Amazon EKS

User Guide

Amazon EKS: User Guide

Table of Contents

What Is Amazon EKS?

Amazon Elastic Container Service for Kubernetes (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.

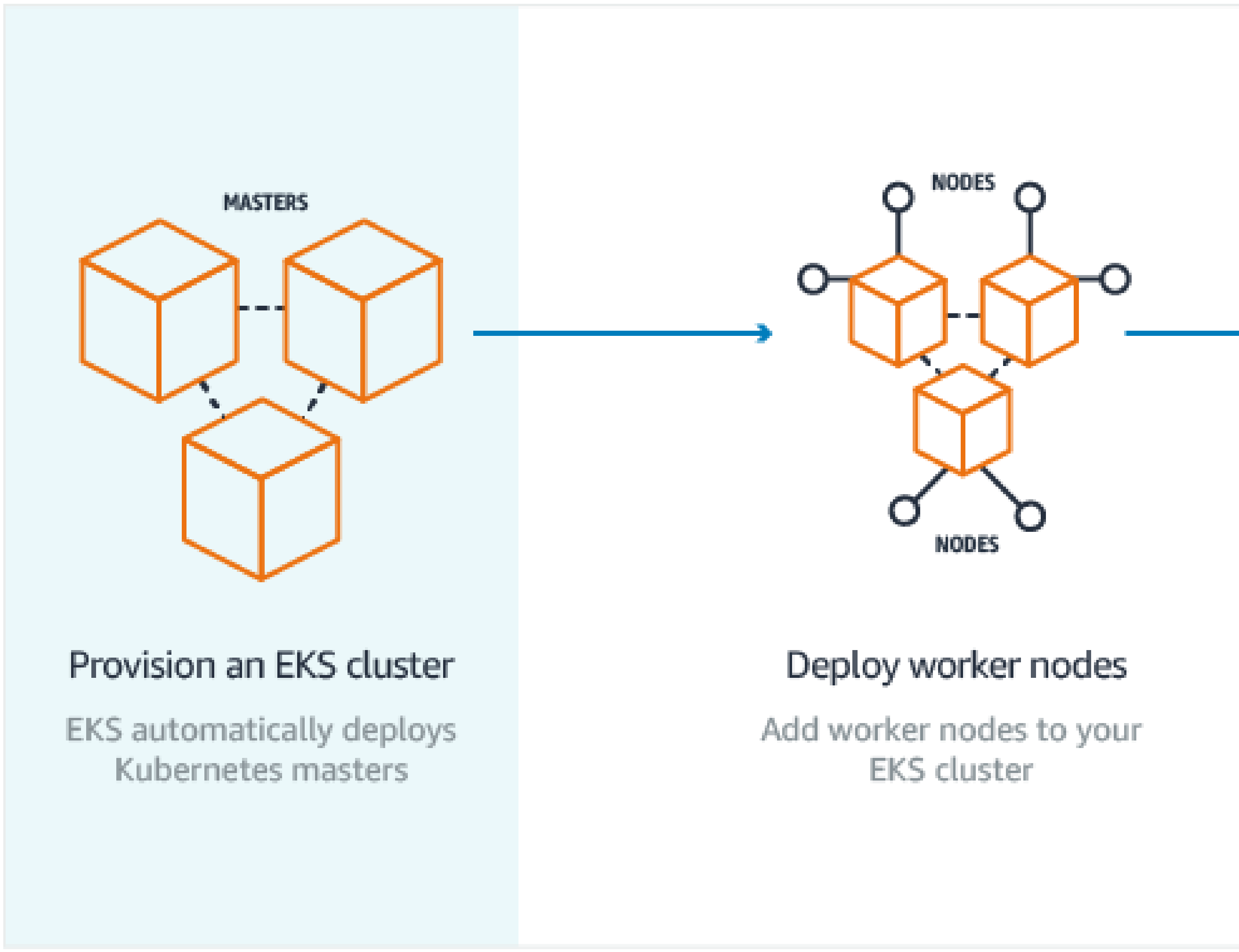
Amazon EKS runs Kubernetes control plane instances across multiple Availability Zones to ensure high availability. Amazon EKS automatically detects and replaces unhealthy control plane instances, and it provides automated version upgrades and patching for them.

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

- 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.

How Does Amazon EKS Work?


MASTERS
NODES
NODES
Provision an EKS cluster
EKS automatically deploys Kubernetes masters
Deploy worker nodes
Add worker nodes 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

This getting started guide helps you to create all of the required resources to get started with Amazon EKS.

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: Review**.
5. For **Role name**, enter a unique name for your role, such as `eksServiceRole`, then choose **Create role**.

Create your Amazon EKS Cluster VPC

To create your cluster VPC

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.

   Note
   Amazon EKS is available in the following Regions at this time:

   - **US West (Oregon)** (`us-west-2`)
   - **US East (N. Virginia)** (`us-east-1`)
   - **US East (Ohio)** (`us-east-2`)
   - **EU (Frankfurt)** (`eu-central-1`)
   - **EU (Stockholm)** (`eu-north-1`)
   - **EU (Ireland)** (`eu-west-1`)
   - **Asia Pacific (Tokyo)** (`ap-northeast-1`)
   - **Asia Pacific (Singapore)** (`ap-southeast-1`)
   - **Asia Pacific (Sydney)** (`ap-southeast-2`)
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/2018-12-10/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 may leave the default value.
   - **Subnet01Block**: Choose a CIDR range for subnet 1. You may leave the default value.
   - **Subnet02Block**: Choose a CIDR range for subnet 2. You may leave the default value.
   - **Subnet03Block**: Choose a CIDR range for subnet 3. You may leave the default value.
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.

Install and Configure **kubectl** for Amazon EKS

Kubernetes uses a command-line utility called `kubectl` for communicating with the cluster API server. Amazon EKS clusters also require the AWS IAM Authenticator for Kubernetes to allow IAM authentication for your Kubernetes cluster. Beginning with Kubernetes version 1.10, you can configure the **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.

Amazon EKS vends **aws-iam-authenticator** binaries that you can use that are identical to the upstream **aws-iam-authenticator** binaries with the same version. Alternatively, you can use **go get** to fetch the binary from the AWS IAM Authenticator for Kubernetes project on GitHub.

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. 60).

To install `aws-iam-authenticator` for Amazon EKS

1. Download and install the `aws-iam-authenticator` binary.

Amazon EKS vends `aws-iam-authenticator` binaries that you can use, or you can use **go get** to fetch the binary from the AWS IAM Authenticator for Kubernetes project on GitHub for other operating systems.

- To download and install the Amazon EKS-vended `aws-iam-authenticator` binary:

  a. Download the Amazon EKS-vended `aws-iam-authenticator` binary from Amazon S3:

  - **Linux**: https://amazon-eks.s3-us-west-2.amazonaws.com/1.11.5/2018-12-06/bin/linux/amd64/aws-iam-authenticator
  - **MacOS**: https://amazon-eks.s3-us-west-2.amazonaws.com/1.11.5/2018-12-06/bin/darwin/amd64/aws-iam-authenticator
  - **Windows**: https://amazon-eks.s3-us-west-2.amazonaws.com/1.11.5/2018-12-06/bin/windows/amd64/aws-iam-authenticator.exe

  Use the command below to download the binary, substituting the correct URL for your platform. The example below is for macOS clients.

```
curl -o aws-iam-authenticator https://amazon-eks.s3-us-
west-2.amazonaws.com/1.11.5/2018-12-06/bin/darwin/amd64/aws-iam-authenticator
```

  b. (Optional) Verify the downloaded binary with the SHA-256 sum provided in the same bucket prefix, substituting the correct URL for your platform.

  i. Download the SHA-256 sum for your system. The example below is to download the SHA-256 sum for macOS clients.

```
curl -o aws-iam-authenticator.sha256 https://amazon-eks.s3-us-
west-2.amazonaws.com/1.11.5/2018-12-06/bin/darwin/amd64/aws-iam-
authenticator.sha256
```

  ii. Check the SHA-256 sum for your downloaded binary. The example `openssl` command below was tested for macOS and Ubuntu clients. Your operating system may use a different command or syntax to check SHA-256 sums. Consult your operating system documentation if necessary.

```
openssl sha -sha256 aws-iam-authenticator
```

  iii. Compare the generated SHA-256 sum in the command output against your downloaded `aws-iam-authenticator.sha256` file. The two should match.

  c. Apply execute permissions to the binary.

```
chmod +x ./aws-iam-authenticator
```

  d. 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`.

```
cp ./aws-iam-authenticator $HOME/bin/aws-iam-authenticator && export PATH=
$HOME/bin:$PATH
```

  e. Add `$HOME/bin` to your `PATH` environment variable.

  - For Bash shells on macOS:

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bash_profile
```

- For Bash shells on Linux:

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
```

f. Test that the `aws-iam-authenticator` binary works.

```
aws-iam-authenticator help
```

- Or, to install the `aws-iam-authenticator` binary from GitHub using **go get**:

  a. Install the Go programming language for your operating system if you do not already have **go** installed. For more information, see Install the Go tools in the Go documentation.

  b. Use **go get** to install the `aws-iam-authenticator` binary.

```
go get -u -v github.com/kubernetes-sigs/aws-iam-authenticator/cmd/aws-iam-
authenticator
```

Note
If you receive the following error, you must upgrade your Go language to 1.7 or greater. For more information, see Install the Go tools in the Go documentation.

```
package context: unrecognized import path "context" (import path does
 not begin with hostname)
```

  c. Add `$HOME/go/bin` to your `PATH` environment variable.

- For Bash shells on macOS:

```
export PATH=$HOME/go/bin:$PATH && echo 'export PATH=$HOME/go/bin:$PATH' >>
 ~/.bash_profile
```

- For Bash shells on Linux:

```
export PATH=$HOME/go/bin:$PATH && echo 'export PATH=$HOME/go/bin:$PATH' >>
 ~/.bashrc
```

  d. Test that the `aws-iam-authenticator` binary works.

```
aws-iam-authenticator help
```

2. 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. 65). Otherwise, see Creating an Amazon EKS Cluster (p. 17) to create a new Amazon EKS cluster.

(Optional) Download and Install the Latest AWS CLI

While the AWS CLI is not explicitly required to use Amazon EKS, the **update-kubeconfig** command greatly simplifies the kubeconfig creation process. To use the AWS CLI with Amazon EKS, you must have at least version 1.16.73 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*.

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 Python 3, or Python 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.

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. 68). 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. These also work for the AWS IAM Authenticator for Kubernetes. 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*.

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 Creating Amazon EKS IAM Policies (p. 74).
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. 3).
   - **VPC**: The VPC you created with Create your Amazon EKS Cluster VPC (p. 3). 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. 3). By default, the available subnets in the above VPC are preselected.

- **Security Groups**: The **SecurityGroups** value from the AWS CloudFormation output that you generated with Create your Amazon EKS Cluster VPC (p. 3). 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 we recommend that you use a dedicated security group for your cluster control plane. If you share it with other resources, you may block or disrupt connections to those resources.

  Note
  You may receive an error that one of the Availability Zones in your request does not 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.

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.

To create your cluster with the AWS CLI

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. 3), and the subnet and security group IDs for the VPC that you created in Create your Amazon EKS Cluster VPC (p. 3).

```
aws eks create-cluster --name devel --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 may 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. Please update your AWS CLI version to the latest available and be sure to delete the custom service model directory at `~/.aws/models/eks`.

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

   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 Creating Amazon EKS IAM Policies (p. 74).

   Output:

```
{
    "cluster": {
        "name": "devel",
        "arn": "arn:aws:eks:us-west-2:111122223333:cluster/devel",
        "createdAt": 1527785885.159,
        "version": "1.10",
        "roleArn": "arn:aws:iam::111122223333:role/eks-service-role-
AWSServiceRoleForAmazonEKS-AFNL4H8HB71F",
        "resourcesVpcConfig": {
            "subnetIds": [
                "subnet-a9189fe2",
                "subnet-50432629"
            ],
            "securityGroupIds": [
                "sg-f5c54184"
            ],
            "vpcId": "vpc-a54041dc"
        },
        "status": "CREATING",
        "certificateAuthority": {}
    }
}
```

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

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

Step 2: Configure `kubectl` for Amazon EKS

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. 65).

To create your `kubeconfig` file with the AWS CLI

1. Ensure that you have at least version 1.16.73 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 Python 3, or Python 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 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`, then your **kubectl** is not configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 63).

   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 On-Demand 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. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
3. From the navigation bar, select a Region that supports Amazon EKS.

   Note
   Amazon EKS is available in the following Regions at this time:

   - **US West (Oregon)** (`us-west-2`)
   - **US East (N. Virginia)** (`us-east-1`)
   - **US East (Ohio)** (`us-east-2`)
   - **EU (Frankfurt)** (`eu-central-1`)
   - **EU (Stockholm)** (`eu-north-1`)
   - **EU (Ireland)** (`eu-west-1`)
   - **Asia Pacific (Tokyo)** (`ap-northeast-1`)
   - **Asia Pacific (Singapore)** (`ap-southeast-1`)
   - **Asia Pacific (Sydney)** (`ap-southeast-2`)
4. Choose **Create stack**.
5. For **Choose a template**, select **Specify an Amazon S3 template URL**.
6. Paste the following URL into the text area and choose **Next**:

```
https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2018-12-10/amazon-eks-nodegroup.yaml
```

7. On the **Specify Details** page, fill out the following parameters accordingly, and choose **Next**.

- **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. 7); 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. 3).
- **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.
- **NodeImageId**: Enter the current Amazon EKS worker node AMI ID for your Region. The AMI IDs for the latest Amazon EKS-optimized AMI (with and without GPU support (p. 27)) are shown in the following table.

  Note
  The Amazon EKS-optimized AMI with GPU support only supports P2 and P3 instance types. Be sure to specify these instance types in your worker node AWS CloudFormation template. Because this AMI includes third-party software that requires an end user license agreement (EULA), you must subscribe to the AMI in the AWS Marketplace and accept the EULA before you can use the AMI in your worker node groups. To subscribe to the AMI, visit the AWS Marketplace.

Kubernetes version 1.11

Region	Amazon EKS-optimized AMI	with GPU support
US West (Oregon) (`us-west-2`)	`ami-094fa4044a2a3cf52`	`ami-014f4e495a19d3e4f`
US East (N. Virginia) (`us-east-1`)	`ami-0b4eb1d8782fc3aea`	`ami-08a0bb74d1c9a5e2f`
US East (Ohio) (`us-east-2`)	`ami-053cbe66e0033ebcf`	`ami-04a758678ae5ebad5`
EU (Frankfurt) (`eu-central-1`)	`ami-0ce0ec06e682ee10e`	`ami-017912381e1ebb308`
EU (Stockholm) (`eu-north-1`)	`ami-082e6cf1c07e60241`	`ami-69b03e17`
EU (Ireland) (`eu-west-1`)	`ami-0a9006fb385703b54`	`ami-050db3f5f9dbd4439`
Asia Pacific (Tokyo) (`ap-northeast-1`)	`ami-063650732b3e8b38c`	`ami-080be783089a635dd`
Asia Pacific (Singapore) (`ap-southeast-1`)	`ami-0549ac6995b998478`	`ami-05bbe4b57e4030910`
Asia Pacific (Sydney) (`ap-southeast-2`)	`ami-03297c04f71690a76`	`ami-0da8916a67c116ace`

Kubernetes version 1.10

Region	Amazon EKS-optimized AMI	with GPU support
US West (Oregon) (`us-west-2`)	`ami-07af9511082779ae7`	`ami-08754f7ac73185331`
US East (N. Virginia) (`us-east-1`)	`ami-027792c3cc6de7b5b`	`ami-03c499c67bc65c089`
US East (Ohio) (`us-east-2`)	`ami-036130f4127a367f7`	`ami-081210a2fd7f3c487`
EU (Frankfurt) (`eu-central-1`)	`ami-06d069282a5fea248`	`ami-03b492cd6806000ff`
EU (Stockholm) (`eu-north-1`)	`ami-04b0f84e5a05e0b30`	`ami-24b43a5a`
EU (Ireland) (`eu-west-1`)	`ami-03612357ac9da2c7d`	`ami-047637529a86c7237`
Asia Pacific (Tokyo) (`ap-northeast-1`)	`ami-06f4af3742fca5998`	`ami-02a0802829f472c55`
Asia Pacific (Singapore) (`ap-southeast-1`)	`ami-0bc97856f0dd86d41`	`ami-0092b6c977d1937f0`
Asia Pacific (Sydney) (`ap-southeast-2`)	`ami-05d25b3f16e685c2e`	`ami-0d0218a832355edf4`

> **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.

- **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. 3).
- **Subnets**: Choose the subnets that you created in Create your Amazon EKS Cluster VPC (p. 3).

8. On the **Options** page, you can choose to tag your stack resources. Choose **Next**.
9. On the **Review** page, review your information, acknowledge that the stack might create IAM resources, and then choose **Create**.
10. When your stack has finished creating, select it in the console and choose the **Outputs** tab.
11. 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.

   ```
   curl -O https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2018-12-10/
   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`, then your **kubectl** is not configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 63).

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 P2 or P3 instance type and the Amazon EKS-optimized AMI with GPU support, you must apply the NVIDIA device plugin for Kubernetes as a daemon set on your cluster with the following command.

   ```
   kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v1.11/
   nvidia-device-plugin.yml
   ```

Step 4: Launch a Guest Book Application

In this section, you create a sample guest book application to test your new 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`, then your **kubectl** is not configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 63).

   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
```

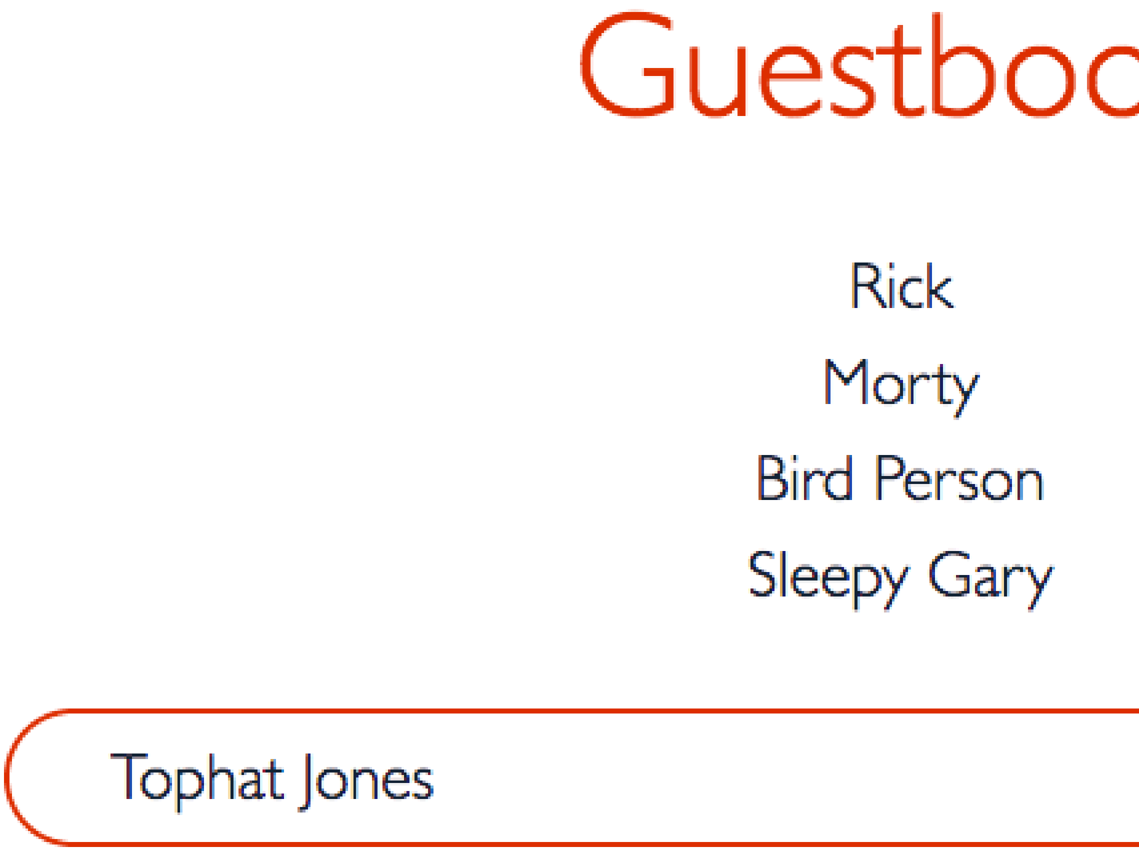
7. Query the services in your cluster and wait until the **External IP** column for the `guestbook` service is populated.

   Note
   It may 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 may take several minutes for DNS to propagate and for your guest book to show up.



Important
If you are unable to connect to the external IP address with your browser, be sure that your corporate firewall is not blocking non-standards ports, like 3000. You can try switching to a guest network to verify.

Step 5: Cleaning Up Guest Book Objects

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`, then your **kubectl** is not configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 63).

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. 22).

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, like `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.

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.

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).

Topics

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 our Getting Started with Amazon EKS (p. 3) guide instead, which provides a complete end-to-end walkthrough from creating an Amazon EKS cluster to deploying a sample Kubernetes application.

This topic 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. 44) and Cluster Security Group Considerations (p. 45). The Getting Started with Amazon EKS (p. 3) guide creates a VPC that meets the requirements, or you can also follow Tutorial: Creating a VPC with Public and Private Subnets for Your Amazon EKS Cluster (p. 83) to create one manually.
- 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 Service IAM Role (p. 75) to create one manually.

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. 68).

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. These also work for the AWS IAM Authenticator for Kubernetes. 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*.

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 Creating Amazon EKS IAM Policies (p. 74).
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**: The Amazon Resource Name (ARN) of your Amazon EKS service role. For more information, see Amazon EKS Service IAM Role (p. 75).
   - **VPC**: The VPC to use for your cluster.
   - **Subnets**: The subnets within the above VPC to use for your cluster. By default, the available subnets in the above VPC are preselected. Your subnets must meet the requirements for an Amazon EKS cluster. For more information, see Cluster VPC Considerations (p. 44).
   - **Security Groups**: Specify one or more (up to a limit of 5) security groups within the above 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. 45).

     Important
     The worker node AWS CloudFormation template modifies the security group that you specify here, so we recommend that you use a dedicated security group for your cluster control plane. If you share it with other resources, you may block or disrupt connections to those resources.

   Note
   You may receive an error that one of the Availability Zones in your request does not 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.
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. When your cluster provisioning is complete (usually less than 10 minutes), and 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. 63) and Create a `kubeconfig` for Amazon EKS (p. 65) to enable communication with your new cluster.

To create your cluster with the AWS CLI

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. 3), and the subnet and security group IDs for the VPC that you created in Create your Amazon EKS Cluster VPC (p. 3).

   ```
   aws eks create-cluster --name devel --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 may 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. Please update your AWS CLI version to the latest available and be sure to delete the custom service model directory at `~/.aws/models/eks`.

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

   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 Creating Amazon EKS IAM Policies (p. 74).

   Output:

   ```
   {
       "cluster": {
           "name": "devel",
           "arn": "arn:aws:eks:us-west-2:111122223333:cluster/devel",
           "createdAt": 1527785885.159,
           "version": "1.10",
           "roleArn": "arn:aws:iam::111122223333:role/eks-service-role-
   AWSServiceRoleForAmazonEKS-AFNL4H8HB71F",
           "resourcesVpcConfig": {
               "subnetIds": [
                   "subnet-a9189fe2",
                   "subnet-50432629"
               ],
               "securityGroupIds": [
                   "sg-f5c54184"
               ],
               "vpcId": "vpc-a54041dc"
           },
           "status": "CREATING",
           "certificateAuthority": {}
       }
   }
   ```

   Note
   You may receive an error that one of the Availability Zones in your request does not 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.

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

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

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

   a. Retrieve the `endpoint`:

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

   b. Retrieve the `certificateAuthority.data`:

   ```
   aws eks 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. 63) and Create a `kubeconfig` for Amazon EKS (p. 65) to enable communication with your new cluster.

Updating an Amazon EKS Cluster

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, and as such, 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.

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

To update an existing cluster 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 update and choose **Update cluster version**.
3. For **Kubernetes version**, select the version to update your cluster to and choose **Update**.
4. For **Cluster name**, type the name of your cluster and choose **Confirm**.

   Note
   The cluster update should finish within a few minutes.
5. Patch the `kube-proxy` daemonset to use the image that corresponds to your current cluster Kubernetes version (in this example, *`1.11.5`*).

```
kubectl patch daemonset kube-proxy \
-n kube-system \
-p '{"spec": {"template": {"spec": {"containers": [{"image": "602401143452.dkr.ecr.us-
west-2.amazonaws.com/eks/kube-proxy:v1.11.5","name":"kube-proxy"}]}}}}'
```

6. 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 1.11, and you would like to use

CoreDNS for DNS and service discovery, you must install CoreDNS and remove `kube-dns`. For more information, see Installing CoreDNS (p. 49)

7. 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. 34).

To update an existing cluster with the AWS CLI

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

   ```
   aws eks update-cluster-version --name prod --kubernetes-version 1.11
   ```

   Output:

   ```
   {
       "update": {
           "id": "b5f0ba18-9a87-4450-b5a0-825e6e84496f",
           "status": "InProgress",
           "type": "VersionUpdate",
           "params": [
               {
                   "type": "Version",
                   "value": "1.11"
               },
               {
                   "type": "PlatformVersion",
                   "value": "eks.1"
               }
           ],
           "createdAt": 1544051347.305,
           "errors": []
       }
   }
   ```

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

   ```
   aws eks describe-update --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": "1.11"
               },
               {
                   "type": "PlatformVersion",
                   "value": "eks.1"
               }
           ],
           "createdAt": 1544051347.305,
           "errors": []
   ```

```
    }
}
```

3. Patch the `kube-proxy` daemonset to use the image that corresponds to your current cluster Kubernetes version (in this example, *1.11.5*).

```
kubectl patch daemonset kube-proxy \
-n kube-system \
-p '{"spec": {"template": {"spec": {"containers": [{"image": "602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/kube-proxy:v1.11.5","name":"kube-proxy"}]}}}}'
```

4. 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 1.11, and you would like to use CoreDNS for DNS and service discovery, you must install CoreDNS and remove `kube-dns`. For more information, see Installing CoreDNS (p. 49)
5. 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. 34).

Deleting a Cluster

When you are done using an Amazon EKS cluster, you should delete the resources associated with it so that you do not 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.

To delete an Amazon EKS cluster

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**, **Delete Stack**.

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

Platform Versions

Starting with the release of Kubernetes aggregation layer support in new clusters, Amazon EKS introduced the concept of platform versions, which allows you to identify the features that are currently enabled for your clusters. The Amazon EKS platform version represents capabilities of the cluster control plane, such as which Kubernetes API server flags are enabled, as well as the current Kubernetes patch version.

New clusters are created with the latest available platform version for the specified Kubernetes version. When a new platform version becomes available for a Kubernetes version, existing clusters are automatically upgraded to the latest platform version for their Kubernetes version in incremental roll outs. To take advantage of the latest platform version features immediately, you can create a new Amazon EKS cluster.

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

Kubernetes 1.11 Platform Versions

Kubernetes Version	Kubernetes Patch Version	Amazon EKS Platform Version	Enabled Admission Controllers	Release Notes
`1.11`	`1.11.5`	`eks.1`	`Initializers, NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook`	Initial release of Kubernetes 1.11 for Amazon EKS.

Kubernetes 1.10 Platform Versions

Kubernetes Version	Kubernetes Patch Version	Amazon EKS Platform Version	Enabled Admission Controllers	Release Notes
`1.10`	`1.10.11`	`eks.3`	`Initializers, NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook`	New platform version updating Kubernetes to patch level 1.10.11 to address CVE-2018-1002105.

Kubernetes Version	Kubernetes Patch Version	Amazon EKS Platform Version	Enabled Admission Controllers	Release Notes
1.10	1.10.3	eks.2	`Initializers, 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	1.10.3	eks.1	`Initializers, NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction`	Initial launch of Amazon EKS.

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 On-Demand 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 that triggers a specialized bootstrap script that allows them to discover and connect to your cluster's control plane automatically. For more information, see Launching Amazon EKS Worker Nodes (p. 30).

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

Topics

Amazon EKS-Optimized AMI

The Amazon EKS-optimized 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. 27)) are shown in the following table.

Note
The Amazon EKS-optimized AMI with GPU support only supports P2 and P3 instance types. Be sure to specify these instance types in your worker node AWS CloudFormation template.

Because this AMI includes third-party software that requires an end user license agreement (EULA), you must subscribe to the AMI in the AWS Marketplace and accept the EULA before you can use the AMI in your worker node groups. To subscribe to the AMI, visit the AWS Marketplace.

Kubernetes version 1.11

Region	Amazon EKS-optimized AMI	with GPU support
US West (Oregon) (`us-west-2`)	`ami-094fa4044a2a3cf52`	`ami-014f4e495a19d3e4f`
US East (N. Virginia) (`us-east-1`)	`ami-0b4eb1d8782fc3aea`	`ami-08a0bb74d1c9a5e2f`
US East (Ohio) (`us-east-2`)	`ami-053cbe66e0033ebcf`	`ami-04a758678ae5ebad5`
EU (Frankfurt) (`eu-central-1`)	`ami-0ce0ec06e682ee10e`	`ami-017912381e1ebb308`
EU (Stockholm) (`eu-north-1`)	`ami-082e6cf1c07e60241`	`ami-69b03e17`
EU (Ireland) (`eu-west-1`)	`ami-0a9006fb385703b54`	`ami-050db3f5f9dbd4439`
Asia Pacific (Tokyo) (`ap-northeast-1`)	`ami-063650732b3e8b38c`	`ami-080be783089a635dd`
Asia Pacific (Singapore) (`ap-southeast-1`)	`ami-0549ac6995b998478`	`ami-05bbe4b57e4030910`
Asia Pacific (Sydney) (`ap-southeast-2`)	`ami-03297c04f71690a76`	`ami-0da8916a67c116ace`

Kubernetes version 1.10

Region	Amazon EKS-optimized AMI	with GPU support
US West (Oregon) (`us-west-2`)	`ami-07af9511082779ae7`	`ami-08754f7ac73185331`
US East (N. Virginia) (`us-east-1`)	`ami-027792c3cc6de7b5b`	`ami-03c499c67bc65c089`
US East (Ohio) (`us-east-2`)	`ami-036130f4127a367f7`	`ami-081210a2fd7f3c487`
EU (Frankfurt) (`eu-central-1`)	`ami-06d069282a5fea248`	`ami-03b492cd6806000ff`
EU (Stockholm) (`eu-north-1`)	`ami-04b0f84e5a05e0b30`	`ami-24b43a5a`
EU (Ireland) (`eu-west-1`)	`ami-03612357ac9da2c7d`	`ami-047637529a86c7237`
Asia Pacific (Tokyo) (`ap-northeast-1`)	`ami-06f4af3742fca5998`	`ami-02a0802829f472c55`
Asia Pacific (Singapore) (`ap-southeast-1`)	`ami-0bc97856f0dd86d41`	`ami-0092b6c977d1937f0`
Asia Pacific (Sydney) (`ap-southeast-2`)	`ami-05d25b3f16e685c2e`	`ami-0d0218a832355edf4`

Important
These AMIs require the latest AWS CloudFormation worker node template. You cannot 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/2018-12-10/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 that allows them to discover and connect to your cluster's control plane automatically. For more information, see Launching Amazon EKS Worker Nodes (p. 30).

Amazon EKS-Optimized AMI Build Scripts

Amazon Elastic Container Service for Kubernetes (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.

Note
The Amazon EKS-optimized AMI with GPU support only supports P2 and P3 instance types. Be sure to specify these instance types in your worker node AWS CloudFormation template. Because this AMI includes third-party software that requires an end user license agreement (EULA), you must subscribe to the AMI in the AWS Marketplace and accept the EULA before you can use the AMI in your worker node groups. To subscribe to the AMI, visit the AWS Marketplace.

Kubernetes version 1.11

Region	Amazon EKS-optimized AMI with GPU support
US West (Oregon) (`us-west-2`)	`ami-014f4e495a19d3e4f`
US East (N. Virginia) (`us-east-1`)	`ami-08a0bb74d1c9a5e2f`
US East (Ohio) (`us-east-2`)	`ami-04a758678ae5ebad5`
EU (Frankfurt) (`eu-central-1`)	`ami-017912381e1ebb308`
EU (Stockholm) (`eu-north-1`)	`ami-69b03e17`
EU (Ireland) (`eu-west-1`)	`ami-050db3f5f9dbd4439`
Asia Pacific (Tokyo) (`ap-northeast-1`)	`ami-080be783089a635dd`
Asia Pacific (Singapore) (`ap-southeast-1`)	`ami-05bbe4b57e4030910`
Asia Pacific (Sydney) (`ap-southeast-2`)	`ami-0da8916a67c116ace`

Kubernetes version 1.10

Region	Amazon EKS-optimized AMI with GPU support
US West (Oregon) (`us-west-2`)	`ami-08754f7ac73185331`
US East (N. Virginia) (`us-east-1`)	`ami-03c499c67bc65c089`
US East (Ohio) (`us-east-2`)	`ami-081210a2fd7f3c487`
EU (Frankfurt) (`eu-central-1`)	`ami-03b492cd6806000ff`
EU (Stockholm) (`eu-north-1`)	`ami-24b43a5a`
EU (Ireland) (`eu-west-1`)	`ami-047637529a86c7237`
Asia Pacific (Tokyo) (`ap-northeast-1`)	`ami-02a0802829f472c55`
Asia Pacific (Singapore) (`ap-southeast-1`)	`ami-0092b6c977d1937f0`
Asia Pacific (Sydney) (`ap-southeast-2`)	`ami-0d0218a832355edf4`

Important
These AMIs require the latest AWS CloudFormation worker node template. You cannot 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/2018-12-10/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 that allows them to discover and connect to your cluster's control plane automatically. For more information, see Launching Amazon EKS Worker Nodes (p. 30).

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

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v1.11/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

Example

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

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

Apply the manifest with the following command:

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

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

```
kubectl logs nvidia-smi
```

Output:

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


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

Amazon EKS 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 Container Service for Kubernetes (EKS) and Optimized Support for Amazon EKS on Ubuntu 18.04.

Launching Amazon EKS Worker Nodes

This topic helps you to launch an Auto Scaling group of 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 worker nodes, we recommend that you follow our Getting Started with Amazon EKS (p. 3) guide instead. The guide provides a complete end-to-end walkthrough from creating an Amazon EKS cluster to deploying a sample Kubernetes application.

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

This topic has the following prerequisites:

- You have created a VPC and security group that meets the requirements for an Amazon EKS cluster. For more information, see Cluster VPC Considerations (p. 44) and Cluster Security Group Considerations (p. 45). The Getting Started with Amazon EKS (p. 3) guide creates a VPC that meets the requirements, or you can also follow Tutorial: Creating a VPC with Public and Private Subnets for Your Amazon EKS Cluster (p. 83) to create one manually.
- You have created an Amazon EKS cluster and specified that it use the above VPC and security group. For more information, see Creating an Amazon EKS Cluster (p. 17).

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. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
3. From the navigation bar, select a Region that supports Amazon EKS.

   Note
   Amazon EKS is available in the following Regions at this time:

   - **US West (Oregon)** (`us-west-2`)
   - **US East (N. Virginia)** (`us-east-1`)
   - **US East (Ohio)** (`us-east-2`)
   - **EU (Frankfurt)** (`eu-central-1`)
   - **EU (Stockholm)** (`eu-north-1`)
   - **EU (Ireland)** (`eu-west-1`)

   - **Asia Pacific (Tokyo)** (`ap-northeast-1`)
   - **Asia Pacific (Singapore)** (`ap-southeast-1`)
   - **Asia Pacific (Sydney)** (`ap-southeast-2`)
4. Choose **Create stack**.
5. For **Choose a template**, select **Specify an Amazon S3 template URL**.
6. Paste the following URL into the text area and choose **Next**:

```
https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2018-12-10/amazon-eks-
nodegroup.yaml
```

7. On the **Specify Details** page, fill out the following parameters accordingly, and choose **Next**:

   - **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 your Amazon EKS cluster name. Otherwise, your worker nodes will be unable to join it.
   - **ClusterControlPlaneSecurityGroup**: Enter the security group or groups that you used when you created your Amazon EKS cluster. This AWS CloudFormation template creates a worker node security group that allows traffic to and from the cluster control plane security group specified.
   - **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 to which your worker node Auto Scaling group can scale in.
   - **NodeAutoScalingGroupDesiredCapacity**: Enter the desired number of nodes to scale to when your stack is created.
   - **NodeAutoScalingGroupMaxSize**: Enter the maximum number of nodes to which your worker node Auto Scaling group can scale out. This value must be at least 1 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 an instance type for your worker nodes.
   - **NodeImageId**: Enter the current Amazon EKS worker node AMI ID for your Region. The AMI IDs for the latest Amazon EKS-optimized AMI (with and without GPU support (p. 27)) are shown in the following table. Be sure to choose the correct AMI ID for your desired Kubernetes version and AWS region.

     Note
     The Amazon EKS-optimized AMI with GPU support only supports P2 and P3 instance types. Be sure to specify these instance types in your worker node AWS CloudFormation template. Because this AMI includes third-party software that requires an end user license agreement (EULA), you must subscribe to the AMI in the AWS Marketplace and accept the EULA before you can use the AMI in your worker node groups. To subscribe to the AMI, visit the AWS Marketplace.

Kubernetes version 1.11

Region	Amazon EKS-optimized AMI	with GPU support
US West (Oregon) (`us-west-2`)	`ami-094fa4044a2a3cf52`	`ami-014f4e495a19d3e4f`
US East (N. Virginia) (`us-east-1`)	`ami-0b4eb1d8782fc3aea`	`ami-08a0bb74d1c9a5e2f`

Region	Amazon EKS-optimized AMI	with GPU support
US East (Ohio) (`us-east-2`)	`ami-053cbe66e0033ebcf`	`ami-04a758678ae5ebad5`
EU (Frankfurt) (`eu-central-1`)	`ami-0ce0ec06e682ee10e`	`ami-017912381e1ebb308`
EU (Stockholm) (`eu-north-1`)	`ami-082e6cf1c07e60241`	`ami-69b03e17`
EU (Ireland) (`eu-west-1`)	`ami-0a9006fb385703b54`	`ami-050db3f5f9dbd4439`
Asia Pacific (Tokyo) (`ap-northeast-1`)	`ami-063650732b3e8b38c`	`ami-080be783089a635dd`
Asia Pacific (Singapore) (`ap-southeast-1`)	`ami-0549ac6995b998478`	`ami-05bbe4b57e4030910`
Asia Pacific (Sydney) (`ap-southeast-2`)	`ami-03297c04f71690a76`	`ami-0da8916a67c116ace`

Kubernetes version 1.10

Region	Amazon EKS-optimized AMI	with GPU support
US West (Oregon) (`us-west-2`)	`ami-07af9511082779ae7`	`ami-08754f7ac73185331`
US East (N. Virginia) (`us-east-1`)	`ami-027792c3cc6de7b5b`	`ami-03c499c67bc65c089`
US East (Ohio) (`us-east-2`)	`ami-036130f4127a367f7`	`ami-081210a2fd7f3c487`
EU (Frankfurt) (`eu-central-1`)	`ami-06d069282a5fea248`	`ami-03b492cd6806000ff`
EU (Stockholm) (`eu-north-1`)	`ami-04b0f84e5a05e0b30`	`ami-24b43a5a`
EU (Ireland) (`eu-west-1`)	`ami-03612357ac9da2c7d`	`ami-047637529a86c7237`
Asia Pacific (Tokyo) (`ap-northeast-1`)	`ami-06f4af3742fca5998`	`ami-02a0802829f472c55`
Asia Pacific (Singapore) (`ap-southeast-1`)	`ami-0bc97856f0dd86d41`	`ami-0092b6c977d1937f0`
Asia Pacific (Sydney) (`ap-southeast-2`)	`ami-05d25b3f16e685c2e`	`ami-0d0218a832355edf4`

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.

- **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 your worker nodes should launch into.
- **Subnets**: Choose the subnets within the above VPC that your worker nodes should launch into.

8. On the **Options** page, you can choose to tag your stack resources. Choose **Next**.
9. On the **Review** page, review your information, acknowledge that the stack might create IAM resources, and then choose **Create**.
10. When your stack has finished creating, select it in the console and choose **Outputs**.
11. 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. Download the configuration map:

   ```
   curl -O https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2018-12-10/
   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`, then your **kubectl** is not configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 63).

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 P2 or P3 instance type and the Amazon EKS-optimized AMI with GPU support, you must apply the NVIDIA device plugin for Kubernetes as a daemon set on your cluster with the following command.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v1.11/
nvidia-device-plugin.yml
```

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. Migrating to a new worker node group is more graceful than simply updating the AMI ID in an existing AWS CloudFormation stack, because it 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

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.

To migrate your applications to a new worker node group

1. Launch a new worker node group by following the steps outlined in Launching Amazon EKS Worker Nodes (p. 30).
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 (for example, to add 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. 45).

   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 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. 34), 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. 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) with the following command:

```
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 2 replicas, scale out the deployment to 2 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 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 (for example, to add 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 1 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. If your cluster is using `kube-dns` for DNS resolution (see step Step 9 (p. 36)), scale in the `kube-dns` deployment to 1 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, so you always have your Auto Scaling group's desired count of active instance in your cluster during the rolling update.

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 2 replicas, scale out the deployment to 2 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 0 replicas to avoid conflicting scaling actions.

   ```
   kubectl scale deployments/cluster-autoscaler --replicas=0 -n kube-system
   ```
4. Determine the instance type and desired instance count of your current 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 **Actions**, **Update stack**.
7. For **Choose a template**, select **Specify an Amazon S3 template URL**.
8. 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 choose **Next**:

   ```
   https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2018-12-10/amazon-eks-
   nodegroup.yaml
   ```
9. On the **Specify Details** page, fill out the following parameters accordingly, and choose **Next**:

   - **NodeAutoScalingGroupDesiredCapacity**: Enter the desired instance count that you recorded in Step 4 (p. 38), 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 1 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. 38), or choose a different instance type for your worker nodes.
- **NodeImageId**: Enter the current Amazon EKS worker node AMI ID for your Region. The AMI IDs for the latest Amazon EKS-optimized AMI (with and without GPU support (p. 27)) are shown in the following table.

  Note
  The Amazon EKS-optimized AMI with GPU support only supports P2 and P3 instance types. Be sure to specify these instance types in your worker node AWS CloudFormation template. Because this AMI includes third-party software that requires an end user license agreement (EULA), you must subscribe to the AMI in the AWS Marketplace and accept the EULA before you can use the AMI in your worker node groups. To subscribe to the AMI, visit the AWS Marketplace.

Kubernetes version 1.11

Region	Amazon EKS-optimized AMI	with GPU support
US West (Oregon) (`us-west-2`)	`ami-094fa4044a2a3cf52`	`ami-014f4e495a19d3e4f`
US East (N. Virginia) (`us-east-1`)	`ami-0b4eb1d8782fc3aea`	`ami-08a0bb74d1c9a5e2f`
US East (Ohio) (`us-east-2`)	`ami-053cbe66e0033ebcf`	`ami-04a758678ae5ebad5`
EU (Frankfurt) (`eu-central-1`)	`ami-0ce0ec06e682ee10e`	`ami-017912381e1ebb308`
EU (Stockholm) (`eu-north-1`)	`ami-082e6cf1c07e60241`	`ami-69b03e17`
EU (Ireland) (`eu-west-1`)	`ami-0a9006fb385703b54`	`ami-050db3f5f9dbd4439`
Asia Pacific (Tokyo) (`ap-northeast-1`)	`ami-063650732b3e8b38c`	`ami-080be783089a635dd`
Asia Pacific (Singapore) (`ap-southeast-1`)	`ami-0549ac6995b998478`	`ami-05bbe4b57e4030910`
Asia Pacific (Sydney) (`ap-southeast-2`)	`ami-03297c04f71690a76`	`ami-0da8916a67c116ace`

Kubernetes version 1.10

Region	Amazon EKS-optimized AMI	with GPU support
US West (Oregon) (`us-west-2`)	`ami-07af9511082779ae7`	`ami-08754f7ac73185331`
US East (N. Virginia) (`us-east-1`)	`ami-027792c3cc6de7b5b`	`ami-03c499c67bc65c089`
US East (Ohio) (`us-east-2`)	`ami-036130f4127a367f7`	`ami-081210a2fd7f3c487`
EU (Frankfurt) (`eu-central-1`)	`ami-06d069282a5fea248`	`ami-03b492cd6806000ff`

Region	Amazon EKS-optimized AMI	with GPU support
EU (Stockholm) (`eu-north-1`)	`ami-04b0f84e5a05e0b30`	`ami-24b43a5a`
EU (Ireland) (`eu-west-1`)	`ami-03612357ac9da2c7d`	`ami-047637529a86c7237`
Asia Pacific (Tokyo) (`ap-northeast-1`)	`ami-06f4af3742fca5998`	`ami-02a0802829f472c55`
Asia Pacific (Singapore) (`ap-southeast-1`)	`ami-0bc97856f0dd86d41`	`ami-0092b6c977d1937f0`
Asia Pacific (Sydney) (`ap-southeast-2`)	`ami-05d25b3f16e685c2e`	`ami-0d0218a832355edf4`

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.

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**.

    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 1 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 1 replica.

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

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.

To create an AWS storage class for your Amazon EKS cluster

1. Create an AWS storage class manifest file for your storage class. The below example 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 in the Kubernetes documentation. For this example, the file is called `gp2-storage-class.yaml`.

```
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
sc1       kubernetes.io/aws-ebs     6s
```

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
sc1             kubernetes.io/aws-ebs   4m
```

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.

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.

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
```

Amazon EKS Networking

This chapter covers networking considerations for running Kubernetes on Amazon EKS.

Topics

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.

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 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:

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`

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 need 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 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	
Recommended inbound traffic	TCP	443	All worker node security groups	
Minimum outbound traffic	TCP	10250		All worker node security groups
Recommended outbound traffic	TCP	1025-65535		All worker node security groups

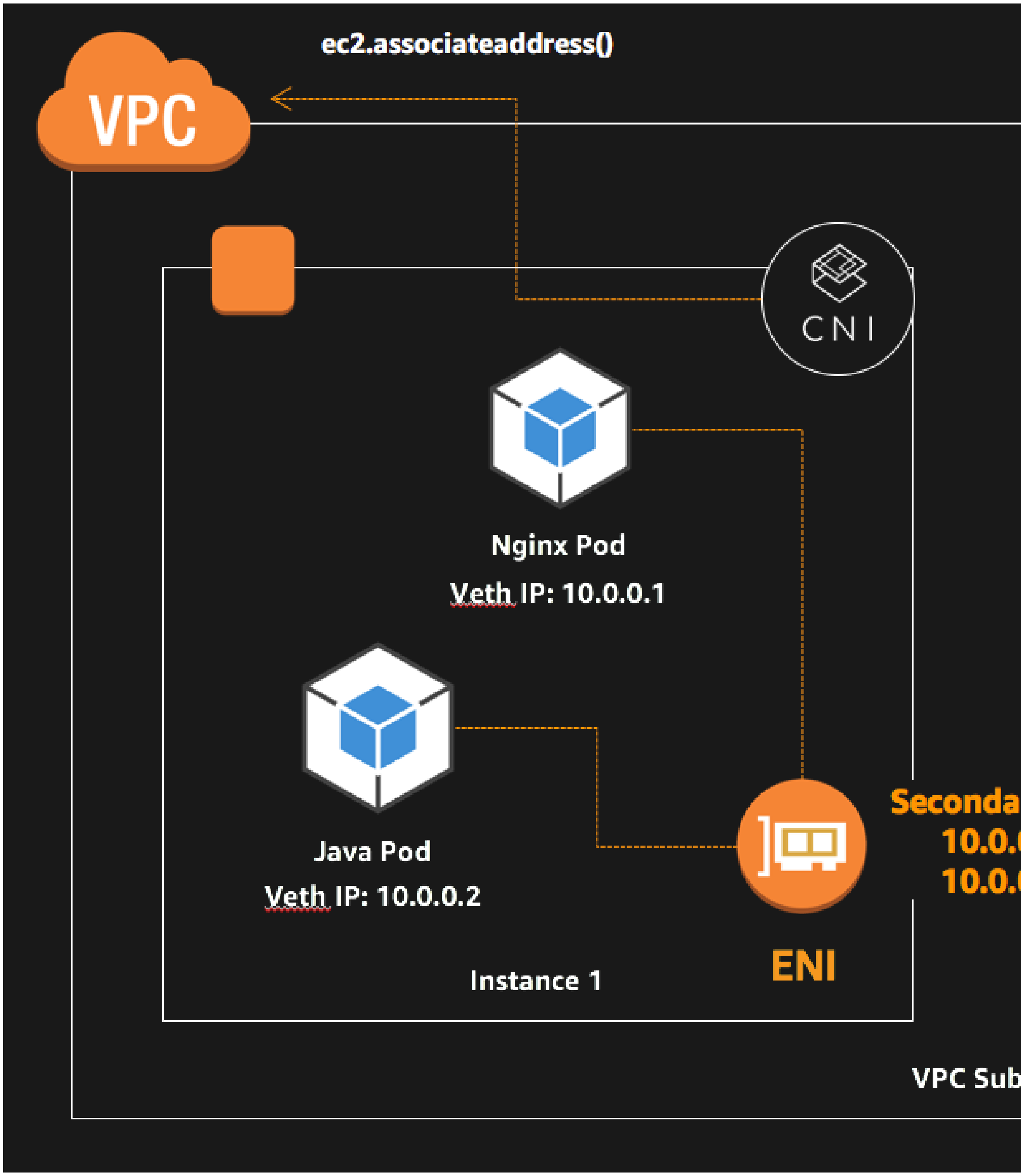
Worker Node Security Groups

	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

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 Proposal: 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*.

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. 53).

`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. 51).

`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.

Installing CoreDNS

New Amazon EKS clusters created with Kubernetes version 1.11 ship with CoreDNS as the default DNS and service discovery 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 1.11, and you would like to use CoreDNS for DNS and service discovery, you must install CoreDNS and remove `kube-dns`.

You can check to see if your cluster is already running CoreDNS with the following command:

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

If the output shows `coredns` in the pod names, then you are already running CoreDNS in your cluster. If not, use the following procedure to update your DNS and service discovery provider to CoreDNS.

To install CoreDNS on an updated Amazon EKS cluster

1. Add the `{"eks.amazonaws.com/component": "kube-dns"}` selector to the `kube-dns` deployment for your cluster (this is to prevent 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 https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2018-12-10/
   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 is receiving requests.

   ```
   kubectl get --raw /api/v1/namespaces/kube-system/pods/$COREDNS_POD:9153/proxy/
   metrics \
   | grep 'coredns_dns_request_count_total'
   ```

   Note
   It may 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. Scale down the `kube-dns` deployment to 0 replicas.

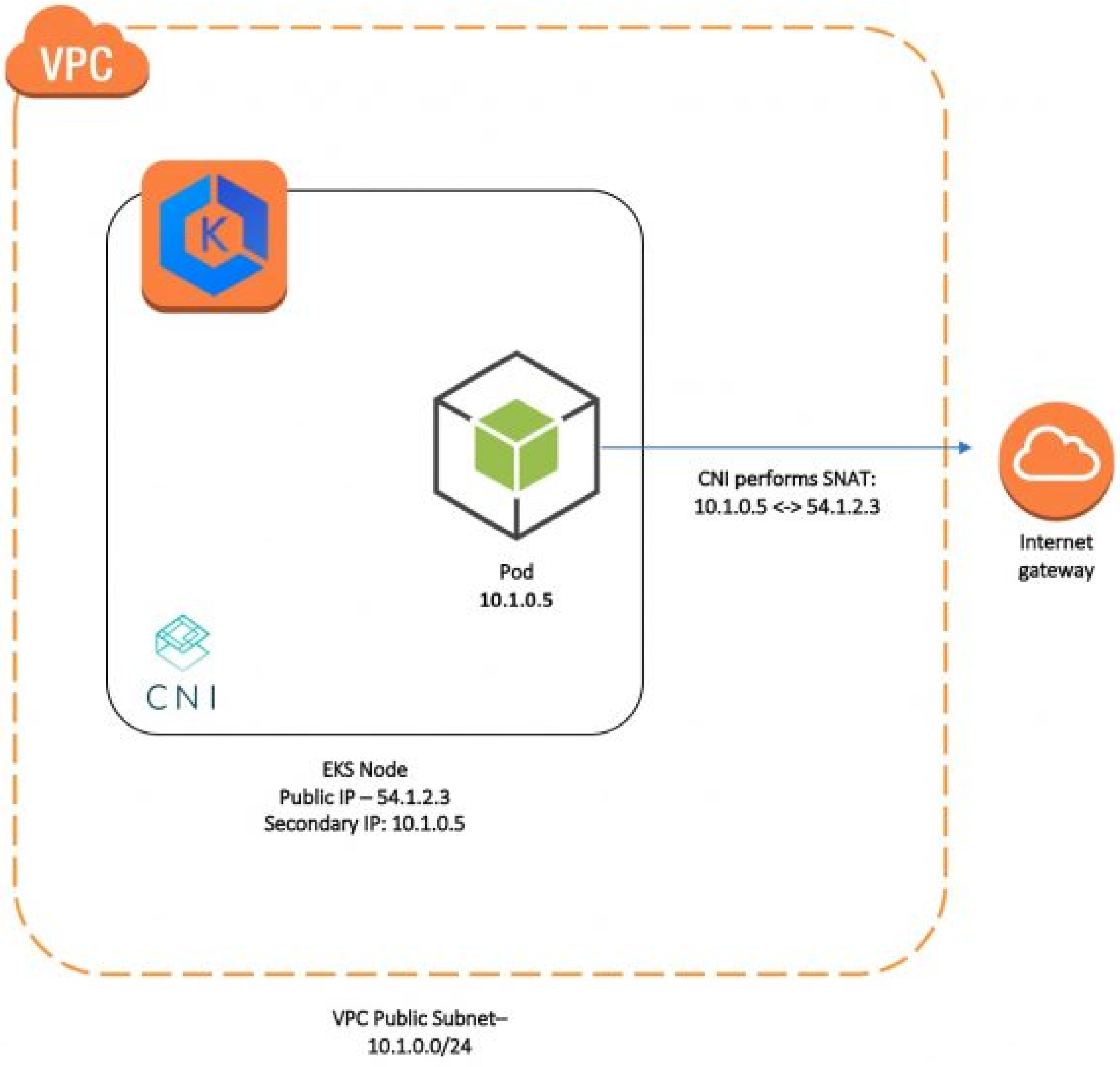
```
kubectl  scale -n kube-system deployment/kube-dns --replicas=0
```

4. Clean up the old `kube-dns` resources.

```
kubectl delete -n kube-system deployment/kube-dns serviceaccount/kube-dns configmap/
kube-dns
```

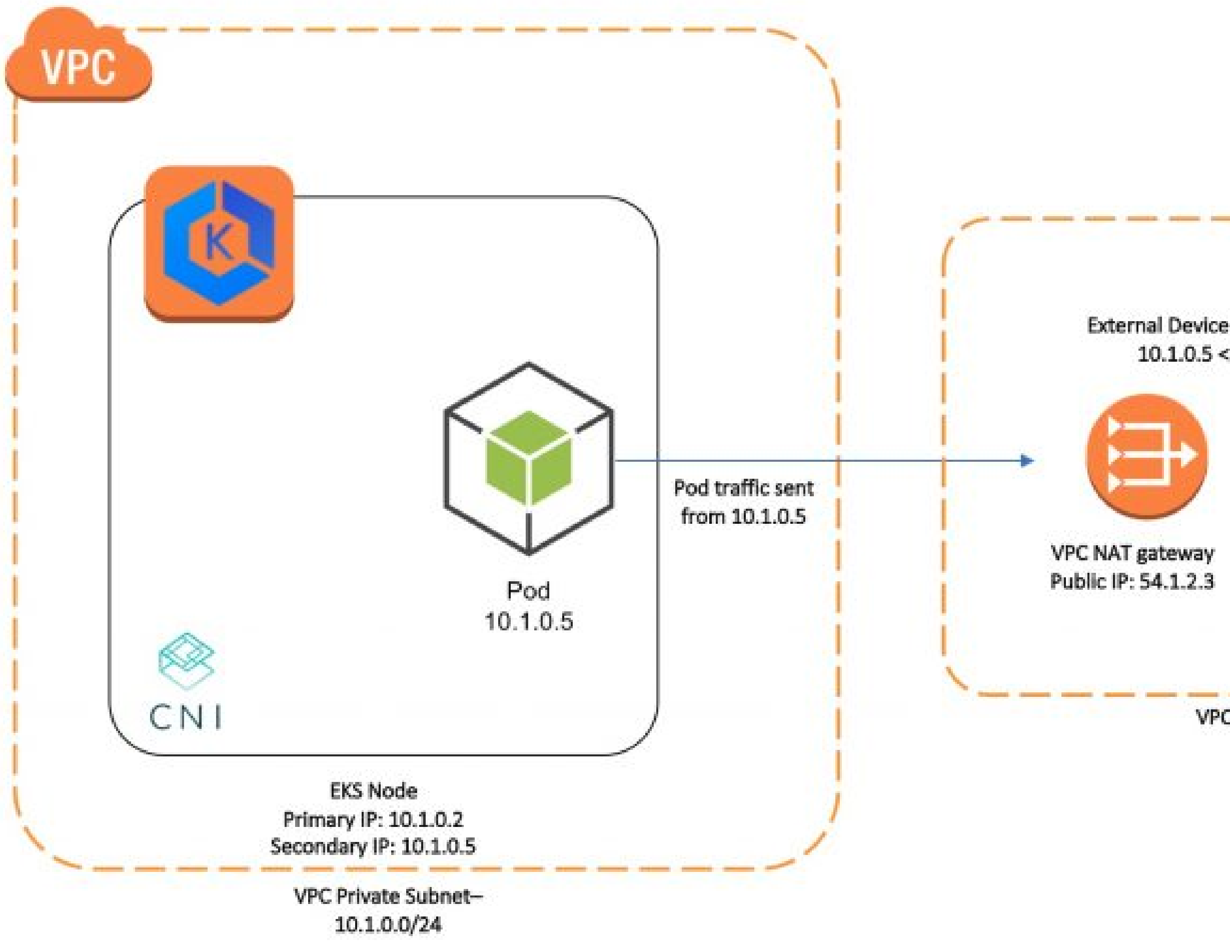
External Source Network Address Translation (SNAT)

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

1. Edit the `aws-node` configmap:

```
kubectl edit daemonset -n kube-system aws-node
```

2. Add the `AWS_VPC_K8S_CNI_EXTERNALSNAT` environment variable to the node container spec and set it to `true`:

```
...
    spec:
      containers:
      - env:
        - name: AWS_VPC_K8S_CNI_EXTERNALSNAT
          value: "true"
        - name: AWS_VPC_K8S_CNI_LOGLEVEL
          value: DEBUG
        - name: MY_NODE_NAME
...
```

3. Save the file and exit your text editor.

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.

Note
This feature requires Amazon VPC CNI plugin for Kubernetes version 1.2.1 or later. To check your CNI version, and upgrade if necessary, see Amazon VPC CNI Plugin for Kubernetes Upgrades (p. 55).

- 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.

To configure CNI custom networking

1. Edit the `aws-node` configmap:

```
kubectl edit daemonset -n kube-system aws-node
```

2. Add the `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG` environment variable to the node container spec and set it to `true`:

```
...
    spec:
      containers:
      - env:
        - name: AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG
          value: "true"
        - name: AWS_VPC_K8S_CNI_LOGLEVEL
          value: DEBUG
        - name: MY_NODE_NAME
...
```

3. Save the file and exit your text editor.
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:
       scope: Cluster
       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 definition for each subnet in which your pods reside.

   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. For this example, the file is called *`group1-pod-netconfig`*`.yaml`.

      Note
      Each subnet and security group combination requires its own custom resource definition.

   ```
   apiVersion: crd.k8s.amazonaws.com/v1alpha1
   kind: ENIConfig
   metadata:
    name: group1-pod-netconfig
   spec:
    subnet: subnet-0c4678ec01ce68b24
    securityGroups:
    - sg-066c7927a794cf7e7
    - sg-08f5f22cfb70d8405
    - sg-0bb293fc16f5c2058
   ```

   b. Apply each custom resource definition file that you created earlier to your cluster with the following command:

   ```
   kubectl apply -f group1-pod-netconfig.yaml
   ```

6. For each node in your cluster, annotate the node with the custom network configuration to use. Worker nodes can only be annotated with a single `ENIConfig` value at a time. The subnet in the `ENIConfig` must belong to the same Availability Zone in which the worker node resides.

```
kubectl annotate node <nodename>.<region>.compute.internal k8s.amazonaws.com/
eniConfig=group1-pod-netconfig
```

Amazon VPC CNI Plugin for Kubernetes Upgrades

When you launch an Amazon EKS cluster, we apply the latest version of the Amazon VPC CNI plugin for Kubernetes to your cluster. 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 current CNI version is 1.3.0. 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:1.2.1
  ```

  In this example output, the CNI version is 1.2.1, which is earlier than the current version, 1.3.0. 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:

  ```
  kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/master/
  config/v1.3/aws-k8s-cni.yaml
  ```

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 daemon sets in the `kube-system` namespace.

   ```
   kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/master/
   config/v1.3/calico.yaml
   ```

2. Watch the `kube-system` daemon sets and wait for the `calico-node` daemon set 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
```

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/v3.1/getting-started/kubernetes/
tutorials/stars-policy/manifests/00-namespace.yaml
kubectl apply -f https://docs.projectcalico.org/v3.1/getting-started/kubernetes/
tutorials/stars-policy/manifests/01-management-ui.yaml
kubectl apply -f https://docs.projectcalico.org/v3.1/getting-started/kubernetes/
tutorials/stars-policy/manifests/02-backend.yaml
kubectl apply -f https://docs.projectcalico.org/v3.1/getting-started/kubernetes/
tutorials/stars-policy/manifests/03-frontend.yaml
kubectl apply -f https://docs.projectcalico.org/v3.1/getting-started/kubernetes/
tutorials/stars-policy/manifests/04-client.yaml
```

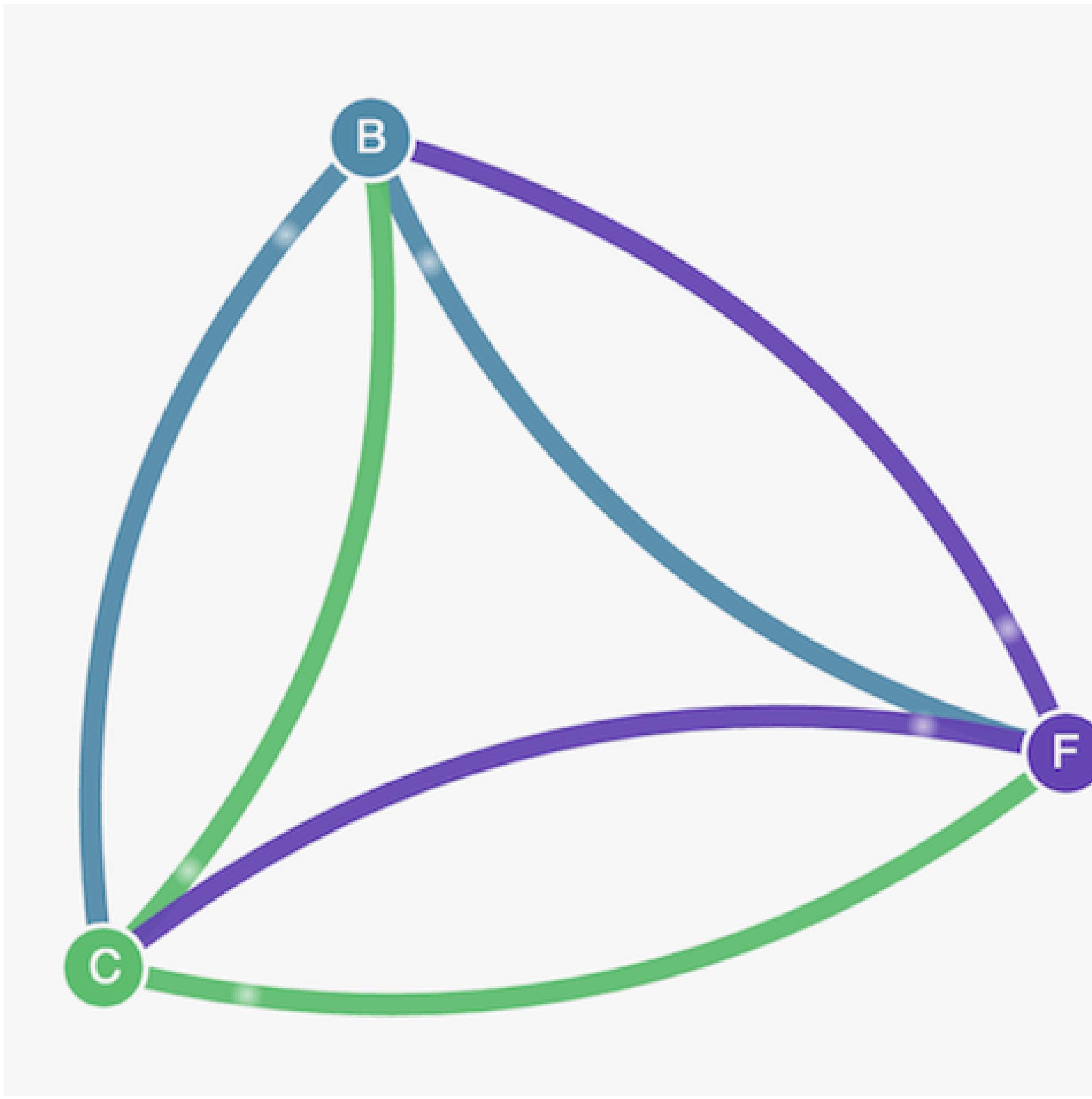
2. Wait for all of the pods to reach the `Running` status:

```
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:

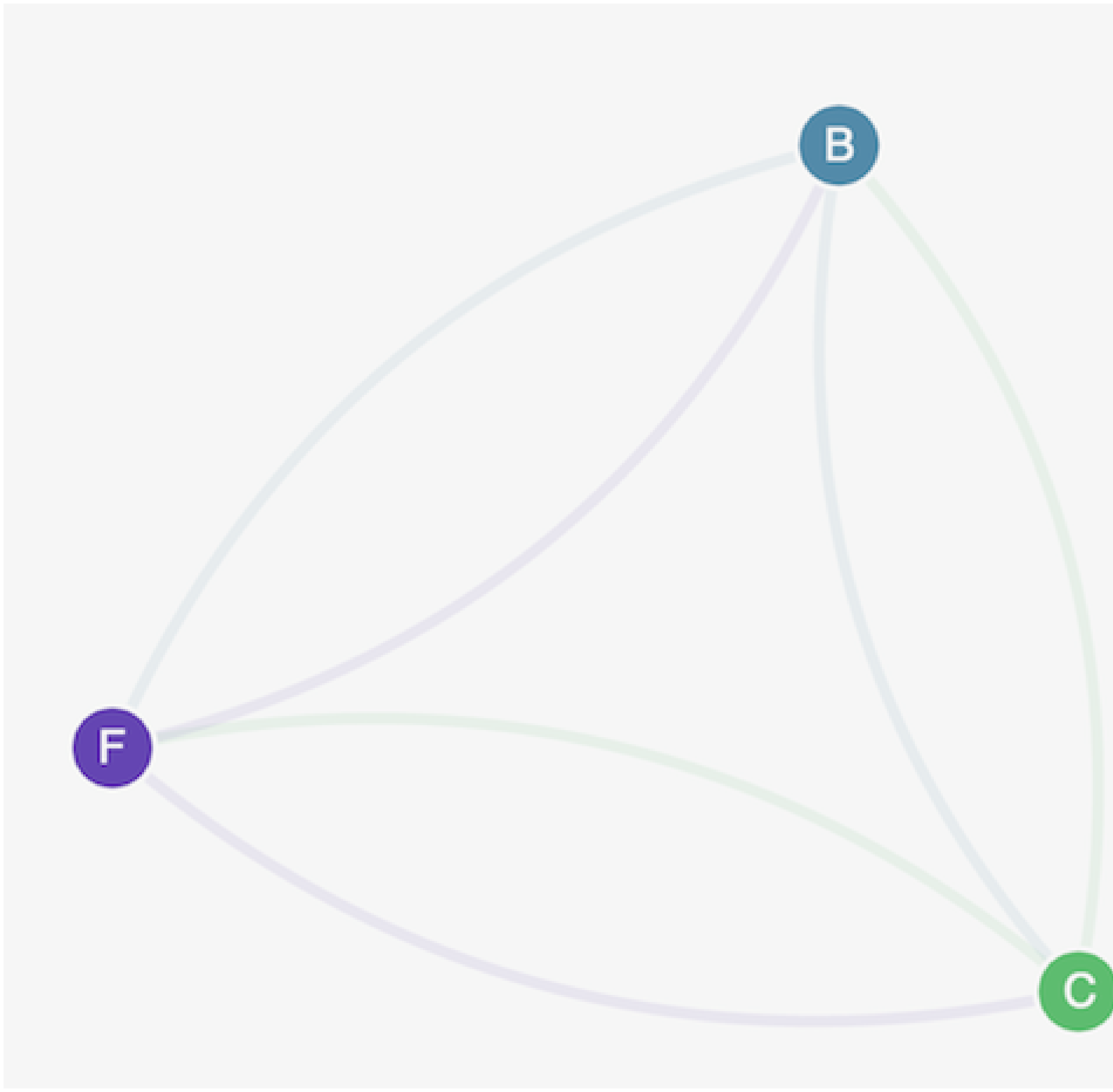
```
kubectl apply -n stars -f https://docs.projectcalico.org/v3.1/getting-started/
kubernetes/tutorials/stars-policy/policies/default-deny.yaml
kubectl apply -n client -f https://docs.projectcalico.org/v3.1/getting-started/
kubernetes/tutorials/stars-policy/policies/default-deny.yaml
```

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/v3.1/getting-started/kubernetes/
tutorials/stars-policy/policies/allow-ui.yaml
```

```
kubectl apply -f https://docs.projectcalico.org/v3.1/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.

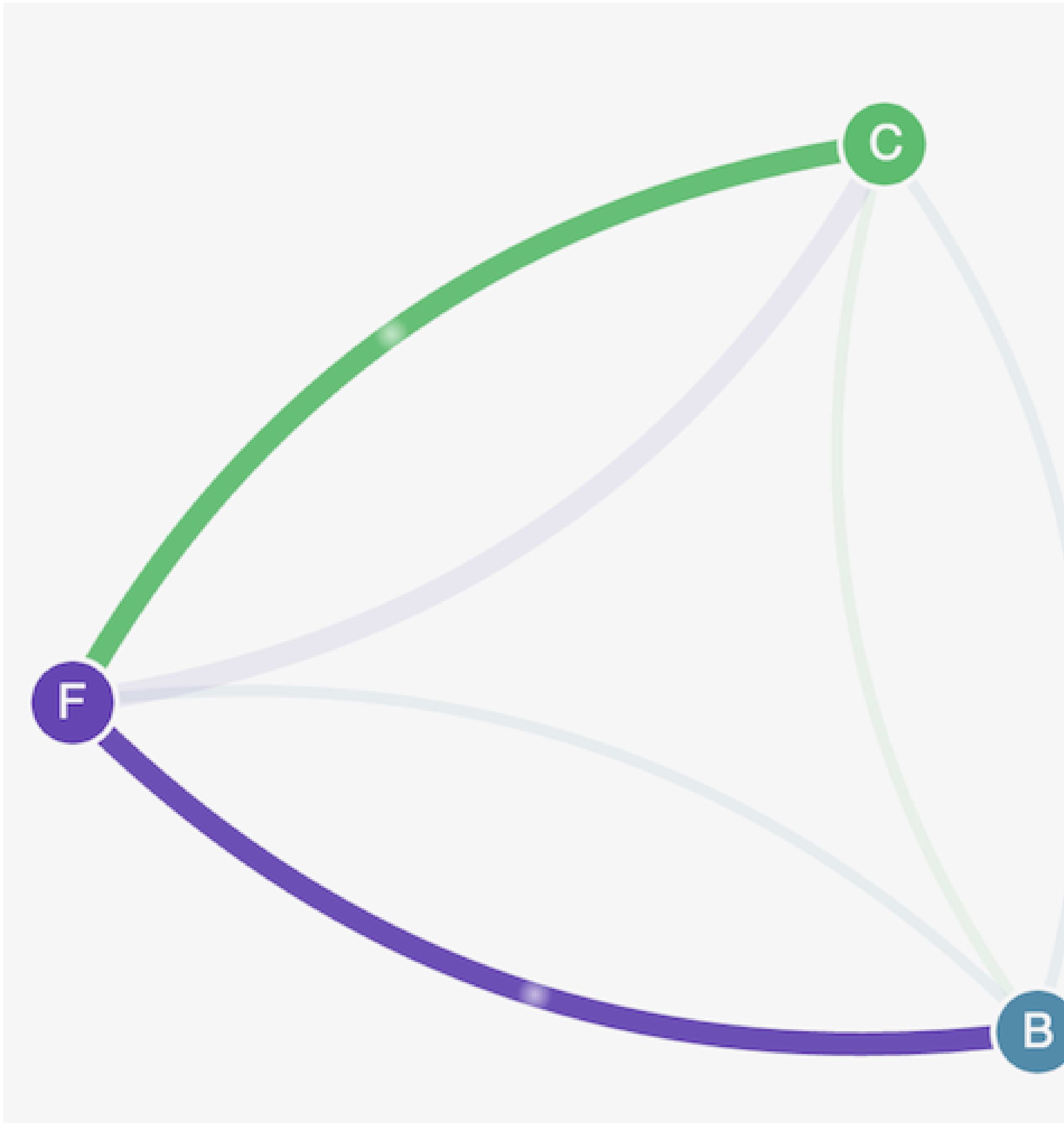


9. Apply the following network policy to allow traffic from the frontend service to the backend service:

```
kubectl apply -f https://docs.projectcalico.org/v3.1/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/v3.1/getting-started/kubernetes/
tutorials/stars-policy/policies/frontend-policy.yaml
```

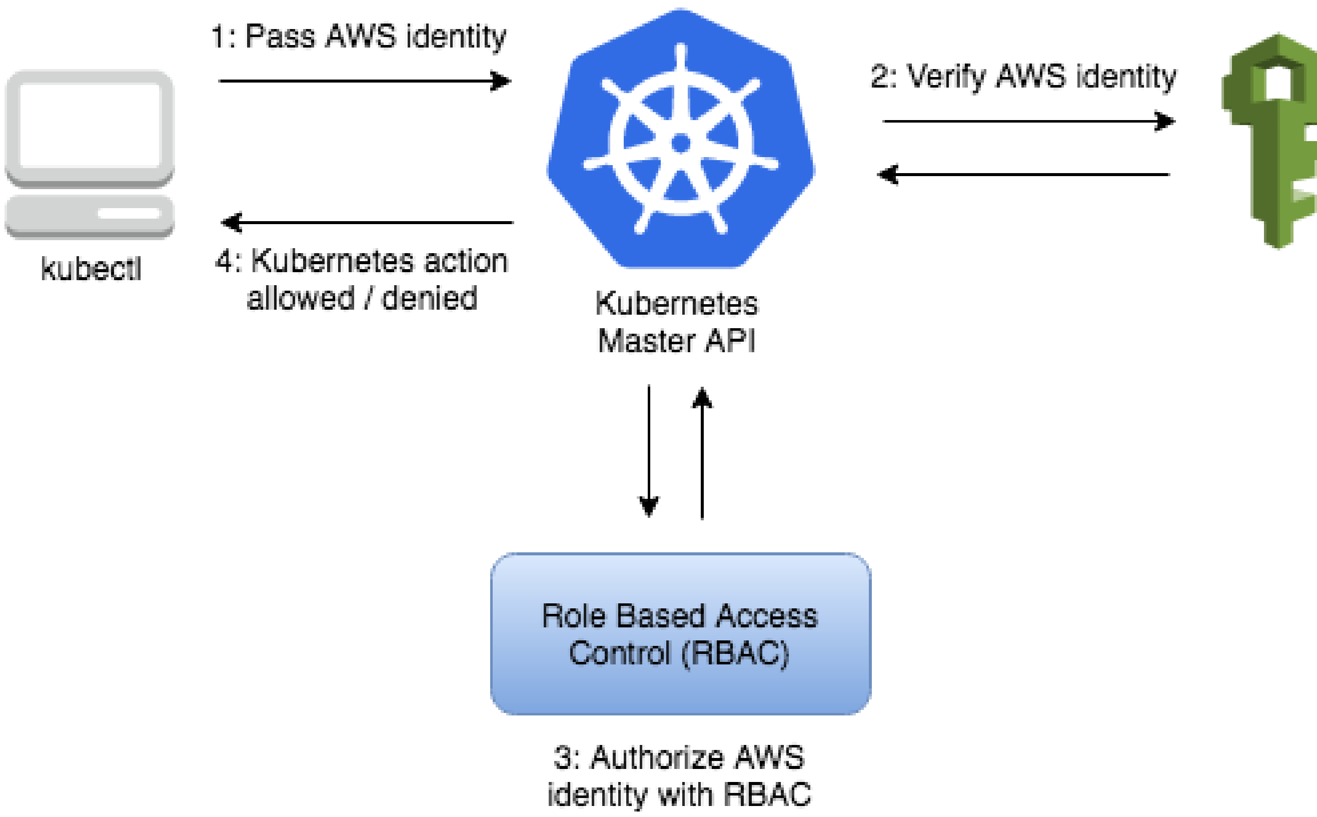


11. (Optional) When you are done with the demo, you can delete its resources with the following command:

```
kubectl delete ns client stars management-ui
```

Managing Cluster Authentication

Amazon EKS uses IAM to provide authentication to your Kubernetes cluster (through 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`

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.

Topics

MacOS

This section helps you to install `kubectl` for MacOS clients.

To install `kubectl` on MacOS

1. Download the Amazon EKS-vended **kubectl** binary from Amazon S3:

   ```
   curl -o kubectl https://amazon-eks.s3-us-west-2.amazonaws.com/1.11.5/2018-12-06/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 MacOS:

      ```
      curl -o kubectl.sha256 https://amazon-eks.s3-us-
      west-2.amazonaws.com/1.11.5/2018-12-06/bin/darwin/amd64/kubectl.sha256
      ```
   b. Check the SHA-256 sum for your downloaded binary.

      ```
      openssl sha -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 $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

This section helps you to install `kubectl` for Linux clients.

To install `kubectl` on Linux

1. Download the Amazon EKS-vended **kubectl** binary from Amazon S3:

```
curl -o kubectl https://amazon-eks.s3-us-west-2.amazonaws.com/1.11.5/2018-12-06/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 Linux:

   ```
   curl -o kubectl.sha256 https://amazon-eks.s3-us-
   west-2.amazonaws.com/1.11.5/2018-12-06/bin/linux/amd64/kubectl.sha256
   ```

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

   ```
   openssl sha -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 $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

This section helps you to install `kubectl` for Windows clients with PowerShell.

To install `kubectl` on Windows

1. Open a PowerShell terminal.
2. Download the Amazon EKS-vended **kubectl** binary from Amazon S3:

```
curl -o kubectl.exe https://amazon-eks.s3-us-
west-2.amazonaws.com/1.11.5/2018-12-06/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 Windows:

```
curl -o kubectl.exe.sha256 https://amazon-eks.s3-us-
west-2.amazonaws.com/1.11.5/2018-12-06/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. Beginning with Kubernetes version 1.10, 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.

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. 60).

To install `aws-iam-authenticator` for Amazon EKS

1. Download and install the `aws-iam-authenticator` binary.

   Amazon EKS vends `aws-iam-authenticator` binaries that you can use, or you can use **go get** to fetch the binary from the AWS IAM Authenticator for Kubernetes project on GitHub for other operating systems.

   - To download and install the Amazon EKS-vended `aws-iam-authenticator` binary:

     a. Download the Amazon EKS-vended `aws-iam-authenticator` binary from Amazon S3:

        - **Linux**: https://amazon-eks.s3-us-west-2.amazonaws.com/1.11.5/2018-12-06/bin/linux/amd64/aws-iam-authenticator

- **MacOS**: https://amazon-eks.s3-us-west-2.amazonaws.com/1.11.5/2018-12-06/bin/darwin/amd64/aws-iam-authenticator
- **Windows**: https://amazon-eks.s3-us-west-2.amazonaws.com/1.11.5/2018-12-06/bin/windows/amd64/aws-iam-authenticator.exe

Use the command below to download the binary, substituting the correct URL for your platform. The example below is for macOS clients.

```
curl -o aws-iam-authenticator https://amazon-eks.s3-us-
west-2.amazonaws.com/1.11.5/2018-12-06/bin/darwin/amd64/aws-iam-authenticator
```

b. (Optional) Verify the downloaded binary with the SHA-256 sum provided in the same bucket prefix, substituting the correct URL for your platform.

   i. Download the SHA-256 sum for your system. The example below is to download the SHA-256 sum for macOS clients.

```
curl -o aws-iam-authenticator.sha256 https://amazon-eks.s3-us-
west-2.amazonaws.com/1.11.5/2018-12-06/bin/darwin/amd64/aws-iam-
authenticator.sha256
```

   ii. Check the SHA-256 sum for your downloaded binary. The example `openssl` command below was tested for macOS and Ubuntu clients. Your operating system may use a different command or syntax to check SHA-256 sums. Consult your operating system documentation if necessary.

```
openssl sha -sha256 aws-iam-authenticator
```

   iii. Compare the generated SHA-256 sum in the command output against your downloaded `aws-iam-authenticator.sha256` file. The two should match.

c. Apply execute permissions to the binary.

```
chmod +x ./aws-iam-authenticator
```

d. 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`.

```
cp ./aws-iam-authenticator $HOME/bin/aws-iam-authenticator && export PATH=
$HOME/bin:$PATH
```

e. Add `$HOME/bin` to your `PATH` environment variable.

- For Bash shells on macOS:

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bash_profile
```

- For Bash shells on Linux:

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
```

f. Test that the `aws-iam-authenticator` binary works.

```
aws-iam-authenticator help
```

- Or, to install the `aws-iam-authenticator` binary from GitHub using **go get**:

a. Install the Go programming language for your operating system if you do not already have **go** installed. For more information, see Install the Go tools in the Go documentation.

b. Use **go get** to install the `aws-iam-authenticator` binary.

```
go get -u -v github.com/kubernetes-sigs/aws-iam-authenticator/cmd/aws-iam-
authenticator
```

Note
If you receive the following error, you must upgrade your Go language to 1.7 or greater. For more information, see Install the Go tools in the Go documentation.

```
package context: unrecognized import path "context" (import path does
 not begin with hostname)
```

c. Add `$HOME/go/bin` to your `PATH` environment variable.

- For Bash shells on macOS:

```
export PATH=$HOME/go/bin:$PATH && echo 'export PATH=$HOME/go/bin:$PATH' >>
 ~/.bash_profile
```

- For Bash shells on Linux:

```
export PATH=$HOME/go/bin:$PATH && echo 'export PATH=$HOME/go/bin:$PATH' >>
 ~/.bashrc
```

d. Test that the `aws-iam-authenticator` binary works.

```
aws-iam-authenticator help
```

2. 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. 65). Otherwise, see Creating an Amazon EKS Cluster (p. 17) 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 IAM Authenticator for Kubernetes with **kubectl** for cluster authentication, which uses the same default AWS credential provider chain as the AWS CLI and AWS SDKs. 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 1.16.73 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 Python 3, or Python 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 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`, then your **kubectl** is not configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 63).

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 the `kubeconfig` code block below into it.

```
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 have the AWS IAM Authenticator for Kubernetes assume a role to perform cluster operations instead of the default AWS credential provider chain, uncomment the `-r` and *`<role-arn>`* lines and substitute an IAM role ARN to use with your user.
7. (Optional) To have the AWS IAM Authenticator for Kubernetes 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`, then your **kubectl** is not configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 63).

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 (for example, for federated users) **that creates the cluster** is automatically granted `system:master` 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*.

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 https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2018-12-10/
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`, then your **kubectl** is not configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 63).

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.
   - **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 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.

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	3

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

Amazon EKS IAM Policies, Roles, and Permissions

By default, IAM users don't have permission to create or modify Amazon EKS resources, or perform tasks using the Amazon EKS API. (This means that they also can't do so using the Amazon EKS console or the AWS CLI.) To allow IAM users to create or modify clusters, you must create IAM policies that grant IAM users permissions to use the specific resources and API actions that they need, and then attach those policies to the IAM users or groups that require those permissions.

When you attach a policy to a user or group of users, it allows or denies the users permission to perform the specified tasks on the specified resources. For more information, see Permissions and Policies in the *IAM User Guide*. For more information about managing and creating custom IAM policies, see Managing IAM Policies.

Likewise, Amazon EKS makes calls to other AWS services on your behalf, so the service must authenticate with your credentials. This authentication is accomplished by creating an IAM role and policy that can provide these permissions and then associating that role with your compute environments when you create them. For more information, see Amazon EKS Service IAM Role (p. 75) and also IAM Roles in the *IAM User Guide*.

Getting Started

An IAM policy must grant or deny permissions to use one or more Amazon EKS actions.

Topics

Policy Structure

The following topics explain the structure of an IAM policy.

Topics

Policy Syntax

An IAM policy is a JSON document that consists of one or more statements. Each statement is structured as follows:

```
{
  "Statement":[{
    "Effect":"effect",
    "Action":"action",
    "Resource":"arn",
    "Condition":{
      "condition":{
        "key":"value"
        }
      }
    }
  ]
}
```

There are various elements that make up a statement:

- **Effect:** The *effect* can be `Allow` or `Deny`. By default, IAM users don't have permission to use resources and API actions, so all requests are denied. An explicit allow overrides the default. An explicit deny overrides any allows.
- **Action**: The *action* is the specific API action for which you are granting or denying permission.
- **Resource**: The resource that's affected by the action. Amazon EKS API operations currently do not support resource level permissions, so you must use the * wildcard to specify that all resources can be affected by the action.
- **Condition**: Conditions are optional. They can be used to control when your policy is in effect.

For more information about example IAM policy statements for Amazon EKS, see Creating Amazon EKS IAM Policies (p. 74).

Actions for Amazon EKS

In an IAM policy statement, you can specify any API action from any service that supports IAM. For Amazon EKS, use the following prefix with the name of the API action: `eks:`. For example: `eks:CreateCluster` and `eks:DeleteCluster`.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": ["eks:action1", "eks:action2"]
```

You can also specify multiple actions using wildcards. For example, you can specify all actions whose name begins with the word "Describe" as follows:

```
"Action": "eks:Describe*"
```

To specify all Amazon EKS API actions, use the * wildcard as follows:

```
"Action": "eks:*"
```

Checking That Users Have the Required Permissions

After you've created an IAM policy, we recommend that you check whether it grants users the permissions to use the particular API actions and resources they need before you put the policy into production.

First, create an IAM user for testing purposes, and then attach the IAM policy that you created to the test user. Then, make a request as the test user. You can make test requests in the console or with the AWS CLI.

Note
You can also test your policies with the IAM Policy Simulator. For more information on the policy simulator, see Working with the IAM Policy Simulator in the *IAM User Guide*.

If the policy doesn't grant the user the permissions that you expected, or is overly permissive, you can adjust the policy as needed and retest until you get the desired results.

Important
It can take several minutes for policy changes to propagate before they take effect. Therefore, we recommend that you allow five minutes to pass before you test your policy updates.

If an authorization check fails, the request returns an encoded message with diagnostic information. You can decode the message using the `DecodeAuthorizationMessage` action. For more information, see DecodeAuthorizationMessage in the *AWS Security Token Service API Reference*, and decode-authorization-message in the *AWS CLI Command Reference*.

Creating Amazon EKS IAM Policies

You can create specific IAM policies to restrict the calls and resources that users in your account have access to, and then attach those policies to IAM users.

When you attach a policy to a user or group of users, it allows or denies the users permission to perform the specified tasks on the specified resources. For more information, see Permissions and Policies in the *IAM User Guide*. For more information about managing and creating custom IAM policies, see Managing IAM Policies.

If your IAM user does not have administrative privileges, you must explicitly add permissions for that user to call the Amazon EKS API operations.

To create an IAM policy for an Amazon EKS admin user

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**, **Create policy**.
3. On the **JSON** tab, paste the following policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:*"
            ],
            "Resource": "*"
        }
    ]
}
```

4. Choose **Review policy**.
5. In the **Name** field, type your own unique name, such as `AmazonEKSAdminPolicy`.
6. Choose **Create Policy** to finish.

To attach an IAM policy to a user

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users** and then choose the user you would like to attach the policy to.
3. Choose **Permissions**, **Add permissions**.
4. In the **Grant permissions** section, choose **Attach existing policies directly**.

5. Select the custom policy that you created in the previous procedure and choose **Next: Review**.
6. Review your details and choose **Add permissions** to finish.

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 use the service, you must create an IAM role with the following IAM policies:

- `AmazonEKSServicePolicy`
- `AmazonEKSClusterPolicy`

Check for an Existing `AWSServiceRoleForAmazonEKS` 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 `AWSServiceRoleForAmazonEKS` 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 `AWSServiceRoleForAmazonEKS`. If the role does not exist, see Creating the `AWSServiceRoleForAmazonEKS` role (p. 75) 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 `AWSServiceRoleForAmazonEKS` 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: Review**.
5. 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

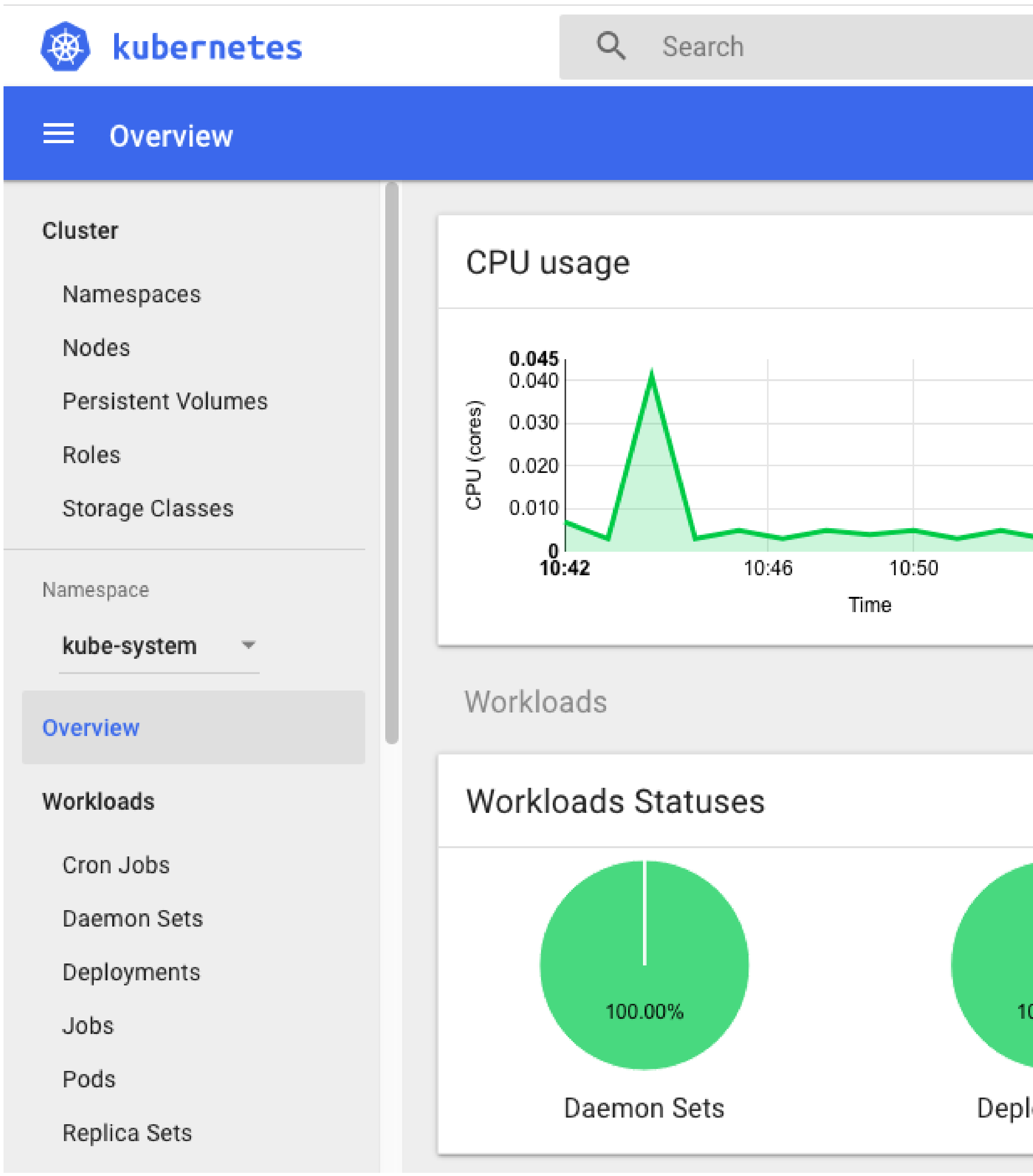
1. Save the following AWS CloudFormation template to a text file on your local system.

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


Resources:

  AWSServiceRoleForAmazonEKS:
    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 EKS will use to create AWS resources for Kubernetes
 clusters
    Value: !GetAtt AWSServiceRoleForAmazonEKS.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 **Choose a template**, select **Upload a template to Amazon S3**, and then choose **Browse**.
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 **Options** page, choose **Next**.
8. On the **Review** page, review your information, acknowledge that the stack might create IAM resources, and then choose **Create**.

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. 45).
- You are using a **kubectl** client that is configured to communicate with your Amazon EKS cluster (p. 9).

Step 1: Deploy the Dashboard

Use the following steps to deploy the Kubernetes dashboard, `heapster`, and the `influxdb` backend for CPU and memory metrics to your cluster.

To deploy the Kubernetes dashboard

1. Deploy the Kubernetes dashboard to your cluster:

   ```
   kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/
   deploy/recommended/kubernetes-dashboard.yaml
   ```

   Output:

   ```
   secret "kubernetes-dashboard-certs" created
   serviceaccount "kubernetes-dashboard" created
   role "kubernetes-dashboard-minimal" created
   rolebinding "kubernetes-dashboard-minimal" created
   deployment "kubernetes-dashboard" created
   service "kubernetes-dashboard" created
   ```
2. Deploy `heapster` to enable container cluster monitoring and performance analysis on your cluster:

   ```
   kubectl apply -f https://raw.githubusercontent.com/kubernetes/heapster/master/deploy/
   kube-config/influxdb/heapster.yaml
   ```

   Output:

   ```
   serviceaccount "heapster" created
   deployment "heapster" created
   service "heapster" created
   ```
3. Deploy the `influxdb` backend for `heapster` to your cluster:

   ```
   kubectl apply -f https://raw.githubusercontent.com/kubernetes/heapster/master/deploy/
   kube-config/influxdb/influxdb.yaml
   ```

   Output:

   ```
   deployment "monitoring-influxdb" created
   service "monitoring-influxdb" created
   ```
4. Create the `heapster` cluster role binding for the dashboard:

   ```
   kubectl apply -f https://raw.githubusercontent.com/kubernetes/heapster/master/deploy/
   kube-config/rbac/heapster-rbac.yaml
   ```

   Output:

```
clusterrolebinding "heapster" created
```

Step 2: Create an `eks-admin` Service Account and Cluster Role Binding

By default, the Kubernetes dashboard user has limited permissions. In this section, you create an `eks-admin` service account and cluster role binding that you can use to securely connect to the dashboard with admin-level permissions. For more information, see Managing Service Accounts in the Kubernetes documentation.

To create the `eks-admin` service account and cluster role binding

Important
The example service account created with this procedure has full `cluster-admin` (superuser) privileges on the cluster. For more information, see Using RBAC Authorization in the Kubernetes documentation.

1. Create a file called `eks-admin-service-account.yaml` with the text below. This manifest defines a service account and cluster role binding called `eks-admin`.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: eks-admin
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/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 3: Connect to the Dashboard

Now that the Kubernetes dashboard is deployed to your cluster, and you have an administrator service account that you can use to view and control your cluster, you can connect to the dashboard with that service account.

To connect to the Kubernetes dashboard

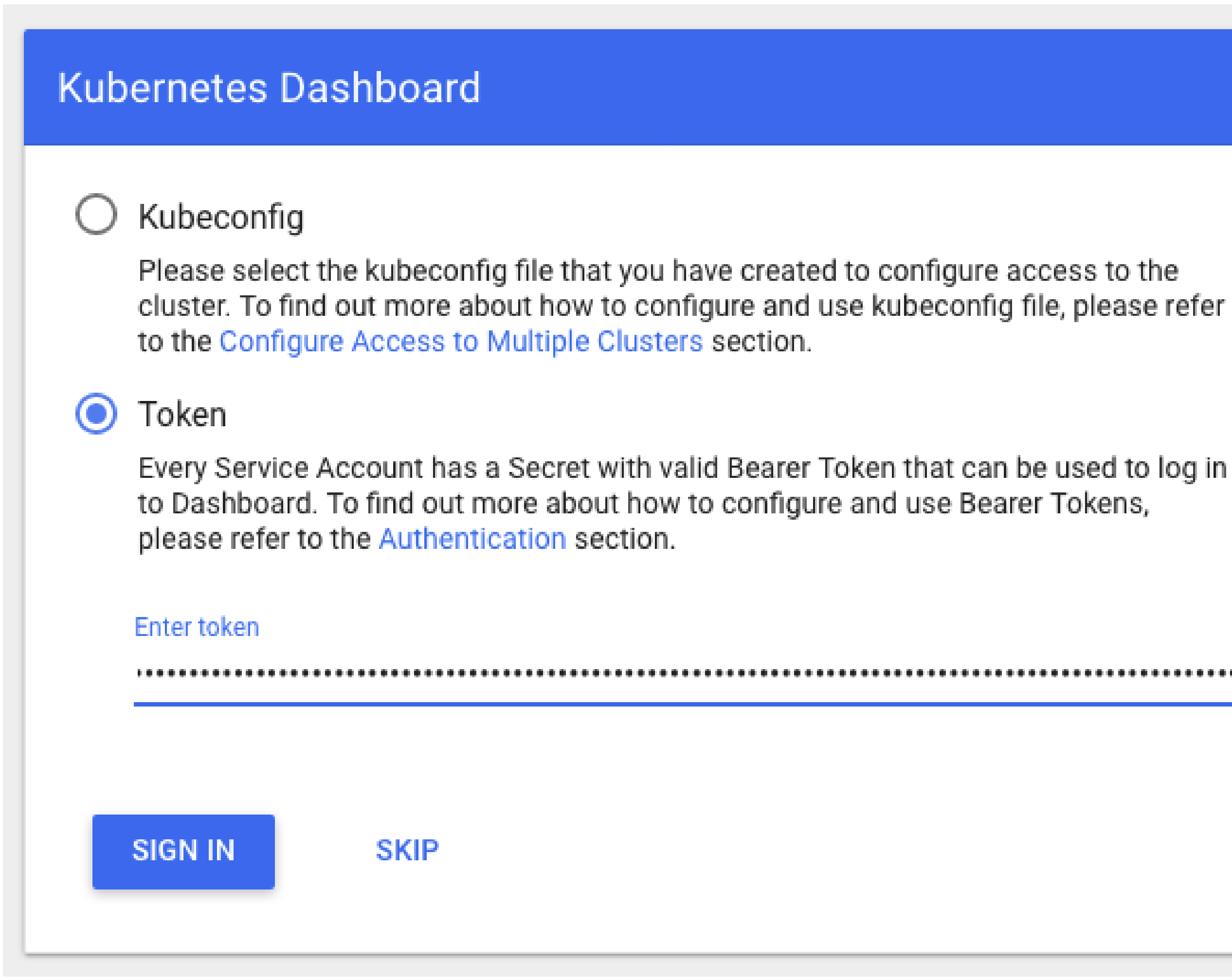
1. Retrieve an authentication token for the `eks-admin` service account. Copy the *<authentication_token>* value from the output. You use this token to connect to the dashboard.

   ```
   kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep eks-
   admin | awk '{print $1}')
   ```

   Output:

   ```
   Name:         eks-admin-token-b5zv4
   Namespace:    kube-system
   Labels:       <none>
   Annotations:  kubernetes.io/service-account.name=eks-admin
                 kubernetes.io/service-account.uid=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. Open the following link with a web browser to access the dashboard endpoint: http://localhost:8001/api/v1/namespaces/kube-system/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 4: 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.

Tutorial: Creating a VPC with Public and Private Subnets for Your Amazon EKS Cluster

This tutorial guides you through creating a VPC with 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.

Topics

Step 1: Create an Elastic IP Address for Your NAT Gateway

Worker nodes in private subnets require a NAT gateway for outbound internet access. A NAT gateway requires an Elastic IP address in your public subnet, but the VPC wizard does not create one for you. Create the Elastic IP address before running the VPC wizard.

To create an Elastic IP address

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the left navigation pane, choose **Elastic IPs**.
3. Choose **Allocate new address**, **Allocate**, **Close**.
4. Note the **Allocation ID** for your newly created Elastic IP address; you enter this later in the VPC wizard.

Step 2: Run the VPC Wizard

The VPC wizard automatically creates and configures most of your VPC resources for you.

To run the VPC wizard

1. In the left navigation pane, choose **VPC Dashboard**.
2. Choose **Start VPC Wizard**, **VPC with Public and Private Subnets**, **Select**.

3. For **VPC name**, give your VPC a unique name.
4. For **Elastic IP Allocation ID**, choose the ID of the Elastic IP address that you created earlier.
5. Choose **Create VPC**.
6. When the wizard is finished, choose **OK**. Note the Availability Zone in which your VPC subnets were created. Your additional subnets should be created in a different Availability Zone.

Step 3: Create Additional Subnets

The wizard creates a VPC with a single public and a single private subnet in a single Availability Zone. For greater availability, you should create at least one more of each subnet type in a different Availability Zone so that your VPC has both public and private subnets across two zones.

To create an additional private subnet

1. In the left navigation pane, choose **Subnets**, **Create Subnet**.
2. For **Name tag**, enter a name for your subnet, such as **Private subnet**.
3. For **VPC**, choose the VPC that you created earlier.
4. For **Availability Zone**, choose a different zone than your original subnets in the VPC.
5. For **IPv4 CIDR block**, enter a valid CIDR block. For example, the wizard creates CIDR blocks in 10.0.0.0/24 and 10.0.1.0/24 by default. You could use **10.0.3.0/24** for your second private subnet.
6. Choose **Yes, Create**.

To create an additional public subnet

1. In the left navigation pane, choose **Subnets**, **Create Subnet**.
2. For **Name tag**, enter a name for your subnet, such as **Public subnet**.
3. For **VPC**, choose the VPC that you created earlier.
4. For **Availability Zone**, choose the same zone as the additional private subnet that you created in the previous procedure.
5. For **IPv4 CIDR block**, enter a valid CIDR block. For example, the wizard creates CIDR blocks in 10.0.0.0/24 and 10.0.1.0/24 by default. You could use **10.0.2.0/24** for your second public subnet.
6. Choose **Yes, Create**.
7. Select the public subnet that you just created and choose **Route Table**, **Edit**.
8. By default, the private route table is selected. Choose the other available route table so that the **0.0.0.0/0** destination is routed to the internet gateway (**igw-*xxxxxxxx***) and choose **Save**.
9. With your second public subnet still selected, choose **Subnet Actions**, **Modify auto-assign IP settings**.
10. Select **Enable auto-assign public IPv4 address** and choose **Save**, **Close**.

Step 4: Tag your Private Subnets

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`

To tag your private subnets

1. Select one of your private subnets and choose **Tags**, **Add/Edit Tags**.
2. Choose **Create Tag**.
3. For **Key**, enter `kubernetes.io/role/internal-elb`.
4. For **Value**, enter `1`.
5. Choose **Save**, and repeat this procedure for any additional private subnets in your VPC.

Step 5: Create a Control Plane Security Group

When you create an Amazon EKS cluster, your cluster control plane creates elastic network interfaces in your subnets to enable communication with the worker nodes. You should create a security group that is dedicated to your Amazon EKS cluster control plane, so that you can apply inbound and outbound rules to govern what traffic is allowed across that connection. When you create the cluster, you specify this security group, and that is applied to the elastic network interfaces that are created in your subnets.

The worker node AWS CloudFormation template used in Step 3: Launch and Configure Amazon EKS Worker Nodes (p. 10) creates a worker node security group, and it applies the necessary rules to allow communication with the control plane automatically, but you must specify the control plane security group when you create a stack from that template.

To create a control plane security group

1. In the left navigation pane, for **Filter by VPC**, select your VPC and choose **Security Groups**, **Create Security Group**.

   Note
   If you don't see your new VPC here, refresh the page to pick it up.
2. Fill in the following fields and choose **Yes, Create**:

   - For **Name tag**, provide a name for your security group. For example, `<cluster-name>-control-plane`.
   - For **Description**, provide a description of your security group to help you identify it later.
   - For **VPC**, choose the VPC that you are using for your Amazon EKS cluster.

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. 3) section and use these subnets and security groups for your cluster.

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"
}
```

Amazon EKS Shared Responsibility Model

Security and compliance is a shared responsibility between AWS and the customer. This shared model helps relieve your operational burden as AWS operates, manages, and controls the components from the host operating system and virtualization layer down to the physical security of the facilities in which the service operates. For more information, see the official AWS Shared Responsibility Model detail page.

AWS is responsible for protecting the infrastructure that runs all services offered in the AWS Cloud. This infrastructure is composed of the hardware, software, networking, and facilities that run AWS Cloud services. For Amazon EKS, AWS is responsible for the Kubernetes control plane, which includes the control plane nodes and `etcd` database.

You assume responsibility and management of the following:

- 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

For more information about using AWS IAM with Amazon EKS, see Amazon EKS IAM Policies, Roles, and Permissions (p. 72).

For more information about managing native Kubernetes Role Based Access Control (RBAC) with Amazon EKS clusters, see Managing Cluster Authentication (p. 60).

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. For more information, see Logging Amazon EKS API Calls with AWS CloudTrail (p. 86).

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. 63).

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 Worker Nodes (p. 30).
- 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.

`hostname doesn't match`

Your system's Python version must be Python 3, or Python 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.

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`.

Document History for Amazon EKS

The following table describes the major updates and new features for the Amazon EKS User Guide. We also update the documentation frequently to address the feedback that you send us.

update-history-change	update-history-description	update-history-date
Amazon EKS region expansion (p. 91)	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. 91)	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. 91)	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. 91)	Initial documentation for service launch	June 5, 2018

AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the *AWS General Reference*.