the *Amazon Elastic Container Registry User Guide*.
- The `DryRun` feature is in beta in Kubernetes 1.13 and is enabled by default for Amazon EKS clusters. For more information, see Dry run in the Kubernetes documentation.
- The `TaintBasedEvictions` feature is in beta in Kubernetes 1.13 and is enabled by default for Amazon EKS clusters. For more information, see Taint based Evictions in the Kubernetes documentation.
- Raw block volume support is in beta in Kubernetes 1.13 and is enabled by default for Amazon EKS clusters. This is accessible via the `volumeDevices` container field in pod specs, and the `volumeMode` field in persistent volume and persistent volume claim definitions. For more information, see Raw Block Volume Support in the Kubernetes documentation.
- Node lease renewal is treated as the heartbeat signal from the node, in addition to its `NodeStatus` update. This reduces load on the control plane for large clusters. For more information, see https://github.com/kubernetes/kubernetes/pull/69241.

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

Amazon EKS Version Deprecation

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

We will announce the deprecation of a given Kubernetes minor version at least 60 days before the deprecation date. Because of the Amazon EKS qualification and release process for new Kubernetes versions, the deprecation of a Kubernetes version on Amazon EKS will be on or after the date the Kubernetes project stops supporting the version upstream.

On the deprecation date, Amazon EKS clusters running the version targeted for deprecation will begin to be updated to the next Amazon EKS-supported version of Kubernetes. This means that if the deprecated version is 1.11, clusters will eventually be automatically updated to version 1.12. If a cluster is automatically updated by Amazon EKS, you must also update the version of your worker nodes after the update is complete. For more information, see Worker Node Updates (p. 89).

Kubernetes supports compatibility between masters and workers for at least 2 minor versions, so 1.11 workers will continue to operate when orchestrated by a 1.12 control plane. For more information, see Kubernetes Version and Version Skew Support Policy in the Kubernetes documentation.

Platform Versions

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

When a new Kubernetes minor version is available in Amazon EKS, such as 1.14, the initial Amazon EKS platform version for that Kubernetes minor version starts at `eks.1`. However, Amazon EKS releases new platform versions periodically to enable new Kubernetes control plane settings and to provide security fixes.

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

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

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

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

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

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

Kubernetes version 1.14

Kubernetes Version	Amazon EKS Platform Version	Enabled Admission Controllers	Release Notes
`1.14.6`	`eks.1`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy`	Initial release of Kubernetes 1.14 for Amazon EKS. For more information, see Kubernetes 1.14 (p. 51).

Kubernetes version 1.13

Kubernetes Version	Amazon EKS Platform Version	Enabled Admission Controllers	Release Notes
`1.13.10`	`eks.4`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy`	New platform version to support IAM roles for service accounts. For more information, see IAM Roles for Service Accounts (p. 208).
`1.13.10`	`eks.3`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy`	New platform version updating Amazon EKS Kubernetes 1.13 clusters to a patched version of 1.13.10 to address CVE-2019-9512 and CVE-2019-9514.
`1.13.8`	`eks.2`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy`	New platform version updating Amazon EKS Kubernetes 1.13 clusters to a patched version of 1.13.8 to address CVE-2019-11247 and CVE-2019-11249.

Kubernetes Version	Amazon EKS Platform Version	Enabled Admission Controllers	Release Notes
`1.13.7`	`eks.1`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy`	Initial release of Kubernetes 1.13 for Amazon EKS. For more information, see Kubernetes 1.13 (p. 52).

Kubernetes version 1.12

Kubernetes Version	Amazon EKS Platform Version	Enabled Admission Controllers	Release Notes
`1.12.10`	`eks.4`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook`	New platform version updating Amazon EKS Kubernetes 1.12 clusters to a patched version of 1.12.10 to address CVE-2019-9512 and CVE-2019-9514.
`1.12.10`	`eks.3`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook`	New platform version updating Amazon EKS Kubernetes 1.12 clusters to a patched version of 1.12.10 to address CVE-2019-11247 and CVE-2019-11249.
`1.12.6`	`eks.2`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook`	New platform version to support custom DNS names in the Kubelet certificate and improve `etcd` performance. This fixes a bug that caused worker node Kubelet daemons to request a new certificate every few seconds.
`1.12.6`	`eks.1`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota,`	Initial release of Kubernetes 1.12 for Amazon EKS.

Kubernetes Version	Amazon EKS Platform Version	Enabled Admission Controllers	Release Notes
		DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook	

Kubernetes version 1.11

Kubernetes Version	Amazon EKS Platform Version	Enabled Admission Controllers	Release Notes
1.11.10	eks.5	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook	New platform version updating Amazon EKS Kubernetes 1.11 clusters to a patched version of 1.11.10 to address CVE-2019-9512 and CVE-2019-9514.
1.11.10	eks.4	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook	New platform version updating Amazon EKS Kubernetes 1.11 clusters to a patched version of 1.11.10 to address CVE-2019-11247 and CVE-2019-11249.
1.11.8	eks.3	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook	New platform version to support custom DNS names in the Kubelet certificate and improve etcd performance.
1.11.8	eks.2	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook	New platform version updating Amazon EKS Kubernetes 1.11 clusters to patch level 1.11.8 to address CVE-2019-1002100.
1.11.5	eks.1	NamespaceLifecycle, LimitRanger, ServiceAccount,	Initial release of Kubernetes 1.11 for Amazon EKS.

Kubernetes Version	Amazon EKS Platform Version	Enabled Admission Controllers	Release Notes
		`DefaultStorageClass`, `ResourceQuota`, `DefaultTolerationSeconds`, `NodeRestriction`, `MutatingAdmissionWebhook`, `ValidatingAdmissionWebhook`	

Kubernetes version 1.10

Important

Amazon EKS will deprecate Kubernetes version 1.11 on November 4th, 2019. On this day, you will no longer be able to create new 1.11 clusters, and all Amazon EKS clusters running Kubernetes version 1.11 will be updated to the latest available platform version of Kubernetes version 1.12. For more information, see Amazon EKS Version Deprecation (p. 53).

Kubernetes version 1.10 is no longer supported on Amazon EKS. You can no longer create new 1.10 clusters, and all existing Amazon EKS clusters running Kubernetes version 1.10 will eventually be automatically updated to the latest available platform version of Kubernetes version 1.11. For more information, see Amazon EKS Version Deprecation (p. 53).

Please update any 1.10 clusters to version 1.11 or higher in order to avoid service interruption. For more information, see Updating an Amazon EKS Cluster Kubernetes Version (p. 33).

Kubernetes Version	Amazon EKS Platform Version	Enabled Admission Controllers	Release Notes
`1.10.13`	`eks.5`	`NamespaceLifecycle`, `LimitRanger`, `ServiceAccount`, `DefaultStorageClass`, `ResourceQuota`, `DefaultTolerationSeconds`, `NodeRestriction`, `MutatingAdmissionWebhook`, `ValidatingAdmissionWebhook`	New platform version to support custom DNS names in the Kubelet certificate and improve `etcd` performance. Updated to `GitVersion:"v1.10.13-eks-c4a9600"` to address CVE-2019-11247 and CVE-2019-11249.
`1.10.13`	`eks.4`	`NamespaceLifecycle`, `LimitRanger`, `ServiceAccount`, `DefaultStorageClass`, `ResourceQuota`, `DefaultTolerationSeconds`, `NodeRestriction`, `MutatingAdmissionWebhook`, `ValidatingAdmissionWebhook`	New platform version updating Kubernetes to patch level 1.10.13 and a patch to address CVE-2019-1002100.
`1.10.11`	`eks.3`	`NamespaceLifecycle`, `LimitRanger`, `ServiceAccount`, `DefaultStorageClass`, `ResourceQuota`, `DefaultTolerationSeconds`, `NodeRestriction`,	New platform version updating Kubernetes to patch level 1.10.11 to address CVE-2018-1002105.

Kubernetes Version	Amazon EKS Platform Version	Enabled Admission Controllers	Release Notes
		MutatingAdmissionWebhook, ValidatingAdmissionWebhook	
1.10.3	eks.2	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook	• Added support for Kubernetes aggregation layer. • Added support for Kubernetes Horizontal Pod Autoscaler (HPA). • Kubernetes Metrics Server 0.3.0 or greater is compatible with EKS platform version eks.2.
1.10.3	eks.1	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction	Initial launch of Amazon EKS.

Windows Support

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

Considerations

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

- Windows workloads are supported with Amazon EKS clusters running Kubernetes version 1.14 or later.
- Amazon EC2 instance types C3, C4, D2, I2, M4 (excluding m4.16xlarge), and R3 instances are not supported for Windows workloads.
- Host networking mode is not supported for Windows workloads.
- Amazon EKS clusters must contain 1 or more Linux worker nodes to run core system pods that only run on Linux, such as `coredns` and the VPC resource controller.
- The `kubelet` and `kube-proxy` event logs are redirected to the Amazon EKS Windows Event Log and are set to a 200 MB limit.
- Windows worker nodes support one elastic network interface per node. The number of pods that you can run per Windows worker node is equal to the number of IP addresses available per elastic network interface for the node's instance type, minus one. For more information, see IP Addresses Per Network Interface Per Instance Type in the *Amazon EC2 User Guide for Linux Instances*.
- Calico network policy enforcement has not been tested with Amazon EKS Windows nodes.
- Group Managed Service Accounts (GMSA) for Windows pods and containers is a Kubernetes 1.14 alpha feature that is not supported by Amazon EKS. You can follow the instructions in the Kubernetes documentation to enable and test this alpha feature on your clusters.

- 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:
        beta.kubernetes.io/os: linux
        beta.kubernetes.io/arch: amd64
```

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

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

Enabling Windows Support

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

eksctl

To enable Windows support for your cluster with `eksctl`

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.7.0`. 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. 161).

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

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

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

2. After you have enabled Windows support, you can launch a Windows node group into your cluster. For more information, see Launching Amazon EKS Windows Worker Nodes (p. 83).

Windows

To enable Windows support for your cluster with a Windows client

In the following steps, replace the *us-west-2* with the region that your cluster resides in.

1. Deploy the VPC resource controller to your cluster.

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

b. Install OpenSSL and jq.

c. Setup the VPC admission webhook.

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

3. Deploy the VPC admission webhook.

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

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

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

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

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

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

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

Apply the configuration to the cluster.

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

5. After you have enabled Windows support, you can launch a Windows node group into your cluster. For more information, see Launching Amazon EKS Windows Worker Nodes (p. 83).

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 be installed on your client system.

In the following steps, replace the *`us-west-2`* with the region that your cluster resides in.

1. Deploy the VPC resource controller to your cluster.

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

   b. Add execute file permissions to the shell scripts.

   ```
   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 whether 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 contents.

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

Apply the configuration to the cluster.

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

5. After you have enabled Windows support, you can launch a Windows node group into your cluster. For more information, see Launching Amazon EKS Windows Worker Nodes (p. 83).

Deploy a Windows Sample Application

To deploy a Windows sample application

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

```
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:
        beta.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 before the IP address is 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. For example, *`http://a341875bfe61311e98376029b52cbbb6-1884437540.us-west-2.elb.amazonaws.com`*

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

Worker Nodes

Worker machines in Kubernetes are called nodes. Amazon EKS worker nodes run in your AWS account and connect to your cluster's control plane via the cluster API server endpoint.

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

By default, Amazon EKS provides AWS CloudFormation templates to spin up worker nodes in your Amazon EKS cluster. This AMI is built on top of Amazon Linux 2, and is configured to serve as the base image for Amazon EKS worker nodes. The AMI is configured to work with Amazon EKS out of the box, and it includes Docker, **kubelet**, and the AWS IAM Authenticator. The AMI also contains a specialized bootstrap script that allows it to discover and connect to your cluster's control plane automatically.

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

The AWS CloudFormation worker node template launches your worker nodes with specialized Amazon EC2 user data. This user data triggers a specialized bootstrap script that allows your worker nodes to discover and connect to your cluster's control plane automatically. For more information, see Launching Amazon EKS Linux Worker Nodes (p. 76).

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

Topics

- Amazon EKS-Optimized Linux AMI (p. 64)
- Amazon EKS-Optimized Windows AMI (p. 73)
- Amazon EKS Partner AMIs (p. 75)
- Launching Amazon EKS Linux Worker Nodes (p. 76)
- Launching Amazon EKS Windows Worker Nodes (p. 83)
- Worker Node Updates (p. 89)

Amazon EKS-Optimized Linux AMI

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

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

The AMI IDs for the latest Amazon EKS-optimized AMI (with and without GPU support (p. 68)) are shown in the following table. You can also retrieve the IDs with an Amazon EC2 Systems Manager parameter using different tools. For more information, see Retrieving Amazon EKS-Optimized AMI IDs (p. 72).

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

Kubernetes version 1.14.7

Region	Amazon EKS-optimized AMI	with GPU support
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID
EU (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID
EU (Ireland) (eu-west-1)	View AMI ID	View AMI ID
EU (London) (eu-west-2)	View AMI ID	View AMI ID
EU (Paris) (eu-west-3)	View AMI ID	View AMI ID
EU (Stockholm) (eu-north-1)	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID

Kubernetes version 1.13.11

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

Region	Amazon EKS-optimized AMI	with GPU support
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID
EU (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID
EU (Ireland) (eu-west-1)	View AMI ID	View AMI ID
EU (London) (eu-west-2)	View AMI ID	View AMI ID
EU (Paris) (eu-west-3)	View AMI ID	View AMI ID
EU (Stockholm) (eu-north-1)	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID

Kubernetes version 1.12.10

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

Region	Amazon EKS-optimized AMI	with GPU support
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID
EU (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID
EU (Ireland) (eu-west-1)	View AMI ID	View AMI ID
EU (London) (eu-west-2)	View AMI ID	View AMI ID
EU (Paris) (eu-west-3)	View AMI ID	View AMI ID
EU (Stockholm) (eu-north-1)	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID

Kubernetes version 1.11.10

Region	Amazon EKS-optimized AMI	with GPU support
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID
EU (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID
EU (Ireland) (eu-west-1)	View AMI ID	View AMI ID
EU (London) (eu-west-2)	View AMI ID	View AMI ID
EU (Paris) (eu-west-3)	View AMI ID	View AMI ID
EU (Stockholm) (eu-north-1)	View AMI ID	View AMI ID

Region	Amazon EKS-optimized AMI	with GPU support
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID

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

```
https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2019-10-08/amazon-eks-nodegroup.yaml
```

The AWS CloudFormation worker node template launches your worker nodes with Amazon EC2 user data that triggers a specialized bootstrap script. This script allows your worker nodes to discover and connect to your cluster's control plane automatically. For more information, see Launching Amazon EKS Linux Worker Nodes (p. 76).

Amazon EKS-Optimized AMI Build Scripts

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

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

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

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

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

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

Amazon EKS-Optimized AMI with GPU Support

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

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

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

The AMI IDs for the latest Amazon EKS-optimized AMI with GPU support are shown in the following table. You can also retrieve the IDs with an Amazon EC2 Systems Manager parameter using different tools. For more information, see Retrieving Amazon EKS-Optimized AMI IDs (p. 72).

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

Kubernetes version 1.14.7

Region	Amazon EKS-optimized AMI with GPU support
US East (Ohio) (`us-east-2`)	View AMI ID
US East (N. Virginia) (`us-east-1`)	View AMI ID
US West (Oregon) (`us-west-2`)	View AMI ID
Asia Pacific (Hong Kong) (`ap-east-1`)	View AMI ID
Asia Pacific (Mumbai) (`ap-south-1`)	View AMI ID
Asia Pacific (Tokyo) (`ap-northeast-1`)	View AMI ID
Asia Pacific (Seoul) (`ap-northeast-2`)	View AMI ID
Asia Pacific (Singapore) (`ap-southeast-1`)	View AMI ID
Asia Pacific (Sydney) (`ap-southeast-2`)	View AMI ID
EU (Frankfurt) (`eu-central-1`)	View AMI ID
EU (Ireland) (`eu-west-1`)	View AMI ID
EU (London) (`eu-west-2`)	View AMI ID
EU (Paris) (`eu-west-3`)	View AMI ID
EU (Stockholm) (`eu-north-1`)	View AMI ID
Middle East (Bahrain) (`me-south-1`)	View AMI ID

Kubernetes version 1.13.11

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

Region	Amazon EKS-optimized AMI with GPU support
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID
EU (Frankfurt) (eu-central-1)	View AMI ID
EU (Ireland) (eu-west-1)	View AMI ID
EU (London) (eu-west-2)	View AMI ID
EU (Paris) (eu-west-3)	View AMI ID
EU (Stockholm) (eu-north-1)	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID

Kubernetes version 1.12.10

Region	Amazon EKS-optimized AMI with GPU support
US East (Ohio) (us-east-2)	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID
EU (Frankfurt) (eu-central-1)	View AMI ID
EU (Ireland) (eu-west-1)	View AMI ID
EU (London) (eu-west-2)	View AMI ID
EU (Paris) (eu-west-3)	View AMI ID
EU (Stockholm) (eu-north-1)	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID

Kubernetes version 1.11.10

Region	Amazon EKS-optimized AMI with GPU support
US East (Ohio) (us-east-2)	View AMI ID

Region	Amazon EKS-optimized AMI with GPU support
US East (N. Virginia) (`us-east-1`)	View AMI ID
US West (Oregon) (`us-west-2`)	View AMI ID
Asia Pacific (Hong Kong) (`ap-east-1`)	View AMI ID
Asia Pacific (Mumbai) (`ap-south-1`)	View AMI ID
Asia Pacific (Tokyo) (`ap-northeast-1`)	View AMI ID
Asia Pacific (Seoul) (`ap-northeast-2`)	View AMI ID
Asia Pacific (Singapore) (`ap-southeast-1`)	View AMI ID
Asia Pacific (Sydney) (`ap-southeast-2`)	View AMI ID
EU (Frankfurt) (`eu-central-1`)	View AMI ID
EU (Ireland) (`eu-west-1`)	View AMI ID
EU (London) (`eu-west-2`)	View AMI ID
EU (Paris) (`eu-west-3`)	View AMI ID
EU (Stockholm) (`eu-north-1`)	View AMI ID
Middle East (Bahrain) (`me-south-1`)	View AMI ID

Important

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

```
https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2019-10-08/amazon-eks-nodegroup.yaml
```

The AWS CloudFormation worker node template launches your worker nodes with Amazon EC2 user data that triggers a specialized bootstrap script. This script allows your worker nodes to discover and connect to your cluster's control plane automatically. For more information, see Launching Amazon EKS Linux Worker Nodes (p. 76).

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

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

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

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

Example GPU Manifest

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

Example Get `nvidia-smi` output

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

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

Apply the manifest with the following command:

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

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

```
kubectl logs nvidia-smi
```

Output:

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

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

Retrieving Amazon EKS-Optimized AMI IDs

You can programmatically retrieve the Amazon Machine Image (AMI) ID for Amazon EKS-optimized AMIs by querying the AWS Systems Manager Parameter Store API. This parameter eliminates the need for you to manually look up Amazon EKS-optimized AMI IDs. For more information about the Systems Manager

Parameter Store API, see GetParameter. Your user account must have the `ssm:GetParameter` IAM permission to retrieve the Amazon EKS-optimized AMI metadata.

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

AWS CLI

You can retrieve the image ID of the latest recommended Amazon EKS-optimized Amazon Linux AMI with the following command by using the sub-parameter `image_id`. Replace *`1.14`* with a supported version (p. 53) and *`us-west-2`* with an Amazon EKS-supported Region for which you want the AMI ID. Replace *`amazon-linux-2`* with `amazon-linux-2-gpu` to see the AMI with GPU ID.

```
aws ssm get-parameter --name /aws/service/eks/optimized-ami/1.14/amazon-linux-2/
recommended/image_id --region us-west-2 --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 *`1.14`* with a supported version (p. 53) and *`us-west-2`* with an Amazon EKS-supported Region for which you want the AMI ID. Replace *`amazon-linux-2`* with `amazon-linux-2-gpu` to see the AMI with GPU ID.

```
https://console.aws.amazon.com/systems-manager/parameters/%252Faws%252Fservice
%252Feks%252Foptimized-ami%252F1.14%252Famazon-linux-2%252Frecommended%252Fimage_id/
description?region=us-west-2
```

AWS CloudFormation

You can launch the AWS CloudFormation console with the Amazon EKS worker node template's `NodeImageIdSSMParam` field pre-populated with the Amazon EC2 Systems Manager parameter value for the Amazon EKS recommended AMI ID . In the following link, replace *`1.14`* with a supported version (p. 53) and *`us-west-2`* in the *`?region=us-west-2#`* part of the URL with the Amazon EKS supported Region for which you want the AMI ID. Replace *`amazon-linux-2`* with `amazon-linux-2-gpu` if you want to use an AMI with GPU. Open an internet browser and enter the modified link for the pre-populated template.

```
https://console.aws.amazon.com/cloudformation/home?region=us-west-2#/stacks/create/
review?templateURL=https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/
2019-10-08/amazon-eks-nodegroup.yaml&param_NodeImageIdSSMParam=/aws/service/eks/
optimized-ami/1.14/amazon-linux-2/recommended/image_id
```

If you want to specify your own custom AMI ID, enter the ID in the `NodeImageId` field of the template instead of using the SSM parameter. The value overrides the value that is specified for the `NodeImageIdSSMParam` field.

Amazon EKS-Optimized Windows AMI

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

Note
You can track security or privacy events for Windows Server with the Microsoft Security Update Guide.

The AMI IDs for the latest Amazon EKS-optimized AMI are shown in the following table. You can also retrieve the IDs with an Amazon EC2 Systems Manager parameter using different tools. For more information, see Retrieving Amazon EKS-Optimized Windows AMI IDs (p. 74).

Kubernetes version 1.14.6

Region	Amazon EKS-optimized Windows Server 2019 Full	Amazon EKS-optimized Windows Server 2019 Core
US East (Ohio) (`us-east-2`)	View AMI ID	View AMI ID
US East (N. Virginia) (`us-east-1`)	View AMI ID	View AMI ID
US West (Oregon) (`us-west-2`)	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (`ap-east-1`)	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (`ap-south-1`)	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (`ap-northeast-1`)	View AMI ID	View AMI ID
Asia Pacific (Seoul) (`ap-northeast-2`)	View AMI ID	View AMI ID
Asia Pacific (Singapore) (`ap-southeast-1`)	View AMI ID	View AMI ID
Asia Pacific (Sydney) (`ap-southeast-2`)	View AMI ID	View AMI ID
EU (Frankfurt) (`eu-central-1`)	View AMI ID	View AMI ID
EU (Ireland) (`eu-west-1`)	View AMI ID	View AMI ID
EU (London) (`eu-west-2`)	View AMI ID	View AMI ID
EU (Paris) (`eu-west-3`)	View AMI ID	View AMI ID
EU (Stockholm) (`eu-north-1`)	View AMI ID	View AMI ID
Middle East (Bahrain) (`me-south-1`)	View AMI ID	View AMI ID

Retrieving Amazon EKS-Optimized Windows AMI IDs

You can programmatically retrieve the Amazon Machine Image (AMI) ID for Amazon EKS-optimized AMIs by querying the AWS Systems Manager Parameter Store API. This parameter eliminates the need for you to manually look up Amazon EKS-optimized AMI IDs. For more information about the Systems Manager

Parameter Store API, see GetParameter. Your user account must have the `ssm:GetParameter` IAM permission to retrieve the Amazon EKS-optimized AMI metadata.

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

AWS CLI

You can retrieve the image ID of the latest recommended Amazon EKS-optimized Windows AMI with the following command by using the sub-parameter `image_id`. Replace *`us-west-2`* 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.

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-2019-
English-Core-EKS_Optimized-1.14/image_id --region us-west-2 --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, replace *`us-west-2`* 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.

```
https://us-west-2.console.aws.amazon.com/systems-manager/parameters/%252Faws
%252Fservice%252Fami-windows-latest%252FWindows_Server-2019-English-Core-
EKS_Optimized-1.14%252Fimage_id/description?region=us-west-2
```

AWS CloudFormation

You can launch the AWS CloudFormation console with the Amazon EKS worker node template's `NodeImageIdSSMParam` field pre-populated with the Amazon EC2 Systems Manager parameter value for the Amazon EKS recommended AMI ID . In the following link, replace *`us-west-2`* in the *`?region=us-west-2#`* part of the URL with the Amazon EKS supported Region for which you want the AMI ID. Replace *`Core`* with `Full` if you want to use the Windows Server full AMI ID. Open an internet browser and enter the modified link for the pre-populated template.

```
https://console.aws.amazon.com/cloudformation/home?region=us-west-2#/stacks/
create/review?templateURL=https://amazon-eks.s3-us-west-2.amazonaws.com/
cloudformation/2019-10-08/amazon-eks-windows-nodegroup.yaml&param_NodeImageIdSSMParam=/
aws/service/ami-windows-latest/Windows_Server-2019-English-Core-EKS_Optimized-1.14/
image_id
```

If you want to specify your own custom AMI ID, enter the ID in the `NodeImageId` field of the template instead of using the SSM parameter. The value overrides the value that is specified for the `NodeImageIdSSMParam` field.

Amazon EKS Partner AMIs

In addition to the official Amazon EKS-optimized, Canonical has partnered with Amazon EKS to create worker node AMIs that you can use in your clusters.

Canonical delivers a built-for-purpose Kubernetes Node OS image. This minimized Ubuntu image is optimized for Amazon EKS and includes the custom AWS kernel that is jointly developed with AWS.

For more information, see Ubuntu and Amazon Elastic Kubernetes Service and Optimized Support for Amazon EKS on Ubuntu 18.04.

Launching Amazon EKS Linux Worker Nodes

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

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

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

Choose the tab below that corresponds to your desired worker node creation method:

eksctl

To launch worker nodes with `eksctl`

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.6.0`. 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. 161).

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

1. Create your worker node group with the following command. Replace the *`example values`* with your own values.

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

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

Output:

```
[#]  using region us-west-2
[#]  will use version 1.12 for new nodegroup(s) based on control plane version
[#]  nodegroup "standard-workers" will use
 "ami-0923e4b35a30a5f53" [AmazonLinux2/1.12]
```

```
[#]  1 nodegroup (standard-workers) was included
[#]  will create a CloudFormation stack for each of 1 nodegroups in cluster
 "default"
[#]  1 task: { create nodegroup "standard-workers" }
[#]  building nodegroup stack "eksctl-default-nodegroup-standard-workers"
[#]  deploying stack "eksctl-default-nodegroup-standard-workers"
[#]  adding role "arn:aws:iam::111122223333:role/eksctl-default-nodegroup-standard-
NodeInstanceRole-12C2JO814XSEE" to auth ConfigMap
[#]  nodegroup "standard-workers" has 0 node(s)
[#]  waiting for at least 1 node(s) to become ready in "standard-workers"
[#]  nodegroup "standard-workers" has 3 node(s)
[#]  node "ip-192-168-52-42.us-west-2.compute.internal" is ready
[#]  node "ip-192-168-7-27.us-west-2.compute.internal" is not ready
[#]  node "ip-192-168-76-138.us-west-2.compute.internal" is not ready
[#]  created 1 nodegroup(s) in cluster "default"
[#]  checking security group configuration for all nodegroups
[#]  all nodegroups have up-to-date configuration
```

2. (Optional) Launch a Guest Book Application (p. 163) — Deploy a sample application to test your cluster and Linux worker nodes.

AWS Management Console

To launch your worker nodes with the AWS Management Console

These procedures have the following prerequisites:

- You have created a VPC and security group that meet the requirements for an Amazon EKS cluster. For more information, see Cluster VPC Considerations (p. 126) and Cluster Security Group Considerations (p. 128). The Getting Started with Amazon EKS (p. 3) guide creates a VPC that meets the requirements, or you can also follow Creating a VPC for Your Amazon EKS Cluster (p. 124) to create one manually.
- You have created an Amazon EKS cluster and specified that it use the VPC and security group that meet the requirements of an Amazon EKS cluster. For more information, see Creating an Amazon EKS Cluster (p. 24).

1. Wait for your cluster status to show as `ACTIVE`. If you launch your worker nodes before the cluster is active, the worker nodes will fail to register with the cluster and you will have to relaunch them.
2. Choose the tab below that corresponds to your cluster's Kubernetes version, then choose a **Launch workers** link that corresponds to your region and AMI type. This opens the AWS CloudFormation console and pre-populates several fields for you.

   Kubernetes version 1.14.7

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

Region	Amazon EKS-optimized AMI	with GPU support
Asia Pacific (Mumbai) (`ap-south-1`)	Launch workers	Launch workers
Asia Pacific (Tokyo) (`ap-northeast-1`)	Launch workers	Launch workers
Asia Pacific (Seoul) (`ap-northeast-2`)	Launch workers	Launch workers
Asia Pacific (Singapore) (`ap-southeast-1`)	Launch workers	Launch workers
Asia Pacific (Sydney) (`ap-southeast-2`)	Launch workers	Launch workers
EU (Frankfurt) (`eu-central-1`)	Launch workers	Launch workers
EU (Ireland) (`eu-west-1`)	Launch workers	Launch workers
EU (London) (`eu-west-2`)	Launch workers	Launch workers
EU (Paris) (`eu-west-3`)	Launch workers	Launch workers
EU (Stockholm) (`eu-north-1`)	Launch workers	Launch workers
Middle East (Bahrain) (`me-south-1`)	Launch workers	Launch workers

Kubernetes version 1.13.11

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

Region	Amazon EKS-optimized AMI	with GPU support
Asia Pacific (Singapore) (`ap-southeast-1`)	Launch workers	Launch workers
Asia Pacific (Sydney) (`ap-southeast-2`)	Launch workers	Launch workers
EU (Frankfurt) (`eu-central-1`)	Launch workers	Launch workers
EU (Ireland) (`eu-west-1`)	Launch workers	Launch workers
EU (London) (`eu-west-2`)	Launch workers	Launch workers
EU (Paris) (`eu-west-3`)	Launch workers	Launch workers
EU (Stockholm) (`eu-north-1`)	Launch workers	Launch workers
Middle East (Bahrain) (`me-south-1`)	Launch workers	Launch workers

Kubernetes version 1.12.10

Region	Amazon EKS-optimized AMI	with GPU support
US East (Ohio) (`us-east-2`)	Launch workers	Launch workers
US East (N. Virginia) (`us-east-1`)	Launch workers	Launch workers
US West (Oregon) (`us-west-2`)	Launch workers	Launch workers
Asia Pacific (Hong Kong) (`ap-east-1`)	Launch workers	Launch workers
Asia Pacific (Mumbai) (`ap-south-1`)	Launch workers	Launch workers
Asia Pacific (Tokyo) (`ap-northeast-1`)	Launch workers	Launch workers
Asia Pacific (Seoul) (`ap-northeast-2`)	Launch workers	Launch workers
Asia Pacific (Singapore) (`ap-southeast-1`)	Launch workers	Launch workers
Asia Pacific (Sydney) (`ap-southeast-2`)	Launch workers	Launch workers
EU (Frankfurt) (`eu-central-1`)	Launch workers	Launch workers
EU (Ireland) (`eu-west-1`)	Launch workers	Launch workers

Region	Amazon EKS-optimized AMI	with GPU support
EU (London) (`eu-west-2`)	Launch workers	Launch workers
EU (Paris) (`eu-west-3`)	Launch workers	Launch workers
EU (Stockholm) (`eu-north-1`)	Launch workers	Launch workers
Middle East (Bahrain) (`me-south-1`)	Launch workers	Launch workers

Kubernetes version 1.11.10

Region	Amazon EKS-optimized AMI	with GPU support
US East (Ohio) (`us-east-2`)	Launch workers	Launch workers
US East (N. Virginia) (`us-east-1`)	Launch workers	Launch workers
US West (Oregon) (`us-west-2`)	Launch workers	Launch workers
Asia Pacific (Hong Kong) (`ap-east-1`)	Launch workers	Launch workers
Asia Pacific (Mumbai) (`ap-south-1`)	Launch workers	Launch workers
Asia Pacific (Tokyo) (`ap-northeast-1`)	Launch workers	Launch workers
Asia Pacific (Seoul) (`ap-northeast-2`)	Launch workers	Launch workers
Asia Pacific (Singapore) (`ap-southeast-1`)	Launch workers	Launch workers
Asia Pacific (Sydney) (`ap-southeast-2`)	Launch workers	Launch workers
EU (Frankfurt) (`eu-central-1`)	Launch workers	Launch workers
EU (Ireland) (`eu-west-1`)	Launch workers	Launch workers
EU (London) (`eu-west-2`)	Launch workers	Launch workers
EU (Paris) (`eu-west-3`)	Launch workers	Launch workers
EU (Stockholm) (`eu-north-1`)	Launch workers	Launch workers
Middle East (Bahrain) (`me-south-1`)	Launch workers	Launch workers

Note
If you intend to only deploy worker nodes to private subnets, you should edit this template in the AWS CloudFormation designer and modify the `AssociatePublicIpAddress` parameter in the `NodeLaunchConfig` to be `false`.

```
AssociatePublicIpAddress: 'false'
```

3. On the **Quick create stack** page, fill out the following parameters accordingly:

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

     Important
     This name must exactly match the name you used in Step 1: Create Your Amazon EKS Cluster (p. 14); otherwise, your worker nodes cannot join the cluster.
   - **ClusterControlPlaneSecurityGroup**: Choose the **SecurityGroups** value from the AWS CloudFormation output that you generated with Create your Amazon EKS Cluster VPC (p. 12).
   - **NodeGroupName**: Enter a name for your node group. This name can be used later to identify the Auto Scaling node group that is created for your worker nodes.
   - **NodeAutoScalingGroupMinSize**: Enter the minimum number of nodes that your worker node Auto Scaling group can scale in to.
   - **NodeAutoScalingGroupDesiredCapacity**: Enter the desired number of nodes to scale to when your stack is created.
   - **NodeAutoScalingGroupMaxSize**: Enter the maximum number of nodes that your worker node Auto Scaling group can scale out to.
   - **NodeInstanceType**: Choose an instance type for your worker nodes.

     Important
     Some instance types might not be available in all regions.
   - **NodeImageIdSSMParam**: Pre-populated based on the version that you launched your worker nodes with in step 2. This value is the Amazon EC2 Systems Manager Parameter Store parameter to use for your worker node AMI ID. For example, the `/aws/service/eks/optimized-ami/`*`1.14`*`/`*`amazon-linux-2`*`/recommended/image_id` parameter is for the latest recommended Kubernetes version 1.14 Amazon EKS-optimized AMI.

     Note
     The Amazon EKS worker node AMI is based on Amazon Linux 2. You can track security or privacy events for Amazon Linux 2 at the Amazon Linux Security Center or subscribe to the associated RSS feed. Security and privacy events include an overview of the issue, what packages are affected, and how to update your instances to correct the issue.
   - **NodeImageId**: (Optional) If you are using your own custom AMI (instead of the Amazon EKS-optimized AMI), enter a worker node AMI ID for your Region. If you specify a value here, it overrides any values in the **NodeImageIdSSMParam** field.
   - **NodeVolumeSize**: Specify a root volume size for your worker nodes, in GiB.
   - **KeyName**: Enter the name of an Amazon EC2 SSH key pair that you can use to connect using SSH into your worker nodes with after they launch. If you don't already have an Amazon EC2 keypair, you can create one in the AWS Management Console. For more information, see Amazon EC2 Key Pairs in the *Amazon EC2 User Guide for Linux Instances*.

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

- **BootstrapArguments**: Specify any optional arguments to pass to the worker node bootstrap script, such as extra **kubelet** arguments. For more information, view the bootstrap script usage information at https://github.com/awslabs/amazon-eks-ami/blob/master/files/bootstrap.sh
- **VpcId**: Enter the ID for the VPC that you created in Create your Amazon EKS Cluster VPC (p. 12).
- **Subnets**: Choose the subnets that you created in Create your Amazon EKS Cluster VPC (p. 12). If you created your VPC using the steps described at Creating a VPC for Your Amazon EKS Cluster (p. 124), then specify only the private subnets within the VPC for your worker nodes to launch into.

4. Acknowledge that the stack might create IAM resources, and then choose **Create stack**.
5. When your stack has finished creating, select it in the console and choose **Outputs**.
6. Record the **NodeInstanceRole** for the node group that was created. You need this when you configure your Amazon EKS worker nodes.

To enable worker nodes to join your cluster

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

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

   ```
   curl -o aws-auth-cm.yaml https://amazon-eks.s3-us-west-2.amazonaws.com/
   cloudformation/2019-10-08/aws-auth-cm.yaml
   ```

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

   Important
   Do not modify any other lines in this file.

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

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

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

   Note
   If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 151).
   If you receive any other authorization or resource type errors, see Unauthorized or Access Denied (`kubectl`) (p. 235) in the troubleshooting section.

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

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

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

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

4. (Optional) Launch a Guest Book Application (p. 163) — Deploy a sample application to test your cluster and Linux worker nodes.

Launching Amazon EKS Windows Worker Nodes

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

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

You must also enable Windows support for your cluster before you launch a Windows worker node group. For more information, see Enabling Windows Support (p. 59).

Choose the tab below that corresponds to your desired worker node creation method:

eksctl

If you don't already have an Amazon EKS cluster and a Linux worker node group to add a Windows worker node group to, then we recommend that you follow the Getting Started with `eksctl` (p. 3) guide instead. The guide provides a complete end-to-end walkthrough for creating an Amazon EKS cluster with Linux and Windows worker nodes. If you have an existing Amazon EKS cluster and a Linux worker node group to add a Windows worker node group to, then complete the following steps to add the Windows worker node group.

To launch Windows worker nodes with `eksctl`

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.7.0`. 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. 161).

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

1. Retrieve the Windows worker node AMI ID that you want to use with your cluster.

   Kubernetes version 1.14.6

Region	Amazon EKS-optimized Windows Server 2019 Full	Amazon EKS-optimized Windows Server 2019 Core
US East (Ohio) (`us-east-2`)	View AMI ID	View AMI ID

Region	Amazon EKS-optimized Windows Server 2019 Full	Amazon EKS-optimized Windows Server 2019 Core
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID
EU (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID
EU (Ireland) (eu-west-1)	View AMI ID	View AMI ID
EU (London) (eu-west-2)	View AMI ID	View AMI ID
EU (Paris) (eu-west-3)	View AMI ID	View AMI ID
EU (Stockholm) (eu-north-1)	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID

2. Create your worker node group with the following command. Replace the *example values* with your own values. Be sure to use the AMI ID that you retrieved in the previous step.

```
eksctl create nodegroup \
--region us-west-2 \
--cluster windows \
--version 1.14 \
--name windows-ng \
--node-type t2.large \
--nodes 3 \
--nodes-min 1 \
--nodes-max 4 \
--node-ami-family WindowsServer2019FullContainer \
--node-ami ami-0c7f1b5f1bebccac2
```

Note

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

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

Output:

```
[#]  using region us-west-2
[#]  1 nodegroup(s) that already exist (ng-9d1cc1f2) will be excluded
[#]  nodegroup "windows-ng" will use
 "ami-0c7f1b5f1bebccac2" [WindowsServer2019FullContainer/1.14]
[#]  1 nodegroup (windows-ng) was included (based on the include/exclude rules)
[#]  combined exclude rules: ng-9d1cc1f2
[#]  no nodegroups present in the current set were excluded by the filter
[#]  will create a CloudFormation stack for each of 1 nodegroups in cluster
 "windows"
[#]  1 task: { create nodegroup "windows-ng" }
[#]  building nodegroup stack "eksctl-windows-nodegroup-windows-ng"
[#]  deploying stack "eksctl-windows-nodegroup-windows-ng"
[#]  adding role "arn:aws:iam::123456789012:role/eksctl-windows-nodegroup-windows-
NodeInstanceRole-1E4JMZRAT9AEZ" to auth ConfigMap
[#]  nodegroup "windows-ng" has 0 node(s)
[#]  waiting for at least 1 node(s) to become ready in "windows-ng"
[#]  nodegroup "windows-ng" has 1 node(s)
[#]  node "ip-192-168-88-105.us-west-2.compute.internal" is ready
[#]  created 1 nodegroup(s) in cluster "windows"
[#]  checking security group configuration for all nodegroups
[#]  all nodegroups have up-to-date configuration
```

3. (Optional) Deploy a Windows Sample Application (p. 62) — Deploy a sample application to test your cluster and Windows worker nodes.

AWS Management Console

To launch your worker nodes with the AWS Management Console

These procedures have the following prerequisites:

- You have an existing Amazon EKS cluster and a Linux worker node group. If you don't have these resources, we recommend that you follow one of our Getting Started with Amazon EKS (p. 3) guides to create them. The guides provide a complete end-to-end walkthrough for creating an Amazon EKS cluster with Linux worker nodes.
- You have created a VPC and security group that meet the requirements for an Amazon EKS cluster. For more information, see Cluster VPC Considerations (p. 126) and Cluster Security Group Considerations (p. 128). The Getting Started with Amazon EKS (p. 3) guide creates a VPC that meets the requirements, or you can also follow Creating a VPC for Your Amazon EKS Cluster (p. 124) to create one manually.

1. Wait for your cluster status to show as `ACTIVE`. If you launch your worker nodes before the cluster is active, the worker nodes will fail to register with the cluster and you will have to relaunch them.
2. Choose a **Launch workers** link that corresponds to your region and AMI type. This opens the AWS CloudFormation console and pre-populates several fields for you.

Kubernetes version 1.14.6

Region	Amazon EKS-optimized Windows Server 2019 Full	Amazon EKS-optimized Windows Server 2019 Core
US East (Ohio) (`us-east-2`)	Launch workers	Launch workers
US East (N. Virginia) (`us-east-1`)	Launch workers	Launch workers
US West (Oregon) (`us-west-2`)	Launch workers	Launch workers
Asia Pacific (Hong Kong) (`ap-east-1`)	Launch workers	Launch workers
Asia Pacific (Mumbai) (`ap-south-1`)	Launch workers	Launch workers
Asia Pacific (Tokyo) (`ap-northeast-1`)	Launch workers	Launch workers
Asia Pacific (Seoul) (`ap-northeast-2`)	Launch workers	Launch workers
Asia Pacific (Singapore) (`ap-southeast-1`)	Launch workers	Launch workers
Asia Pacific (Sydney) (`ap-southeast-2`)	Launch workers	Launch workers
EU (Frankfurt) (`eu-central-1`)	Launch workers	Launch workers
EU (Ireland) (`eu-west-1`)	Launch workers	Launch workers
EU (London) (`eu-west-2`)	Launch workers	Launch workers
EU (Paris) (`eu-west-3`)	Launch workers	Launch workers
EU (Stockholm) (`eu-north-1`)	Launch workers	Launch workers
Middle East (Bahrain) (`me-south-1`)	Launch workers	Launch workers

Note
If you intend to only deploy worker nodes to private subnets, you should edit this template in the AWS CloudFormation designer and modify the `AssociatePublicIpAddress` parameter in the `NodeLaunchConfig` to be `false`.

```
AssociatePublicIpAddress: 'false'
```

3. On the **Quick create stack** page, fill out the following parameters accordingly:

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

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

  Important
  This name must exactly match the name you used in Step 1: Create Your Amazon EKS Cluster (p. 14); otherwise, your worker nodes cannot join the cluster.
- **ClusterControlPlaneSecurityGroup**: Choose the **SecurityGroups** value from the AWS CloudFormation output that you generated with Create your Amazon EKS Cluster VPC (p. 12).
- **NodeGroupName**: Enter a name for your node group. This name can be used later to identify the Auto Scaling node group that is created for your worker nodes.
- **NodeAutoScalingGroupMinSize**: Enter the minimum number of nodes that your worker node Auto Scaling group can scale in to.
- **NodeAutoScalingGroupDesiredCapacity**: Enter the desired number of nodes to scale to when your stack is created.
- **NodeAutoScalingGroupMaxSize**: Enter the maximum number of nodes that your worker node Auto Scaling group can scale out to.
- **NodeInstanceType**: Choose an instance type for your worker nodes.

  Important
  Some instance types might not be available in all regions.
- **NodeImageIdSSMParam**: Pre-populated based on the version that you launched your worker nodes with in step 2. This value is the Amazon EC2 Systems Manager Parameter Store parameter to use for your worker node AMI ID. For example, the `aws/service/ami-windows-latest/Windows_Server-2019-English-Core-EKS-1.14_Optimized/image_id` parameter is for the latest recommended Kubernetes version 1.14 Amazon EKS-optimized Windows AMI. If you want to use the full version of Windows, then replace *`Core`* with `Full`.

  Note
  The Amazon EKS worker node AMI is based on Amazon Linux 2. You can track security or privacy events for Amazon Linux 2 at the Amazon Linux Security Center or subscribe to the associated RSS feed. Security and privacy events include an overview of the issue, what packages are affected, and how to update your instances to correct the issue.
- **NodeImageId**: (Optional) If you are using your own custom AMI (instead of the Amazon EKS-optimized AMI), enter a worker node AMI ID for your Region. If you specify a value here, it overrides any values in the **NodeImageIdSSMParam** field.
- **NodeVolumeSize**: Specify a root volume size for your worker nodes, in GiB.
- **KeyName**: Enter the name of an Amazon EC2 SSH key pair that you can use to connect using SSH into your worker nodes with after they launch. If you don't already have an Amazon EC2 keypair, you can create one in the AWS Management Console. For more information, see Amazon EC2 Key Pairs in the *Amazon EC2 User Guide for Linux Instances*.

  Note
  If you do not provide a keypair here, the AWS CloudFormation stack creation fails.
- **BootstrapArguments**: Specify any optional arguments to pass to the worker node bootstrap script, such as extra **kubelet** arguments. For more information, view the bootstrap script usage information at https://github.com/awslabs/amazon-eks-ami/blob/master/files/bootstrap.sh
- **VpcId**: Select the ID for the VPC that you created in Create your Amazon EKS Cluster VPC (p. 12).
- **NodeSecurityGroups**: Select the security group that was created for your Linux worker node group in Create your Amazon EKS Cluster VPC (p. 12). If your Linux worker nodes have more than one security group attached to them (for example, if the Linux worker node group was created with `eksctl`), specify all of them here.

- **Subnets**: Choose the subnets that you created in Create your Amazon EKS Cluster VPC (p. 12). If you created your VPC using the steps described at Creating a VPC for Your Amazon EKS Cluster (p. 124), then specify only the private subnets within the VPC for your worker nodes to launch into.

4. Acknowledge that the stack might create IAM resources, and then choose **Create stack**.
5. When your stack has finished creating, select it in the console and choose **Outputs**.
6. Record the **NodeInstanceRole** for the node group that was created. You need this when you configure your Amazon EKS Windows worker nodes.

To enable worker nodes to join your cluster

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

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

   ```
   curl -o aws-auth-cm-windows.yaml https://amazon-eks.s3-us-west-2.amazonaws.com/
   cloudformation/2019-10-08/aws-auth-cm-windows.yaml
   ```

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

   Important
   Do not modify any other lines in this file.

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

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

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

   Note
   If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 151).
   If you receive any other authorization or resource type errors, see Unauthorized or Access Denied (`kubectl`) (p. 235) in the troubleshooting section.

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

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

3. (Optional) Deploy a Windows Sample Application (p. 62) — Deploy a sample application to test your cluster and Windows worker nodes.

Worker Node Updates

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

There are two basic ways to update the worker nodes in your clusters to use a new AMI: create a new worker node group and migrate your pods to that group, or update the AWS CloudFormation stack for an existing worker node group to use the new AMI. This latter method is not supported for worker node groups that were created with `eksctl`.

Migrating to a new worker node group is more graceful than simply updating the AMI ID in an existing AWS CloudFormation stack, because the migration process taints the old node group as `NoSchedule` and drains the nodes after a new stack is ready to accept the existing pod workload.

Topics

- Migrating to a New Worker Node Group (p. 89)
- Updating an Existing Worker Node Group (p. 94)

Migrating to a New Worker Node Group

This topic helps you to create a new worker node group, gracefully migrate your existing applications to the new group, and then remove the old worker node group from your cluster.

eksctl

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

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.6.0`. 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. 161).

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

1. Retrieve the name of your existing worker node groups, substituting *`default`* with your cluster name.

   ```
   eksctl get nodegroups --cluster=default
   ```

   Output:

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

2. Launch a new worker node group with `eksctl` with the following command, substituting the *`example`* values with your own values.

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

```
eksctl create nodegroup \
--cluster default \
--version 1.14 \
--name standard-1-14 \
--node-type t3.medium \
--nodes 3 \
--nodes-min 1 \
--nodes-max 4 \
--node-ami auto
```

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

```
kubectl get nodes
```

4. Delete the original node group with the following command, substituting the *`example`* values with your cluster and nodegroup names:

```
eksctl delete nodegroup --cluster default --name standard-workers
```

AWS Management Console

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

1. Launch a new worker node group by following the steps outlined in Launching Amazon EKS Linux Worker Nodes (p. 76).
2. When your stack has finished creating, select it in the console and choose **Outputs**.
3. Record the **NodeInstanceRole** for the node group that was created. You need this to add the new Amazon EKS worker nodes to your cluster.

   Note
   If you have attached any additional IAM policies to your old node group IAM role, such as adding permissions for the Kubernetes Cluster Autoscaler, you should attach those same policies to your new node group IAM role to maintain that functionality on the new group.
4. Update the security groups for both worker node groups so that they can communicate with each other. For more information, see Cluster Security Group Considerations (p. 128).

   a. Record the security group IDs for both worker node groups. This is shown as the **NodeSecurityGroup** value in the AWS CloudFormation stack outputs.

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

```
oldNodes="<old_node_CFN_stack_name>"
newNodes="<new_node_CFN_stack_name>"

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

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

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

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

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

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

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

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
    - rolearn: arn:aws:iam::111122223333:role/workers-1-10-NodeInstanceRole-
U11V27W93CX5
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
```

   Replace the `<ARN of instance role (not instance profile)>` snippet with the **NodeInstanceRole** value that you recorded in Step 3 (p. 90), then save and close the file to apply the updated configmap.

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

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

7. (Optional) If you are using the Kubernetes Cluster Autoscaler, scale the deployment down to 0 replicas to avoid conflicting scaling actions.

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

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

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

   If you are upgrading your worker nodes to a new Kubernetes version, you can identify and taint all of the nodes of a particular Kubernetes version (in this case, 1.10.3) with the following code snippet.

   ```
   K8S_VERSION=1.10.3
   nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==
   \"v$K8S_VERSION\")].metadata.name}")
   for node in ${nodes[@]}
   do
       echo "Tainting $node"
       kubectl taint nodes $node key=value:NoSchedule
   done
   ```

9. Determine your cluster's DNS provider.

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

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

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

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

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

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

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

    If you are upgrading your worker nodes to a new Kubernetes version, you can identify and drain all of the nodes of a particular Kubernetes version (in this case, 1.10.3) with the following code snippet.

    ```
    K8S_VERSION=1.10.3
    nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==
    \"v$K8S_VERSION\")].metadata.name}")
    for node in ${nodes[@]}
    do
        echo "Draining $node"
        kubectl drain $node --ignore-daemonsets --delete-local-data
    done
    ```

12. After your old worker nodes have finished draining, revoke the security group ingress rules you authorized earlier, and then delete the AWS CloudFormation stack to terminate the instances.

    Note

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

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

    ```
    oldNodes="<old_node_CFN_stack_name>"
    newNodes="<new_node_CFN_stack_name>"

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

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

    c. Select your old worker node stack.

    d. Choose **Actions**, then **Delete stack**.

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

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

    Delete the `mapRoles` entry for the old worker node group.

    ```
    apiVersion: v1
    data:
      mapRoles: |
        - rolearn: arn:aws:iam::111122223333:role/workers-1-11-NodeInstanceRole-
    W70725MZQFF8
          username: system:node:{{EC2PrivateDNSName}}
          groups:
            - system:bootstrappers
            - system:nodes
        - rolearn: arn:aws:iam::111122223333:role/workers-1-10-NodeInstanceRole-
    U11V27W93CX5
          username: system:node:{{EC2PrivateDNSName}}
          groups:
            - system:bootstrappers
            - system:nodes
    ```

    Save and close the file to apply the updated configmap.

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

Note
You must also tag your new Auto Scaling group appropriately (for example, `k8s.io/cluster-autoscaler/enabled,k8s.io/cluster-autoscaler/<YOUR CLUSTER NAME>`) and update your Cluster Autoscaler deployment's command to point to the newly tagged Auto Scaling group. For more information, see Cluster Autoscaler on AWS.

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

15. (Optional) Verify that you are using the latest version of the Amazon VPC CNI plugin for Kubernetes. You may need to update your CNI version to take advantage of the latest supported instance types. For more information, see Amazon VPC CNI Plugin for Kubernetes Upgrades (p. 139).
16. If your cluster is using `kube-dns` for DNS resolution (see step Step 9 (p. 92)), scale in the `kube-dns` deployment to one replica.

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

Updating an Existing Worker Node Group

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

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

Note
This method is not supported for worker node groups that were created with `eksctl`. If you created your cluster or worker node group with `eksctl`, see Migrating to a New Worker Node Group (p. 89).

To update an existing worker node group

1. Determine your cluster's DNS provider.

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

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

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

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

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

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

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

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

   a. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

   b. Choose **Launch Configurations** in the left navigation, and note the instance type for your existing worker node launch configuration.

   c. Choose **Auto Scaling Groups** in the left navigation and note the **Desired** instance count for your existing worker node Auto Scaling group.

5. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
6. Select your worker node group stack, and then choose **Update**.
7. Select **Replace current template** and select **Amazon S3 URL**.
8. For **Amazon S3 URL**, paste the following URL into the text area to ensure that you are using the latest version of the worker node AWS CloudFormation template, and then choose **Next**:

```
https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2019-10-08/amazon-eks-nodegroup.yaml
```

9. On the **Specify stack details** page, fill out the following parameters, and choose **Next**:

   - **NodeAutoScalingGroupDesiredCapacity** – Enter the desired instance count that you recorded in Step 4 (p. 95), or enter a new desired number of nodes to scale to when your stack is updated.
   - **NodeAutoScalingGroupMaxSize** – Enter the maximum number of nodes to which your worker node Auto Scaling group can scale out. **This value must be at least one node greater than your desired capacity so that you can perform a rolling update of your worker nodes without reducing your node count during the update.**
   - **NodeInstanceType** – Choose the instance type your recorded in Step 4 (p. 95), or choose a different instance type for your worker nodes.

     Note

     The supported instance types for the latest version of the Amazon VPC CNI plugin for Kubernetes are shown here. You may need to update your CNI version to take advantage of the latest supported instance types. For more information, see Amazon VPC CNI Plugin for Kubernetes Upgrades (p. 139).

     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 1.14.

     ```
     /aws/service/eks/optimized-ami/1.14/amazon-linux-2/recommended/image_id
     ```

     You can change the *`1.14`* value to any supported Kubernetes version (p. 53). If you want to use the Amazon EKS-optimized AMI with GPU support, then change *`amazon-linux-2`* to *`amazon-linux-2-gpu`*.

     Note

     Using the Amazon EC2 Systems Manager parameter enables you to update your worker nodes in the future without having to lookup and specify an AMI ID. If your AWS CloudFormation stack is using this value, any stack update will always launch the latest recommended Amazon EKS-optimized AMI for your specified Kubernetes version, even if you don't change any values in the template.
   - **NodeImageId** – To use your own custom AMI, enter the ID for the AMI to use.

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

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

    Note
    Wait for the update to complete before performing the next steps.
12. If your cluster's DNS provider is `kube-dns`, scale in the `kube-dns` deployment to one replica.

    ```
    kubectl scale deployments/kube-dns --replicas=1 -n kube-system
    ```
13. (Optional) If you are using the Kubernetes Cluster Autoscaler, scale the deployment back to one replica.

    ```
    kubectl scale deployments/cluster-autoscaler --replicas=1 -n kube-system
    ```
14. (Optional) Verify that you are using the latest version of the Amazon VPC CNI plugin for Kubernetes. You may need to update your CNI version to take advantage of the latest supported instance types. For more information, see Amazon VPC CNI Plugin for Kubernetes Upgrades (p. 139).

Storage

This chapter covers storage options for Amazon EKS clusters.

The Storage Classes (p. 97) topic uses the in-tree Amazon EBS storage provisioner. For Kubernetes 1.14 and above clusters, the Amazon EBS CSI Driver (p. 98) is available for managing storage.

> **Note**
> The existing in-tree Amazon EBS plugin is still supported, but by using a CSI driver, you benefit from the decoupling of Kubernetes upstream release cycle and CSI driver release cycle. Eventually, the in-tree plugin will be deprecated in favor of the CSI driver.

Topics

- Storage Classes (p. 97)
- Amazon EBS CSI Driver (p. 98)
- Amazon EFS CSI Driver (p. 101)

Storage Classes

Amazon EKS clusters that were created prior to Kubernetes version 1.11 were not created with any storage classes. You must define storage classes for your cluster to use and you should define a default storage class for your persistent volume claims. For more information, see Storage Classes in the Kubernetes documentation.

> **Note**
> This topic uses the in-tree Amazon EBS storage provisioner. For Kubernetes 1.14 and above clusters, the Amazon EBS CSI Driver (p. 98) is available for managing storage. The existing in-tree Amazon EBS plugin is still supported, but by using a CSI driver, you benefit from the decoupling of Kubernetes upstream release cycle and CSI driver release cycle. Eventually, the in-tree plugin will be deprecated in favor of the CSI driver.

To create an AWS storage class for your Amazon EKS cluster

1. Create an AWS storage class manifest file for your storage class. The `gp2-storage-class.yaml` example below defines a storage class called `gp2` that uses the Amazon EBS `gp2` volume type.

   For more information about the options available for AWS storage classes, see AWS EBS in the Kubernetes documentation.

   ```
   kind: StorageClass
   apiVersion: storage.k8s.io/v1
   metadata:
     name: gp2
     annotations:
       storageclass.kubernetes.io/is-default-class: "true"
   provisioner: kubernetes.io/aws-ebs
   parameters:
     type: gp2
     fsType: ext4
   ```

2. Use **kubectl** to create the storage class from the manifest file.

   ```
   kubectl create -f gp2-storage-class.yaml
   ```

   Output:

```
storageclass "gp2" created
```

To define a default storage class

1. List the existing storage classes for your cluster. A storage class must be defined before you can set it as a default.

   ```
   kubectl get storageclass
   ```

   Output:

   ```
   NAME       PROVISIONER             AGE
   gp2        kubernetes.io/aws-ebs   8m
   ```

2. Choose a storage class and set it as your default by setting the `storageclass.kubernetes.io/is-default-class=true` annotation.

   ```
   kubectl patch storageclass gp2 -p '{"metadata": {"annotations":
   {"storageclass.kubernetes.io/is-default-class":"true"}}}'
   ```

   Output:

   ```
   storageclass "gp2" patched
   ```

3. Verify that the storage class is now set as default.

   ```
   kubectl get storageclass
   ```

   Output:

   ```
   gp2 (default)   kubernetes.io/aws-ebs   12m
   ```

Amazon EBS CSI Driver

The Amazon EBS Container Storage Interface (CSI) Driver provides a CSI interface that allows Amazon EKS clusters to manage the lifecycle of Amazon EBS volumes for persistent volumes.

This topic shows you how to deploy the Amazon EBS CSI Driver to your Amazon EKS cluster and verify that it works. We recommend using version v0.4.0 of the driver.

> **Note**
> This driver is only supported on Kubernetes version 1.14 and above Amazon EKS clusters. Alpha features of the Amazon EBS 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 EBS Container Storage Interface (CSI) Driver project on GitHub.

To deploy the Amazon EBS CSI Driver to an Amazon EKS cluster

1. Create an IAM policy called `Amazon_EBS_CSI_Driver` for your worker node instance profile that allows the Amazon EBS CSI Driver to make calls to AWS APIs on your behalf. Use the following AWS

CLI commands to create the IAM policy in your AWS account. You can view the policy document on GitHub.

a. Download the policy document from GitHub.

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-ebs-csi-driver/
v0.4.0/docs/example-iam-policy.json
```

b. Create the policy.

```
aws iam create-policy --policy-name Amazon_EBS_CSI_Driver \
--policy-document file://example-iam-policy.json
```

Take note of the policy ARN that is returned.

2. Get the IAM role name for your worker nodes. Use the following command to print the `aws-auth` configmap.

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

Output:

```
Name:          aws-auth
Namespace:     kube-system
Labels:        <none>
Annotations:   <none>

Data
====
mapRoles:
----
- groups:
  - system:bootstrappers
  - system:nodes
  rolearn: arn:aws:iam::111122223333:role/eksctl-alb-nodegroup-ng-b1f603c5-
NodeInstanceRole-GKNS581EASPU
  username: system:node:{{EC2PrivateDNSName}}

Events:  <none>
```

Record the role name for any `rolearn` values that have the `system:nodes` group assigned to them. In the previous example output, the role name is *`eksctl-alb-nodegroup-ng-b1f603c5-NodeInstanceRole-GKNS581EASPU`*. You should have one value for each node group in your cluster.

3. Attach the new `Amazon_EBS_CSI_Driver` IAM policy to each of the worker node IAM roles you identified earlier with the following command, substituting the red text with your own AWS account number and worker node IAM role name.

```
aws iam attach-role-policy \
--policy-arn arn:aws:iam::111122223333:policy/Amazon_EBS_CSI_Driver \
--role-name eksctl-alb-nodegroup-ng-b1f603c5-NodeInstanceRole-GKNS581EASPU
```

4. Deploy the Amazon EBS CSI Driver with the following command.

```
kubectl apply -k "github.com/kubernetes-sigs/aws-ebs-csi-driver/deploy/kubernetes/
overlays/stable/?ref=master"
```

To deploy a sample application and verify that the CSI driver is working

This procedure uses the Dynamic Volume Provisioning example from the Amazon EBS Container Storage Interface (CSI) Driver GitHub repository to consume a dynamically-provisioned Amazon EBS volume.

1. Clone the Amazon EBS Container Storage Interface (CSI) Driver GitHub repository to your local system.

   ```
   git clone https://github.com/kubernetes-sigs/aws-ebs-csi-driver.git
   ```

2. Navigate to the `dynamic-provisioning` example directory.

   ```
   cd aws-ebs-csi-driver/examples/kubernetes/dynamic-provisioning/
   ```

3. Deploy the `ebs-sc` storage class, `ebs-claim` persistent volume claim, and `app` sample application from the `specs` directory.

   ```
   kubectl apply -f specs/
   ```

4. Describe the `ebs-sc` storage class.

   ```
   kubectl describe storageclass ebs-sc
   ```

   Output:

   ```
   Name:            ebs-sc
   IsDefaultClass:  No
   Annotations:     kubectl.kubernetes.io/last-applied-
   configuration={"apiVersion":"storage.k8s.io/v1","kind":"StorageClass","metadata":
   {"annotations":{},"name":"ebs-
   sc"},"provisioner":"ebs.csi.aws.com","volumeBindingMode":"WaitForFirstConsumer"}

   Provisioner:           ebs.csi.aws.com
   Parameters:            <none>
   AllowVolumeExpansion:  <unset>
   MountOptions:          <none>
   ReclaimPolicy:         Delete
   VolumeBindingMode:     WaitForFirstConsumer
   Events:                <none>
   ```

   Note that the storage class uses the `WaitForFirstConsumer` volume binding mode. This means that volumes are not dynamically provisioned until a pod makes a persistent volume claim. For more information, see Volume Binding Mode in the Kubernetes documentation.

5. Watch the pods in the default namespace and wait for the `app` pod to become ready.

   ```
   kubectl get pods --watch
   ```

6. List the persistent volumes in the default namespace. Look for a persistent volume with the `default/ebs-claim` claim.

   ```
   kubectl get pv
   ```

   Output:

   ```
   NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
    STATUS   CLAIM                    STORAGECLASS   REASON   AGE
   ```

```
pvc-37717cd6-d0dc-11e9-b17f-06fad4858a5a   4Gi        RWO            Delete
 Bound    default/ebs-claim   ebs-sc                  30s
```

7. Describe the persistent volume.

   ```
   kubectl describe pv pvc-37717cd6-d0dc-11e9-b17f-06fad4858a5a
   ```

   Output:

   ```
   Name:              pvc-37717cd6-d0dc-11e9-b17f-06fad4858a5a
   Labels:            <none>
   Annotations:       pv.kubernetes.io/provisioned-by: ebs.csi.aws.com
   Finalizers:        [kubernetes.io/pv-protection external-attacher/ebs-csi-aws-com]
   StorageClass:      ebs-sc
   Status:            Bound
   Claim:             default/ebs-claim
   Reclaim Policy:    Delete
   Access Modes:      RWO
   VolumeMode:        Filesystem
   Capacity:          4Gi
   Node Affinity:
     Required Terms:
       Term 0:        topology.ebs.csi.aws.com/zone in [us-west-2a]
   Message:
   Source:
       Type:              CSI (a Container Storage Interface (CSI) volume source)
       Driver:            ebs.csi.aws.com
       VolumeHandle:      vol-0d651e157c6d93445
       ReadOnly:          false
       VolumeAttributes:      storage.kubernetes.io/
   csiProvisionerIdentity=1567792483192-8081-ebs.csi.aws.com
   Events:                <none>
   ```

   The Amazon EBS volume ID is listed as the `VolumeHandle`.

8. Verify that the pod is successfully writing data to the volume.

   ```
   kubectl exec -it app cat /data/out.txt
   ```

   Output:

   ```
   Fri Sep 6 19:26:53 UTC 2019
   Fri Sep 6 19:26:58 UTC 2019
   Fri Sep 6 19:27:03 UTC 2019
   Fri Sep 6 19:27:08 UTC 2019
   Fri Sep 6 19:27:13 UTC 2019
   Fri Sep 6 19:27:18 UTC 2019
   ```

9. When you finish experimenting, delete the resources for this sample application to clean up.

   ```
   kubectl delete -f specs/
   ```

Amazon EFS CSI Driver

The Amazon EFS Container Storage Interface (CSI) Driver provides a CSI interface that allows Amazon EKS clusters to manage the lifecycle of Amazon EFS file systems.

This topic shows you how to deploy the Amazon EFS CSI Driver to your Amazon EKS cluster and verify that it works.

> **Note**
> This driver is supported on Kubernetes version 1.14 and later Amazon EKS clusters. 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.

To deploy the Amazon EFS CSI Driver to an Amazon EKS cluster

- Deploy the Amazon EFS CSI Driver with the following command.

  ```
  kubectl apply -k "github.com/kubernetes-sigs/aws-efs-csi-driver/deploy/kubernetes/
  overlays/stable/?ref=master"
  ```

To create an Amazon EFS file system for your Amazon EKS cluster

1. Locate the VPC ID for your Amazon EKS cluster. You can find this ID in the Amazon EKS console, or you can use the following AWS CLI command.

   ```
   aws eks describe-cluster --name cluster_name --query cluster.resourcesVpcConfig.vpcId
    --output text
   ```

   Output:

   ```
   vpc-exampledb76d3e813
   ```

2. Locate the CIDR range for your cluster's VPC. You can find this in the Amazon VPC console, or you can use the following AWS CLI command.

   ```
   aws ec2 describe-vpcs --vpc-ids vpc-exampledb76d3e813 --query Vpcs[].CidrBlock --output
    text
   ```

   Output:

   ```
   192.168.0.0/16
   ```

3. Create a security group that allows inbound NFS traffic for your Amazon EFS mount points.

   a. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
   b. Choose **Security Groups** in the left navigation pane, and then **Create security group**.
   c. Enter a name and description for your security group, and choose the VPC that your Amazon EKS cluster is using.
   d. Choose **Create** and then **Close** to finish.

4. Add a rule to your security group to allow inbound NFS traffic from your VPC CIDR range.

   a. Choose the security group that you created in the previous step.
   b. Choose the **Inbound Rules** tab and then choose **Edit rules**.
   c. Choose **Add Rule**, fill out the following fields, and then choose **Save rules**.

      - **Type**: NFS
      - **Source**: Custom. Paste the VPC CIDR range.
      - **Description**: Add a description, such as "Allows inbound NFS traffic from within the VPC."

5. Create the Amazon EFS file system for your Amazon EKS cluster.

   a. Open the Amazon Elastic File System console at https://console.aws.amazon.com/efs/.
   b. Choose **Create file system**.
   c. On the **Configure file system access** page, choose the VPC that your Amazon EKS cluster is using.
   d. For **Security groups**, add the security group that you created in the previous step to each mount target and choose **Next step**.
   e. Configure any optional settings for your file system, and then choose **Next step** and **Create File System** to finish.

      Important
      By default, new Amazon EFS file systems are owned by `root:root`, and only the `root` user (UID 0) has read-write-execute permissions. If your containers are not running as `root`, you must change the Amazon EFS file system permissions to allow other users to modify the file system. For more information, see Working with Users, Groups, and Permissions at the Network File System (NFS) Level in the *Amazon Elastic File System User Guide*.

To deploy a sample application and verify that the CSI driver is working

This procedure uses the Multiple Pods Read Write Many example from the Amazon EFS Container Storage Interface (CSI) Driver GitHub repository to consume a statically provisioned Amazon EFS persistent volume and access it from multiple pods with the `ReadWriteMany` access mode.

1. Clone the Amazon EFS Container Storage Interface (CSI) Driver GitHub repository to your local system.

```
git clone https://github.com/kubernetes-sigs/aws-efs-csi-driver.git
```

2. Navigate to the `multiple_pods` example directory.

```
cd aws-efs-csi-driver/examples/kubernetes/multiple_pods/
```

3. Retrieve your Amazon EFS file system ID. You can find this in the Amazon EFS console, or use the following AWS CLI command.

```
aws efs describe-file-systems --query "FileSystems[*].FileSystemId" --output text
```

   Output:

```
fs-582a03f3
```

4. Edit the `specs/pv.yaml` file and replace the `volumeHandle` value with your Amazon EFS file system ID.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: efs-pv
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: efs-sc
  csi:
    driver: efs.csi.aws.com
    volumeHandle: fs-582a03f3
```

Note

Because Amazon EFS is an elastic file system, it does not enforce any file system capacity limits. The actual storage capacity value in persistent volumes and persistent volume claims is not used when creating the file system. However, since storage capacity is a required field in Kubernetes, you must specify a valid value, such as, *5Gi* in this example. This value does not limit the size of your Amazon EFS file system.

5. Deploy the `efs-sc` storage class, `efs-claim` persistent volume claim, `efs-pv` persistent volume, and `app1` and `app2` sample applications from the `specs` directory.

```
kubectl apply -f specs/
```

6. Watch the pods in the default namespace and wait for the `app1` and `app2` pods to become ready.

```
kubectl get pods --watch
```

7. List the persistent volumes in the default namespace. Look for a persistent volume with the `default/efs-claim` claim.

```
kubectl get pv
```

Output:

```
NAME     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
 STORAGECLASS   REASON   AGE
efs-pv   5Gi        RWX            Retain           Bound    default/efs-claim   efs-sc
                  2m50s
```

8. Describe the persistent volume.

```
kubectl describe pv efs-pv
```

Output:

```
Name:            efs-pv
Labels:          <none>
Annotations:     kubectl.kubernetes.io/last-applied-configuration:
                   {"apiVersion":"v1","kind":"PersistentVolume","metadata":
{"annotations":{},"name":"efs-pv"},"spec":{"accessModes":["ReadWriteMany"],"capaci...
                 pv.kubernetes.io/bound-by-controller: yes
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:    efs-sc
Status:          Bound
Claim:           default/efs-claim
Reclaim Policy:  Retain
Access Modes:    RWX
VolumeMode:      Filesystem
Capacity:        5Gi
Node Affinity:   <none>
Message:
Source:
    Type:              CSI (a Container Storage Interface (CSI) volume source)
    Driver:            efs.csi.aws.com
    VolumeHandle:      fs-582a03f3
    ReadOnly:           false
    VolumeAttributes:   <none>
Events:                 <none>
```

The Amazon EFS file system ID is listed as the `VolumeHandle`.

9. Verify that the `app1` pod is successfully writing data to the volume.

   ```
   kubectl exec -ti app1 -- tail /data/out1.txt
   ```

   Output:

   ```
   Wed Sep 18 20:30:48 UTC 2019
   Wed Sep 18 20:30:53 UTC 2019
   Wed Sep 18 20:30:58 UTC 2019
   Wed Sep 18 20:31:03 UTC 2019
   Wed Sep 18 20:31:08 UTC 2019
   Wed Sep 18 20:31:13 UTC 2019
   ```

10. Verify that the `app2` pod is shows the same data in the volume.

    ```
    kubectl exec -ti app2 -- tail /data/out1.txt
    ```

    Output:

    ```
    Wed Sep 18 20:30:48 UTC 2019
    Wed Sep 18 20:30:53 UTC 2019
    Wed Sep 18 20:30:58 UTC 2019
    Wed Sep 18 20:31:03 UTC 2019
    Wed Sep 18 20:31:08 UTC 2019
    Wed Sep 18 20:31:13 UTC 2019
    ```

11. When you finish experimenting, delete the resources for this sample application to clean up.

    ```
    kubectl delete -f specs/
    ```

Autoscaling

This chapter covers various autoscaling configurations for your Amazon EKS cluster. There are several types of Kubernetes autoscaling supported in Amazon EKS:

- Cluster Autoscaler (p. 106) — The Kubernetes Cluster Autoscaler automatically adjusts the number of nodes in your cluster when pods fail to launch due to lack of resources or when nodes in the cluster are underutilized and their pods can be rescheduled on to other nodes in the cluster.
- Horizontal Pod Autoscaler (p. 109) — The Kubernetes Horizontal Pod Autoscaler automatically scales the number of pods in a deployment, replication controller, or replica set based on that resource's CPU utilization.
- Vertical Pod Autoscaler (p. 113) — The Kubernetes Vertical Pod Autoscaler automatically adjusts the CPU and memory reservations for your pods to help "right size" your applications. This can help you to better use your cluster resources and free up CPU and memory for other pods.

Cluster Autoscaler

The Kubernetes Cluster Autoscaler automatically adjusts the number of nodes in your cluster when pods fail to launch due to lack of resources or when nodes in the cluster are underutilized and their pods can be rescheduled onto other nodes in the cluster.

This topic shows you how to deploy the Cluster Autoscaler to your Amazon EKS cluster and how to configure it to modify your Amazon EC2 Auto Scaling groups. The Cluster Autoscaler modifies your worker node groups so that they scale out when you need more resources and scale in when you have underutilized resources.

Create an Amazon EKS Cluster

Create an Amazon EKS cluster with no node groups with the following `eksctl` command. For more information, see Creating an Amazon EKS Cluster (p. 24). Note the Availability Zones that the cluster is created in. You will use these Availability Zones when you create your node groups. Substitute the red variable text with your own values.

```
eksctl create cluster --name my-cluster --version 1.14 --without-nodegroup
```

Output:

```
[#]  using region us-west-2
[#]  setting availability zones to [us-west-2a us-west-2c us-west-2b]
[#]  subnets for us-west-2a - public:192.168.0.0/19 private:192.168.96.0/19
[#]  subnets for us-west-2c - public:192.168.32.0/19 private:192.168.128.0/19
[#]  subnets for us-west-2b - public:192.168.64.0/19 private:192.168.160.0/19
[#]  using Kubernetes version 1.14
[#]  creating EKS cluster "my-cluster" in "us-west-2" region
[#]  will create a CloudFormation stack for cluster itself and 0 nodegroup stack(s)
[#]  if you encounter any issues, check CloudFormation console or try 'eksctl utils
 describe-stacks --region=us-west-2 --name=my-cluster'
[#]  CloudWatch logging will not be enabled for cluster "my-cluster" in "us-west-2"
[#]  you can enable it with 'eksctl utils update-cluster-logging --region=us-west-2 --
name=my-cluster'
[#]  1 task: { create cluster control plane "my-cluster" }
[#]  building cluster stack "eksctl-my-cluster-cluster"
[#]  deploying stack "eksctl-my-cluster-cluster"
[#]  all EKS cluster resource for "my-cluster" had been created
[#]  saved kubeconfig as "/Users/ericn/.kube/config"
[#]  kubectl command should work with "/Users/ericn/.kube/config", try 'kubectl get nodes'
[#]  EKS cluster "my-cluster" in "us-west-2" region is ready
```

This cluster was created in the following Availability Zones: *us-west-2a us-west-2c us-west-2b*.

Create Node Groups for your Cluster

Create single-zone node groups for each Availability Zone that your cluster was created in. For more information, see Launching Amazon EKS Linux Worker Nodes (p. 76).

The Cluster Autoscaler does not support Auto Scaling groups that span multiple Availability Zones. Instead, use an Auto Scaling group for each Availability Zone. You can later enable the `--balance-similar-node-groups` feature to keep your cluster's node count relatively even across Availability Zones.

For each Availability Zone in your cluster, use the following `eksctl` command to create a node group. Substitute the red variable text with your own values. This command creates an Auto Scaling group with a minimum count of one and a maximum count of ten.

```
eksctl create nodegroup --cluster my-cluster --node-zones us-west-2a --name us-west-2a --asg-access --nodes-min 1 --nodes-max 10
```

Node Group IAM Policy

The Cluster Autoscaler requires the following IAM permissions to make calls to AWS APIs on your behalf.

If you used the previous `eksctl` command to create your node groups, these permissions are automatically provided and attached to your worker node IAM roles. If you did not use `eksctl`, you must create an IAM policy with the following document and attach it to your worker node IAM roles. For more information, see Modifying a Role in the *IAM User Guide*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeAutoScalingInstances",
                "autoscaling:DescribeLaunchConfigurations",
                "autoscaling:DescribeTags",
                "autoscaling:SetDesiredCapacity",
                "autoscaling:TerminateInstanceInAutoScalingGroup",
                "ec2:DescribeLaunchTemplateVersions"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

Auto Scaling Group Tags

The Cluster Autoscaler requires the following tags on your node group Auto Scaling groups so that they can be auto-discovered.

If you used the previous `eksctl` command to create your node groups, these tags are automatically applied. If not, you must manually tag your Auto Scaling groups with the following tags. For more information, see Tagging Your Amazon EC2 Resources in the *Amazon EC2 User Guide for Linux Instances*.

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

Deploy the Cluster Autoscaler

To deploy the Cluster Autoscaler

1. Deploy the Cluster Autoscaler to your cluster with the following command.

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

2. Add the `cluster-autoscaler.kubernetes.io/safe-to-evict` annotation to the deployment with the following command.

```
kubectl -n kube-system annotate deployment.apps/cluster-autoscaler cluster-
autoscaler.kubernetes.io/safe-to-evict="false"
```

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

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

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

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

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

Save and close the file to apply the changes.

4. Open the Cluster Autoscaler releases page in a web browser and find the Cluster Autoscaler version that matches your cluster's Kubernetes major and minor version. For example, if your cluster's Kubernetes version is 1.14, find the Cluster Autoscaler release that begins with 1.14. Record the semantic version number (1.14.*n*) for that release to use in the next step.

5. Set the Cluster Autoscaler image tag to the version you recorded in the previous step with the following command. Replace the red variable text with your own value.

   ```
   kubectl -n kube-system set image deployment.apps/cluster-autoscaler cluster-
   autoscaler=k8s.gcr.io/cluster-autoscaler:v1.14.5
   ```

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 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.us-
west-2.compute.internal - node group min size reached
I0926 23:15:55.166365       1 utils.go:552] Skipping ip-192-168-71-83.us-
west-2.compute.internal - node group min size reached
I0926 23:15:55.166373       1 utils.go:552] Skipping ip-192-168-60-191.us-
west-2.compute.internal - node group min size reached
I0926 23:15:55.166435       1 static_autoscaler.go:393] Scale down status:
 unneededOnly=false lastScaleUpTime=2019-09-26 21:42:40.908059094 ...
I0926 23:15:55.166458       1 static_autoscaler.go:403] Starting scale down
I0926 23:15:55.166488       1 scale_down.go:706] No candidates for scale down
```

Horizontal Pod Autoscaler

The Kubernetes Horizontal Pod Autoscaler automatically scales the number of pods in a deployment, replication controller, or replica set based on that resource's CPU utilization. This can help your applications scale out to meet increased demand or scale in when resources are not needed, thus freeing up your worker nodes for other applications. When you set a target CPU utilization percentage, the Horizontal Pod Autoscaler scales your application in or out to try to meet that target.

The Horizontal Pod Autoscaler is a standard API resource in Kubernetes that simply requires that a metrics source (such as the Kubernetes metrics server) is installed on your Amazon EKS cluster to work. You do not need to deploy or install the Horizontal Pod Autoscaler on your cluster to begin scaling your applications. For more information, see Horizontal Pod Autoscaler in the Kubernetes documentation.

Use this topic to prepare the Horizontal Pod Autoscaler for your Amazon EKS cluster and to verify that it is working with a sample application.

> **Note**
> This topic is based on the Horizontal Pod Autoscaler Walkthrough in the Kubernetes documentation.

Install the Metrics Server

The Kubernetes metrics server is an aggregator of resource usage data in your cluster. The metrics server is not deployed by default in Amazon EKS clusters, but it provides metrics that are required by the Horizontal Pod Autoscaler. This topic explains how to deploy the Kubernetes metrics server on your Amazon EKS cluster.

If you have already deployed the metrics server to your cluster, you can move on to the next section. You can check for the metrics server with the following command.

```
kubectl -n kube-system get deployment/metrics-server
```

If this command returns a `NotFound` error, then you must deploy the metrics server to your Amazon EKS cluster. Choose the tab below that corresponds to your preferred installation method.

curl and jq

To install `metrics-server` from GitHub on an Amazon EKS cluster using `curl` and `jq`

If you have a macOS or Linux system with `curl`, `tar`, `gzip`, and the `jq` JSON parser installed, you can download, extract, and install the latest release with the following commands. Otherwise, use the next procedure to download the latest version using a web browser.

1. Open a terminal window and navigate to a directory where you would like to download the latest `metrics-server` release.
2. Copy and paste the commands below into your terminal window and type **Enter** to execute them. These commands download the latest release, extract it, and apply the version 1.8+ manifests to your cluster.

```
DOWNLOAD_URL=$(curl --silent "https://api.github.com/repos/kubernetes-incubator/
metrics-server/releases/latest" | jq -r .tarball_url)
DOWNLOAD_VERSION=$(grep -o '[^/v]*$' <<< $DOWNLOAD_URL)
curl -Ls $DOWNLOAD_URL -o metrics-server-$DOWNLOAD_VERSION.tar.gz
mkdir metrics-server-$DOWNLOAD_VERSION
tar -xzf metrics-server-$DOWNLOAD_VERSION.tar.gz --directory metrics-server-
$DOWNLOAD_VERSION --strip-components 1
kubectl apply -f metrics-server-$DOWNLOAD_VERSION/deploy/1.8+/
```

3. 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             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
metrics-server   1         1         1            1           56m
```

Web browser

To install `metrics-server` from GitHub on an Amazon EKS cluster using a web browser

1. Download and extract the latest version of the metrics server code from GitHub.

   a. Navigate to the latest release page of the `metrics-server` project on GitHub (https://github.com/kubernetes-incubator/metrics-server/releases/latest), then choose a source code archive for the latest release to download it.

      Note
      If you are downloading to a remote server, you can use the following `curl` command, substituting the red text with the latest version number.

      ```
      curl --remote-name --location https://github.com/kubernetes-incubator/
      metrics-server/archive/v0.3.4.tar.gz
      ```

   b. Navigate to your downloads location and extract the source code archive. For example, if you downloaded the `.tar.gz` archive on a macOS or Linux system, use the following command to extract (substituting your release version).

      ```
      tar -xzf v0.3.4.tar.gz
      ```

2. Apply all of the YAML manifests in the `metrics-server-0.3.4/deploy/1.8+` directory (substituting your release version).

   ```
   kubectl apply -f metrics-server-0.3.4/deploy/1.8+/
   ```

3. 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             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
   metrics-server   1         1         1            1           56m
   ```

Run a Horizontal Pod Autoscaler Test Application

In this section, you deploy a sample application to verify that the Horizontal Pod Autoscaler is working.

Note
This example is based on the Horizontal Pod Autoscaler Walkthrough in the Kubernetes documentation.

To test your Horizontal Pod Autoscaler installation

1. Create a simple Apache web server application with the following command.

   ```
   kubectl run httpd --image=httpd --requests=cpu=100m --limits=cpu=200m --expose --
   port=80
   ```

   This Apache web server pod is given 100 millicpu and 200 megabytes of memory, and it is serving on port 80.

2. Create a Horizontal Pod Autoscaler resource for the `httpd` deployment.

   ```
   kubectl autoscale deployment httpd --cpu-percent=50 --min=1 --max=10
   ```

This command creates an autoscaler that targets 50 percent CPU utilization for the deployment, with a minimum of one pod and a maximum of ten pods. When the average CPU load is below 50 percent, the autoscaler tries to reduce the number of pods in the deployment, to a minimum of one. When the load is greater than 50 percent, the autoscaler tries to increase the number of pods in the deployment, up to a maximum of ten. For more information, see How does the Horizontal Pod Autoscaler work? in the Kubernetes documentation.

3. Describe the autoscaler with the following command to view its details.

```
kubectl describe hpa/httpd
```

Output:

```
Name:                                                  httpd
Namespace:                                             default
Labels:                                                <none>
Annotations:                                           <none>
CreationTimestamp:                                     Fri, 27 Sep 2019 13:32:15 -0700
Reference:                                             Deployment/httpd
Metrics:                                               ( current / target )
  resource cpu on pods  (as a percentage of request):  1% (1m) / 50%
Min replicas:                                          1
Max replicas:                                          10
Deployment pods:                                       1 current / 1 desired
Conditions:
  Type            Status  Reason              Message
  ----            ------  ------              -------
  AbleToScale     True    ReadyForNewScale    recommended size matches current size
  ScalingActive   True    ValidMetricFound    the HPA was able to successfully
 calculate a replica count from cpu resource utilization (percentage of request)
  ScalingLimited  False   DesiredWithinRange  the desired count is within the
 acceptable range
Events:           <none>
```

As you can see, the current CPU load is only one percent, but the pod count is already at its lowest boundary (one), so it cannot scale in.

4. Create a load for the web server. The following command uses the Apache Bench program to send hundreds of thousands of requests to the `httpd` server. This should significantly increase the load and cause the autoscaler to scale out the deployment.

```
kubectl run apache-bench -i --tty --rm --image=httpd -- ab -n 500000 -c 1000 http://
httpd.default.svc.cluster.local/
```

5. Watch the `httpd` deployment scale out while the load is generated. To watch the deployment and the autoscaler, periodically run the following command.

```
kubectl get horizontalpodautoscaler.autoscaling/httpd
```

Output:

```
NAME    REFERENCE          TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
httpd   Deployment/httpd   76%/50%   1         10        10         4m50s
```

When the load finishes, the deployment should scale back down to 1.

6. When you are done experimenting with your sample application, delete the `httpd` resources.

```
kubectl delete deployment.apps/httpd service/httpd horizontalpodautoscaler.autoscaling/
httpd
```

Vertical Pod Autoscaler

The Kubernetes Vertical Pod Autoscaler automatically adjusts the CPU and memory reservations for your pods to help "right size" your applications. This adjustment can improve cluster resource utilization and free up CPU and memory for other pods. This topic helps you to deploy the Vertical Pod Autoscaler to your cluster and verify that it is working.

Install the Metrics Server

The Kubernetes metrics server is an aggregator of resource usage data in your cluster. It is not deployed by default in Amazon EKS clusters, but it provides metrics that are required by the Vertical Pod Autoscaler. This topic explains how to deploy the Kubernetes metrics server on your Amazon EKS cluster.

> **Note**
> You can also use Prometheus to provide metrics for the Vertical Pod Autoscaler. For more information, see Control Plane Metrics with Prometheus (p. 168).

If you have already deployed the metrics server to your cluster, you can move on to the next section. You can check for the metrics server with the following command.

```
kubectl -n kube-system get deployment/metrics-server
```

If this command returns a `NotFound` error, then you must deploy the metrics server to your Amazon EKS cluster. Choose the tab below that corresponds to your preferred installation method.

curl and jq

To install `metrics-server` from GitHub on an Amazon EKS cluster using `curl` and `jq`

If you have a macOS or Linux system with `curl`, `tar`, `gzip`, and the `jq` JSON parser installed, you can download, extract, and install the latest release with the following commands. Otherwise, use the next procedure to download the latest version using a web browser.

1. Open a terminal window and navigate to a directory where you would like to download the latest `metrics-server` release.
2. Copy and paste the commands below into your terminal window and type **Enter** to execute them. These commands download the latest release, extract it, and apply the version 1.8+ manifests to your cluster.

```
DOWNLOAD_URL=$(curl --silent "https://api.github.com/repos/kubernetes-incubator/
metrics-server/releases/latest" | jq -r .tarball_url)
DOWNLOAD_VERSION=$(grep -o '[^/v]*$' <<< $DOWNLOAD_URL)
curl -Ls $DOWNLOAD_URL -o metrics-server-$DOWNLOAD_VERSION.tar.gz
mkdir metrics-server-$DOWNLOAD_VERSION
tar -xzf metrics-server-$DOWNLOAD_VERSION.tar.gz --directory metrics-server-
$DOWNLOAD_VERSION --strip-components 1
kubectl apply -f metrics-server-$DOWNLOAD_VERSION/deploy/1.8+/
```

3. 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             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
metrics-server   1         1         1            1           56m
```

Web browser

To install `metrics-server` from GitHub on an Amazon EKS cluster using a web browser

1. Download and extract the latest version of the metrics server code from GitHub.

   a. Navigate to the latest release page of the `metrics-server` project on GitHub (https://github.com/kubernetes-incubator/metrics-server/releases/latest), then choose a source code archive for the latest release to download it.

      Note
      If you are downloading to a remote server, you can use the following `curl` command, substituting the red text with the latest version number.

      ```
      curl --remote-name --location https://github.com/kubernetes-incubator/
      metrics-server/archive/v0.3.4.tar.gz
      ```

   b. Navigate to your downloads location and extract the source code archive. For example, if you downloaded the `.tar.gz` archive on a macOS or Linux system, use the following command to extract (substituting your release version).

      ```
      tar -xzf v0.3.4.tar.gz
      ```

2. Apply all of the YAML manifests in the `metrics-server-0.3.4/deploy/1.8+` directory (substituting your release version).

   ```
   kubectl apply -f metrics-server-0.3.4/deploy/1.8+/
   ```

3. 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             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
   metrics-server   1         1         1            1           56m
   ```

Deploy the Vertical Pod Autoscaler

In this section, you deploy the Vertical Pod Autoscaler to your cluster.

To deploy the Vertical Pod Autoscaler

1. Open a terminal window and navigate to a directory where you would like to download the Vertical Pod Autoscaler source code.

2. Clone the kubernetes/autoscaler GitHub repository.

```
git clone https://github.com/kubernetes/autoscaler.git
```

3. Change to the `vertical-pod-autoscaler` directory.

```
cd autoscaler/vertical-pod-autoscaler/
```

4. (Optional) If you have already deployed another version of the Vertical Pod Autoscaler, remove it with the following command.

```
./hack/vpa-down.sh
```

5. Deploy the Vertical Pod Autoscaler to your cluster with the following command.

```
./hack/vpa-up.sh
```

6. Verify that the Vertical Pod Atoscaler 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.us-west-2.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.us-west-2.compute.internal/192.168.9.44
Start Time:     Fri, 27 Sep 2019 10:37:08 -0700
Labels:         app=hamster
                pod-template-hash=c7d89d6db
Annotations:    kubernetes.io/psp: eks.privileged
                vpaUpdates: Pod resources updated by hamster-vpa: container 0: cpu
 request, memory request
Status:         Running
IP:             192.168.3.140
IPs:            <none>
Controlled By:  ReplicaSet/hamster-c7d89d6db
Containers:
  hamster:
    Container ID:
 docker://2c3e7b6fb7ce0d8c86444334df654af6fb3fc88aad4c5d710eac3b1e7c58f7db
    Image:         k8s.gcr.io/ubuntu-slim:0.1
    Image ID:      docker-pullable://k8s.gcr.io/ubuntu-
slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
    Args:
      -c
      while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
    State:          Running
      Started:      Fri, 27 Sep 2019 10:37:08 -0700
    Ready:          True
    Restart Count:  0
    Requests:
      cpu:        587m
      memory:     262144k
...
```

Here you can see that the CPU reservation has increased to 587 millicpu, which is over five times the original value. The memory has increased to 262,144 Kilobytes, which is around 250 Mebibytes, or five times the original value. This pod was under-resourced, and the Vertical Pod Autoscaler corrected our estimate with a much more appropriate value.

6. Describe the `hamster-vpa` resource to view the new recommendation.

```
kubectl describe vpa/hamster-vpa
```

Output:

```
Name:         hamster-vpa
Namespace:    default
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"autoscaling.k8s.io/
v1beta2","kind":"VerticalPodAutoscaler","metadata":{"annotations":{},"name":"hamster-
vpa","namespace":"d...
API Version:  autoscaling.k8s.io/v1beta2
Kind:         VerticalPodAutoscaler
Metadata:
  Creation Timestamp:  2019-09-27T18:22:51Z
  Generation:          23
  Resource Version:    14411
  Self Link:           /apis/autoscaling.k8s.io/v1beta2/namespaces/default/
verticalpodautoscalers/hamster-vpa
  UID:                 d0d85fb9-e153-11e9-ae53-0205785d75b0
Spec:
  Target Ref:
    API Version:  apps/v1
    Kind:         Deployment
    Name:         hamster
Status:
  Conditions:
    Last Transition Time:  2019-09-27T18:23:28Z
    Status:                True
    Type:                  RecommendationProvided
  Recommendation:
    Container Recommendations:
      Container Name:  hamster
      Lower Bound:
        Cpu:     550m
        Memory:  262144k
      Target:
        Cpu:     587m
        Memory:  262144k
      Uncapped Target:
        Cpu:     587m
        Memory:  262144k
      Upper Bound:
        Cpu:     21147m
        Memory:  387863636
Events:          <none>
```

7. When you finish experimenting with the example application, you can delete it with the following command.

```
kubectl delete -f examples/hamster.yaml
```

Load Balancing and Ingress

This chapter covers common load balancing and Ingress configuration for Amazon EKS clusters.

Topics

- Load Balancing (p. 119)
- ALB Ingress Controller on Amazon EKS (p. 120)

Load Balancing

Amazon EKS supports the Network Load Balancer and the Classic Load Balancer through the Kubernetes service of type `LoadBalancer`. The configuration of your load balancer is controlled by annotations that are added to the manifest for your service.

By default, Classic Load Balancers are used for `LoadBalancer` type services. To use the Network Load Balancer instead, apply the following annotation to your service:

```
service.beta.kubernetes.io/aws-load-balancer-type: nlb
```

For more information about using Network Load Balancer with Kubernetes, see Network Load Balancer support on AWS in the Kubernetes documentation.

By default, services of type `LoadBalancer` create public-facing load balancers. To use an internal load balancer, apply the following annotation to your service:

```
service.beta.kubernetes.io/aws-load-balancer-internal: 0.0.0.0/0
```

For internal load balancers, your Amazon EKS cluster must be configured to use at least one private subnet in your VPC. Kubernetes examines the route table for your subnets to identify whether they are public or private. Public subnets have a route directly to the internet using an internet gateway, but private subnets do not.

Subnet Tagging for Load Balancers

Public subnets in your VPC may be tagged accordingly 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):

Key	Value
`kubernetes.io/role/elb`	`1`

Private subnets in your VPC should be tagged accordingly so that Kubernetes knows that it can use them for internal load balancers:

Key	Value
`kubernetes.io/role/internal-elb`	1

ALB Ingress Controller on Amazon EKS

The AWS ALB Ingress Controller for Kubernetes is a controller that triggers the creation of an Application Load Balancer and the necessary supporting AWS resources whenever an Ingress resource is created on the cluster with the `kubernetes.io/ingress.class: alb` annotation. The Ingress resource uses the ALB to route HTTP or HTTPS traffic to different endpoints within the cluster. The ALB Ingress Controller is supported for production workloads running on Amazon EKS clusters.

To ensure that your Ingress objects use the ALB Ingress Controller, add the following annotation to your Ingress specification. For more information, see Ingress specification in the documentation.

```
annotations:
    kubernetes.io/ingress.class: alb
```

Your Kubernetes service can be of the following types:

- NodePort
- ClusterIP (with the `alb.ingress.kubernetes.io/target-type: ip` annotation to put the service into IP mode)
- LoadBalancer (this creates two load balancers; one for the service, and one for the ingress)

For other available annotations supported by the ALB Ingress Controller, see Ingress annotations.

This topic show you how to configure the ALB Ingress Controller to work with your Amazon EKS cluster.

To deploy the ALB Ingress Controller to an Amazon EKS cluster

1. Tag the subnets in your VPC that you want to use for your load balancers so that the ALB Ingress Controller knows that it can use them.

   - Public subnets in your VPC should be tagged accordingly so that Kubernetes knows to use only those subnets for external load balancers.

Key	Value
`kubernetes.io/role/elb`	1

   - Private subnets in your VPC should be tagged accordingly so that Kubernetes knows that it can use them for internal load balancers:

Key	Value
`kubernetes.io/role/internal-elb`	1

2. Create an IAM policy called `ALBIngressControllerIAMPolicy` for your worker node instance profile that allows the ALB Ingress Controller to make calls to AWS APIs on your behalf. Use the following AWS CLI commands to create the IAM policy in your AWS account. You can view the policy document on GitHub.

  a. Download the policy document from GitHub.

  ```
  curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-controller/v1.1.2/docs/examples/iam-policy.json
  ```

  b. Create the policy.

  ```
  aws iam create-policy \
  --policy-name ALBIngressControllerIAMPolicy \
  --policy-document file://iam-policy.json
  ```

  Take note of the policy ARN that is returned.

3. Get the IAM role name for your worker nodes. Use the following command to print the `aws-auth` configmap.

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

   Output:

   ```
   Name:         aws-auth
   Namespace:    kube-system
   Labels:       <none>
   Annotations:  <none>

   Data
   ====
   mapRoles:
   ----
   - groups:
     - system:bootstrappers
     - system:nodes
     rolearn: arn:aws:iam::111122223333:role/eksctl-alb-nodegroup-ng-b1f603c5-NodeInstanceRole-GKNS581EASPU
     username: system:node:{{EC2PrivateDNSName}}

   Events:  <none>
   ```

   Record the role name for any `rolearn` values that have the `system:nodes` group assigned to them. In the above example output, the role name is *`eksctl-alb-nodegroup-ng-b1f603c5-NodeInstanceRole-GKNS581EASPU`*. You should have one value for each node group in your cluster.

4. Attach the new `ALBIngressControllerIAMPolicy` IAM policy to each of the worker node IAM roles you identified earlier with the following command, substituting the red text with your own AWS account number and worker node IAM role name.

   ```
   aws iam attach-role-policy \
   --policy-arn arn:aws:iam::111122223333:policy/ALBIngressControllerIAMPolicy \
   --role-name eksctl-alb-nodegroup-ng-b1f603c5-NodeInstanceRole-GKNS581EASPU
   ```

5. Create a service account, cluster role, and cluster role binding for the ALB Ingress Controller to use with the following command.

   ```
   kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-controller/v1.1.2/docs/examples/rbac-role.yaml
   ```

6. Deploy the ALB Ingress Controller with the following command.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.2/docs/examples/alb-ingress-controller.yaml
```

7. Open the ALB Ingress Controller deployment manifest for editing with the following command.

```
kubectl edit deployment.apps/alb-ingress-controller -n kube-system
```

8. Add the cluster name, VPC ID, and AWS Region name for your cluster after the `--ingress-class=alb` line and then save and close the file.

```
    spec:
      containers:
      - args:
        - --ingress-class=alb
        - --cluster-name=my_cluster
        - --aws-vpc-id=vpc-03468a8157edca5bd
        - --aws-region=us-west-2
```

To deploy a sample application

1. Deploy a sample application to verify that the ALB Ingress Controller creates an Application Load Balancer as a result of the Ingress object. Use the following commands to deploy the game 2048 as a sample application.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.2/docs/examples/2048/2048-namespace.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.2/docs/examples/2048/2048-deployment.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.2/docs/examples/2048/2048-service.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.2/docs/examples/2048/2048-ingress.yaml
```

2. After a few minutes, verify that the Ingress resource was created with the following command.

```
kubectl get ingress/2048-ingress -n 2048-game
```

Output:

```
NAME           HOSTS   ADDRESS
        PORTS   AGE
2048-ingress   *       example-2048game-2048ingr-6fa0-352729433.us-
west-2.elb.amazonaws.com   80      24h
```

3. Open a browser and navigate to the `ADDRESS` URL from the previous command output to see the sample application.

4. When you finish experimenting with your sample application, delete it with the following commands.

```
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.2/docs/examples/2048/2048-ingress.yaml
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.2/docs/examples/2048/2048-service.yaml
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.2/docs/examples/2048/2048-deployment.yaml
```

```
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.2/docs/examples/2048/2048-namespace.yaml
```

Amazon EKS Networking

This chapter covers networking considerations for running Kubernetes on Amazon EKS.

Topics

- Creating a VPC for Your Amazon EKS Cluster (p. 124)
- Cluster VPC Considerations (p. 126)
- Cluster Security Group Considerations (p. 128)
- Pod Networking (CNI) (p. 130)
- Installing CoreDNS (p. 140)
- Installing Calico on Amazon EKS (p. 141)

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.

This topic guides you through creating a VPC for your cluster with either 3 public subnets, or two public subnets and two private subnets, which are provided with internet access through a NAT gateway. You can use this VPC for your Amazon EKS cluster. We recommend a network architecture that uses private subnets for your worker nodes, and public subnets for Kubernetes to create public load balancers within.

Choose the tab below that represents your desired VPC configuration.

Only public subnets

To create your cluster VPC with only public subnets

1. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. From the navigation bar, select a Region that supports Amazon EKS.
3. Choose **Create stack**.
4. For **Choose a template**, select **Specify an Amazon S3 template URL**.
5. Paste the following URL into the text area and choose **Next**:

```
https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2019-10-08/amazon-eks-
vpc-sample.yaml
```

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

   - **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it **eks-vpc**.
   - **VpcBlock**: Choose a CIDR range for your VPC. You can keep the default value.
   - **Subnet01Block**: Specify a CIDR range for subnet 1. We recommend that you keep the default value so that you have plenty of IP addresses for pods to use.
   - **Subnet02Block**: Specify a CIDR range for subnet 2. We recommend that you keep the default value so that you have plenty of IP addresses for pods to use.
   - **Subnet03Block**: Specify a CIDR range for subnet 3. We recommend that you keep the default value so that you have plenty of IP addresses for pods to use.
7. (Optional) On the **Options** page, tag your stack resources. Choose **Next**.

8. On the **Review** page, choose **Create**.
9. When your stack is created, select it in the console and choose **Outputs**.
10. Record the **SecurityGroups** value for the security group that was created. You need this when you create your EKS cluster; this security group is applied to the cross-account elastic network interfaces that are created in your subnets that allow the Amazon EKS control plane to communicate with your worker nodes.
11. Record the **VpcId** for the VPC that was created. You need this when you launch your worker node group template.
12. Record the **SubnetIds** for the subnets that were created. You need this when you create your EKS cluster; these are the subnets that your worker nodes are launched into.

Public and private subnets

To create your cluster VPC with public and private subnets

1. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. From the navigation bar, select a Region that supports Amazon EKS.
3. Choose **Create stack**.
4. For **Choose a template**, select **Specify an Amazon S3 template URL**.
5. Paste the following URL into the text area and choose **Next**:

```
https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2019-10-08/amazon-eks-
vpc-private-subnets.yaml
```

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

   - **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it **eks-vpc**.
   - **VpcBlock**: Choose a CIDR range for your VPC. You can keep the default value.
   - **PublicSubnet01Block**: Specify a CIDR range for public subnet 1. We recommend that you keep the default value so that you have plenty of IP addresses for pods to use.
   - **PublicSubnet02Block**: Specify a CIDR range for public subnet 2. We recommend that you keep the default value so that you have plenty of IP addresses for pods to use.
   - **PrivateSubnet01Block**: Specify a CIDR range for private subnet 1. We recommend that you keep the default value so that you have plenty of IP addresses for pods to use.
   - **PrivateSubnet02Block**: Specify a CIDR range for private subnet 2. We recommend that you keep the default value so that you have plenty of IP addresses for pods to use.
7. (Optional) On the **Options** page, tag your stack resources. Choose **Next**.
8. On the **Review** page, choose **Create**.
9. When your stack is created, select it in the console and choose **Outputs**.
10. Record the **SecurityGroups** value for the security group that was created. You need this when you create your EKS cluster; this security group is applied to the cross-account elastic network interfaces that are created in your subnets that allow the Amazon EKS control plane to communicate with your worker nodes.
11. Record the **VpcId** for the VPC that was created. You need this when you launch your worker node group template.
12. Record the **SubnetIds** for the subnets that were created. You need this when you create your EKS cluster; these are the subnets that your worker nodes are launched into.
13. Tag your private subnets so that Kubernetes knows that it can use them for internal load balancers.

    a. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.

b. Choose **Subnets** in the left navigation.

c. Select one of the private subnets for your Amazon EKS cluster's VPC (you can filter them with the string `PrivateSubnet`), and choose the **Tags** tab, and then **Add/Edit Tags**.

d. Choose **Create Tag** and add the following key and value, and then choose **Save**.

Key	Value
`kubernetes.io/role/internal-elb`	`1`

e. Repeat these substeps for each private subnet in your VPC.

Next Steps

After you have created your VPC, you can try the Getting Started with Amazon EKS (p. 3) walkthrough, but you can skip the Create your Amazon EKS Cluster VPC (p. 12) section and use these subnets and security groups for your cluster.

Cluster VPC Considerations

When you create an Amazon EKS cluster, you specify the Amazon VPC subnets for your cluster to use. Amazon EKS requires subnets in at least two Availability Zones. We recommend a network architecture that uses private subnets for your worker nodes and public subnets for Kubernetes to create internet-facing load balancers within.

When you create your cluster, specify all of the subnets that will host resources for your cluster (such as worker nodes and load balancers).

> **Note**
> Internet-facing load balancers require a public subnet in your cluster. Worker nodes also require outbound internet access to the Amazon EKS APIs for cluster introspection and node registration at launch time. To pull container images, they require access to the Amazon S3 and Amazon ECR APIs (and any other container registries, such as DockerHub). For more information, see Cluster Security Group Considerations (p. 128) and AWS IP Address Ranges in the *AWS General Reference*.

The subnets that you pass when you create the cluster influence where Amazon EKS places elastic network interfaces that are used for the control plane to worker node communication.

It is possible to specify only public or private subnets when you create your cluster, but there are some limitations associated with these configurations:

- **Private-only**: Everything runs in a private subnet and Kubernetes cannot create internet-facing load balancers for your pods.
- **Public-only**: Everything runs in a public subnet, including your worker nodes.

Amazon EKS creates an elastic network interface in your private subnets to facilitate communication to your worker nodes. This communication channel supports Kubernetes functionality such as **kubectl exec** and **kubectl logs**. The security group that you specify when you create your cluster is applied to the elastic network interfaces that are created for your cluster control plane.

Your VPC must have DNS hostname and DNS resolution support. Otherwise, your worker nodes cannot register with your cluster. For more information, see Using DNS with Your VPC in the *Amazon VPC User Guide*.

VPC IP Addressing

You can define both private (RFC 1918) and public (non-RFC 1918) CIDR ranges within the VPC used for your Amazon EKS cluster. For more information, see VPCs and Subnets and IP Addressing in Your VPC in the *Amazon VPC User Guide*.

The Amazon EKS control plane creates up to 4 cross-account elastic network interfaces in your VPC for each cluster. Be sure that the subnets you specify have enough available IP addresses for the cross-account elastic network interfaces and your pods.

> **Important**
> Docker runs in the `172.17.0.0/16` CIDR range in Amazon EKS clusters. We recommend that your cluster's VPC subnets do not overlap this range. Otherwise, you will receive the following error:

```
Error: : error upgrading connection: error dialing backend: dial tcp
 172.17.nn.nn:10250: getsockopt: no route to host
```

VPC Tagging Requirement

When you create your Amazon EKS cluster, Amazon EKS tags the VPC containing the subnets you specify in the following way so that Kubernetes can discover it:

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

- **Key**: The `<cluster-name>` value matches your Amazon EKS cluster's name.
- **Value**: The `shared` value allows more than one cluster to use this VPC.

Subnet Tagging Requirement

When you create your Amazon EKS cluster, Amazon EKS tags the subnets you specify in the following way so that Kubernetes can discover them:

> **Note**
> All subnets (public and private) that your cluster uses for resources should have this tag.

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

- **Key**: The `<cluster-name>` value matches your Amazon EKS cluster.
- **Value**: The `shared` value allows more than one cluster to use this subnet.

Private Subnet Tagging Requirement for Internal Load Balancers

Private subnets in your VPC should be tagged accordingly so that Kubernetes knows that it can use them for internal load balancers:

Key	Value
`kubernetes.io/role/internal-elb`	`1`

Public Subnet Tagging Option for External Load Balancers

Public subnets in your VPC may be tagged accordingly 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):

Key	Value
`kubernetes.io/role/elb`	`1`

Cluster Security Group Considerations

If you create your VPC and worker node groups with the AWS CloudFormation templates provided in the Getting Started with Amazon EKS (p. 3) walkthrough, then your control plane and worker node security groups are configured with our recommended settings.

The security group for the worker nodes and the security group for the control plane communication to the worker nodes have been set up to prevent communication to privileged ports in the worker nodes. If your applications require added inbound or outbound access from the control plane or worker nodes, you must add these rules to the security groups associated with your cluster. For more information, see Security Groups for Your VPC in the *Amazon VPC User Guide*.

> **Note**
> To allow proxy functionality on privileged ports or to run the CNCF conformance tests yourself, you must edit the security groups for your control plane and the worker nodes. The security group on the worker nodes' side needs to allow inbound access for ports 0-65535 from the control plane, and the control plane side needs to allow outbound access to the worker nodes on ports 0-65535.

The worker node AWS CloudFormation template modifies the cluster control plane security group when you launch worker nodes (p. 76). **Amazon EKS strongly recommends that you use a dedicated security group for each cluster control plane (one per cluster)**. If you share a cluster control plane security group with other Amazon EKS clusters or resources, you may block or disrupt connections to those resources.

The following tables show the minimum required and recommended security group settings for the control plane and worker node security groups for your cluster:

Control Plane Security Group

	Protocol	Port Range	Source	Destination
Minimum inbound traffic	TCP	443	All worker node security groups **When cluster endpoint private access (p. 42) is enabled:** Any security groups	

	Protocol	Port Range	Source	Destination
			that generate API server client traffic (such as `kubectl` commands on a bastion host within your cluster's VPC)	
Recommended inbound traffic	TCP	443	All worker node security groups **When cluster endpoint private access (p. 42) is enabled:** Any security groups that generate API server client traffic (such as `kubectl` commands on a bastion host within your cluster's VPC)	
Minimum outbound traffic	TCP	10250		All worker node security groups
Recommended outbound traffic	TCP	1025-65535		All worker node security groups

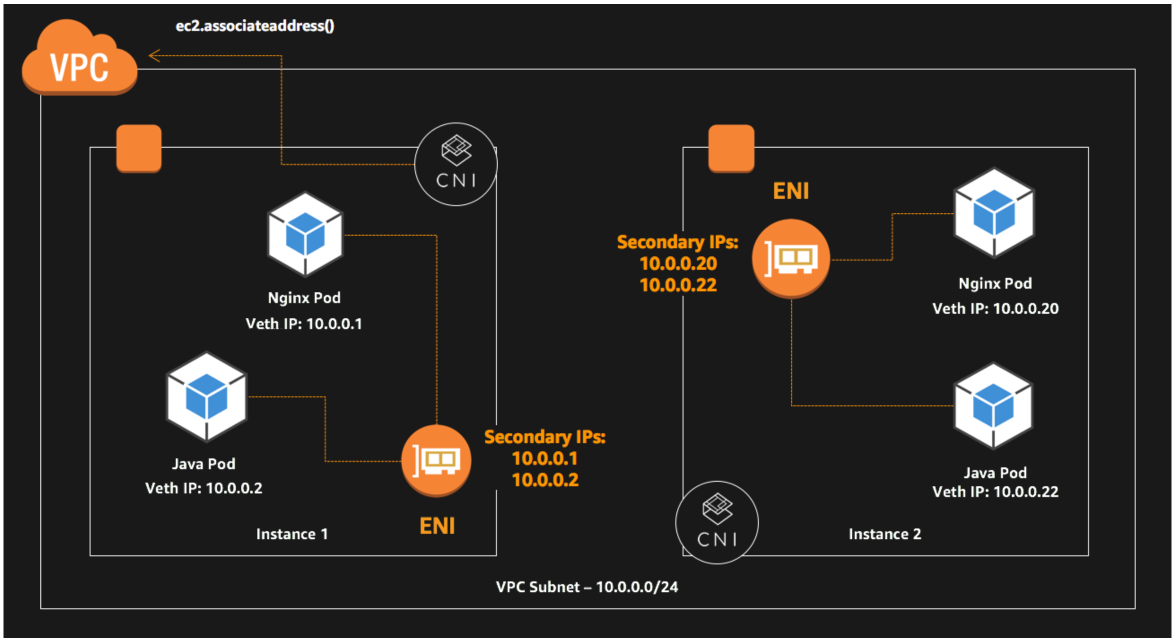
Worker Node Security Groups

	Protocol	Port Range	Source	Destination
Minimum inbound traffic (from other worker nodes)	Any protocol you expect your worker nodes to use for inter-worker communication	Any ports you expect your worker nodes to use for inter-worker communication	All worker node security groups	
Minimum inbound traffic (from control plane)	TCP	10250	Control plane security group	
Recommended inbound traffic	All TCP	All 443, 1025-65535	All worker node security groups Control plane security group	
Minimum outbound traffic*	TCP	443		Control plane security group
Recommended outbound traffic	All	All		0.0.0.0/0

* Worker nodes also require outbound internet access to the Amazon EKS APIs for cluster introspection and node registration at launch time. To pull container images, they require access to the Amazon S3 and Amazon ECR APIs (and any other container registries, such as DockerHub). For more information, see AWS IP Address Ranges in the *AWS General Reference*.

Pod Networking (CNI)

Amazon EKS supports native VPC networking via the Amazon VPC CNI plugin for Kubernetes. Using this CNI plugin allows Kubernetes pods to have the same IP address inside the pod as they do on the VPC network. This CNI plugin is an open-source project that is maintained on GitHub.



The CNI plugin is responsible for allocating VPC IP addresses to Kubernetes nodes and configuring the necessary networking for pods on each node. The plugin consists of two primary components:

- The L-IPAM daemon is responsible for attaching elastic network interfaces to instances, assigning secondary IP addresses to elastic network interfaces, and maintaining a "warm pool" of IP addresses on each node for assignment to Kubernetes pods when they are scheduled.
- The CNI plugin itself is responsible for wiring the host network (for example, configuring the interfaces and virtual Ethernet pairs) and adding the correct interface to the pod namespace.

For more information about the design and networking configuration, see CNI plugin for Kubernetes networking over AWS VPC.

Elastic network interface and secondary IP address limitations by Amazon EC2 instance types are applicable. In general, larger instances can support more IP addresses. For more information, see IP Addresses Per Network Interface Per Instance Type in the *Amazon EC2 User Guide for Linux Instances*.

Topics

- CNI Configuration Variables (p. 131)
- External Source Network Address Translation (SNAT) (p. 132)
- CNI Custom Networking (p. 133)
- CNI Metrics Helper (p. 136)

- Amazon VPC CNI Plugin for Kubernetes Upgrades (p. 139)

CNI Configuration Variables

The Amazon VPC CNI plugin for Kubernetes supports a number of configuration options, which are set through environment variables. The following environment variables are available, and all of them are optional.

`AWS_VPC_CNI_NODE_PORT_SUPPORT`

Type: Boolean

Default: `true`

Specifies whether `NodePort` services are enabled on a worker node's primary network interface. This requires additional `iptables` rules and that the kernel's reverse path filter on the primary interface is set to `loose`.

`AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG`

Type: Boolean

Default: `false`

Specifies that your pods may use subnets and security groups (within the same VPC as your control plane resources) that are independent of your cluster's `resourcesVpcConfig`. By default, pods share the same subnet and security groups as the worker node's primary interface. Setting this variable to `true` causes `ipamD` to use the security groups and subnets in a worker node's `ENIConfig` for elastic network interface allocation. You must create an `ENIConfig` custom resource definition for each subnet that your pods will reside in, and then annotate each worker node to use a specific `ENIConfig` (multiple worker nodes can be annotated with the same `ENIConfig`). Worker nodes can only be annotated with a single `ENIConfig` at a time, and the subnet in the `ENIConfig` must belong to the same Availability Zone that the worker node resides in. For more information, see CNI Custom Networking (p. 133).

`AWS_VPC_K8S_CNI_EXTERNALSNAT`

Type: Boolean

Default: `false`

Specifies whether an external NAT gateway should be used to provide SNAT of secondary ENI IP addresses. If set to `true`, the SNAT `iptables` rule and off-VPC IP rule are not applied, and these rules are removed if they have already been applied.

Disable SNAT if you need to allow inbound communication to your pods from external VPNs, direct connections, and external VPCs, and your pods do not need to access the Internet directly via an Internet Gateway. However, your nodes must be running in a private subnet and connected to the internet through an AWS NAT Gateway or another external NAT device.

For more information, see External Source Network Address Translation (SNAT) (p. 132).

`WARM_ENI_TARGET`

Type: Integer

Default: `1`

Specifies the number of free elastic network interfaces (and all of their available IP addresses) that the `ipamD` daemon should attempt to keep available for pod assignment on the node. By default, `ipamD` attempts to keep 1 elastic network interface and all of its IP addresses available for pod assignment.

> **Note**
>
> The number of IP addresses per network interface varies by instance type. For more information, see IP Addresses Per Network Interface Per Instance Type in the *Amazon EC2 User Guide for Linux Instances*.

For example, an `m4.4xlarge` launches with 1 network interface and 30 IP addresses. If 5 pods are placed on the node and 5 free IP addresses are removed from the IP address warm pool, then `ipamD` attempts to allocate more interfaces until `WARM_ENI_TARGET` free interfaces are available on the node.

> **Note**
>
> If `WARM_IP_TARGET` is set, then this environment variable is ignored and the `WARM_IP_TARGET` behavior is used instead.

`WARM_IP_TARGET`

Type: Integer

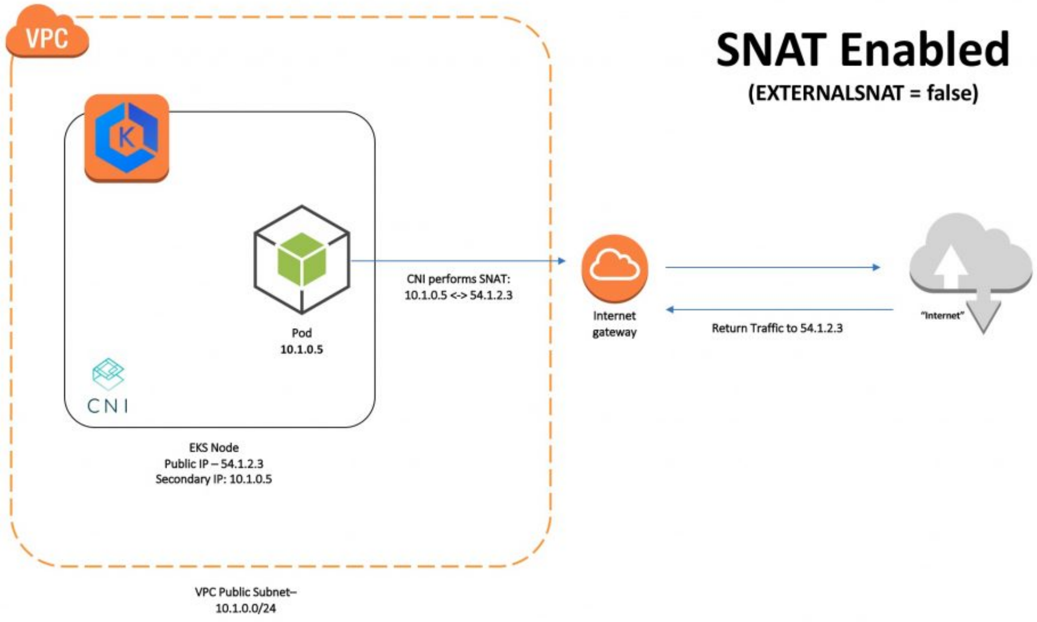
Default: None

Specifies the number of free IP addresses that the `ipamD` daemon should attempt to keep available for pod assignment on the node. For example, if `WARM_IP_TARGET` is set to 10, then `ipamD` attempts to keep 10 free IP addresses available at all times. If the elastic network interfaces on the node are unable to provide these free addresses, `ipamD` attempts to allocate more interfaces until `WARM_IP_TARGET` free IP addresses are available.

> **Note**
>
> This environment variable overrides `WARM_ENI_TARGET` behavior.

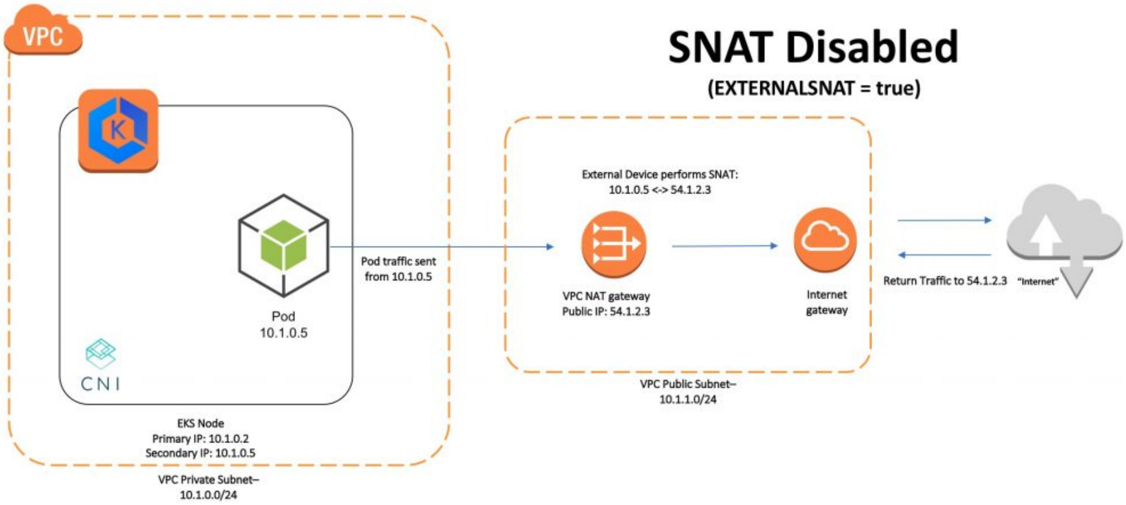
External Source Network Address Translation (SNAT)

By default, the Amazon VPC CNI plugin for Kubernetes configures pods with source network address translation (SNAT) enabled. This sets the return address for a packet to the primary public IP of the instance and allows for communication with the internet. In this default configuration, when you use an internet gateway and a public address, the return packet is routed to the correct Amazon EC2 instance.



However, SNAT can cause issues if traffic from another private IP space (for example, VPC peering, Transit VPC, or Direct Connect) attempts to communicate directly to a pod that is not attached to the primary elastic network interface of the Amazon EC2 instance. To specify that NAT be handled by an external device (such as a NAT gateway, and not on the instance itself), you can disable SNAT on the instance by setting the `AWS_VPC_K8S_CNI_EXTERNALSNAT` environment variable to `true`. Disable SNAT to allow inbound communication to your pods from external VPNs, direct connections, and external VPCs, and your pods do not need to access the internet directly via an internet gateway.

> **Note**
> SNAT is required for nodes that reside in a public subnet. To use external SNAT, your nodes must reside in a private subnet and connect to the internet through a NAT gateway or another external NAT device.



To disable SNAT on your worker nodes

- Set the `AWS_VPC_K8S_CNI_EXTERNALSNAT` environment variable to `true` in the `aws-node` DaemonSet:

```
kubectl set env daemonset -n kube-system aws-node AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

CNI Custom Networking

By default, when new network interfaces are allocated for pods, ipamD uses the worker node's primary elastic network interface's security groups and subnet. However, there are use cases where your pod network interfaces should use a different security group or subnet, within the same VPC as your control plane security group. For example:

- There are a limited number of IP addresses available in a subnet. This limits the number of pods can be created in the cluster. Using different subnets for pod groups allows you to increase the number of available IP addresses.
- For security reasons, your pods must use different security groups or subnets than the node's primary network interface.

- The worker nodes are configured in public subnets and you want the pods to be placed in private subnets using a NAT Gateway. For more information, see External Source Network Address Translation (SNAT) (p. 132).

> **Note**
> The use cases discussed in this topic require 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. 139).

Enabling this feature effectively removes an available elastic network interface (and all of its available IP addresses for pods) from each worker node that uses it. The primary network interface for the worker node is not used for pod placement when this feature is enabled. You should choose larger instance types with more available elastic network interfaces if you choose to enable this feature.

To configure CNI custom networking

1. Associate a secondary CIDR block to your cluster's VPC. For more information, see Associating a Secondary IPv4 CIDR Block with Your VPC in the *Amazon VPC User Guide*.
2. Create a subnet in your VPC for each Availability Zone, using your secondary CIDR block. Your custom subnets must be from a different VPC CIDR block than the subnet that your worker nodes were launched into. For more information, see Creating a Subnet in Your VPC in the *Amazon VPC User Guide*.
3. Set the `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true` environment variable to `true` in the `aws-node` DaemonSet:

```
kubectl set env daemonset aws-node -n kube-system
 AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
```

4. 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
```

5. Create an `ENIConfig` custom resource for each subnet that you want to schedule pods in.

   a. Create a unique file for each elastic network interface configuration to use with the following information. Replacing the subnet and security group IDs with your own values. If you don't have a specific security group that you want to attach for your pods, you can leave that value empty for now. Later, you will specify the worker node security group in the `ENIConfig`.

      For this example, the file is called *`custom-pod-netconfig`*`.yaml`.

> **Note**
> Each subnet and security group combination requires its own custom resource.

```
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
  name: custom-pod-netconfig
spec:
  securityGroups:
    - sg-0dff363a7d37c3c61
  subnet: subnet-017b472c2f79fdf96
```

b. Apply each custom resource file that you created earlier to your cluster with the following command:

```
kubectl apply -f custom-pod-netconfig.yaml
```

6. Create a new worker node group for each `ENIConfig` that you configured, and limit the Auto Scaling group to the same Availability Zone as the `ENIConfig`.

   Follow the steps in Launching Amazon EKS Linux Worker Nodes (p. 76) to create each new worker node group. When you create each group, apply the `k8s.amazonaws.com/eniConfig` label to the node group, and set the value to the name of the `ENIConfig` to use for that worker node group.

   - If you use `eksctl` to create your worker node groups, add the following flag to your `create cluster` command:

     ```
     --node-labels k8s.amazonaws.com/eniConfig=custom-pod-netconfig
     ```

   - If you use the Amazon EKS-provided AWS CloudFormation templates to create your worker node groups, add the following option to the **BootstrapArguments** field in the AWS CloudFormation console:

     ```
     --kubelet-extra-args '--node-labels=k8s.amazonaws.com/eniConfig=custom-pod-netconfig'
     ```

7. After your worker node groups are created, record the security group that was created for each worker node group and apply it to its associated `ENIConfig`. Edit each `ENIConfig` with the following command, replacing the red text with your value):

   ```
   kubectl edit eniconfig.crd.k8s.amazonaws.com/custom-pod-netconfig
   ```

   The `spec` section should look like this:

   ```
   spec:
     securityGroups:
     - sg-08052d900a2c7fb0a
     subnet: subnet-017b472c2f79fdf96
   ```

8. If you have any worker nodes in your cluster that had pods placed on them before you completed this procedure, you should terminate them. Only new nodes that are registered with the `k8s.amazonaws.com/eniConfig` label will use the new custom networking feature.

To automatically apply an ENIConfig to a node based on its Availability Zone

- By default, Kubernetes applies the availability zone of a node to the `failure-domain.beta.kubernetes.io/zone` label. You can name your `ENIConfig` custom resources after each Availability Zone in your VPC, and then specify this label as the value of the

ENI_CONFIG_LABEL_DEF environment variable in the aws-node container spec for your worker nodes.

```
...
    spec:
      containers:
      - env:
        - name: AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG
          value: "true"
        - name: ENI_CONFIG_LABEL_DEF
          value: failure-domain.beta.kubernetes.io/zone
        - name: AWS_VPC_K8S_CNI_LOGLEVEL
          value: DEBUG
        - name: MY_NODE_NAME
...
```

For example, if subnet-0c4678ec01ce68b24 is in the us-east-1a Availability Zone, you could use the following ENIConfig for that Availability Zone by naming it us-east-1a:

```
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
 name: us-east-1a
spec:
  securityGroups:
  - sg-08052d900a2c7fb0a
  subnet: subnet-0c4678ec01ce68b24
```

CNI Metrics Helper

The CNI metrics helper is a tool that you can use to scrape elastic network interface and IP address information, aggregate metrics at the cluster level, and publish the metrics to Amazon CloudWatch.

When managing an Amazon EKS cluster, you may want to know how many IP addresses have been assigned and how many are available. The CNI metrics helper helps you to:

- Track these metrics over time
- Troubleshoot and diagnose issues related to IP assignment and reclamation
- Provide insights for capacity planning

When a worker node is provisioned, the CNI plugin automatically allocates a pool of secondary IP addresses from the node’s subnet to the primary elastic network interface (eth0). This pool of IP addresses is known as the *warm pool*, and its size is determined by the worker node’s instance type. For example, a c4.large instance can support three elastic network interfaces and nine IP addresses per interface. The number of IP addresses available for a given pod is one less than the maximum (of ten) because one of the IP addresses is reserved for the elastic network interface itself. For more information, see IP Addresses Per Network Interface Per Instance Type in the *Amazon EC2 User Guide for Linux Instances*.

As the pool of IP addresses is depleted, the plugin automatically attaches another elastic network interface to the instance and allocates another set of secondary IP addresses to that interface. This process continues until the node can no longer support additional elastic network interfaces.

The following metrics are collected for your cluster and exported to CloudWatch:

- The maximum number of elastic network interfaces that the cluster can support

- The number of elastic network interfaces have been allocated to pods
- The number of IP addresses currently assigned to pods
- The total and maximum numbers of IP addresses available
- The number of ipamD errors

Deploying the CNI Metrics Helper

The CNI metrics helper requires `cloudwatch:PutMetricData` permissions to send metric data to CloudWatch. This section helps you to create an IAM policy with those permissions, apply it to your worker node instance role, and then deploy the CNI metrics helper.

To create an IAM policy for the CNI metrics helper

1. Create a file called `allow_put_metrics_data.json` and populate it with the following policy document.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "cloudwatch:PutMetricData",
      "Resource": "*"
    }
  ]
}
```

2. Create an IAM policy called `CNIMetricsHelperPolicy` for your worker node instance profile that allows the CNI metrics helper to make calls to AWS APIs on your behalf. Use the following AWS CLI command to create the IAM policy in your AWS account.

```
aws iam create-policy --policy-name CNIMetricsHelperPolicy \
--description "Grants permission to write metrics to CloudWatch" \
--policy-document file://allow_put_metrics_data.json
```

   Take note of the policy ARN that is returned.

3. Get the IAM role name for your worker nodes. Use the following command to print the `aws-auth` configmap.

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

   Output:

```
Name:         aws-auth
Namespace:    kube-system
Labels:       <none>
Annotations:  <none>

Data
====
mapRoles:
----
- groups:
  - system:bootstrappers
  - system:nodes
  rolearn: arn:aws:iam::111122223333:role/eksctl-prod-nodegroup-standard-wo-
NodeInstanceRole-GKNS581EASPU
```

```
  username: system:node:{{EC2PrivateDNSName}}

Events:  <none>
```

Record the role name for any rolearn values that have the system:nodes group assigned to them. In the above example output, the role name is *ekscti-prod-nodegroup-standard-wo-NodeInstanceRole-GKNS581EASPU*. You should have one value for each node group in your cluster.

4. Attach the new CNIMetricsHelperPolicy IAM policy to each of the worker node IAM roles you identified earlier with the following command, substituting the red text with your own AWS account number and worker node IAM role name.

```
aws iam attach-role-policy \
--policy-arn arn:aws:iam::111122223333:policy/CNIMetricsHelperPolicy \
--role-name eksctl-prod-nodegroup-standard-wo-NodeInstanceRole-GKNS581EASPU
```

To deploy the CNI metrics helper

- Apply the CNI metrics helper manifest with the following command.

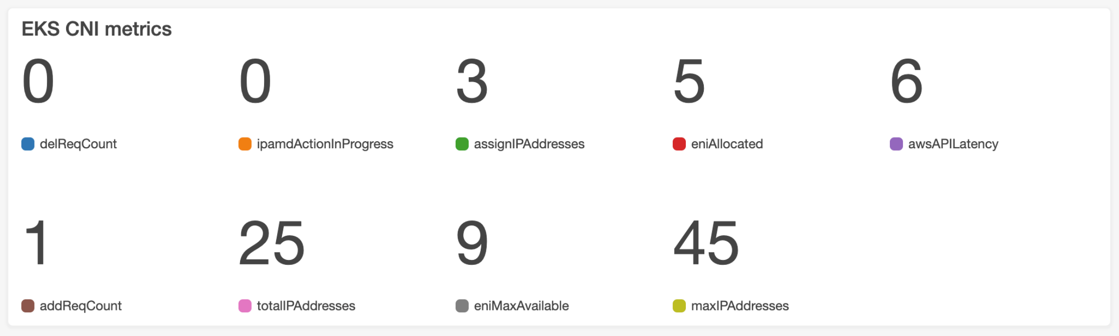
```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/release-1.5/
config/v1.5/cni-metrics-helper.yaml
```

Creating a Metrics Dashboard

After you have deployed the CNI metrics helper, you can view the CNI metrics in the CloudWatch console. This topic helps you to create a dashboard for viewing your cluster's CNI metrics.

To create a CNI metrics dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation, choose **Metrics**.
3. Under **Custom Namespaces**, choose **Kubernetes**.
4. Choose **CLUSTER_ID**.
5. On the **All metrics** tab, select the metrics you want to add to the dashboard.
6. Choose **Actions**, and then **Add to dashboard**.
7. In the **Select a dashboard** section, choose **Create new** and enter a name for your dashboard, such as "EKS-CNI-metrics".
8. In the **Select a widget type** section, choose **Number**.
9. In the **Customize the widget title** section, enter a logical name for your dashboard title, such as "ECS 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
delReqCount

0
ipamdActionInProgress

3
assignIPAddresses

5
eniAllocated

6
awsAPILatency

1
addReqCount

25
totalIPAddresses

9
eniMaxAvailable

45
maxIPAddresses

Amazon VPC CNI Plugin for Kubernetes Upgrades

When you launch an Amazon EKS cluster, we apply a recent version of the Amazon VPC CNI plugin for Kubernetes to your cluster (the absolute latest version of the plugin is available on GitHub for a short grace period before new clusters are switched over to use it). However, Amazon EKS does not automatically upgrade the CNI plugin on your cluster when new versions are released. You must upgrade the CNI plugin manually to get the latest version on existing clusters.

The latest CNI version available on GitHub is 1.5.4. You can view the different releases available for the plugin, and read the release notes for each version on GitHub.

Use the following procedures to check your CNI version and upgrade to the latest version.

To check your Amazon VPC CNI Plugin for Kubernetes version

- Use the following command to print your cluster's CNI version:

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

  Output:

  ```
  amazon-k8s-cni:1.5.3
  ```

  In this example output, the CNI version is 1.5.3, which is earlier than the current version, 1.5.4. Use the following procedure to upgrade the CNI.

To upgrade the Amazon VPC CNI Plugin for Kubernetes

- Use the following command to upgrade your CNI version to the latest version:
  - For Kubernetes 1.10 clusters:

    ```
    kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
    release-1.5/config/v1.5/aws-k8s-cni-1.10.yaml
    ```
  - For all other Kubernetes versions:

    ```
    kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
    release-1.5/config/v1.5/aws-k8s-cni.yaml
    ```

Installing CoreDNS

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, 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, use the following procedure to update your DNS and service discovery provider to CoreDNS.

> **Note**
> The service for CoreDNS is still called `kube-dns` for backward compatibility.

To install CoreDNS on an updated Amazon EKS cluster with `kubectl`

1. Add the `{"eks.amazonaws.com/component": "kube-dns"}` selector to the `kube-dns` deployment for your cluster. This prevents the two DNS deployments from competing for control of the same set of labels.

   ```
   kubectl patch -n kube-system deployment/kube-dns --patch \
   '{"spec":{"selector":{"matchLabels":{"eks.amazonaws.com/component":"kube-dns"}}}}'
   ```

2. Deploy CoreDNS to your cluster.

   a. Set your cluster's DNS IP address to the `DNS_CLUSTER_IP` environment variable.

   ```
   export DNS_CLUSTER_IP=$(kubectl get svc -n kube-system kube-dns -o
    jsonpath='{.spec.clusterIP}')
   ```

   b. Set your cluster's AWS Region to the `REGION` environment variable.

   ```
   export REGION="us-west-2"
   ```

   c. Download the CoreDNS manifest from the Amazon EKS resource bucket.

   ```
   curl -o dns.yaml https://amazon-eks.s3-us-west-2.amazonaws.com/
   cloudformation/2019-10-08/dns.yaml
   ```

   d. Replace the variable placeholders in the `dns.yaml` file with your environment variable values and apply the updated manifest to your cluster. The following command completes this in one step.

   ```
   cat dns.yaml | sed -e "s/REGION/$REGION/g" | sed -e "s/DNS_CLUSTER_IP/
   $DNS_CLUSTER_IP/g" | kubectl apply -f -
   ```

   e. Fetch the `coredns` pod name from your cluster.

   ```
   COREDNS_POD=$(kubectl get pod -n kube-system -l eks.amazonaws.com/component=coredns
    \
   -o jsonpath='{.items[0].metadata.name}')
   ```

   f. Query the `coredns` pod to ensure that it's receiving requests.

   ```
   kubectl get --raw /api/v1/namespaces/kube-system/pods/$COREDNS_POD:9153/proxy/
   metrics \
   ```

```
| grep 'coredns_dns_request_count_total'
```

Note
It might take several minutes for the expected output to return properly, depending on the rate of DNS requests in your cluster.

Expected output (the number in red 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. Check the current version of your cluster's `coredns` deployment.

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

   Output:

   ```
   coredns:v1.1.3
   ```

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

   - **Kubernetes 1.14:** `1.3.1`
   - **Kubernetes 1.13:** `1.2.6`
   - **Kubernetes 1.12:** `1.2.2`
   - **Kubernetes 1.11:** `1.1.3`

   If your current `coredns` version doesn't match the recommendation for your cluster version, update the `coredns` deployment to use the recommended image with the following command, replacing the red text with your `coredns` version:

   ```
   kubectl set image --namespace kube-system deployment.apps/coredns \
   coredns=602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/coredns:v1.3.1
   ```

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
   ```

Installing Calico on Amazon EKS

Project Calico is a network policy engine for Kubernetes. With Calico network policy enforcement, you can implement network segmentation and tenant isolation. This is useful in multi-tenant environments where you must isolate tenants from each other or when you want to create separate environments for development, staging, and production. Network policies are similar to AWS security groups in that you can create network ingress and egress rules. Instead of assigning instances to a security group, you assign network policies to pods using pod selectors and labels. The following procedure shows you how to install Calico on your Amazon EKS cluster.

To install Calico on your Amazon EKS cluster

1. Apply the Calico manifest from the `aws/amazon-vpc-cni-k8s` GitHub project. This manifest creates DaemonSets in the `kube-system` namespace.

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/release-1.5/
config/v1.5/calico.yaml
```

2. Watch the `kube-system` DaemonSets and wait for the `calico-node` DaemonSet to have the `DESIRED` number of pods in the `READY` state. When this happens, Calico is working.

```
kubectl get daemonset calico-node --namespace kube-system
```

Output:

```
NAME           DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE   NODE SELECTOR
 AGE
calico-node    3         3         3         3            3           <none>
 38s
```

To delete Calico from your Amazon EKS cluster

- If you are done using Calico in your Amazon EKS cluster, you can delete the DaemonSet with the following command:

```
kubectl delete -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/release-1.5/
config/v1.5/calico.yaml
```

Stars Policy Demo

This section walks through the Stars Policy Demo provided by the Project Calico documentation. The demo creates a frontend, backend, and client service on your Amazon EKS cluster. The demo also creates a management GUI that shows the available ingress and egress paths between each service.

Before you create any network policies, all services can communicate bidirectionally. After you apply the network policies, you can see that the client can only communicate with the frontend service, and the backend can only communicate with the frontend.

To run the Stars Policy demo

1. Apply the frontend, backend, client, and management UI services:

```
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/00-namespace.yaml
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/01-management-ui.yaml
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/02-backend.yaml
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/03-frontend.yaml
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/04-client.yaml
```

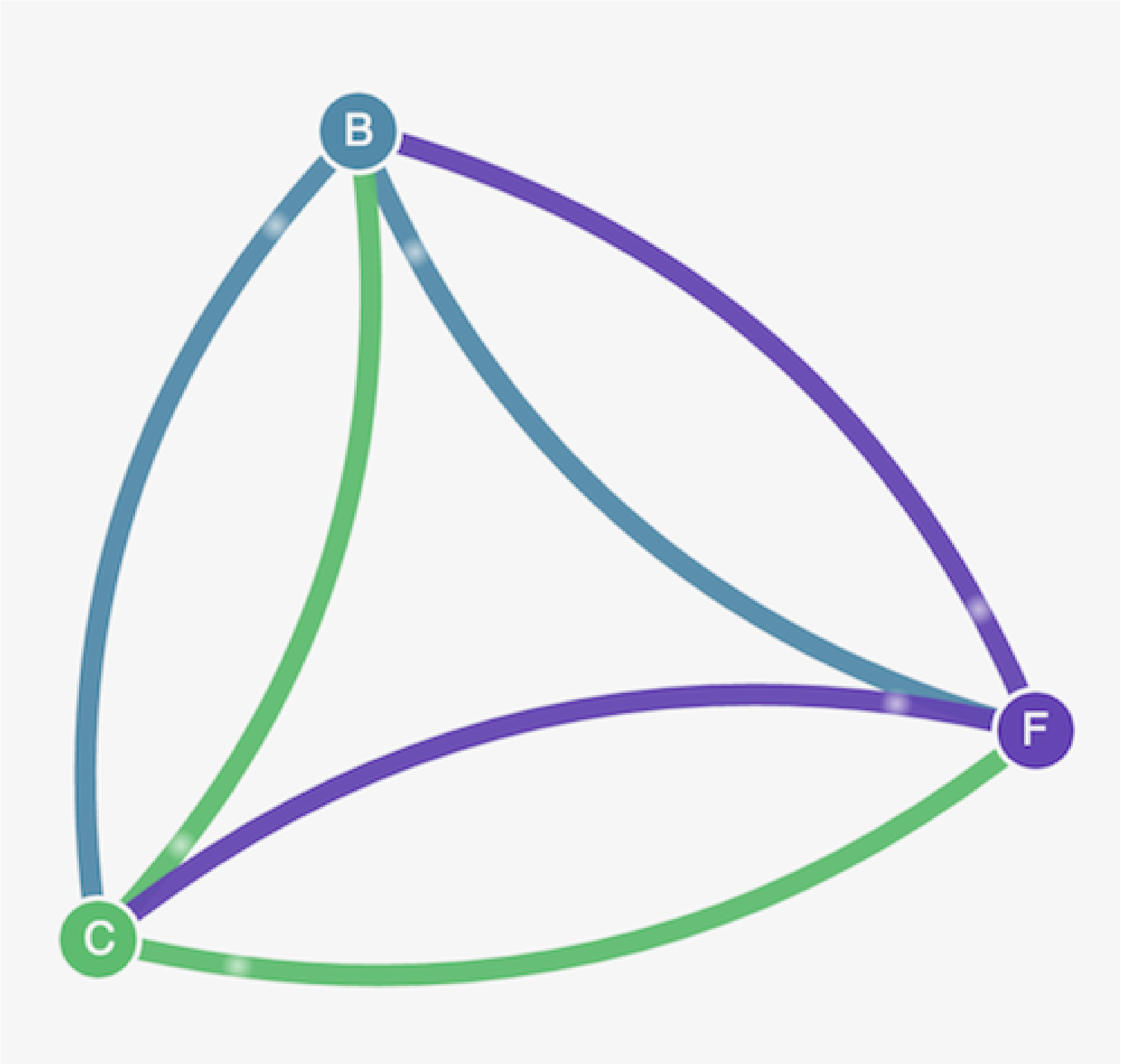
2. Wait for all of the pods to reach the `Running` status:

```
kubectl get pods --all-namespaces --watch
```

3. To connect to the management UI, forward your local port 9001 to the `management-ui` service running on your cluster:

```
kubectl port-forward service/management-ui -n management-ui 9001
```

4. Open a browser on your local system and point it to http://localhost:9001/. You should see the management UI. The **C** node is the client service, the **F** node is the frontend service, and the **B** node is the backend service. Each node has full communication access to all other nodes (as indicated by the bold, colored lines).



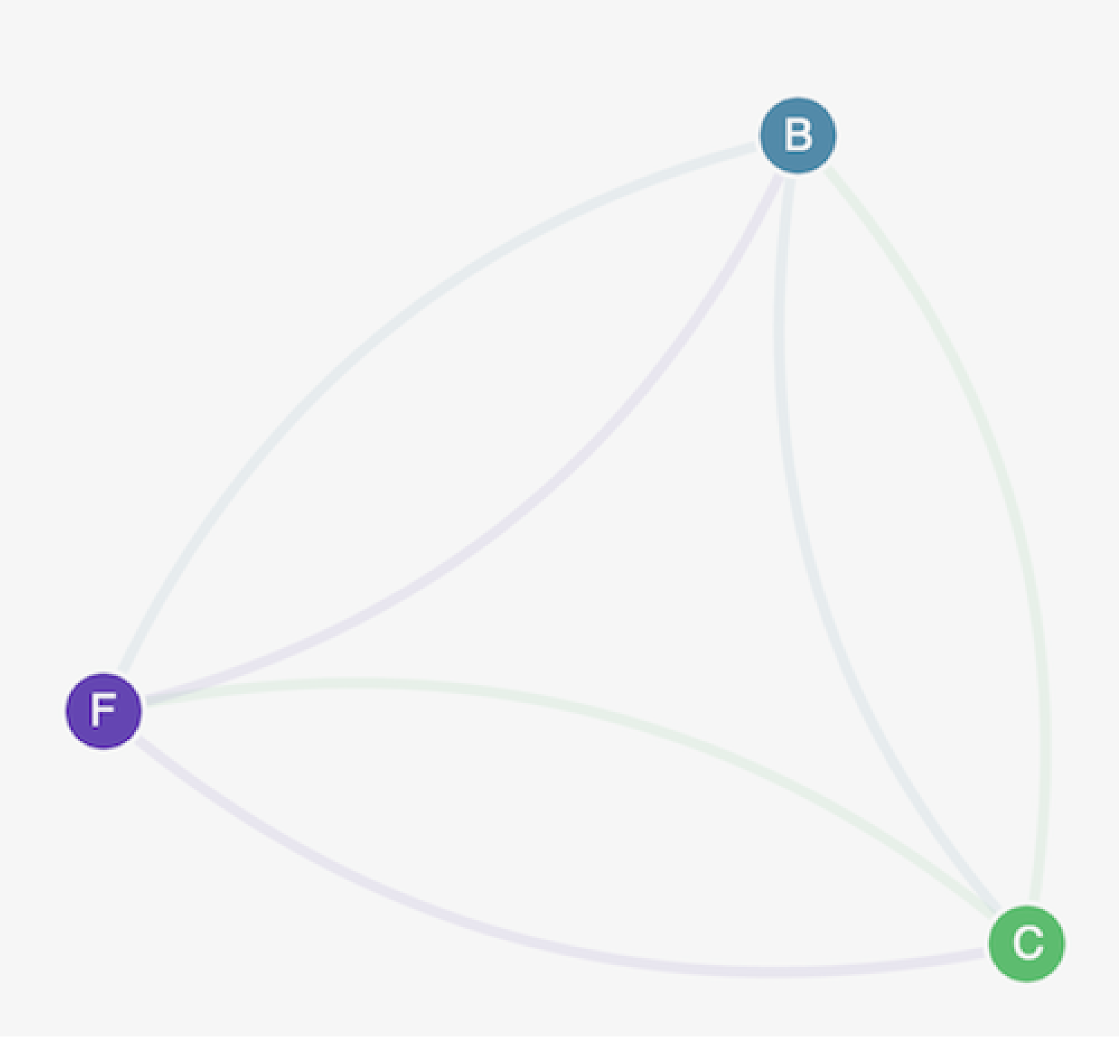
5. Apply the following network policies to isolate the services from each other:

```
kubectl apply -n stars -f https://docs.projectcalico.org/v3.3/getting-started/
kubernetes/tutorials/stars-policy/policies/default-deny.yaml
kubectl apply -n client -f https://docs.projectcalico.org/v3.3/getting-started/
kubernetes/tutorials/stars-policy/policies/default-deny.yaml
```

6. Refresh your browser. You see that the management UI can no longer reach any of the nodes, so they don't show up in the UI.
7. Apply the following network policies to allow the management UI to access the services:

```
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/policies/allow-ui.yaml
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/policies/allow-ui-client.yaml
```

8. Refresh your browser. You see that the management UI can reach the nodes again, but the nodes cannot communicate with each other.

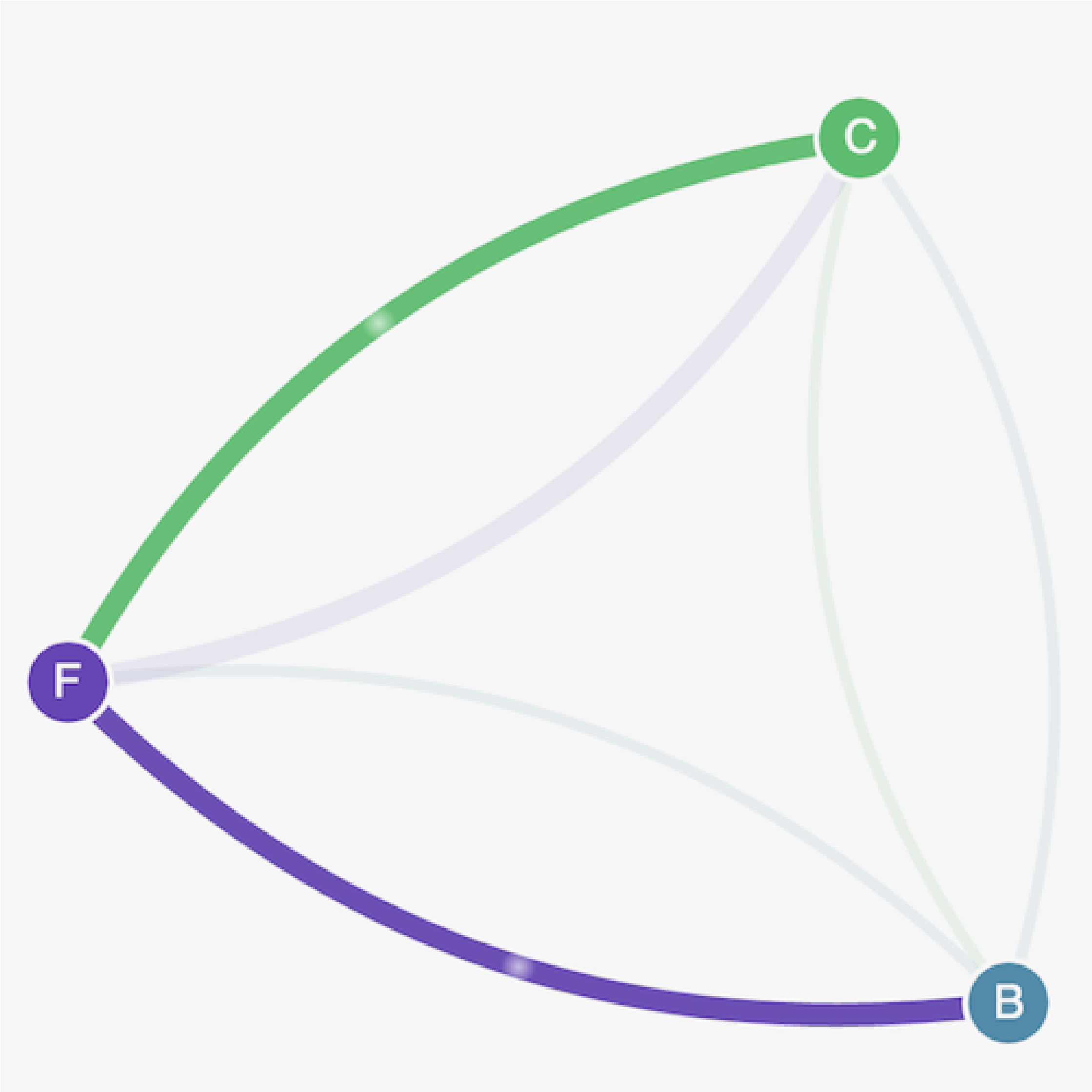


9. Apply the following network policy to allow traffic from the frontend service to the backend service:

```
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/policies/backend-policy.yaml
```

10. Apply the following network policy to allow traffic from the `client` namespace to the frontend service:

```
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/policies/frontend-policy.yaml
```

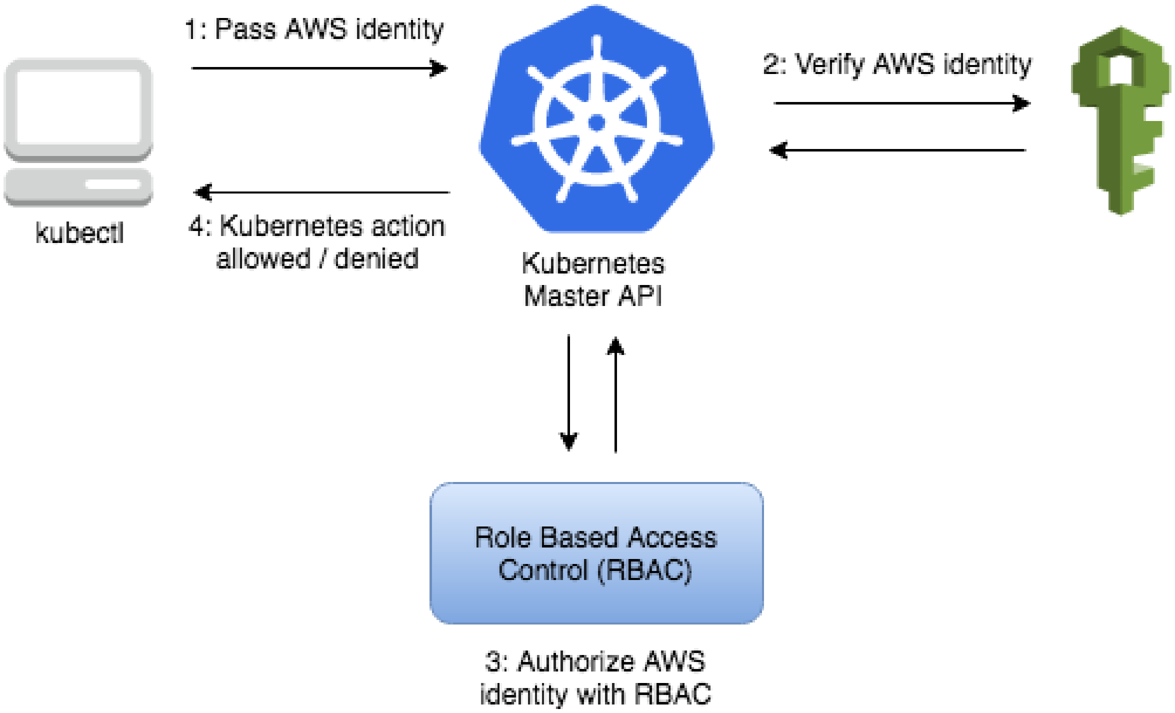


11. (Optional) When you are done with the demo, you can delete its resources with the following commands:

```
kubectl delete -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/04-client.yaml
kubectl delete -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/03-frontend.yaml
kubectl delete -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/02-backend.yaml
kubectl delete -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/01-management-ui.yaml
kubectl delete -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/00-namespace.yaml
```

Managing Cluster Authentication

Amazon EKS uses IAM to provide authentication to your Kubernetes cluster (through the **aws eks get-token** command, available in version 1.16.232 or greater 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 Master API
Role Based Access Control (RBAC)
3: Authorize AWS identity with RBAC

Topics

- Installing kubectl (p. 146)
- Installing aws-iam-authenticator (p. 151)
- Create a kubeconfig for Amazon EKS (p. 154)
- Managing Users or IAM Roles for your Cluster (p. 157)

Installing `kubectl`

Kubernetes uses a command line utility called `kubectl` for communicating with the cluster API server. The `kubectl` binary is available in many operating system package managers, and this option is often much easier than a manual download and install process. You can follow the instructions for your specific operating system or package manager in the Kubernetes documentation to install.

This topic helps you to download and install the Amazon EKS-vended **kubectl** binaries for macOS, Linux, and Windows operating systems. These binaries are identical to the upstream community versions, and are not unique to Amazon EKS or AWS.

Note

You must use a `kubectl` version that is within one minor version difference of your Amazon EKS cluster control plane . For example, a 1.12 `kubectl` client should work with Kubernetes 1.11, 1.12, and 1.13 clusters.

macOS

To install `kubectl` on macOS

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

   - **Kubernetes 1.14:**

     ```
     curl -o kubectl https://amazon-eks.s3-us-
     west-2.amazonaws.com/1.14.6/2019-08-22/bin/darwin/amd64/kubectl
     ```
   - **Kubernetes 1.13:**

     ```
     curl -o kubectl https://amazon-eks.s3-us-
     west-2.amazonaws.com/1.13.8/2019-08-14/bin/darwin/amd64/kubectl
     ```
   - **Kubernetes 1.12:**

     ```
     curl -o kubectl https://amazon-eks.s3-us-
     west-2.amazonaws.com/1.12.10/2019-08-14/bin/darwin/amd64/kubectl
     ```
   - **Kubernetes 1.11:**

     ```
     curl -o kubectl https://amazon-eks.s3-us-
     west-2.amazonaws.com/1.11.10/2019-08-14/bin/darwin/amd64/kubectl
     ```
2. (Optional) Verify the downloaded binary with the SHA-256 sum for your binary.

   a. Download the SHA-256 sum for your cluster's Kubernetes version for macOS:

      - **Kubernetes 1.14:**

        ```
        curl -o kubectl.sha256 https://amazon-eks.s3-us-
        west-2.amazonaws.com/1.14.6/2019-08-22/bin/darwin/amd64/kubectl.sha256
        ```
      - **Kubernetes 1.13:**

        ```
        curl -o kubectl.sha256 https://amazon-eks.s3-us-
        west-2.amazonaws.com/1.13.8/2019-08-14/bin/darwin/amd64/kubectl.sha256
        ```
      - **Kubernetes 1.12:**

        ```
        curl -o kubectl.sha256 https://amazon-eks.s3-us-
        west-2.amazonaws.com/1.12.10/2019-08-14/bin/darwin/amd64/kubectl.sha256
        ```
      - **Kubernetes 1.11:**

        ```
        curl -o kubectl.sha256 https://amazon-eks.s3-us-
        west-2.amazonaws.com/1.11.10/2019-08-14/bin/darwin/amd64/kubectl.sha256
        ```

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

```
openssl sha1 -sha256 kubectl
```

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

3. Apply execute permissions to the binary.

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

4. Copy the binary to a folder in your `PATH`. If you have already installed a version of **kubectl**, then we recommend creating a `$HOME/bin/kubectl` and ensuring that `$HOME/bin` comes first in your `$PATH`.

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$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=$HOME/bin:$PATH' >> ~/.bash_profile
```

6. After you install **kubectl**, you can verify its version with the following command:

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

Linux

To install `kubectl` on Linux

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

   - **Kubernetes 1.14:**

   ```
   curl -o kubectl https://amazon-eks.s3-us-
   west-2.amazonaws.com/1.14.6/2019-08-22/bin/linux/amd64/kubectl
   ```

   - **Kubernetes 1.13:**

   ```
   curl -o kubectl https://amazon-eks.s3-us-
   west-2.amazonaws.com/1.13.8/2019-08-14/bin/linux/amd64/kubectl
   ```

   - **Kubernetes 1.12:**

   ```
   curl -o kubectl https://amazon-eks.s3-us-
   west-2.amazonaws.com/1.12.10/2019-08-14/bin/linux/amd64/kubectl
   ```

   - **Kubernetes 1.11:**

   ```
   curl -o kubectl https://amazon-eks.s3-us-
   west-2.amazonaws.com/1.11.10/2019-08-14/bin/linux/amd64/kubectl
   ```

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

   a. Download the SHA-256 sum for your cluster's Kubernetes version for Linux:

      - **Kubernetes 1.14:**

```
curl -o kubectl.sha256 https://amazon-eks.s3-us-
west-2.amazonaws.com/1.14.6/2019-08-22/bin/linux/amd64/kubectl.sha256
```

- **Kubernetes 1.13:**

```
curl -o kubectl.sha256 https://amazon-eks.s3-us-
west-2.amazonaws.com/1.13.8/2019-08-14/bin/linux/amd64/kubectl.sha256
```

- **Kubernetes 1.12:**

```
curl -o kubectl.sha256 https://amazon-eks.s3-us-
west-2.amazonaws.com/1.12.10/2019-08-14/bin/linux/amd64/kubectl.sha256
```

- **Kubernetes 1.11:**

```
curl -o kubectl.sha256 https://amazon-eks.s3-us-
west-2.amazonaws.com/1.11.10/2019-08-14/bin/linux/amd64/kubectl.sha256
```

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

```
openssl sha1 -sha256 kubectl
```

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

3. Apply execute permissions to the binary.

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

4. Copy the binary to a folder in your `PATH`. If you have already installed a version of **kubectl**, then we recommend creating a `$HOME/bin/kubectl` and ensuring that `$HOME/bin` comes first in your `$PATH`.

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$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.

   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=$HOME/bin:$PATH' >> ~/.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 1.14:**

  ```
  curl -o kubectl.exe https://amazon-eks.s3-us-
  west-2.amazonaws.com/1.14.6/2019-08-22/bin/windows/amd64/kubectl.exe
  ```
- **Kubernetes 1.13:**

  ```
  curl -o kubectl.exe https://amazon-eks.s3-us-
  west-2.amazonaws.com/1.13.8/2019-08-14/bin/windows/amd64/kubectl.exe
  ```
- **Kubernetes 1.12:**

  ```
  curl -o kubectl.exe https://amazon-eks.s3-us-
  west-2.amazonaws.com/1.12.10/2019-08-14/bin/windows/amd64/kubectl.exe
  ```
- **Kubernetes 1.11:**

  ```
  curl -o kubectl.exe https://amazon-eks.s3-us-
  west-2.amazonaws.com/1.11.10/2019-08-14/bin/windows/amd64/kubectl.exe
  ```

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

   a. Download the SHA-256 sum for your cluster's Kubernetes version for Windows:

   - **Kubernetes 1.14:**

     ```
     curl -o kubectl.exe.sha256 https://amazon-eks.s3-us-
     west-2.amazonaws.com/1.14.6/2019-08-22/bin/windows/amd64/kubectl.exe.sha256
     ```
   - **Kubernetes 1.13:**

     ```
     curl -o kubectl.exe.sha256 https://amazon-eks.s3-us-
     west-2.amazonaws.com/1.13.8/2019-08-14/bin/windows/amd64/kubectl.exe.sha256
     ```
   - **Kubernetes 1.12:**

     ```
     curl -o kubectl.exe.sha256 https://amazon-eks.s3-us-
     west-2.amazonaws.com/1.12.10/2019-08-14/bin/windows/amd64/kubectl.exe.sha256
     ```
   - **Kubernetes 1.11:**

     ```
     curl -o kubectl.exe.sha256 https://amazon-eks.s3-us-
     west-2.amazonaws.com/1.11.10/2019-08-14/bin/windows/amd64/kubectl.exe.sha256
     ```

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

   ```
   Get-FileHash kubectl.exe
   ```

   c. Compare the generated SHA-256 sum in the command output against your downloaded SHA-256 file. The two should match, although the PowerShell output will be uppercase.

4. Copy the binary to a folder in your `PATH`. If you have an existing directory in your PATH that you use for command line utilities, copy the binary to that directory. Otherwise, complete the following steps.

   a. Create a new directory for your command line binaries, such as `C:\bin`.

   b. Copy the `kubectl.exe` binary to your new directory.

   c. Edit your user or system PATH environment variable to add the new directory to your PATH.

d. Close your PowerShell terminal and open a new one to pick up the new PATH variable.

5. After you install **kubectl**, you can verify its version with the following command:

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

Installing `aws-iam-authenticator`

Amazon EKS uses IAM to provide authentication to your Kubernetes cluster through the AWS IAM Authenticator for Kubernetes. You can configure the stock **kubectl** client to work with Amazon EKS by installing the AWS IAM Authenticator for Kubernetes and modifying your **kubectl** configuration file to use it for authentication.

macOS

To install `aws-iam-authenticator` with Homebrew

The easiest way to install the **`aws-iam-authenticator`** is with Homebrew.

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

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
master/install)"
```

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/1.14.6/2019-08-22/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/1.14.6/2019-08-22/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 $HOME/bin/aws-iam-authenticator &&
    export PATH=$HOME/bin:$PATH
   ```

5. Add `$HOME/bin` to your `PATH` environment variable.

   ```
   echo 'export PATH=$HOME/bin:$PATH' >> ~/.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/1.14.6/2019-08-22/bin/linux/amd64/aws-iam-authenticator
   ```

2. (Optional) Verify the downloaded binary with the SHA-256 sum provided in the same bucket prefix.

   a. Download the SHA-256 sum for your system.

      ```
      curl -o aws-iam-authenticator.sha256 https://amazon-eks.s3-us-
      west-2.amazonaws.com/1.14.6/2019-08-22/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 $HOME/bin/aws-iam-authenticator &&
    export PATH=$HOME/bin:$PATH
   ```

5. Add `$HOME/bin` to your `PATH` environment variable.

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.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/1.14.6/2019-08-22/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/1.14.6/2019-08-22/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. 154). Otherwise, see Creating an Amazon EKS Cluster (p. 24) to create a new Amazon EKS cluster.

Create a `kubeconfig` for Amazon EKS

In this section, you create a `kubeconfig` file for your cluster (or update an existing one).

This section offers two procedures to create or update your kubeconfig. You can quickly create or update a kubeconfig with the AWS CLI **update-kubeconfig** command by using the first procedure, or you can create a kubeconfig manually with the second procedure.

Amazon EKS uses the **aws eks get-token** command, available in version 1.16.232 or greater of the AWS CLI or the AWS IAM Authenticator for Kubernetes with **kubectl** for cluster authentication. If you have installed the AWS CLI on your system, then by default the AWS IAM Authenticator for Kubernetes will use the same credentials that are returned with the following command:

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

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

To create your `kubeconfig` file with the AWS CLI

1. Ensure that you have at least version 1.16.232 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 greater. 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.

   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*.

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

3. Test your configuration.

```
kubectl get svc
```

> **Note**
> If you receive the error "aws-iam-authenticator": executable file not found in $PATH, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing aws-iam-authenticator (p. 151).
> If you receive any other authorization or resource type errors, see Unauthorized or Access Denied (kubectl) (p. 235) in the troubleshooting section.

Output:

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

To create your kubeconfig file manually

1. Create the default ~/.kube directory if it does not already exist.

```
mkdir -p ~/.kube
```

2. Open your favorite text editor and copy one of the kubeconfig code blocks below into it, depending on your preferred client token method.

- To use the AWS CLI **aws eks get-token** command (requires at least version 1.16.232 of the AWS CLI):

```
apiVersion: v1
clusters:
- cluster:
    server: <endpoint-url>
    certificate-authority-data: <base64-encoded-ca-cert>
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: aws
  name: aws
current-context: aws
kind: Config
preferences: {}
users:
- name: aws
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1alpha1
      command: aws
      args:
        - "eks"
        - "get-token"
        - "--cluster-name"
        - "<cluster-name>"
        # - "--role"
        # - "<role-arn>"
      # env:
        # - name: AWS_PROFILE
        #   value: "<aws-profile>"
```

- To use the AWS IAM Authenticator for Kubernetes:

```
apiVersion: v1
clusters:
- cluster:
    server: <endpoint-url>
    certificate-authority-data: <base64-encoded-ca-cert>
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: aws
  name: aws
current-context: aws
kind: Config
preferences: {}
users:
- name: aws
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1alpha1
      command: aws-iam-authenticator
      args:
        - "token"
        - "-i"
        - "<cluster-name>"
        # - "-r"
        # - "<role-arn>"
      # env:
        # - name: AWS_PROFILE
        #   value: "<aws-profile>"
```

3. Replace the `<endpoint-url>` with the endpoint URL that was created for your cluster.
4. Replace the `<base64-encoded-ca-cert>` with the `certificateAuthority.data` that was created for your cluster.
5. Replace the `<cluster-name>` with your cluster name.
6. (Optional) To assume an IAM role to perform cluster operations instead of the default AWS credential provider chain, uncomment the `-r` or `--role` and `<role-arn>` lines and substitute an IAM role ARN to use with your user.
7. (Optional) To always use a specific named AWS credential profile (instead of the default AWS credential provider chain), uncomment the `env` lines and substitute `<aws-profile>` with the profile name to use.
8. Save the file to the default **kubectl** folder, with your cluster name in the file name. For example, if your cluster name is `devel`, save the file to `~/.kube/config-devel`.
9. Add that file path to your `KUBECONFIG` environment variable so that **kubectl** knows where to look for your cluster configuration.

```
export KUBECONFIG=$KUBECONFIG:~/.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
```

11. Test your configuration.

```
kubectl get svc
```

Note
If you receive the error "aws-iam-authenticator": executable file not found in $PATH, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing aws-iam-authenticator (p. 151).
If you receive any other authorization or resource type errors, see Unauthorized or Access Denied (kubectl) (p. 235) in the troubleshooting section.

Output:

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

Managing Users or IAM Roles for your Cluster

When you create an Amazon EKS cluster, the IAM entity user or role, such as a federated user that creates the cluster, is automatically granted system:masters permissions in the cluster's RBAC configuration. To grant additional AWS users or roles the ability to interact with your cluster, you must edit the aws-auth ConfigMap within Kubernetes.

Note
For more information about different IAM identities, see Identities (Users, Groups, and Roles) in the *IAM User Guide*. For more information on Kubernetes RBAC configuration, see Using RBAC Authorization.

The aws-auth ConfigMap is applied as part of the Getting Started with Amazon EKS (p. 3) guide which provides a complete end-to-end walkthrough from creating an Amazon EKS cluster to deploying a sample Kubernetes application. It is initially created to allow your worker nodes to join your cluster, but you also use this ConfigMap to add RBAC access to IAM users and roles. If you have not launched worker nodes and applied the aws-auth ConfigMap, you can do so with the following procedure.

To apply the aws-auth ConfigMap to your cluster

1. Check to see if you have already applied the aws-auth ConfigMap.

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

If you receive an error stating "Error from server (NotFound): configmaps "aws-auth" not found", then proceed with the following steps to apply the stock ConfigMap.

2. Download, edit, and apply the AWS authenticator configuration map.

   a. Download the configuration map:

```
curl -o aws-auth-cm.yaml https://amazon-eks.s3-us-west-2.amazonaws.com/
cloudformation/2019-10-08/aws-auth-cm.yaml
```

   b. Open the file with your favorite text editor. Replace the *<ARN of instance role (not instance profile)>* snippet with the Amazon Resource Name (ARN) of the IAM role that is associated with your worker nodes, and save the file. You can inspect the AWS CloudFormation stack outputs for your worker node groups and look for the following values:

- **InstanceRoleARN** (for worker node groups that were created with `eksctl`)
- **NodeInstanceRole** (for worker node groups that were created with Amazon EKS-vended AWS CloudFormation templates in the AWS Management Console)

Important
Do not modify any other lines in this file.

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

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

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

Note
If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 151).
If you receive any other authorization or resource type errors, see Unauthorized or Access Denied (`kubectl`) (p. 235) in the troubleshooting section.

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

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

To add an IAM user or role to an Amazon EKS cluster

1. Ensure that the AWS credentials that **kubectl** is using are already authorized for your cluster. The IAM user that created the cluster has these permissions by default.
2. Open the `aws-auth` ConfigMap.

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

Note
If you receive an error stating "`Error from server (NotFound): configmaps "aws-auth" not found`", then use the previous procedure to apply the stock ConfigMap.

Example ConfigMap:

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will
 be
# reopened with the relevant failures.
```

```
#
apiVersion: v1
data:
  mapRoles: |
    - rolearn: arn:aws:iam::111122223333:role/doc-test-worker-nodes-NodeInstanceRole-
WDO5P42N3ETB
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
kind: ConfigMap
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{"mapRoles":"- rolearn: arn:aws:iam::111122223333:role/
doc-test-worker-nodes-NodeInstanceRole-WDO5P42N3ETB\n  username: system:node:
{{EC2PrivateDNSName}}\n  groups:\n    - system:bootstrappers\n    -
 system:nodes\n"},"kind":"ConfigMap","metadata":{"annotations":{},"name":"aws-
auth","namespace":"kube-system"}}
  creationTimestamp: 2018-04-04T18:49:10Z
  name: aws-auth
  namespace: kube-system
  resourceVersion: "780"
  selfLink: /api/v1/namespaces/kube-system/configmaps/aws-auth
  uid: dcc31de5-3838-11e8-af26-02e00430057c
```

3. Add your IAM users, roles, or AWS accounts to the configMap.

   - **To add an IAM user:** add the user details to the `mapUsers` section of the ConfigMap, under `data`. Add this section if it does not already exist in the file. Each entry supports the following parameters:
     - **userarn**: The ARN of the IAM user to add.
     - **username**: The user name within Kubernetes to map to the IAM user. By default, the user name is the ARN of the IAM user.
     - **groups**: A list of groups within Kubernetes to which the user is mapped to. For more information, see Default Roles and Role Bindings in the Kubernetes documentation.
   - **To add an IAM role (for example, for federated users):** add the role details to the `mapRoles` section of the ConfigMap, under `data`. Add this section if it does not already exist in the file. Each entry supports the following parameters:
     - **rolearn**: The ARN of the IAM role to add.
     - **username**: The user name within Kubernetes to map to the IAM role. By default, the user name is the ARN of the IAM role.
     - **groups**: A list of groups within Kubernetes to which the role is mapped. For more information, see Default Roles and Role Bindings in the Kubernetes documentation.

   For example, the block below contains:

   - A `mapRoles` section that adds the worker node instance role so that worker nodes can register themselves with the cluster.
   - A `mapUsers` section with the AWS users `admin` from the default AWS account, and `ops-user` from another AWS account. Both users are added to the `system:masters` group.

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will
 be
# reopened with the relevant failures.
#
apiVersion: v1
```

```
data:
  mapRoles: |
    - rolearn: arn:aws:iam::555555555555:role/devel-worker-nodes-
NodeInstanceRole-74RF4UBDUKL6
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
  mapUsers: |
    - userarn: arn:aws:iam::555555555555:user/admin
      username: admin
      groups:
        - system:masters
    - userarn: arn:aws:iam::111122223333:user/ops-user
      username: ops-user
      groups:
        - system:masters
```

4. Save the file and exit your text editor.

The `eksctl` Command Line Utility

This chapter covers `eksctl`, a simple command line utility for creating and managing Kubernetes clusters on Amazon EKS. The `eksctl` command line utility provides the fastest and easiest way to create a new cluster with worker nodes for Amazon EKS.

For more information and to see the official documentation, visit https://eksctl.io/.

Installing or Upgrading `eksctl`

This section helps you to install or upgrade the `eksctl` command line utility.

Choose the tab below that best represents your client setup.

macOS

To install or upgrade `eksctl` on macOS using Homebrew

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

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

   ```
   /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
   master/install)"
   ```

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 `0.6.0`. If not, check your terminal output for any installation or upgrade errors.

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/download/
latest_release/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 `0.6.0`. If not, check your terminal output for any installation or upgrade errors.

Windows

To install or upgrade `eksctl` on Windows using Chocolatey

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

   - Install the binaries with the following command:

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

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

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

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

```
eksctl version
```

Note
The `GitTag` version should be at least `0.7.0`. If not, check your terminal output for any installation or upgrade errors.

Launch a Guest Book Application

In this topic, you create a sample guest book application to test your Amazon EKS cluster.

> **Note**
> For more information about setting up the guest book example, see https://github.com/kubernetes/examples/blob/master/guestbook-go/README.md in the Kubernetes documentation.

To create your guest book application

1. Create the Redis master replication controller.

   ```
   kubectl apply -f https://raw.githubusercontent.com/kubernetes/examples/master/
   guestbook-go/redis-master-controller.json
   ```

   > **Note**
   > If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 151).
   > If you receive any other authorization or resource type errors, see Unauthorized or Access Denied (`kubectl`) (p. 235) in the troubleshooting section.

   Output:

   ```
   replicationcontroller "redis-master" created
   ```

2. Create the Redis master service.

   ```
   kubectl apply -f https://raw.githubusercontent.com/kubernetes/examples/master/
   guestbook-go/redis-master-service.json
   ```

   Output:

   ```
   service "redis-master" created
   ```

3. Create the Redis slave replication controller.

   ```
   kubectl apply -f https://raw.githubusercontent.com/kubernetes/examples/master/
   guestbook-go/redis-slave-controller.json
   ```

   Output:

   ```
   replicationcontroller "redis-slave" created
   ```

4. Create the Redis slave service.

   ```
   kubectl apply -f https://raw.githubusercontent.com/kubernetes/examples/master/
   guestbook-go/redis-slave-service.json
   ```

Output:

```
service "redis-slave" created
```

5. Create the guestbook replication controller.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/examples/master/
guestbook-go/guestbook-controller.json
```

   Output:

```
replicationcontroller "guestbook" created
```

6. Create the guestbook service.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/examples/master/
guestbook-go/guestbook-service.json
```

   Output:

```
service "guestbook" created
```

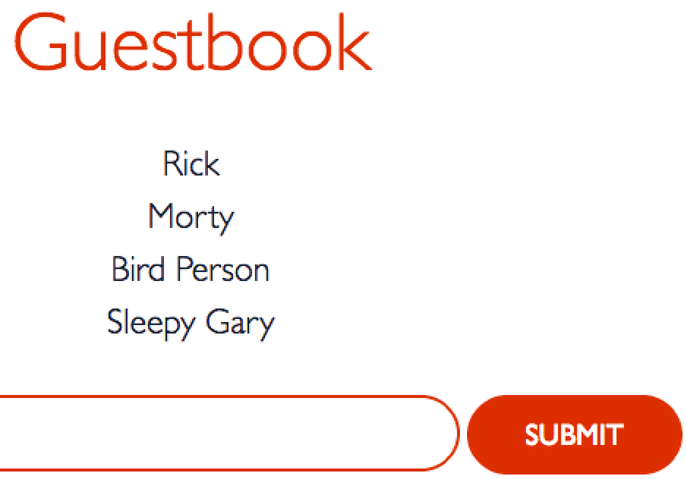
7. Query the services in your cluster and wait until the **External IP** column for the `guestbook` service is populated.

   Note
   It might take several minutes before the IP address is available.

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

8. After your external IP address is available, point a web browser to that address at port 3000 to view your guest book. For example, *`http://a7a95c2b9e69711e7b1a3022fdcfdf2e-1985673473.us-west-2.elb.amazonaws.com:3000`*

   Note
   It might take several minutes for DNS to propagate and for your guest book to show up.



Guestbook

Rick

Morty

Bird Person

Sleepy Gary

Tophat Jones

Important

If you are unable to connect to the external IP address with your browser, be sure that your corporate firewall is not blocking non-standards ports, like 3000. You can try switching to a guest network to verify.

To clean up your guest book application

When you are finished experimenting with your guest book application, you should clean up the resources that you created for it.

- The following command deletes all of the services and replication controllers for the guest book application:

```
kubectl delete rc/redis-master rc/redis-slave rc/guestbook svc/redis-master svc/redis-
slave svc/guestbook
```

Note

If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 151).

If you receive any other authorization or resource type errors, see Unauthorized or Access Denied (`kubectl`) (p. 235) in the troubleshooting section.

If you are done with your Amazon EKS cluster, you should delete it and its resources so that you do not incur additional charges. For more information, see Deleting a Cluster (p. 48).

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. This topic explains how to deploy the Kubernetes metrics server on your Amazon EKS cluster.

> **Note**
> The Kubernetes metrics server must be installed on your cluster to use the Horizontal Pod Autoscaler.

Choose the tab below that corresponds to your desired installation method:

curl and jq

To install `metrics-server` from GitHub on an Amazon EKS cluster using `curl` and `jq`

If you have a macOS or Linux system with `curl`, `tar`, `gzip`, and the `jq` JSON parser installed, you can download, extract, and install the latest release with the following commands. Otherwise, use the next procedure to download the latest version using a web browser.

1. Open a terminal window and navigate to a directory where you would like to download the latest `metrics-server` release.
2. Copy and paste the commands below into your terminal window and type **Enter** to execute them. These commands download the latest release, extract it, and apply the version 1.8+ manifests to your cluster.

```
DOWNLOAD_URL=$(curl --silent "https://api.github.com/repos/kubernetes-incubator/
metrics-server/releases/latest" | jq -r .tarball_url)
DOWNLOAD_VERSION=$(grep -o '[^/v]*$' <<< $DOWNLOAD_URL)
curl -Ls $DOWNLOAD_URL -o metrics-server-$DOWNLOAD_VERSION.tar.gz
mkdir metrics-server-$DOWNLOAD_VERSION
tar -xzf metrics-server-$DOWNLOAD_VERSION.tar.gz --directory metrics-server-
$DOWNLOAD_VERSION --strip-components 1
kubectl apply -f metrics-server-$DOWNLOAD_VERSION/deploy/1.8+/
```

3. 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             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
metrics-server   1         1         1            1           56m
```

Web browser

To install `metrics-server` from GitHub on an Amazon EKS cluster using a web browser

1. Download and extract the latest version of the metrics server code from GitHub.

a. Navigate to the latest release page of the `metrics-server` project on GitHub (https://github.com/kubernetes-incubator/metrics-server/releases/latest), then choose a source code archive for the latest release to download it.

   Note
   If you are downloading to a remote server, you can use the following `curl` command, substituting the red text with the latest version number.

   ```
   curl --remote-name --location https://github.com/kubernetes-incubator/
   metrics-server/archive/v0.3.4.tar.gz
   ```

b. Navigate to your downloads location and extract the source code archive. For example, if you downloaded the `.tar.gz` archive on a macOS or Linux system, use the following command to extract (substituting your release version).

   ```
   tar -xzf v0.3.4.tar.gz
   ```

2. Apply all of the YAML manifests in the `metrics-server-0.3.4/deploy/1.8+` directory (substituting your release version).

   ```
   kubectl apply -f metrics-server-0.3.4/deploy/1.8+/
   ```

3. 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             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
   metrics-server   1         1         1            1           56m
   ```

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. Helm is a package manager for Kubernetes clusters. For more information, see Using Helm with Amazon EKS (p. 172).

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. Follow the steps in Using Helm with Amazon EKS (p. 172) to get working `helm` and `tiller` terminal windows, so that you can install Helm charts.
2. In the Helm terminal window, run the following commands to deploy Prometheus.

   a. Create a Prometheus namespace.

   ```
   kubectl create namespace prometheus
   ```

   b. Deploy Prometheus.

   ```
   helm install stable/prometheus \
   --name prometheus \
   --namespace prometheus \
   --set
    alertmanager.persistentVolume.storageClass="gp2",server.persistentVolume.storageClass="gp2"
   ```

3. Verify that all of the pods in the `prometheus` namespace are in the `READY` state.

   ```
   kubectl get pods -n prometheus
   ```

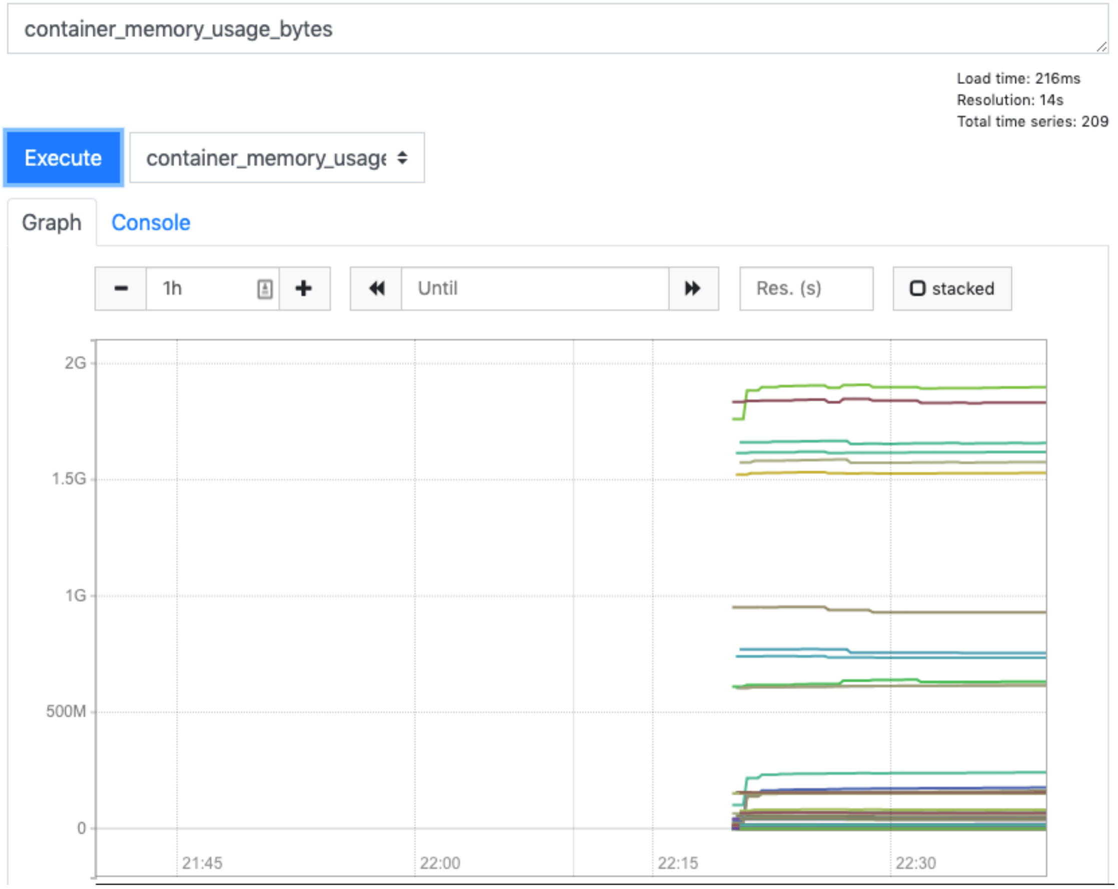
   Output:

   ```
   NAME                                             READY   STATUS    RESTARTS   AGE
   prometheus-alertmanager-848fb754f5-2wpbm         2/2     Running   0          85s
   prometheus-kube-state-metrics-86cbcf9b6f-drnfq   1/1     Running   0          85s
   prometheus-node-exporter-8qpcl                   1/1     Running   0          85s
   prometheus-node-exporter-czz9g                   1/1     Running   0          85s
   prometheus-node-exporter-ffsl9                   1/1     Running   0          85s
   prometheus-pushgateway-564f65fcc8-hmzp6          1/1     Running   0          85s
   prometheus-server-5b65bd569b-6wgwx               2/2     Running   0          85s
   ```

4. Use `kubectl` to port forward the Prometheus console to your local machine.

   ```
   kubectl --namespace=prometheus port-forward deploy/prometheus-server 9090
   ```

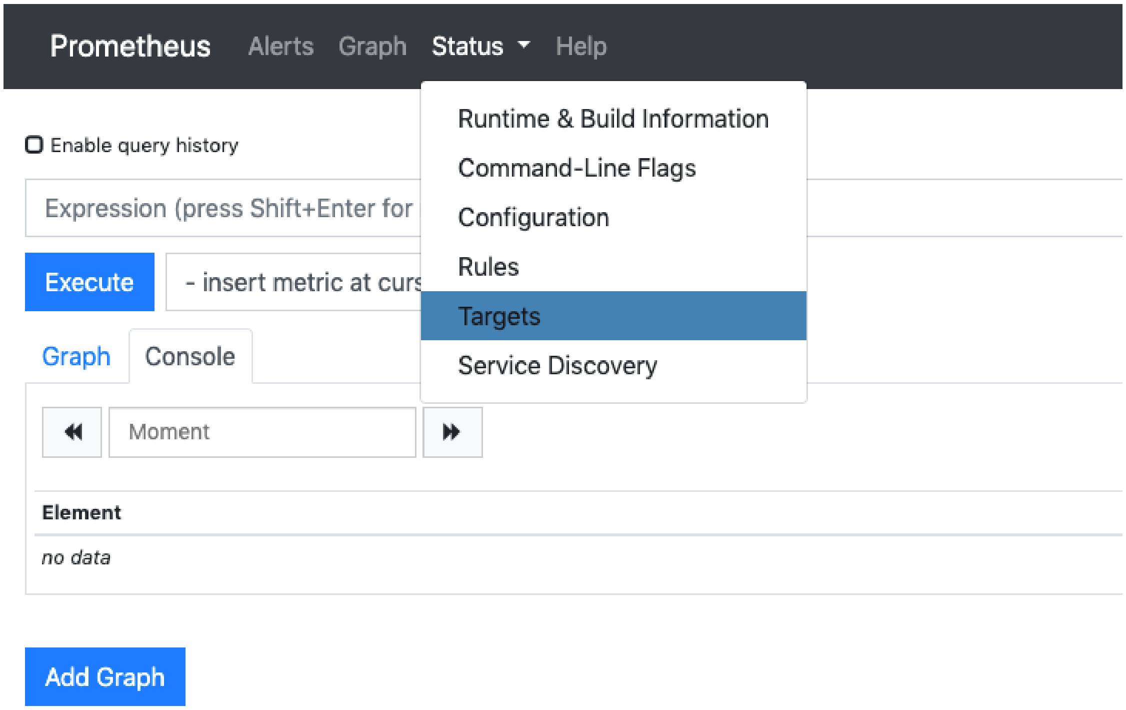
5. Point a web browser to localhost:9090 to view the Prometheus console.
6. Choose a metric from the **- insert metric at cursor** menu, then choose **Execute**. Choose the **Graph** tab to show the metric over time. The following image shows `container_memory_usage_bytes` over time.



```
container_memory_usage_bytes
```

Load time: 216ms
Resolution: 14s
Total time series: 209

7. From the top navigation bar, choose **Status**, then **Targets**.



All of the Kubernetes endpoints that are connected to Prometheus using service discovery are displayed.

Using Helm with Amazon EKS

The `helm` package manager for Kubernetes helps you install and manage applications on your Kubernetes cluster. For more information, see the Helm documentation. This topic helps you install and run the `helm` and `tiller` binaries locally so that you can install and manage charts using the `helm` CLI on your local system.

Although you can run the server-side `tiller` component in your cluster (and many public Helm installation articles offer only this option), running `tiller` locally in its own namespace as described in this topic reduces the risk of exploit for your cluster in the following ways:

- When you run the `tiller` server on your cluster, it gets its own Kubernetes Identity and associated permission set, often with full Kubernetes administrator permissions. This opens up the possibility for a privilege escalation, where an unprivileged Kubernetes user who has network access to the `tiller` server can gain additional Kubernetes permissions by way of installing a chart.
- When you run the `tiller` server on your local machine, users don't inherit the `tiller` server permissions on the cluster (likely full-admin), but instead `tiller` inherits the Kubernetes permissions of the end-user.
- Running `tiller` in its own namespace allows you to control access to the Kubernetes secrets that the `tiller` server stores by controlling access to that namespace.

Important
Before you can install Helm charts on your Amazon EKS cluster, you must configure **kubectl** to work for Amazon EKS. If you have not already done this, see Installing `aws-iam-authenticator` (p. 151) and Create a `kubeconfig` for Amazon EKS (p. 154) before proceeding. If the following command succeeds for your cluster, you're properly configured.

```
kubectl get svc
```

To install the `helm` and `tiller` binaries on your local system

1. - If you're using macOS with Homebrew, install the binaries with the following command.

     ```
     brew install kubernetes-helm
     ```

   - If you're using Windows with Chocolatey, install the binaries with the following command.

     ```
     choco install kubernetes-helm
     ```

   - Otherwise, see Installing the Helm Client in the Helm documentation.

     Important
     Don't proceed to install the `tiller` server-side component with the Helm documentation (stop before you reach Installing Tiller). This topic explains how to run `tiller` locally in its own namespace, which reduces the risk of exploit for your cluster.
2. To pick up the new binaries in your `PATH`, Close your current terminal window and open a new one.

To run `helm` and `tiller` locally

1. Create a namespace called `tiller` with the following command.

```
kubectl create namespace tiller
```

Note

By default, `tiller` stores its secrets in the `kube-system` namespace. Creating a namespace for `tiller` and specifying that namespace when you run it gives you more specific access controls to who is authorized to view the Helm chart secrets that `tiller` stores in your cluster.

2. Open a new terminal window for the `tiller` server. For the following steps, you need a terminal window for the `tiller` server and another window for the `helm` client.

   Important

   You should ensure that you are the only active user for the system that you use for the `tiller` server (such as a local laptop or desktop where you are the only user that is logged in). Otherwise, any user on your system could make requests to the `tiller` server with your Kubernetes permissions. For Linux and macOS systems, you can see the current users with the following command:

   ```
   users
   ```

   Output:

   ```
   ericn
   ```

   In the above example, there is only a single user named *`ericn`* on the system, so it is safe to proceed. If there are more than one user logged in to your system, you should use a different system, or consider launching an Amazon EC2 instance for this procedure so that you can ensure that you are the only active user.

3. In the `tiller` server terminal, set the `TILLER_NAMESPACE` environment variable to `tiller` and then start the `tiller` server.

   a. Set the `TILLER_NAMESPACE` environment variable to `tiller`.

      - **macOS and Linux**:

        ```
        export TILLER_NAMESPACE=tiller
        ```

      - **Windows (PowerShell)**:

        ```
        $env:TILLER_NAMESPACE = 'tiller'
        ```

   b. Start the `tiller` server.

      - **macOS and Linux**:

        ```
        tiller -listen=localhost:44134 -storage=secret -logtostderr
        ```

      - **Windows (PowerShell)**:

        ```
        tiller -listen=localhost:44134 -storage=secret
        ```

      Note

      By default, `tiller` stores release information in ConfigMaps; however, the latest Helm documentation recommends that you use the `-storage=secret` flag to store this information with Kubernetes secrets instead. For more information, see Tiller's Release Information in Securing your Helm Installation. The `-listen=localhost:44134` flag ensures that the `tiller` server only accepts requests from your local machine (this prevents unauthorized network users from accessing your local `tiller` process).

4. In the `helm` client terminal window, set the `HELM_HOST` environment variable to `:44134`.

   - **macOS and Linux**:

     ```
     export HELM_HOST=:44134
     ```

   - **Windows (PowerShell)**:

     ```
     $env:HELM_HOST = ':44134'
     ```

5. In the `helm` client terminal window, initialize the `helm` client.

   ```
   helm init --client-only
   ```

6. In the `helm` client terminal window, verify that `helm` is communicating with the `tiller` server properly.

   ```
   helm repo update
   ```

   Output:

   ```
   Hang tight while we grab the latest from your chart repositories...
   ...Skip local chart repository
   ...Successfully got an update from the "stable" chart repository
   Update Complete. # Happy Helming!#
   ```

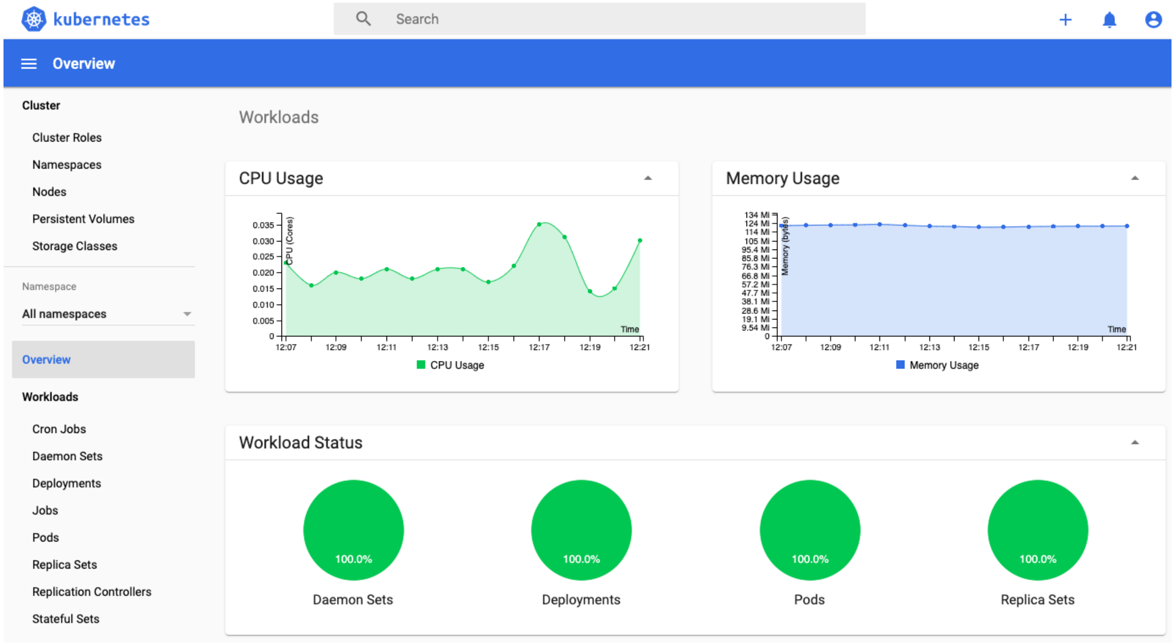
7. At this point, you can run any `helm` commands in your `helm` client terminal window (such as `helm install` *`chart_name`*) to install, modify, delete, or query Helm charts in your cluster. As you run `helm` commands, you can follow the `tiller` logs for those commands in its server terminal window. For more information, see Helm Commands and Charts in the Helm documentation.

   If you're just experimenting with `helm` and you don't have a specific chart to install, you can see Install an Example Chart in the Helm Quickstart Guide.

8. When you're finished, close your `helm` client and `tiller` server terminal windows. Repeat this procedure when you want to use `helm` with your cluster.

Tutorial: Deploy the Kubernetes Web UI (Dashboard)

This tutorial guides you through deploying the Kubernetes dashboard to your Amazon EKS cluster, complete with CPU and memory metrics. It also helps you to create an Amazon EKS administrator service account that you can use to securely connect to the dashboard to view and control your cluster.



Prerequisites

This tutorial assumes the following:

- You have created an Amazon EKS cluster by following the steps in Getting Started with Amazon EKS (p. 3).
- The security groups for your control plane elastic network interfaces and worker nodes follow the recommended settings in Cluster Security Group Considerations (p. 128).
- You are using a **kubectl** client that is configured to communicate with your Amazon EKS cluster (p. 16).

Step 1: Deploy the Kubernetes Metrics Server

The Kubernetes metrics server is an aggregator of resource usage data in your cluster, and it is not deployed by default in Amazon EKS clusters. The Kubernetes dashboard uses the metrics server to gather metrics for your cluster, such as CPU and memory usage over time. Choose the tab below that corresponds to your desired deployment method.

curl and jq

To install `metrics-server` from GitHub on an Amazon EKS cluster using `curl` and `jq`

If you have a macOS or Linux system with `curl`, `tar`, `gzip`, and the `jq` JSON parser installed, you can download, extract, and install the latest release with the following commands. Otherwise, use the next procedure to download the latest version using a web browser.

1. Open a terminal window and navigate to a directory where you would like to download the latest `metrics-server` release.
2. Copy and paste the commands below into your terminal window and type **Enter** to execute them. These commands download the latest release, extract it, and apply the version 1.8+ manifests to your cluster.

   ```
   DOWNLOAD_URL=$(curl --silent "https://api.github.com/repos/kubernetes-incubator/metrics-server/releases/latest" | jq -r .tarball_url)
   DOWNLOAD_VERSION=$(grep -o '[^/v]*$' <<< $DOWNLOAD_URL)
   curl -Ls $DOWNLOAD_URL -o metrics-server-$DOWNLOAD_VERSION.tar.gz
   mkdir metrics-server-$DOWNLOAD_VERSION
   tar -xzf metrics-server-$DOWNLOAD_VERSION.tar.gz --directory metrics-server-$DOWNLOAD_VERSION --strip-components 1
   kubectl apply -f metrics-server-$DOWNLOAD_VERSION/deploy/1.8+/
   ```
3. 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             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
   metrics-server   1         1         1            1           56m
   ```

Web browser

To install `metrics-server` from GitHub on an Amazon EKS cluster using a web browser

1. Download and extract the latest version of the metrics server code from GitHub.

   a. Navigate to the latest release page of the `metrics-server` project on GitHub (https://github.com/kubernetes-incubator/metrics-server/releases/latest), then choose a source code archive for the latest release to download it.

      Note
      If you are downloading to a remote server, you can use the following `curl` command, substituting the red text with the latest version number.

      ```
      curl --remote-name --location https://github.com/kubernetes-incubator/metrics-server/archive/v0.3.4.tar.gz
      ```

   b. Navigate to your downloads location and extract the source code archive. For example, if you downloaded the `.tar.gz` archive on a macOS or Linux system, use the following command to extract (substituting your release version).

      ```
      tar -xzf v0.3.4.tar.gz
      ```

2. Apply all of the YAML manifests in the `metrics-server-0.3.4/deploy/1.8+` directory (substituting your release version).

   ```
   kubectl apply -f metrics-server-0.3.4/deploy/1.8+/
   ```

3. 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             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
   metrics-server   1         1         1            1           56m
   ```

Step 2: Deploy the Dashboard

Use the following command to deploy the Kubernetes dashboard.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-beta4/aio/
deploy/recommended.yaml
```

Output:

```
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created
```

Step 3: Create an `eks-admin` Service Account and Cluster Role Binding

By default, the Kubernetes dashboard user has limited permissions. In this section, you create an `eks-admin` service account and cluster role binding that you can use to securely connect to the dashboard with admin-level permissions. For more information, see Managing Service Accounts in the Kubernetes documentation.

To create the `eks-admin` service account and cluster role binding

> **Important**
> The example service account created with this procedure has full `cluster-admin` (superuser) privileges on the cluster. For more information, see Using RBAC Authorization in the Kubernetes documentation.

1. Create a file called eks-admin-service-account.yaml with the text below. This manifest defines a service account and cluster role binding called eks-admin.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: eks-admin
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: eks-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: eks-admin
  namespace: kube-system
```

2. Apply the service account and cluster role binding to your cluster.

```
kubectl apply -f eks-admin-service-account.yaml
```

Output:

```
serviceaccount "eks-admin" created
clusterrolebinding.rbac.authorization.k8s.io "eks-admin" created
```

Step 4: Connect to the Dashboard

Now that the Kubernetes dashboard is deployed to your cluster, and you have an administrator service account that you can use to view and control your cluster, you can connect to the dashboard with that service account.

To connect to the Kubernetes dashboard

1. Retrieve an authentication token for the eks-admin service account. Copy the *<authentication_token>* value from the output. You use this token to connect to the dashboard.

```
kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep eks-admin | awk '{print $1}')
```

Output:

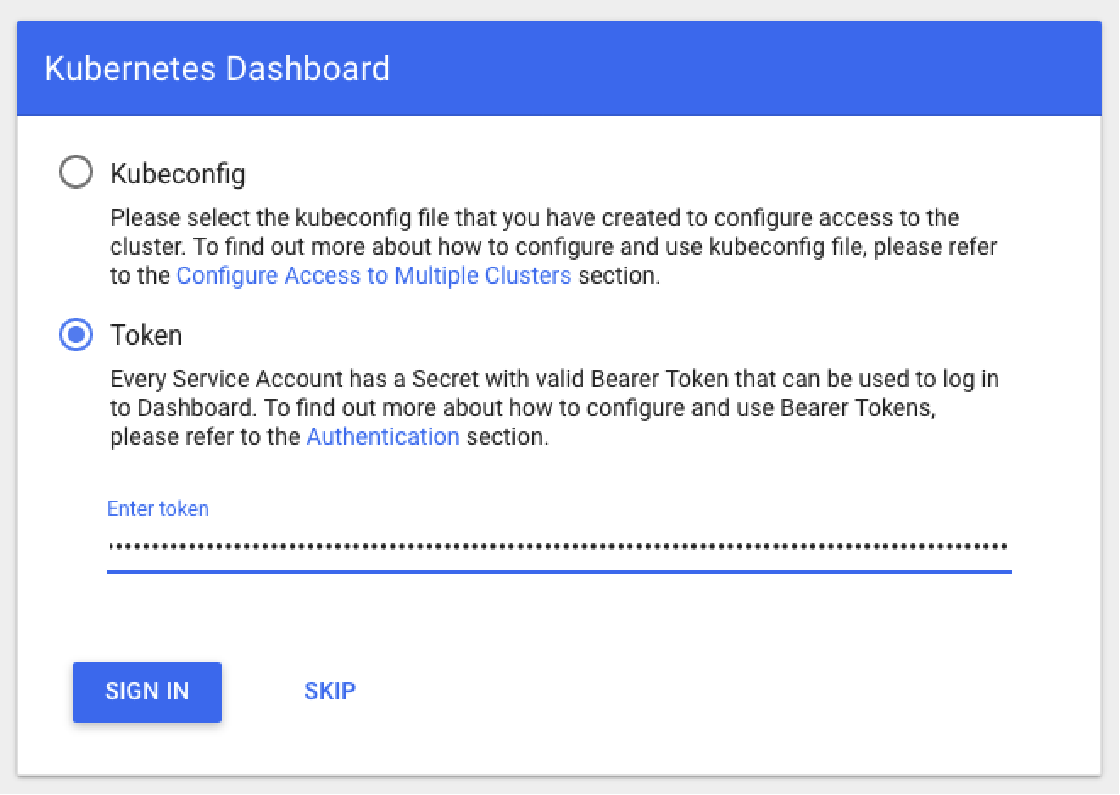
```
Name:         eks-admin-token-b5zv4
Namespace:    kube-system
Labels:       <none>
Annotations:  kubernetes.io/service-account.name=eks-admin
              kubernetes.io/service-account.uid=bcfe66ac-39be-11e8-97e8-026dce96b6e8

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:      1025 bytes
namespace:   11 bytes
token:       <authentication_token>
```

2. Start the **kubectl proxy**.

```
kubectl proxy
```

3. To access the dashboard endpoint, open the following link with a web browser: http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/#!/login

4. Choose **Token**, paste the *<authentication_token>* output from the previous command into the **Token** field, and choose **SIGN IN**.



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 cluster dashboard, you can view and control your cluster using your `eks-admin` service account. For more information about using the dashboard, see the project documentation on GitHub.

Getting Started with AWS App Mesh and Kubernetes

AWS App Mesh is a service mesh based on the Envoy proxy that makes it easy to monitor and control microservices. App Mesh standardizes how your microservices 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 microservice in an application. For more information, see the App Mesh User Guide.

This topic helps you to use AWS App Mesh with an existing microservice application running on Amazon EKS or Kubernetes on Amazon EC2. You can either integrate Kubernetes with App Mesh resources by completing the steps in this topic, or by installing the App Mesh Kubernetes integration components. The integration components automatically complete the tasks in this topic for you, enabling you to integrate with App Mesh directly from Kubernetes. For more information, see Configure App Mesh Integration with Kubernetes.

Prerequisites

App Mesh supports microservice applications that use service discovery naming for their components. To use this getting started guide, you must have a microservice application running on Amazon EKS or Kubernetes on AWS.

Kubernetes `kube-dns` and `coredns` are supported. For more information, see DNS for Services and Pods in the Kubernetes documentation.

Step 1: Create Your Service Mesh

A service mesh is a logical boundary for network traffic between the services that reside within it. For more information, see Service Meshes in the *AWS App Mesh User Guide*.

After you create your service mesh, you can create virtual services, virtual nodes, virtual routers, and routes to distribute traffic between the applications in your mesh.

To create a new service mesh with the AWS Management Console

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose **Create mesh**.
3. For **Mesh name**, specify a name for your service mesh.
4. Choose **Create mesh** to finish.

Step 2: Create Your Virtual Nodes

A virtual node acts as a logical pointer to a particular task group, such as a Kubernetes deployment. For more information, see Virtual Nodes in the *AWS App Mesh User Guide*.

When you create a virtual node, you must specify the DNS service discovery hostname for your task group. Any inbound traffic that your virtual node expects should be specified as a *listener*. Any outbound traffic that your virtual node expects to reach should be specified as a *backend*.

You must create virtual nodes for each microservice in your application.

To create a virtual node in the AWS Management Console.

1. Choose the mesh that you created in the previous steps.
2. Choose **Virtual nodes** in the left navigation.
3. Choose **Create virtual node**.
4. For **Virtual node name**, choose a name for your virtual node.
5. For **Service discovery method**, choose **DNS** for services that use DNS service discovery and then specify the hostname for **DNS hostname**. Otherwise, choose **None** if your virtual node doesn't expect any ingress traffic.
6. To specify any backends (for egress traffic) for your virtual node, or to configure inbound and outbound access logging information, choose **Additional configuration**.

   a. To specify a backend, choose **Add backend** and enter a virtual service name or full Amazon Resource Name (ARN) for the virtual service that your virtual node communicates with. Repeat this step until all of your virtual node backends are accounted for.

   b. To configure logging, enter the HTTP access logs path that you want Envoy to use. We recommend the `/dev/stdout` path so that you can use Docker log drivers to export your Envoy logs to a service such as Amazon CloudWatch Logs.

      Note

      Logs must still be ingested by an agent in your application and sent to a destination. This file path only instructs Envoy where to send the logs.
7. If your virtual node expects ingress traffic, specify a **Port** and **Protocol** for that **Listener**.
8. If you want to configure health checks for your listener, ensure that **Health check enabled** is selected and then complete the following substeps. If not, clear this check box.

   a. For **Health check protocol**, choose to use an HTTP or TCP health check.

   b. For **Health check port**, specify the port that the health check should run on.

   c. For **Healthy threshold**, specify the number of consecutive successful health checks that must occur before declaring the listener healthy.

   d. For **Health check interval**, specify the time period in milliseconds between each health check execution.

   e. For **Path**, specify the destination path for the health check request. This is required only if the specified protocol is HTTP. If the protocol is TCP, this parameter is ignored.

   f. For **Timeout period**, specify the amount of time to wait when receiving a response from the health check, in milliseconds.

   g. For **Unhealthy threshold**, specify the number of consecutive failed health checks that must occur before declaring the listener unhealthy.
9. Chose **Create virtual node** to finish.
10. Repeat this procedure as necessary to create virtual nodes for each remaining microservice in your application.

Step 3: Create Your Virtual Routers

Virtual routers handle traffic for one or more virtual services within your mesh. After you create a virtual router, you can create and associate routes for your virtual router that direct incoming requests to different virtual nodes. For more information, see Virtual Routers in the *AWS App Mesh User Guide*.

Create virtual routers for each microservice in your application.

Creating a virtual router in the AWS Management Console.

1. Choose **Virtual routers** in the left navigation.
2. Choose **Create virtual router**.
3. For **Virtual router name**, specify a name for your virtual router. Up to 255 letters, numbers, hyphens, and underscores are allowed.
4. For **Listener**, specify a **Port** and **Protocol** for your virtual router.
5. Choose **Create virtual router** to finish.
6. Repeat this procedure as necessary to create virtual routers for each remaining microservice in your application.

Step 4: Create Your Routes

A route is associated with a virtual router, and it's used to match requests for a virtual router and distribute traffic accordingly to its associated virtual nodes. For more information, see Routes in the *AWS App Mesh User Guide*.

Create routes for each microservice in your application.

Creating a route in the AWS Management Console.

1. Choose **Virtual routers** in the left navigation.
2. Choose the router that you want to associate a new route with.
3. In the **Routes** table, choose **Create route**.
4. For **Route name**, specify the name to use for your route.
5. For **Route type**, choose the protocol for your route.
6. For **Virtual node name**, choose the virtual node that this route will serve traffic to.
7. For **Weight**, choose a relative weight for the route. The total weight for all routes must be less than 100.
8. To use HTTP path-based routing, choose **Additional configuration** and then specify the path that the route should match. For example, if your virtual service name is `my-service.local` and you want the route to match requests to `my-service.local/metrics`, your prefix should be `/metrics`.
9. Choose **Create route** to finish.
10. Repeat this procedure as necessary to create routes for each remaining microservice in your application.

Step 5: Create Your Virtual Services

A virtual service is an abstraction of a real service that is provided by a virtual node directly or indirectly by means of a virtual router. Dependent services call your virtual service by its `virtualServiceName`, and those requests are routed to the virtual node or virtual router that is specified as the provider for the virtual service. For more information, see Virtual Services in the *AWS App Mesh User Guide*.

Create virtual services for each microservice in your application.

Creating a virtual service in the AWS Management Console.

1. Choose **Virtual services** in the left navigation.

2. Choose **Create virtual service**.
3. For **Virtual service name**, choose a name for your virtual service. We recommend that you use the service discovery name of the real service that you're targeting (such as `my-service.default.svc.cluster.local`).
4. For **Provider**, choose the provider type for your virtual service:
   - If you want the virtual service to spread traffic across multiple virtual nodes, select **Virtual router** and then choose the virtual router to use from the drop-down menu.
   - If you want the virtual service to reach a virtual node directly, without a virtual router, select **Virtual node** and then choose the virtual node to use from the drop-down menu.
   - If you don't want the virtual service to route traffic at this time (for example, if your virtual nodes or virtual router doesn't exist yet), choose **None**. You can update the provider for this virtual service later.
5. Choose **Create virtual service** to finish.
6. Repeat this procedure as necessary to create virtual services for each remaining microservice in your application.

Step 6: Updating Your Microservice Pod Specifications

App Mesh is a service mesh based on the Envoy proxy. After you create your service mesh, virtual services, virtual nodes, virtual routers, and routes, you must update your microservices to be compatible with App Mesh.

App Mesh vends the following custom container images that you must add to your Kubernetes pod specifications:

- App Mesh Envoy container image – `840364872350.dkr.ecr.`*`us-west-2`*`.amazonaws.com/aws-appmesh-envoy:v1.11.2.0-prod`. You can replace the *`Region`* with any Region that App Mesh is supported in. For a list of supported regions, see AWS Service Endpoints. Envoy uses the configuration defined in the App Mesh control plane to determine where to send your application traffic.

  You must use the App Mesh Envoy container image until the Envoy project team merges changes that support App Mesh. For additional details, see the GitHub roadmap issue.
- App Mesh proxy route manager – `111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-proxy-route-manager:v2`. The route manager sets up a pod's network namespace with `iptables` rules that route ingress and egress traffic through Envoy.

The following text is an example Kubernetes pod specification that you can merge with your existing application. Substitute your mesh name and virtual node name for the `APPMESH_VIRTUAL_NODE_NAME` value, and a list of ports that your application listens on for the `APPMESH_APP_PORTS` value. Substitute the Amazon EC2 instance AWS Region for the `AWS_REGION` value.

Update each microservice pod specification in your application to include these containers, and then deploy the new specifications to update your microservices and start using App Mesh with your Kubernetes application.

Example Kubernetes pod spec

```
spec:
  containers:
    - name: envoy
      image: 840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.11.2.0-prod
      securityContext:
        runAsUser: 1337
      env:
        - name: "APPMESH_VIRTUAL_NODE_NAME"
          value: "mesh/meshName/virtualNode/virtualNodeName"
        - name: "ENVOY_LOG_LEVEL"
          value: "info"
        - name: "AWS_REGION"
          value: "aws_region_name"
  initContainers:
    - name: proxyinit
      image: 111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-proxy-route-
manager:v2
      securityContext:
        capabilities:
          add:
            - NET_ADMIN
      env:
        - name: "APPMESH_START_ENABLED"
          value: "1"
        - name: "APPMESH_IGNORE_UID"
          value: "1337"
        - name: "APPMESH_ENVOY_INGRESS_PORT"
          value: "15000"
        - name: "APPMESH_ENVOY_EGRESS_PORT"
          value: "15001"
        - name: "APPMESH_APP_PORTS"
          value: "application_port_list"
        - name: "APPMESH_EGRESS_IGNORED_IP"
          value: "169.254.169.254"
```

Tutorial: Configure App Mesh Integration with Kubernetes

AWS App Mesh is a service mesh based on the Envoy proxy that makes it easy to monitor and control microservices. App Mesh standardizes how your microservices 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 microservice in an application. For more information, see the App Mesh User Guide.

When you use AWS App Mesh with Kubernetes, you manage App Mesh resources, such as virtual services and virtual nodes, that align to Kubernetes resources, such as services and deployments. You also add the App Mesh sidecar container images to Kubernetes pod specifications. This tutorial guides you through the installation of the following open source components that automatically complete these tasks for you when you work with Kubernetes resources:

- **App Mesh controller for Kubernetes** – The controller is accompanied by the deployment of three Kubernetes custom resource definitions: `mesh`, `virtual service`, and `virtual node`. The controller watches for creation, modification, and deletion of the custom resources and makes changes to the corresponding App Mesh `mesh`, `virtual service` (including `virtual router` and `route`), and `virtual node` resources through the App Mesh API. To learn more or contribute to the controller, see the GitHub project.
- **App Mesh sidecar injector for Kubernetes** – The injector installs as a webhook and injects the App Mesh sidecar container images into Kubernetes pods running in specific, labeled namespaces. To learn more or contribute, see the GitHub project.

The features discussed in this topic are available as an open-source beta. This means that these features are well tested. Support for the features will not be dropped, though details may change. If the schema or schematics of a feature changes, instructions for migrating to the next version will be provided. This migration may require deleting, editing, and re-creating Kubernetes API objects.

Prerequisites

To use the controller and sidecar injector, you must have the following resources:

- An existing Kubernetes cluster running version 1.11 or later. If you don't have an existing cluster, you can deploy one using the Getting Started with Amazon EKS guide.
- A `kubectl` client that is configured to communicate with your Kubernetes cluster. If you're using Amazon Elastic Kubernetes Service, you can use the instructions for installing `kubectl` and configuring a `kubeconfig` file.
- jq and Open SSL installed.

Step 1: Install the Controller and Custom Resources

To install the controller and Kubernetes custom resource definitions, complete the following steps.

1. The controller requires that your account and your Kubernetes worker nodes are able to work with App Mesh resources. Attach the AWSAppMeshFullAccess policy to the role that is attached to your Kubernetes worker nodes. If you are using a pod identity solution, make sure that the controller pod is bound to the policy.
2. To create the Kubernetes custom resources and launch the controller, download the following yaml file and apply it to your cluster with the following command.

   ```
   curl https://raw.githubusercontent.com/aws/aws-app-mesh-controller-for-k8s/master/
   deploy/all.yaml | kubectl apply -f -
   ```

   A Kubernetes namespace named `appmesh-system` is created and a container running the controller is deployed into the namespace.
3. Confirm that the controller is running with the following command.

   ```
   kubectl rollout status deployment app-mesh-controller -n appmesh-system
   ```

   If the controller is running, the following output is returned.

   ```
   deployment "app-mesh-controller" successfully rolled out
   ```
4. Confirm that the Kubernetes custom resources for App Mesh were created with the following command.

   ```
   kubectl get crd
   ```

   If the custom resources were created, output similar to the following is returned.

   ```
   NAME                               CREATED AT
   meshes.appmesh.k8s.aws             2019-05-08T14:17:26Z
   virtualnodes.appmesh.k8s.aws       2019-05-08T14:17:26Z
   virtualservices.appmesh.k8s.aws    2019-05-08T14:17:26Z
   ```

Step 2: Install the Sidecar Injector

To install the sidecar injector, complete the following steps. If you'd like to see the controller and injector in action, complete the steps in this section, but replace *`my-mesh`* in the first step with `color-mesh`, and then see the section called “Deploy a Mesh Connected Service” (p. 189).

1. Export the name of the mesh you want to create with the following command.

   ```
   export MESH_NAME=my-mesh
   ```
2. Download and execute the sidecar injector installation script with the following command.

   ```
   curl https://raw.githubusercontent.com/aws/aws-app-mesh-inject/master/scripts/
   install.sh | bash
   ```

   A Kubernetes namespace named `appmesh-inject` was created and a container running the injector was deployed into the namespace. If the injector successfully installed, the last several lines of the output returned are similar to the following text.

   ```
   deployment.apps/aws-app-mesh-inject configured
   ```

```
mutatingwebhookconfiguration.admissionregistration.k8s.io/aws-app-mesh-inject
 configured
waiting for aws-app-mesh-inject to start
deployment "aws-app-mesh-inject" successfully rolled out
Mesh name has been set up
App Mesh image has been set up
The injector is ready
```

Step 3: Configure App Mesh

When you deploy an application in Kubernetes, you also create the Kubernetes custom resources so that the controller can create the corresponding App Mesh resources. Additionally, you must enable sidecar injection so that the App Mesh sidecar container images are deployed in each Kubernetes pod.

Create Kubernetes Custom Resources

You can deploy mesh, virtual service, and virtual node custom resources in Kubernetes, which then triggers the controller to create the corresponding resources in App Mesh through the App Mesh API.

Create a Mesh

When you create a mesh custom resource, you trigger the creation of an App Mesh mesh. The mesh name that you specify must be the same as the mesh name you exported when you installed the sidecar injector (p. 186). If the mesh name that you specify already exists, a new mesh is not created.

```
apiVersion: appmesh.k8s.aws/v1beta1
kind: Mesh
metadata:
  name: my-mesh
```

Create a Virtual Service

When you create a virtual service custom resource, you trigger the creation of an App Mesh virtual service, virtual router, and one or more routes containing a route configuration. The virtual service allows requests from one application in the mesh to be routed to a number of virtual nodes that make up a service.

```
apiVersion: appmesh.k8s.aws/v1beta1
kind: VirtualService
metadata:
  name: my-svc-a
  namespace: my-namespace
spec:
  meshName: my-mesh
  routes:
    - name: route-to-svc-a
      http:
        match:
          prefix: /
        action:
          weightedTargets:
            - virtualNodeName: my-app-a
              weight: 1
```

Create a Virtual Node

When you create a virtual node custom resource, you trigger the creation of an App Mesh virtual node. The virtual node contains listener, back-end, and service discovery configuration.

```
apiVersion: appmesh.k8s.aws/v1beta1
kind: VirtualNode
metadata:
  name: my-app-a
  namespace: my-namespace
spec:
  meshName: my-mesh
  listeners:
    - portMapping:
        port: 9000
        protocol: http
  serviceDiscovery:
    dns:
      hostName: my-app-a.my-namespace.svc.cluster.local
  backends:
    - virtualService:
        virtualServiceName: my-svc-a
```

Sidecar Injection

You enable sidecar injection for a Kubernetes namespace. When necessary, you can override the injector's default behavior for each pod you deploy in a Kubernetes namespace that you've enabled the injector for.

Enable Sidecar Injection for a Namespace

To enable the sidecar injector for a Kubernetes namespace, label the namespace with the following command.

```
kubectl label namespace my-namespace appmesh.k8s.aws/sidecarInjectorWebhook=enabled
```

The App Mesh sidecar container images will be automatically injected into each pod that you deploy into the namespace.

Override Sidecar Injector Default Behavior

To override the default behavior of the injector when deploying a pod in a namespace that you've enabled the injector for, add any of the following annotations to your pod spec.

- *appmesh.k8s.aws/mesh:* `mesh-name` – Add when you want to use a different mesh name than the one that you specified when you installed the injector.
- *appmesh.k8s.aws/ports: "*`ports`*"* – Specify particular ports when you don't want all of the container ports defined in a pod spec passed to the sidecars as application ports.
- *appmesh.k8s.aws/egressIgnoredPorts:* `ports` – Specify a comma separated list of port numbers for outbound traffic that you want ignored. By default all outbound traffic ports will be routed, except port 22 (SSH).
- *appmesh.k8s.aws/virtualNode:* `virtual-node-name` – Specify your own name if you don't want the virtual node name passed to the sidecars to be `<deployment name>--<namespace>`.
- *appmesh.k8s.aws/sidecarInjectorWebhook: disabled* – Add when you don't want the injector enabled for a pod.

```
apiVersion: appmesh.k8s.aws/v1beta1
kind: Deployment
spec:
    metadata:
      annotations:
        appmesh.k8s.aws/mesh: my-mesh2
        appmesh.k8s.aws/ports: "8079,8080"
        appmesh.k8s.aws/egressIgnoredPorts: "3306"
        appmesh.k8s.aws/virtualNode: my-app
        appmesh.k8s.aws/sidecarInjectorWebhook: disabled
```

Step 4: Remove Integration Components (Optional)

If you need to remove the Kubernetes integration components, run the following commands.

```
kubectl delete crd meshes.appmesh.k8s.aws
kubectl delete crd virtualnodes.appmesh.k8s.aws
kubectl delete crd virtualservices.appmesh.k8s.aws
kubectl delete namespace appmesh-system
kubectl delete namespace appmesh-inject
```

Deploy a Mesh Connected Service

In this topic, you deploy a sample application on Kubernetes. The application deploys mesh, virtual service, and virtual node Kubernetes custom resources. Kubernetes automatically creates mesh, virtual service, and virtual node resources in App Mesh and injects the App Mesh sidecar images into Kubernetes pods.

Prerequisites

Before you deploy the sample application, you must meet the following prerequisites:

- Meet all of the prerequisites in *Tutorial: Configure App Mesh Integration with Kubernetes* (p. 185).
- Have the App Mesh controller for Kubernetes and the App Mesh sidecar injector for Kubernetes installed and configured. When you install the sidecar injector, specify *color-mesh* as the name of your mesh. To learn more about the controller and sidecar injector and how to install and configure them, see *Tutorial: Configure App Mesh Integration with Kubernetes* (p. 185).

Deploy a Sample Application

The sample application consists of two components:

- **ColorGateway** – A simple http service written in Go that is exposed to external clients and that responds to *http://service-name:port/color*. The gateway responds with a color retrieved from *color-teller* and a histogram of colors observed at the server that responded up to the point when you made the request.
- **ColorTeller** – A simple http service written in Go that is configured to return a color. Multiple variants of the service are deployed. Each service is configured to return a specific color.

1. To deploy the color mesh sample application, download the following file and apply it to your Kubernetes cluster with the following command.

```
curl https://raw.githubusercontent.com/aws/aws-app-mesh-controller-for-k8s/v0.1.0/
examples/color.yaml | kubectl apply -f -
```

2. View the resources deployed by the sample application with the following command.

```
kubectl -n appmesh-demo get all
```

In the output, you see a collection of virtual services, virtual nodes, and mesh custom resources along with native Kubernetes deployments, pods, and services. Your output will be similar to the following output.

```
NAME                                     READY     STATUS    RESTARTS   AGE
pod/colorgateway-cc6464d75-4ktj4         2/2       Running   0          37s
pod/colorteller-86664b5956-6h26c         2/2       Running   0          36s
pod/colorteller-black-6787756c7b-dw82f   2/2       Running   0          36s
pod/colorteller-blue-55d6f99dc6-f5wgd    2/2       Running   0          36s
pod/colorteller-red-578866ffb-x9m7w      2/2       Running   0          35s

NAME                        TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/colorgateway        ClusterIP   10.100.21.147    <none>        9080/TCP   37s
service/colorteller         ClusterIP   10.100.187.50    <none>        9080/TCP   37s
service/colorteller-black   ClusterIP   10.100.61.36     <none>        9080/TCP   36s
service/colorteller-blue    ClusterIP   10.100.254.230   <none>        9080/TCP   36s
service/colorteller-red     ClusterIP   10.100.90.38     <none>        9080/TCP   36s

NAME                                DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/colorgateway        1         1         1            1           37s
deployment.apps/colorteller         1         1         1            1           36s
deployment.apps/colorteller-black   1         1         1            1           36s
deployment.apps/colorteller-blue    1         1         1            1           36s
deployment.apps/colorteller-red     1         1         1            1           36s

NAME                                           DESIRED   CURRENT   READY     AGE
replicaset.apps/colorgateway-cc6464d75         1         1         1         37s
replicaset.apps/colorteller-86664b5956         1         1         1         36s
replicaset.apps/colorteller-black-6787756c7b   1         1         1         36s
replicaset.apps/colorteller-blue-55d6f99dc6    1         1         1         36s
replicaset.apps/colorteller-red-578866ffb      1         1         1         35s

NAME                                                       AGE
virtualservice.appmesh.k8s.aws/colorgateway.appmesh-demo   37s
virtualservice.appmesh.k8s.aws/colorteller.appmesh-demo    37s

NAME                              AGE
mesh.appmesh.k8s.aws/color-mesh   38s

NAME                                            AGE
virtualnode.appmesh.k8s.aws/colorgateway        39s
virtualnode.appmesh.k8s.aws/colorteller         39s
virtualnode.appmesh.k8s.aws/colorteller-black   39s
virtualnode.appmesh.k8s.aws/colorteller-blue    39s
virtualnode.appmesh.k8s.aws/colorteller-red     38s
```

You can use the AWS Management Console or AWS CLI to see the App Mesh `mesh`, `virtual service`, `virtual router`, `route`, and `virtual node` resources that were automatically created by the controller. All of the resources were deployed to the `appmesh-demo` namespace, which was labelled with `appmesh.k8s.aws/sidecarInjectorWebhook: enabled`. Since the injector saw this label for the namespace, it injected the App Mesh sidecar container images into each of the pods. Using `kubectl describe pod <pod-name> -n appmesh-demo`, you can see that the App Mesh sidecar container images are included in each of the pods that were deployed.

Run Application

Complete the following steps to run the application.

1. In a terminal, use the following command to create a container in the *appmesh-demo* namespace that has `curl` installed and open a shell to it. In later steps, this terminal is referred to as *Terminal A*.

```
kubectl run -n appmesh-demo -it curler --image=tutum/curl /bin/bash
```

2. From *Terminal A*, run the following command to curl the color gateway in the color mesh application 100 times. The gateway routes traffic to separate virtual nodes that return either white, black, or blue as a response.

```
for i in {1..100}; do curl colorgateway:9080/color; echo; done
```

   100 responses are returned. Each response looks similar to the following text:

```
{"color":"blue", "stats": {"black":0.36,"blue":0.32,"white":0.32}}
```

   In this line of output, the colorgateway routed the request to the blue virtual node. The numbers for each color denote the percentage of responses from each virtual node. The number for each color in each response is cumulative over time. The percentage is similar for each color because, by default, the weighting defined for each virtual node is the same in the *color.yaml* file you used to install the sample application.

   Leave *Terminal A* open.

Change Configuration

Change the configuration and run the application again to see the effect of the changes.

1. In a separate terminal from *Terminal A*, edit the *colorteller.appmesh-demo* virtual service with the following command.

```
kubectl edit VirtualService colorteller.appmesh-demo -n appmesh-demo
```

   In the editor, you can see that the *weight* value of each **virtualNodeName** is *1*. Because the weight of each virtual node is the same, traffic routed to each virtual node is approximately even. To route all traffic to the black node only, change the values for **colorteller.appmesh-demo** and **colorteller-blue** to *0*, as shown in the following text. Save the configuration and exit the editor.

```
spec:
  meshName: color-mesh
  routes:
  - http:
      action:
        weightedTargets:
        - virtualNodeName: colorteller.appmesh-demo
          weight: 0
        - virtualNodeName: colorteller-blue
          weight: 0
        - virtualNodeName: colorteller-black.appmesh-demo
          weight: 1
```

2. In *Terminal A*, run `curl` again with the following command.

   ```
   for i in {1..100}; do curl colorgateway:9080/color; echo; done
   ```

   This time, all lines of output look similar to the following text.

   ```
   {"color":"black", "stats": {"black":0.64,"blue":0.18,"white":0.19}}
   ```

   Black is the response every time because the gateway is now routing all traffic to the black virtual node. Even though all traffic is now going to black, the white and blue virtual nodes still have response percentages, because the numbers are based on relative percentages over time. When you executed the requests in a previous step, white and blue responded, which is why they still have response percentages. You can see that the relative percentages decrease for white and blue with each response, while the percentage for black increases.

Remove Application

When you've finished with the sample application, you can remove it by completing the following steps.

1. Use the following commands to remove the sample application and the App Mesh resources that were created.

   ```
   kubectl delete namespace appmesh-demo
   kubectl delete mesh color-mesh
   ```

2. Optional: If you want to remove the controller and sidecar injector, see Remove integration components (p.).

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 ECR). Deep Learning Containers provide optimized environments with TensorFlow, Nvidia CUDA (for GPU instances), and Intel MKL (for CPU instances) libraries and are available in Amazon ECR.

To get started using AWS Deep Learning Containers on Amazon EKS, see AWS Deep Learning Containers on Amazon EKS in the *AWS Deep Learning AMI Developer Guide*.

Security in Amazon EKS

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. For Amazon EKS, AWS is responsible for the Kubernetes control plane, which includes the control plane nodes and `etcd` database. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon EKS, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility includes the following areas.
  - The security configuration of the data plane, including the configuration of the security groups that allow traffic to pass from the Amazon EKS control plane into the customer VPC
  - The configuration of the worker nodes and the containers themselves
  - The worker node guest operating system (including updates and security patches)
  - Other associated application software:
    - Setting up and managing network controls, such as firewall rules
    - Managing platform-level identity and access management, either with or in addition to IAM
  - The sensitivity of your data, your company's requirements, and applicable laws and regulations

This documentation helps you understand how to apply the shared responsibility model when using Amazon EKS. The following topics show you how to configure Amazon EKS to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon EKS resources.

Topics

- Identity and Access Management for Amazon EKS (p. 194)
- Logging and Monitoring in Amazon EKS (p. 220)
- Compliance Validation for Amazon EKS (p. 220)
- Resilience in Amazon EKS (p. 220)
- Infrastructure Security in Amazon EKS (p. 221)
- Configuration and Vulnerability Analysis in Amazon EKS (p. 222)
- Pod Security Policy (p. 222)

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.

Topics

- Audience (p. 195)
- Authenticating With Identities (p. 195)
- Managing Access Using Policies (p. 197)
- How Amazon EKS Works with IAM (p. 198)
- Amazon EKS Identity-Based Policy Examples (p. 201)
- Amazon EKS Service IAM Role (p. 203)
- Amazon EKS Worker Node IAM Role (p. 205)
- IAM Roles for Service Accounts (p. 208)
- Troubleshooting Amazon EKS Identity and Access (p. 219)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in Amazon EKS.

Service user – If you use the Amazon EKS service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon EKS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon EKS, see Troubleshooting Amazon EKS Identity and Access (p. 219).

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. 198).

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. 201).

Authenticating With Identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see The IAM Console and Sign-in Page in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication, or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email or your IAM user name. You can access AWS programmatically using your root user or IAM user access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don’t use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using Multi-Factor Authentication (MFA) in AWS in the *IAM User Guide*.

AWS Account Root User

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM Users and Groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing Access Keys for IAM Users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to Create an IAM User (Instead of a Role) in the *IAM User Guide*.

IAM Roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM Roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.
- **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role

for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances in the *IAM User Guide*.

To learn whether to use IAM roles, see When to Create an IAM Role (Instead of a User) in the *IAM User Guide*.

Managing Access Using Policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, IAM user, or IAM role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON Policies in the *IAM User Guide*.

An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-Based Policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM Policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing Between Managed Policies and Inline Policies in the *IAM User Guide*.

Resource-Based Policies

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

Access Control Lists (ACLs)

Access control policies (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they are the only policy type that does not use the JSON policy document format. Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access Control List (ACL) Overview in the *Amazon Simple Storage Service Developer Guide*.

Other Policy Types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions Boundaries for IAM Entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs Work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session Policies in the *IAM User Guide*.

Multiple Policy Types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy Evaluation Logic in the *IAM User Guide*.

How Amazon EKS Works with IAM

Before you use IAM to manage access to Amazon EKS, you should understand what IAM features are available to use with Amazon EKS. To get a high-level view of how Amazon EKS and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

Topics

- Amazon EKS Identity-Based Policies (p. 199)
- Amazon EKS Resource-Based Policies (p. 200)
- Authorization Based on Amazon EKS Tags (p. 200)
- Amazon EKS IAM Roles (p. 200)

Amazon EKS Identity-Based Policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon EKS supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

Actions

The `Action` element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon EKS use the following prefix before the action: `eks:`. For example, to grant someone permission to get descriptive information about an Amazon EKS cluster, you include the `DescribeCluster` action in their policy. Policy statements must include either an `Action` or `NotAction` element.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": ["eks:action1", "eks:action2"]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "eks:Describe*"
```

To see a list of Amazon EKS actions, see Actions Defined by Amazon Elastic Kubernetes Service in the *IAM User Guide*.

Resources

The `Resource` element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. You specify a resource using an ARN or using the wildcard (*) to indicate that the statement applies to all resources.

The Amazon EKS cluster resource has the following ARN:

```
arn:${Partition}:eks:${Region}:${Account}:cluster/${ClusterName}
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

For example, to specify the `dev` cluster in your statement, use the following ARN:

```
"Resource": "arn:aws:eks:us-east-1:123456789012:cluster/dev"
```

To specify all clusters that belong to a specific account and Region, use the wildcard (*):

```
"Resource": "arn:aws:eks:us-east-1:123456789012:cluster/*"
```

Some Amazon EKS actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

To see a list of Amazon EKS resource types and their ARNs, see Resources Defined by Amazon Elastic Kubernetes Service in the *IAM User Guide*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon Elastic Kubernetes Service.

Condition Keys

Amazon EKS does not provide any service-specific condition keys, but it does support using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the *IAM User Guide*.

Examples

To view examples of Amazon EKS identity-based policies, see Amazon EKS Identity-Based Policy Examples (p. 201).

When you create an Amazon EKS cluster, the IAM entity user or role, such as a federated user that creates the cluster, is automatically granted `system:masters` permissions in the cluster's RBAC configuration. To grant additional AWS users or roles the ability to interact with your cluster, you must edit the `aws-auth` ConfigMap within Kubernetes.

For additional information about working with the ConfigMap, see Managing Users or IAM Roles for your Cluster (p. 157).

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. 226).

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

Amazon EKS does not support service-linked roles.

Service Roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Amazon EKS supports service roles. For more information, see the section called “Service IAM Role” (p. 203) and the section called “Worker Node IAM Role” (p. 205).

Choosing an IAM Role in Amazon EKS

When you create a cluster resource in Amazon EKS, you must choose a role to allow Amazon EKS to access several other AWS resources on your behalf. If you have previously created a service role, then Amazon EKS provides you with a list of roles to choose from. It's important to choose a role that has the Amazon EKS managed policies attached to it. For more information, see the section called “Check for an Existing Service Role” (p. 204) and the section called “Check for an Existing Worker Node Role” (p. 206).

Amazon EKS Identity-Based Policy Examples

By default, IAM users and roles don't have permission to create or modify Amazon EKS resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the *IAM User Guide*.

When you create an Amazon EKS cluster, the IAM entity user or role, such as a federated user that creates the cluster, is automatically granted `system:masters` permissions in the cluster's RBAC configuration. To grant additional AWS users or roles the ability to interact with your cluster, you must edit the `aws-auth` ConfigMap within Kubernetes.

For additional information about working with the ConfigMap, see Managing Users or IAM Roles for your Cluster (p. 157).

Topics

- Policy Best Practices (p. 201)
- Using the Amazon EKS Console (p. 202)
- Allow Users to View Their Own Permissions (p. 202)
- Update a Kubernetes cluster (p. 203)
- List or describe all clusters (p. 203)

Policy Best Practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon EKS resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get Started Using AWS Managed Policies** – To start using Amazon EKS quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get Started Using Permissions With AWS Managed Policies in the *IAM User Guide*.
- **Grant Least Privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant Least Privilege in the *IAM User Guide*.
- **Enable MFA for Sensitive Operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using Multi-Factor Authentication (MFA) in AWS in the *IAM User Guide*.

- **Use Policy Conditions for Extra Security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON Policy Elements: Condition in the *IAM User Guide*.

Using the Amazon EKS Console

To access the Amazon EKS console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon EKS resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the Amazon EKS console, create a policy with your own unique name, such as `AmazonEKSAdminPolicy`. Attach the policy to the entities. For more information, see Adding Permissions to a User in the *IAM User Guide*:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:*"
            ],
            "Resource": "*"
        }
    ]
}
```

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Allow Users to View Their Own Permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
       "Version": "2012-10-17",
       "Statement": [
           {
               "Sid": "ViewOwnUserInfo",
               "Effect": "Allow",
               "Action": [
                   "iam:GetUserPolicy",
                   "iam:ListGroupsForUser",
                   "iam:ListAttachedUserPolicies",
                   "iam:ListUserPolicies",
                   "iam:GetUser"
               ],
               "Resource": [
                   "arn:aws:iam::*:user/${aws:username}"
               ]
           },
           {
               "Sid": "NavigateInConsole",
               "Effect": "Allow",
               "Action": [
                   "iam:GetGroupPolicy",
                   "iam:GetPolicyVersion",
                   "iam:GetPolicy",
                   "iam:ListAttachedGroupPolicies",
                   "iam:ListGroupPolicies",
                   "iam:ListPolicyVersions",
                   "iam:ListPolicies",
                   "iam:ListUsers"
               ],
               "Resource": "*"
           }
       ]
   }
```

Update a Kubernetes cluster

This example shows how you can create a policy that allows a user to update the Kubernetes version of any *dev* cluster for an account, in any region.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:UpdateClusterVersion",
            "Resource": "arn:aws:eks:*:111122223333:cluster/dev"
        }
    ]
}
```

List or describe all clusters

This example shows how you can create a policy that allows a user read-only access to list or describe all clusters. An account must be able to list and describe clusters to use the `update-kubeconfig` AWS CLI command.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:ListClusters"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon EKS Service IAM Role

Amazon EKS makes 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:

- `AmazonEKSServicePolicy`
- `AmazonEKSClusterPolicy`

Check for an Existing Service Role

You can use the following procedure to check and see if your account already has the Amazon EKS service role.

To check for the `eksServiceRole` in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Search the list of roles for `eksServiceRole` or `AWSServiceRoleForAmazonEKS`. If the role does not exist, see Creating the Amazon EKS Service Role (p. 204) to create the role. If the role does exist, select the role to view the attached policies.
4. Choose **Permissions**.
5. Ensure that the **AmazonEKSServicePolicy** and **AmazonEKSClusterPolicy** managed policies are attached to the role. If the policies are attached, your Amazon EKS service 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 Service Role

You can use the following procedure to create the Amazon EKS service role if you do not already have one for your account.

To create your Amazon EKS service 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 **Allows Amazon EKS to manage your clusters on your behalf** for your use case, 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 `eksServiceRole`, then choose **Create role**.

To create your Amazon EKS service 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 Service Role'


Resources:

  eksServiceRole:
    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/AmazonEKSServicePolicy
        - 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 eksServiceRole.Arn
    Export:
      Name: !Sub "${AWS::StackName}-RoleArn"
```

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 `eksServiceRole`, and then choose **Next**.
7. On the **Configure stack options** page, choose **Next**.
8. On the **Review** page, review your information, acknowledge that the stack might create IAM resources, and then choose **Create stack**.

Amazon EKS Worker Node IAM Role

The Amazon EKS worker node `kubelet` daemon makes calls to AWS APIs on your behalf. Worker nodes receive permissions for these API calls through an IAM instance profile and associated policies. Before you can launch worker nodes and register them into a cluster, you must create an IAM role for those worker nodes to use when they are launched. This requirement applies to worker nodes launched with the Amazon EKS-optimized AMI provided by Amazon, or with any other worker node AMIs that you intend to use. Before you create worker nodes, you must create an IAM role with the following IAM policies:

- `AmazonEKSWorkerNodePolicy`
- `AmazonEKS_CNI_Policy`
- `AmazonEC2ContainerRegistryReadOnly`

Check for an Existing Worker Node Role

You can use the following procedure to check and see if your account already has the Amazon EKS worker node role.

To check for the `NodeInstanceRole` in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Search the list of roles for `NodeInstanceRole`. If the role does not exist, see Creating the Amazon EKS Worker Node Role (p. 206) to create the role. If the role does exist, select the role to view the attached policies.
4. Choose **Permissions**.
5. Ensure that the **AmazonEKSWorkerNodePolicy**, **AmazonEKS_CNI_Policy**, and **AmazonEC2ContainerRegistryReadOnly** managed policies are attached to the role. If the policies are attached, your Amazon EKS worker node role is properly configured.
6. Choose **Trust Relationships**, **Edit Trust Relationship**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

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

Creating the Amazon EKS Worker Node Role

If you created your worker nodes by following the steps in the Getting Started with the AWS Management Console (p. 11) or Getting Started with `eksctl` (p. 3) topics, then the worker node role account already exists and you don't need to manually create it. You can use the following procedure to create the Amazon EKS worker node role if you do not already have one for your account.

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

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Roles**, then **Create role**.

3. Choose **EC2** from the list of services, then **Next: Permissions**.
4. Select the following policies:

   - **AmazonEKSWorkerNodePolicy**
   - **AmazonEKS_CNI_Policy**
   - **AmazonEC2ContainerRegistryReadOnly**
5. Choose **Next: Tags**.
6. (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*.
7. Choose **Next: Review**.
8. For **Role name**, enter a unique name for your role, such as `NodeInstanceRole`, then choose **Create role**.

To create your Amazon EKS instance 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 Worker Node Role'


Resources:

  NodeInstanceRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Principal:
              Service: ec2.amazonaws.com
            Action: sts:AssumeRole
      Path: "/"
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
        - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
        - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly

Outputs:

  RoleArn:
    Description: The role that the worker node kubelet uses to make calls to the Amazon
 EKS API on your behalf
    Value: !GetAtt NodeInstanceRole.Arn
    Export:
      Name: !Sub "${AWS::StackName}-RoleArn"
```

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 `NodeInstanceRole`, 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**.

IAM Roles for Service Accounts

With IAM roles for service accounts on Amazon EKS clusters, you can associate an IAM role with a Kubernetes service account. This service account can then provide AWS permissions to the containers in any pod that uses that service account. With this feature, you no longer need to provide extended permissions to the worker node IAM role so that pods on that node can call AWS APIs.

Applications must sign their AWS API requests with AWS credentials. This feature provides a strategy for managing credentials for your applications, similar to the way that Amazon EC2 instance profiles provide credentials to Amazon EC2 instances. Instead of creating and distributing your AWS credentials to the containers or using the Amazon EC2 instance's role, you can associate an IAM role with a Kubernetes service account. The applications in the pod's containers can then use an AWS SDK or the AWS CLI to make API requests to authorized AWS services.

The IAM roles for service accounts feature provides the following benefits:

- **Least privilege —** By using the IAM roles for service accounts feature, you no longer need to provide extended permissions to the worker node IAM role so that pods on that node can call AWS APIs. You can scope IAM permissions to a service account, and only pods that use that service account have access to those permissions. This feature also eliminates the need for third-party solutions such as `kiam` or `kube2iam`.
- **Credential isolation —** A container can only retrieve credentials for the IAM role that is associated with the service account to which it belongs. A container never has access to credentials that are intended for another container that belongs to another pod.
- **Auditability —** Access and event logging is available through CloudTrail to help ensure retrospective auditing.

To get started, see Enabling IAM Roles for Service Accounts on your Cluster (p. 212).

For an end-to-end walkthrough using `eksctl`, see Walkthrough: Amazon VPC CNI Plugin for Kubernetes (p. 218).

Topics

- IAM Roles for Service Accounts Technical Overview (p. 208)
- Using a Supported AWS SDK (p. 211)
- Enabling IAM Roles for Service Accounts on your Cluster (p. 212)
- Creating an IAM Role and Policy for your Service Account (p. 213)
- Specifying an IAM Role for your Service Account (p. 217)
- Restricting Access to Amazon EC2 Instance Profile Credentials (p. 217)
- Walkthrough: Amazon VPC CNI Plugin for Kubernetes (p. 218)

IAM Roles for Service Accounts Technical Overview

In 2014, AWS Identity and Access Management added support for federated identities using OpenID Connect (OIDC). This feature allows you to authenticate AWS API calls with supported identity providers and receive a valid OIDC JSON web token (JWT). You can pass this token to the AWS STS `AssumeRoleWithWebIdentity` API operation and receive IAM temporary role credentials. You can use these credentials to interact with any AWS service, like Amazon S3 and DynamoDB.

Kubernetes has long used service accounts as its own internal identity system. Pods can authenticate with the Kubernetes API server using an auto-mounted token (which was a non-OIDC JWT) that only the Kubernetes API server could validate. These legacy service account tokens do not expire, and rotating the signing key is a difficult process. In Kubernetes version 1.12, support was added for a new

ProjectedServiceAccountToken feature, which is an OIDC JSON web token that also contains the service account identity, and supports a configurable audience.

Amazon EKS now hosts a public OIDC discovery endpoint per cluster containing the signing keys for the ProjectedServiceAccountToken JSON web tokens so external systems, like IAM, can validate and accept the OIDC tokens issued by Kubernetes.

IAM Role Configuration

In IAM, you create an IAM role with a trust relationship that is scoped to your cluster's OIDC provider, the service account namespace, and (optionally) the service account name, and then attach the IAM policy that you want to associate with the service account. You can add multiple entries in the StringEquals and StringLike conditions below to use multiple service accounts or namespaces with the role.

- To scope a role to a specific service account:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::AWS_ACCOUNT_ID:oidc-provider/OIDC_PROVIDER"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "OIDC_PROVIDER:sub":
 "system:serviceaccount:SERVICE_ACCOUNT_NAMESPACE:SERVICE_ACCOUNT_NAME"
        }
      }
    }
  ]
}
```

- To scope a role to an entire namespace (to use the namespace as a boundary):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::AWS_ACCOUNT_ID:oidc-provider/OIDC_PROVIDER"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringLike": {
          "OIDC_PROVIDER:sub": "system:serviceaccount:SERVICE_ACCOUNT_NAMESPACE:*"
        }
      }
    }
  ]
}
```

Service Account Configuration

In Kubernetes, you define the IAM role to associate with a service account in your cluster by adding the eks.amazonaws.com/role-arn annotation to the service account.

```
apiVersion: v1
```

```
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::AWS_ACCOUNT_ID:role/IAM_ROLE_NAME
```

Pod Configuration

The Amazon EKS Pod Identity Webhook on the cluster watches for pods that are associated with service accounts with this annotation and applies the following environment variables to them.

```
AWS_ROLE_ARN=arn:aws:iam::AWS_ACCOUNT_ID:role/IAM_ROLE_NAME
AWS_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/token
```

> **Note**
> Your cluster does not need to use the mutating web hook to configure the environment variables and token file mounts; you can use a PodPreset to do this, or configure pods manually.

Supported versions of the AWS SDK (p. 211) look for these environment variables first in the credential chain provider. The role credentials are used for pods that meet this criteria.

> **Note**
> When a pod uses AWS credentials from an IAM role associated with a service account, the AWS CLI or other SDKs in the containers for that pod use the credentials provided by that role exclusively. They no longer inherit any IAM permissions from the worker node IAM role.

Cross-Account IAM Permissions

You can configure cross-account IAM permissions either by creating an identity provider from another account's cluster or by using chained AssumeRole operations. In the following examples, Account A owns an Amazon EKS cluster that supports IAM roles for service accounts. Pods running on that cluster need to assume IAM permissions from Account B.

Example : Create an identity provider from another account's cluster

Example

In this example, Account A would provide Account B with the OIDC issuer URL from their cluster. Account B follows the instructions in Enabling IAM Roles for Service Accounts on your Cluster (p. 212) and Creating an IAM Role and Policy for your Service Account (p. 213) 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.us-
west-2.amazonaws.com/id/EXAMPLEC061A78C479E31025A21AC4CDE191335D05820BE5CE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity"
    }
  ]
}
```

Account A attaches a policy to that role with the following permissions to assume the role that Account B created.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::222222222222:role/account-b-role"
        }
    ]
}
```

The application code for pods to assume Account B's role uses two profiles: `account_b_role` and `account_a_role`. The `account_b_role` profile uses the `account_a_role` profile as its source. For the AWS CLI, the `~/.aws/config` file would look like the following example.

```
[profile account_b_role]
source_profile = account_a_role
role_arn=arn:aws:iam::222222222222:role/account-b-role

[profile account_a_role]
web_identity_token_file = /var/run/secrets/eks.amazonaws.com/serviceaccount/token
role_arn=arn:aws:iam::111111111111:role/account-a-role
```

To specify chained profiles for other AWS SDKs, consult their documentation.

Using a Supported AWS SDK

The containers in your pods must use an AWS SDK version that supports assuming an IAM role via an OIDC web identity token file. AWS SDKs that are included in Linux distribution package managers may not be new enough to support this feature. Be sure to use at least the minimum SDK versions listed below:

- Java — 1.11.623
- Java (Version 2) — 2.7.36
- 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 — 3.3.580.0
- PHP — 3.110.7

Note that many popular Kubernetes add-ons, such the Cluster Autoscaler or the ALB Ingress Controller will not work with this feature until they have been updated to use a supported version of their respective AWS SDKs. The Amazon VPC CNI plugin for Kubernetes has been updated with a supported version of the AWS SDK for Go, and you can use the IAM roles for service accounts feature to provide the required permissions for the CNI to work.

To ensure that you are using a supported SDK, follow the installation instructions for your preferred SDK at Tools for Amazon Web Services when you build your containers.

Enabling IAM Roles for Service Accounts on your Cluster

The IAM roles for service accounts feature is available on new Amazon EKS Kubernetes version 1.14 clusters, and clusters that were updated to versions 1.14 or 1.13 on or after September 3rd, 2019. Existing clusters can update to version 1.13 or 1.14 to take advantage of this feature. For more information, see Updating an Amazon EKS Cluster Kubernetes Version (p. 33).

If your cluster supports IAM roles for service accounts, it will have an OpenID Connect issuer URL associated with it. You can view this URL in the Amazon EKS console, or you can use the following AWS CLI command to retrieve it.

> **Important**
> You must use at least version 1.16.232 of the AWS CLI to receive the proper output from this command. For more information, see Installing the AWS CLI in the *AWS Command Line Interface User Guide*.

```
aws eks describe-cluster --name cluster_name --query cluster.identity.oidc.issuer --output
 text
```

Output:

```
https://oidc.eks.region.amazonaws.com/id/EXAMPLED539D4633E53DE1B716D3041E
```

To use IAM roles for service accounts in your cluster, you must create an OIDC identity provider in the IAM console.

eksctl

To create an IAM OIDC identity provider for your cluster with `eksctl`

1. Check your `eksctl` version with the following command. This procedure assumes that you have installed `eksctl` and that your `eksctl` version is at least `0.6.0`.

   ```
   eksctl version
   ```

   For more information about installing or upgrading `eksctl`, see Installing or Upgrading `eksctl` (p. 161).
2. Create your OIDC identity provider for your cluster with the following command. Substitute the red text with your own values.

   ```
   eksctl utils associate-iam-oidc-provider --name cluster_name --approve
   ```

AWS Management Console

To create an IAM OIDC identity provider for your cluster with the AWS Management Console

1. Retrieve the OIDC issuer URL from the Amazon EKS console description of your cluster or use the following AWS CLI command.

   > **Important**
   > You must use at least version 1.16.232 of the AWS CLI to receive the proper output from this command. For more information, see Installing the AWS CLI in the *AWS Command Line Interface User Guide*.

   ```
   aws eks describe-cluster --name cluster_name --query cluster.identity.oidc.issuer
    --output text
   ```
2. Open the IAM console at https://console.aws.amazon.com/iam/.
3. In the navigation pane, choose **Identity Providers**, and then choose **Create Provider**.
4. For **Provider Type**, choose **Choose a provider type**, and then choose **OpenID Connect**.
5. For **Provider URL**, paste the OIDC issuer URL for your cluster.
6. For Audience, type `sts.amazonaws.com` and choose **Next Step**.
7. Verify that the provider information is correct, and then choose **Create** to create your identity provider.

After you have enabled the IAM OIDC identity provider for your cluster, you can create IAM roles to associate with a service account in your cluster. For more information, see Creating an IAM Role and Policy for your Service Account (p. 213)

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 a role for your 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.

To create an IAM policy for your service accounts

In this procedure, we offer two example policies that you can use for your application:

- A policy to allow read-only access to an Amazon S3 bucket. You could store configuration information or a bootstrap script in this bucket, and the containers in your pod can read the file from the bucket and load it into your application.
- A policy to allow paid container images from AWS Marketplace.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies** and then choose **Create policy**.
3. Choose the **JSON** tab.
4. In the **Policy Document** field, paste one of the following policies to apply to your service accounts, or paste your own policy document into the field. You can also use the visual editor to construct your own policy.

   The example below allows permission to the *my-pod-secrets-bucket* Amazon S3 bucket. You can modify the policy document to suit your specific needs.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::my-pod-secrets-bucket/*"
      ]
    }
  ]
}
```

   The example below gives the required permissions to use a paid container image from AWS Marketplace.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "aws-marketplace:RegisterUsage"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

5. Choose **Review policy**.
6. Enter a name and description for your policy and then choose **Create policy**.
7. Record the Amazon Resource Name (ARN) of the policy to use later when you create your role.

eksctl

To create an IAM role for your service accounts with eksctl

- Create your role with the following command. Substitute the red text 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
```

AWS Management Console

To create an IAM role for your service accounts in the console

1. Retrieve the OIDC issuer URL from the Amazon EKS console description of your cluster, or use the following AWS CLI command.

   Important
   You must use at least version 1.16.232 of the AWS CLI to receive the proper output from this command. For more information, see Installing the AWS CLI in the *AWS Command Line Interface User Guide*.

```
aws eks describe-cluster --name cluster_name --query cluster.identity.oidc.issuer
 --output text
```

2. Open the IAM console at https://console.aws.amazon.com/iam/.
3. In the navigation pane, choose **Roles**, **Create New Role**.
4. In the **Select type of trusted entity** section, choose **Web identity**.
5. In the **Choose a web identity provider** section:
   - For **Identity provider**, choose the URL for your cluster.
   - For **Audience**, type `sts.amazonaws.com`.
6. Choose **Next: Permissions**.
7. In the **Attach Policy** section, select the policy to use for your service account. In this example the policy is `AmazonEKSPodS3BucketPolicy`. Choose **Next Step**.
8. For **Role Name**, enter a name for your role. For this example, type `AmazonEKSPodS3BucketRole` to name the role, 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**.
    - Edit the OIDC provider suffix and change it from `:aud` to `:sub`.
    - Replace `sts.amazonaws.com` to your service account ID.

    The resulting line should look like this.

```
"oidc.eks.region.amazonaws.com/id/EXAMPLED539D4633E53DE1B716D3041E:sub":
 "system:serviceaccount:SERVICE_ACCOUNT_NAMESPACE:SERVICE_ACCOUNT_NAME"
```

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

AWS CLI

To create an IAM role for your service account with the AWS CLI

1. Set your AWS account ID to an environment variable with the following command.

   ```
   AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
   ```

2. Set your OIDC identity provider to an environment variable with the following command, replacing your cluster name.

   > **Important**
   > You must use at least version 1.16.232 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. Set the service account namespace to an environment variable with the following command, replacing your namespace name.

   ```
   SERVICE_ACCOUNT_NAMESPACE=kube-system
   ```

4. Set the service account name to an environment variable with the following command, replacing your service account name.

   ```
   SERVICE_ACCOUNT_NAME=aws-node
   ```

5. Copy the block of text below into a terminal and run the commands to create a file called `trust.json`.

   ```
   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": {
             "$OIDC_PROVIDER:sub": "system:serviceaccount:$SERVICE_ACCOUNT_NAMESPACE:
   $SERVICE_ACCOUNT_NAME"
           }
         }
       }
     ]
   }
   EOF
   echo "$TRUST_RELATIONSHIP" > trust.json
   ```

6. Run the following AWS CLI command to create the role, replacing your IAM role name and description.

   ```
   aws iam create-role --role-name IAM_ROLE_NAME --assume-role-policy-document file://
   trust.json --description "IAM_ROLE_DESCRIPTION"
   ```

7. Run the following command to attach your IAM policy to your role, replacing your IAM role name and policy ARN.

   ```
   aws iam attach-role-policy --role-name IAM_ROLE_NAME --policy-arn=IAM_POLICY_ARN
   ```

After you have created an IAM role, you must associate that role with a service account. For more information, see Specifying an IAM Role for your Service Account (p. 217).

Specifying an IAM Role for your Service Account

In Kubernetes, you define the IAM role to associate with a service account in your cluster by adding the following annotation to the service account.

> **Note**
> If you created an IAM role to use with your service account using eksctl, this has already been done for you with the service account you specified when creating the role.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::AWS_ACCOUNT_ID:role/IAM_ROLE_NAME
```

To patch a service account to use with IAM roles

1. Use the following command to annotate your service account with the ARN of the IAM role that you want to use with your service account. Be sure to substitute the service account namespace, name, and IAM role ARN for the service account and IAM role to use with your pods.

   ```
   kubectl annotate serviceaccount -n SERVICE_ACCOUNT_NAMESPACE SERVICE_ACCOUNT_NAME \
   eks.amazonaws.com/role-arn=arn:aws:iam::AWS_ACCOUNT_ID:role/IAM_ROLE_NAME
   ```
2. Delete and re-create any existing pods that are associated with the service account to apply the credential environment variables. The mutating web hook does not apply them to pods that are already running. The following command triggers a rollout of the aws-node DaemonSet. You can modify the namespace and deployment type to update your specific pods.

   ```
   kubectl rollout restart -n kube-system daemonset.apps/aws-node
   ```

Restricting Access to Amazon EC2 Instance Profile Credentials

By default, containers that are running on your worker nodes are not prevented from accessing the credentials that are supplied to the worker node's instance profile through the Amazon EC2 instance metadata server. This section helps you to block pod access to Amazon EC2 instance profile credentials.

To prevent containers in pods from accessing the credential information supplied to the worker node instance profile (while still allowing the permissions that are provided by the service account) by running the following iptables commands on your worker nodes (as root) or include them in your instance bootstrap user data script.

> **Important**
> These commands block ALL containers from using the instance profile credentials.

```
yum install -y iptables-services
iptables --insert FORWARD 1 --in-interface eni+ --destination 169.254.169.254/32 --jump
 DROP
```

```
iptables-save | tee /etc/sysconfig/iptables
systemctl enable --now iptables
```

Walkthrough: Amazon VPC CNI Plugin for Kubernetes

The Amazon VPC CNI plugin for Kubernetes is the networking plugin for pod networking in Amazon EKS clusters. The CNI plugin is responsible for allocating VPC IP addresses to Kubernetes nodes and configuring the necessary networking for pods on each node. The plugin requires IAM permissions, provided by the AWS managed policy `AmazonEKS_CNI_Policy`, to make calls to AWS APIs on your behalf. By default, this policy is attached to your worker node IAM role. However, using this method, all pods on the worker nodes have the same permissions as the CNI plugin. You can use the IAM roles for service accounts feature to provide the `AmazonEKS_CNI_Policy` permissions, and then remove the policy from the worker node IAM role.

For ease of use, this topic uses `eksctl` to configure IAM roles for service accounts. However, if you would rather use the AWS Management Console, the AWS CLI, or one of the AWS SDKs, the same basic concepts apply, but you will have to modify the steps to use the procedures in Enabling IAM Roles for Service Accounts on your Cluster (p. 212)

To configure the CNI plugin to use IAM roles for service accounts

1. Check your `eksctl` version with the following command. This procedure assumes that you have installed `eksctl` and that your `eksctl` version is at least `0.6.0`.

   ```
   eksctl version
   ```

   For more information about installing or upgrading `eksctl`, see Installing or Upgrading `eksctl` (p. 161).
2. Create your OIDC identity provider for your cluster with the following command. Substitute the cluster name with your own value.

   ```
   eksctl utils associate-iam-oidc-provider --name cluster_name --approve
   ```
3. Check the version of your cluster's Amazon VPC CNI Plugin for Kubernetes. Use the following command to print your cluster's CNI version.

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

   Output:

   ```
   amazon-k8s-cni:1.5.3
   ```

   If your CNI version is earlier than 1.5.4, use the following command to upgrade your CNI version to the latest version:

   - For Kubernetes 1.10 clusters:

     ```
     kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
     release-1.5/config/v1.5/aws-k8s-cni-1.10.yaml
     ```
   - For all other Kubernetes versions:

     ```
     kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
     release-1.5/config/v1.5/aws-k8s-cni.yaml
     ```

4. Create a role for your CNI plugin and annotate the `aws-node` service account with the following command. Substitute the cluster name with your own value.

```
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
```

5. Trigger a roll out of the `aws-node` daemonset to apply the credential environment variables. The mutating web hook does not apply them to pods that are already running.

```
kubectl rollout restart -n kube-system daemonset.apps/aws-node
```

6. Watch the roll out, and wait for the `DESIRED` count of the deployment matches the `UP-TO-DATE` count. Press **Ctrl + c** to exit.

```
kubectl get -n kube-system daemonset.apps/aws-node --watch
```

7. List the pods in the `aws-node` daemonset.

```
kubectl get pods -n kube-system  -l k8s-app=aws-node
```

Output:

```
NAME             READY   STATUS    RESTARTS   AGE
aws-node-9rgzw   1/1     Running   0          87m
aws-node-czjxf   1/1     Running   0          86m
aws-node-lm2r8   1/1     Running   0          86m
```

8. 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_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/token
AWS_VPC_K8S_CNI_LOGLEVEL=DEBUG
AWS_ROLE_ARN=arn:aws:iam::111122223333:role/eksctl-prod-addon-iamserviceaccount-kube-
sys-Role1-13LTY0S1XC7Q9
```

9. Remove the `AmazonEKS_CNI_Policy` policy from your worker node IAM role.

   a. Open the IAM console at https://console.aws.amazon.com/iam/.
   b. In the left navigation, choose **Roles**, and then search for your node instance role.
   c. Choose the **Permissions** tab for your node instance role and then choose the **X** to the right of the `AmazonEKS_CNI_Policy`.
   d. Choose **Detach** to finish.

Now your CNI plugin pods are getting their IAM permissions from their own role, and the instance role no longer can provide those permissions to other pods.

Troubleshooting Amazon EKS Identity and Access

To diagnose and fix common issues that you might encounter when working with Amazon EKS and IAM see Troubleshooting IAM (p. 237).

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. 46).

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. 230).

Compliance Validation for Amazon EKS

Third-party auditors assess the security and compliance of Amazon EKS as part of multiple AWS compliance programs. These include SOC, PCI, ISO, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using Amazon EKS is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in Amazon EKS

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

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

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

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure Security in Amazon EKS

As a managed service, Amazon EKS is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Amazon EKS through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

When you create an Amazon EKS cluster, you specify the Amazon VPC subnets for your cluster to use. Amazon EKS requires subnets in at least two Availability Zones. We recommend a network architecture that uses private subnets for your worker nodes and public subnets for Kubernetes to create internet-facing load balancers within.

For more information about VPC considerations, see Cluster VPC Considerations (p. 126).

If you create your VPC and worker node groups with the AWS CloudFormation templates provided in the Getting Started with Amazon EKS (p. 3) walkthrough, then your control plane and worker node security groups are configured with our recommended settings.

For more information about security group considerations, see Cluster Security Group Considerations (p. 128).

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

You can enable private access to the Kubernetes API server so that all communication between your worker nodes and the API server stays within your VPC. You can also completely disable public access to your API server so that it's not accessible from the internet.

For more information about modifying cluster endpoint access, see Modifying Cluster Endpoint Access (p. 43).

You can implement network policies with tools such as Project Calico (p. 141). Project Calico is a third party open source project. For more information, see the Project Calico documentation.

Configuration and Vulnerability Analysis in Amazon EKS

Amazon EKS platform versions represent the capabilities of the cluster control plane, including which Kubernetes API server flags are enabled and the current Kubernetes patch version. New clusters are deployed with the latest platform version. For details, see Platform Versions (p. 53).

You can update an Amazon EKS cluster (p. 33) 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. 51).

Track security or privacy events for Amazon Linux 2 at the Amazon Linux Security Center or subscribe to the associated RSS feed. Security and privacy events include an overview of the issue affected, packages, and instructions for updating your instances to correct the issue.

You can use Amazon Inspector to check for unintended network accessibility of your worker nodes and for vulnerabilities on those Amazon EC2 instances.

Pod Security Policy

The Kubernetes pod security policy admission controller validates pod creation and update requests against a set of rules. By default, Amazon EKS clusters ship with a fully permissive security policy with no restrictions. For more information, see Pod Security Policies in the Kubernetes documentation.

> **Note**
> The pod security policy admission controller is only enabled on Amazon EKS clusters running Kubernetes version 1.13 or later. You must update your cluster's Kubernetes version to at least 1.13 to use pod security policies. For more information, see Updating an Amazon EKS Cluster Kubernetes Version (p. 33).

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 restore the default pod security policy

If you 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 full `eks.privileged` YAML file contents from the preceeding example into it.
2. Apply the YAML with the following command.

```
kubectl apply -f privileged-podsecuritypolicy.yaml
```

Tagging Your Amazon EKS Resources

To help you manage your Amazon EKS clusters, you can assign your own metadata to each resource in the form of *tags*. This topic describes tags and shows you how to create them.

Contents

- Tag Basics (p. 226)
- Tagging Your Resources (p. 226)
- Tag Restrictions (p. 227)
- Working with Tags Using the Console (p. 227)
- Working with Tags Using the CLI or API (p. 228)

Tag Basics

A tag is a label that you assign to an AWS resource. Each tag consists of a *key* and an optional *value*, both of which you define.

Tags enable you to categorize your AWS resources by, for example, purpose, owner, or environment. When you have many resources of the same type, you can quickly identify a specific resource based on the tags you've assigned to it. For example, you can define a set of tags for your Amazon EKS clusters to help you track each cluster's owner and stack level. We recommend that you devise a consistent set of tag keys for each resource type. You can then search and filter the resources based on the tags that you add.

Tags are not automatically assigned to your resources. After you add a tag, you can edit tag keys and values or remove tags from a resource at any time. If you delete a resource, any tags for the resource are also deleted.

Tags don't have any semantic meaning to Amazon EKS and are interpreted strictly as a string of characters. You can set the value of a tag to an empty string, but you can't set the value of a tag to null. If you add a tag that has the same key as an existing tag on that resource, the new value overwrites the old value.

You can work with tags using the AWS Management Console, the AWS CLI, and the Amazon EKS API.

> **Note**
> Amazon EKS tags are not currently supported by `eksctl`.

If you're using AWS Identity and Access Management (IAM), you can control which users in your AWS account have permission to create, edit, or delete tags.

Tagging Your Resources

You can tag new or existing Amazon EKS clusters.

If you're using the Amazon EKS console, you can apply tags to new resources when they are created or to existing resources at any time using the **Tags** tab on the relevant resource page.

If you're using the Amazon EKS API, the AWS CLI, or an AWS SDK, you can apply tags to new resources using the `tags` parameter on the relevant API action or to existing resources using the `TagResource` API action. For more information, see TagResource.

Some resource-creating actions enable you to specify tags for a resource when the resource is created. If tags cannot be applied during resource creation, the resource creation process fails. This ensures that resources you intended to tag on creation are either created with specified tags or not created at all. If you tag resources at the time of creation, you don't need to run custom tagging scripts after resource creation.

The following table describes the Amazon EKS resources that can be tagged, and the resources that can be tagged on creation.

Tagging Support for Amazon EKS Resources

Resource	Supports tags	Supports tag propagation	Supports tagging on creation (Amazon EKS API, AWS CLI, AWS SDK)
Amazon EKS clusters	Yes	No. Cluster tags do not propagate to any other resources associated with the cluster.	Yes

Tag Restrictions

The following basic restrictions apply to tags:

- Maximum number of tags per resource – 50
- For each resource, each tag key must be unique, and each tag key can have only one value.
- Maximum key length – 128 Unicode characters in UTF-8
- Maximum value length – 256 Unicode characters in UTF-8
- If your tagging schema is used across multiple AWS services and resources, remember that other services may have restrictions on allowed characters. Generally allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: + - = . _ : / @.
- Tag keys and values are case sensitive.
- Don't use `aws:`, `AWS:`, or any upper or lowercase combination of such as a prefix for either keys or values, as it is reserved for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix do not count against your tags-per-resource limit.

Working with Tags Using the Console

Using the Amazon EKS console, you can manage the tags associated with new or existing clusters.

When you select a resource-specific page in the Amazon EKS console, it displays a list of those resources. For example, if you select **Clusters** from the navigation pane, the console displays a list of Amazon EKS clusters. When you select a resource from one of these lists (for example, a specific cluster), if the resource supports tags, you can view and manage its tags on the **Tags** tab.

Adding Tags on an Individual Resource On Creation

You can add tags to Amazon EKS clusters when you create them. For more information, see Creating an Amazon EKS Cluster (p. 24)

Adding and Deleting Tags on an Individual Resource

Amazon EKS allows you to add or delete tags associated with your clusters directly from the resource's page.

To add or delete a tag on an individual resource

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. From the navigation bar, select the region to use.
3. In the navigation pane, choose **Clusters**.
4. Choose a specific cluster, then scroll down and choose **Manage tags**.
5. On the **Update tags** page, add or delete your tags as necessary.
   - To add a tag — choose **Add tag** and then specify the key and value for each tag.
   - To delete a tag — choose **Remove tag**.
6. Repeat this process for each tag you want to add or delete, and then choose **Update** to finish.

Working with Tags Using the CLI or API

Use the following AWS CLI commands or Amazon EKS API operations to add, update, list, and delete the tags for your resources.

Tagging Support for Amazon EKS Resources

Task	AWS CLI	API Action
Add or overwrite one or more tags.	tag-resource	TagResource
Delete one or more tags.	untag-resource	UntagResource

The following examples show how to tag or untag resources using the AWS CLI.

Example 1: Tag an existing cluster

The following command tags an existing cluster.

```
aws eks tag-resource --resource-arn resource_ARN --tags team=devs
```

Example 2: Untag an existing cluster

The following command deletes a tag from an existing cluster.

```
aws eks untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

Example 3: List tags for a resource

The following command lists the tags associated with an existing resource.

```
aws eks list-tags-for-resource --resource-arn resource_ARN
```

Some resource-creating actions enable you to specify tags when you create the resource. The following actions support tagging on creation.

Task	AWS CLI	AWS Tools for Windows PowerShell	API Action
Create a cluster	create-cluster	New-EKSCluster	CreateCluster

Logging Amazon EKS API Calls with AWS CloudTrail

Amazon EKS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon EKS. CloudTrail captures all API calls for Amazon EKS as events. The calls captured include calls from the Amazon EKS console and code calls to the Amazon EKS API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon EKS. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon EKS, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Amazon EKS Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon EKS, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon EKS, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All Amazon EKS actions are logged by CloudTrail and are documented in the Amazon EKS API Reference. For example, calls to the `CreateCluster`, `ListClusters` and `DeleteCluster` sections generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understanding Amazon EKS Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateCluster` action.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws:iam::111122223333:user/ericn",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "ericn"
  },
  "eventTime": "2018-05-28T19:16:43Z",
  "eventSource": "eks.amazonaws.com",
  "eventName": "CreateCluster",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "205.251.233.178",
  "userAgent": "PostmanRuntime/6.4.0",
  "requestParameters": {
    "resourcesVpcConfig": {
      "subnetIds": [
        "subnet-a670c2df",
        "subnet-4f8c5004"
      ]
    },
    "roleArn": "arn:aws:iam::111122223333:role/AWSServiceRoleForAmazonEKS-CAC1G1VH3ZKZ",
    "clusterName": "test"
  },
  "responseElements": {
    "cluster": {
      "clusterName": "test",
      "status": "CREATING",
      "createdAt": 1527535003.208,
      "certificateAuthority": {},
      "arn": "arn:aws:eks:us-west-2:111122223333:cluster/test",
      "roleArn": "arn:aws:iam::111122223333:role/AWSServiceRoleForAmazonEKS-CAC1G1VH3ZKZ",
      "version": "1.10",
      "resourcesVpcConfig": {
        "securityGroupIds": [],
        "vpcId": "vpc-21277358",
        "subnetIds": [
          "subnet-a670c2df",
          "subnet-4f8c5004"
        ]
      }
    }
  },
  "requestID": "a7a0735d-62ab-11e8-9f79-81ce5b2b7d37",
  "eventID": "eab22523-174a-499c-9dd6-91e7be3ff8e3",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

Related Projects

These open source projects extend the functionality of Kubernetes clusters running on AWS, including clusters managed by Amazon EKS.

Management Tools

Related management tools for Amazon EKS and Kubernetes clusters.

eksctl

`eksctl` is a simple CLI tool for creating clusters on Amazon EKS.

- Project URL: https://eksctl.io/
- Project documentation: https://eksctl.io/
- AWS open source blog: eksctl: Amazon EKS Cluster with One Command

AWS Service Operator

AWS Service Operator allows you to create AWS resources using `kubectl`.

- Project URL: https://github.com/awslabs/aws-service-operator
- Project documentation: https://github.com/awslabs/aws-service-operator/blob/master/readme.adoc
- AWS open source blog: AWS Service Operator for Kubernetes Now Available

Networking

Related networking projects for Amazon EKS and Kubernetes clusters.

Amazon VPC CNI plugin for Kubernetes

Amazon EKS supports native VPC networking via the Amazon VPC CNI plugin for Kubernetes. Using this CNI plugin allows Kubernetes pods to have the same IP address inside the pod as they do on the VPC network. For more information, see Pod Networking (CNI) (p. 130) and CNI Configuration Variables (p. 131).

- Project URL: https://github.com/aws/amazon-vpc-cni-k8s
- Project documentation: https://github.com/aws/amazon-vpc-cni-k8s/blob/master/README.md

AWS Application Load Balancer (ALB) Ingress Controller for Kubernetes

The AWS ALB Ingress Controller satisfies Kubernetes ingress resources by provisioning Application Load Balancers.

- Project URL: https://github.com/kubernetes-sigs/aws-alb-ingress-controller
- Project documentation: https://github.com/kubernetes-sigs/aws-alb-ingress-controller/tree/master/docs
- AWS open source blog: Kubernetes Ingress with AWS ALB Ingress Controller

ExternalDNS

ExternalDNS synchronizes exposed Kubernetes services and ingresses with DNS providers including Amazon Route 53 and AWS Service Discovery.

- Project URL: https://github.com/kubernetes-incubator/external-dns
- Project documentation: https://github.com/kubernetes-incubator/external-dns/blob/master/docs/tutorials/aws.md

Security

Related security projects for Amazon EKS and Kubernetes clusters.

AWS IAM Authenticator

A tool to use AWS IAM credentials to authenticate to a Kubernetes cluster. For more information, see Installing `aws-iam-authenticator` (p. 151).

- 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/monitoring/

Continuous Integration / Continuous Deployment

Related CI/CD projects for Amazon EKS and Kubernetes clusters.

Jenkins X

CI/CD solution for modern cloud applications on Amazon EKS and Kubernetes clusters.

- Project URL: https://jenkins-x.io/
- Project documentation: https://jenkins-x.io/documentation/
- AWS open source blog: Continuous Delivery with Amazon EKS and Jenkins X

Amazon EKS Troubleshooting

This chapter covers some common errors that you may see while using Amazon EKS and how to work around them.

Insufficient Capacity

If you receive the following error while attempting to create an Amazon EKS cluster, then one of the Availability Zones you specified does not have sufficient capacity to support a cluster.

```
Cannot create cluster 'example-cluster' because us-east-1d, the targeted
availability zone, does not currently have sufficient capacity to support
the cluster. Retry and choose from these availability zones: us-east-1a, us-
east-1b, us-east-1c
```

Retry creating your cluster with subnets in your cluster VPC that are hosted in the Availability Zones returned by this error message.

`aws-iam-authenticator` Not Found

If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, then your **kubectl** is not configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 151).

Worker Nodes Fail to Join Cluster

There are two common reasons that prevent worker nodes from joining the cluster:

- The `aws-auth-cm.yaml` file does not have the correct IAM role ARN for your worker nodes. Ensure that the worker node IAM role ARN (not the instance profile ARN) is specified in your `aws-auth-cm.yaml` file. For more information, see Launching Amazon EKS Linux Worker Nodes (p. 76).
- The **ClusterName** in your worker node AWS CloudFormation template does not exactly match the name of the cluster you want your worker nodes to join. Passing an incorrect value to this field results in an incorrect configuration of the worker node's `/var/lib/kubelet/kubeconfig` file, and the nodes will not join the cluster.

Unauthorized or Access Denied (`kubectl`)

If you receive one of the following errors while running **kubectl** commands, then your **kubectl** is not configured properly for Amazon EKS or the IAM user or role credentials that you are using do not map to a Kubernetes RBAC user with sufficient permissions in your Amazon EKS cluster.

- `could not get token: AccessDenied: Access denied`
- `error: You must be logged in to the server (Unauthorized)`
- `error: the server doesn't have a resource type "svc"`

This could be because the cluster was created with one set of AWS credentials (from an IAM user or role), and **kubectl** is using a different set of credentials.

When an Amazon EKS cluster is created, the IAM entity (user or role) that creates the cluster is added to the Kubernetes RBAC authorization table as the administrator (with `system:master` permissions). Initially, only that IAM user can make calls to the Kubernetes API server using **kubectl**. For more information, see Managing Users or IAM Roles for your Cluster (p. 157). Also, the AWS IAM Authenticator for Kubernetes uses the AWS SDK for Go to authenticate against your Amazon EKS cluster. If you use the console to create the cluster, you must ensure that the same IAM user credentials are in the AWS SDK credential chain when you are running **kubectl** commands on your cluster.

If you install and configure the AWS CLI, you can configure the IAM credentials for your user. If the AWS CLI is configured properly for your user, then the AWS IAM Authenticator for Kubernetes can find those credentials as well. For more information, see Configuring the AWS CLI in the *AWS Command Line Interface User Guide*.

If you assumed a role to create the Amazon EKS cluster, you must ensure that **kubectl** is configured to assume the same role. Use the following command to update your kubeconfig file to use an IAM role. For more information, see Create a `kubeconfig` for Amazon EKS (p. 154).

```
aws --region region 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. 157).

hostname doesn't match

Your system's Python version must be 2.7.9 or greater. 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
```

CNI Log Collection Tool

The Amazon VPC CNI plugin for Kubernetes has its own troubleshooting script (which is available on worker nodes at `/opt/cni/bin/aws-cni-support.sh`) that you can use to collect diagnostic logs for support cases and general troubleshooting.

The script collects the following diagnostic information:

- L-IPAMD introspection data
- Metrics
- Kubelet introspection data

- `ifconfig` output
- `ip rule show` output
- `iptables-save` output
- `iptables -nvL` output
- `iptables -nvL -t nat` output
- A dump of the CNI configuration
- Kubelet logs
- Stored `/var/log/messages`
- Worker node's route table information (via `ip route`)
- The `sysctls` output of `/proc/sys/net/ipv4/conf/{all,default,eth0}/rp_filter`

Use the following command to run the script on your worker node:

```
sudo bash /opt/cni/bin/aws-cni-support.sh
```

> **Note**
> If the script is not present at that location, then the CNI container failed to run. You can manually download and run the script with the following command:
>
> ```
> curl https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/master/scripts/aws-
> cni-support.sh | sudo bash
> ```

The diagnostic information is collected and stored at `/var/log/aws-routed-eni/aws-cni-support.tar.gz`.

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:us-west-2:111122223333: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. 201).

For more general information about IAM, see Controlling Access Using Policies in the *IAM User Guide*.

I Am Not Authorized to Perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Amazon EKS.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon EKS. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I Want to View My Access Keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing Access Keys in the *IAM User Guide*.

I'm an Administrator and Want to Allow Others to Access Amazon EKS

To allow others to access Amazon EKS, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amazon EKS.

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. 198).

- To learn how to provide access to your resources across AWS accounts that you own, see Providing Access to an IAM User in Another AWS Account That You Own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing Access to AWS Accounts Owned by Third Parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing Access to Externally Authenticated Users (Identity Federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.

Amazon EKS Service Limits

The following table provides the default limits for Amazon EKS for an AWS account that can be changed. For more information, see AWS Service Limits in the *Amazon Web Services General Reference*.

Resource	Default Limit
Maximum number of Amazon EKS clusters per region, per account	50

The following table provides limitations for Amazon EKS that cannot be changed.

Resource	Default Limit
Maximum number of control plane security groups per cluster (these are specified when you create the cluster)	5

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
Windows Support	Amazon EKS clusters running Kubernetes version 1.14 now support Windows workloads.	October 7, 2019
Autoscaling	Added a chapter to cover some of the different types of Kubernetes autoscaling that are supported on Amazon EKS clusters.	September 30, 2019
Kubernetes Dashboard Update	Updated topic for installing the Kubernetes dashboard on Amazon EKS clusters to use the beta 2.0 version.	September 28, 2019
Amazon EFS CSI Driver	Added topic for installing the Amazon EFS CSI Driver on Kubernetes 1.14 Amazon EKS clusters.	September 19, 2019
Amazon EC2 Systems Manager parameter for Amazon EKS-optimized AMI ID	Added topic for retrieving the Amazon EKS-optimized AMI ID using an Amazon EC2 Systems Manager parameter. The parameter eliminates the need for you to look up AMI IDs.	September 18, 2019
Amazon EKS resource tagging	Manage tagging of your Amazon EKS clusters.	September 16, 2019
Amazon EBS CSI Driver	Added topic for installing the Amazon EBS CSI Driver on Kubernetes 1.14 Amazon EKS clusters.	September 9, 2019
New Amazon EKS-optimized AMI patched for CVE-2019-9512 and CVE-2019-9514	Amazon EKS has updated the Amazon EKS-optimized AMI to address CVE-2019-9512 and CVE-2019-9514.	September 6, 2019
Announcing deprecation of Kubernetes 1.11 in Amazon EKS	Amazon EKS will deprecate Kubernetes version 1.11 on November 4, 2019. On this day, you will no longer be able to create new 1.11 clusters and all Amazon EKS clusters running Kubernetes version 1.11 will be updated to the latest available platform version of Kubernetes version 1.12.	September 4, 2019

Kubernetes Version 1.14	Added Kubernetes version 1.14 support for new clusters and version upgrades.	September 3, 2019
IAM Roles for Service Accounts	With IAM roles for service accounts on Amazon EKS clusters, you can associate an IAM role with a Kubernetes service account. With this feature, you no longer need to provide extended permissions to the worker node IAM role so that pods on that node can call AWS APIs.	September 3, 2019
Amazon EKS region expansion (p. 241)	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. 241)	Amazon EKS is now available in the Asia Pacific (Hong Kong) (`ap-east-1`) region.	July 31, 2019
Kubernetes 1.10 deprecated on Amazon EKS	Kubernetes version 1.10 is no longer supported on Amazon EKS. Please update any 1.10 clusters to version 1.11 or higher in order to avoid service interruption.	July 30, 2019
Added topic on ALB Ingress Controller	The AWS ALB Ingress Controller for Kubernetes is a controller that triggers the creation of an Application Load Balancer when Ingress resources are created.	July 11, 2019
New Amazon EKS-optimized AMI	Removing unnecessary `kubectl` binary from AMIs.	July 3, 2019
Kubernetes Version 1.13	Added Kubernetes version 1.13 support for new clusters and version upgrades.	June 18, 2019
New Amazon EKS-optimized AMI patched for AWS-2019-005	Amazon EKS has updated the Amazon EKS-optimized AMI to address the vulnerabilities described in AWS-2019-005.	June 17, 2019

Announcing deprecation of Kubernetes 1.10 in Amazon EKS	Amazon EKS will deprecate Kubernetes version 1.10 on July 22, 2019. On this day, you will no longer be able to create new 1.10 clusters and all Amazon EKS clusters running Kubernetes version 1.10 will be updated to the latest available platform version of Kubernetes version 1.11.	May 21, 2019
Amazon EKS platform version update	New platform version for Kubernetes 1.11 and 1.10 clusters to support custom DNS names in the Kubelet certificate and improve `etcd` performance.	May 21, 2019
Getting Started with eksctl	This getting started guide helps you to install all of the required resources to get started with Amazon EKS using `eksctl`, a simple command line utility for creating and managing Kubernetes clusters on Amazon EKS.	May 10, 2019
AWS CLI get-token command (p. 241)	The **aws eks get-token** command was added to the AWS CLI so that you no longer need to install the AWS IAM Authenticator for Kubernetes to create client security tokens for cluster API server communication. Upgrade your AWS CLI installation to the latest version to take advantage of this new functionality. For more information, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.	May 10, 2019
Amazon EKS platform version update	New platform version for Kubernetes 1.12 clusters to support custom DNS names in the Kubelet certificate and improve `etcd` performance. This fixes a bug that caused worker node Kubelet daemons to request a new certificate every few seconds.	May 8, 2019
Prometheus tutorial	Added topic for deploying Prometheus to your Amazon EKS cluster.	April 5, 2019

Amazon EKS Control Plane Logging	Amazon EKS control plane logging makes it easy for you to secure and run your clusters by providing audit and diagnostic logs directly from the Amazon EKS control plane to CloudWatch Logs in your account.	April 4, 2019
Kubernetes Version 1.12 (p. 241)	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. 241)	Amazon EKS is now available in the EU (London) (`eu-west-2`), EU (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. 241)	Amazon EKS is now available in the Asia Pacific (Seoul) (`ap-northeast-2`) region.	January 9, 2019
Amazon EKS region expansion (p. 241)	Amazon EKS is now available in the following additional regions: EU (Frankfurt) (`eu-central-1`), Asia Pacific (Tokyo) (`ap-northeast-1`), Asia Pacific (Singapore) (`ap-southeast-1`), and Asia Pacific (Sydney) (`ap-southeast-2`).	December 19, 2018
Amazon EKS cluster updates	Added documentation for Amazon EKS cluster Kubernetes version updates and worker node replacement.	December 12, 2018
Amazon EKS region expansion (p. 241)	Amazon EKS is now available in the EU (Stockholm) (`eu-north-1`) region.	December 11, 2018
Amazon EKS platform version update	New platform version updating Kubernetes to patch level 1.10.11 to address CVE-2018-1002105.	December 4, 2018
Added version 1.0.0 support for the Application Load Balancer ingress controller	The Application Load Balancer ingress controller releases version 1.0.0 with formal support from AWS.	November 20, 2018
Added support for CNI network configuration	The Amazon VPC CNI plugin for Kubernetes version 1.2.1 now supports custom network configuration for secondary pod network interfaces.	October 16, 2018
Added support for MutatingAdmissionWebhook and ValidatingAdmissionWebhook	Amazon EKS platform version `1.10-eks.2` now supports `MutatingAdmissionWebhook` and `ValidatingAdmissionWebhook` admission controllers.	October 10, 2018
Added Partner AMI information	Canonical has partnered with Amazon EKS to create worker node AMIs that you can use in your clusters.	October 3, 2018

Added instructions for AWS CLI update-kubeconfig command	Amazon EKS has added the `update-kubeconfig` to the AWS CLI to simplify the process of creating a `kubeconfig` file for accessing your cluster.	September 21, 2018
New Amazon EKS-optimized AMIs	Amazon EKS has updated the Amazon EKS-optimized AMIs (with and without GPU support) to provide various security fixes and AMI optimizations.	September 13, 2018
Amazon EKS region expansion (p. 241)	Amazon EKS is now available in the EU (Ireland) (`eu-west-1`) region.	September 5, 2018
Amazon EKS platform version update	New platform version with support for Kubernetes aggregation layer and the Horizontal Pod Autoscaler(HPA).	August 31, 2018
New Amazon EKS-optimized AMIs and GPU support	Amazon EKS has updated the Amazon EKS-optimized AMI to use a new AWS CloudFormation worker node template and bootstrap script. In addition, a new Amazon EKS-optimized AMI with GPU support is available.	August 22, 2018
New Amazon EKS-optimized AMI patched for ALAS2-2018-1058	Amazon EKS has updated the Amazon EKS-optimized AMI to address the CVEs referenced in ALAS2-2018-1058.	August 14, 2018
Amazon EKS-optimized AMI build scripts	Amazon EKS has open-sourced the build scripts that are used to build the Amazon EKS-optimized AMI. These build scripts are now available on GitHub.	July 10, 2018
Amazon EKS initial release (p. 241)	Initial documentation for service launch	June 5, 2018

AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the *AWS General Reference*.