User Guide

Amazon EKS

Amazon EKS: User Guide

Table of Contents

Help improve this page

To contribute to this user guide, choose the **Edit this page on GitHub** link that is located in the right pane of every page.

What is Amazon EKS?

Amazon Elastic Kubernetes Service (Amazon EKS) is a managed Kubernetes service that eliminates the need to operate and maintain the availability and scalability of Kubernetes clusters in Amazon Web Services (AWS) and in your own data centers. Kubernetes is an open source system that automates the management, scaling, and deployment of containerized applications. To get started, see the *Quickstart* page in the Amazon EKS User Guide.

Features of Amazon EKS

Fully Managed Kubernetes

Amazon EKS provides a scalable and highly-available Kubernetes control plane running across multiple AWS Availability Zones (AZs). Amazon EKS automatically manages availability and scalability of Kubernetes API servers and etcd persistence layer. Amazon EKS runs the Kubernetes control plane across multiple AZs to ensure high availability, and automatically detects and replaces unhealthy control plane nodes.

Amazon EKS Auto Mode fully automates Kubernetes cluster infrastructure management for compute, storage, and networking on AWS. It simplifies Kubernetes management by automatically provisioning infrastructure, selecting optimal compute instances, dynamically scaling resources, continuously optimizing costs, patching operating systems, and integrating with AWS security services.

Kubernetes Compatibility and Support

Amazon EKS runs upstream Kubernetes and is certified Kubernetes-conformant, so you can use all the existing plug-ins 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 refactoring your code. Amazon EKS supports Kubernetes versions longer than they are supported upstream, with standard support for Kubernetes minor versions for 14 months from the time they are released in Amazon EKS, and extended support for Kubernetes minor versions for an additional 12 months of support (26 total months per version). See the section called “Kubernetes versions” for more information.

Machine Learning

Amazon EKS has become a cornerstone for deploying and managing AI/ML workloads in the cloud. With its ability to handle complex, resource-intensive tasks, Amazon EKS provides a scalable and flexible foundation for running AI/ML models, making it an ideal choice for organizations aiming to harness the full potential of machine learning. Whether you're training large language models that require vast amounts of compute power or deploying inference pipelines that need to handle unpredictable traffic patterns, Amazon EKS scales up and down efficiently, optimizing resource use and cost. Amazon EKS supports a wide range of compute options including GPU-powered instances and AWS Neuron, allowing for high-performance training and low-latency inference, ensuring that models run efficiently in production environments. For more information, see *Machine Learning on EKS*.

Hybrid Deployments

You can use the same Amazon EKS clusters to run nodes on AWS-hosted infrastructure in AWS Regions, AWS Local Zones, AWS Wavelength Zones, or in your own on-premises environments with AWS Outposts and Amazon EKS Hybrid Nodes. AWS Outposts is AWS-managed infrastructure that you run in your data centers or co-location facilities, whereas Amazon EKS Hybrid Nodes runs on virtual machines or bare metal infrastructure that you manage in your on-premises or edge environments. If you need to run in isolated or air-gapped environments, you can use Amazon EKS Anywhere, which is AWS-supported Kubernetes management software that runs on infrastructure you manage. With Amazon EKS Anywhere, you are responsible for cluster lifecycle operations and maintenance of your Amazon EKS Anywhere clusters. The *Amazon EKS Connector* can be used to view any Kubernetes cluster and their resources in the Amazon EKS console. *Amazon EKS Distro* is the AWS distribution of the underlying Kubernetes components that power all Amazon EKS offerings.

Compute

You can use the full range of Amazon EC2 instance types and AWS innovations such as Nitro and Graviton with Amazon EKS for you to optimize the compute for your workloads. You can use on-demand or Spot instances and your savings plans with compute you use with your Amazon EKS clusters. See *Manage compute* for more information.

Networking

Amazon EKS integrates with Amazon VPC allowing you to use your own Amazon VPC security groups and network access control lists (ACLs) with Amazon EKS clusters. Amazon EKS provides the Amazon VPC container network interface (CNI), allowing Kubernetes pods to receive IP addresses directly from the VPC. Amazon EKS supports IPv4 and IPv6 for workloads and dual-stack endpoints for the Amazon EKS APIs and Kubernetes API. You can use Application Load Balancers (ALB) and Network Load Balancers (NLB) managed by the AWS Load Balancer Controller for application ingress and load balancing. You can also use Amazon VPC Lattice, a managed application networking service built directly into the AWS networking infrastructure, for cross-cluster connectivity with standard Kubernetes semantics in a simple and consistent manner. See *Configure networking* for more information.

Security

Amazon EKS integrates with AWS Identity and Access Management (IAM) for you to secure your clusters and applications. Amazon EKS makes it easy to map AWS IAM permissions to Kubernetes Role Based Access Control (RBAC). You can use AWS IAM for cluster authentication and authorization with Amazon EKS Cluster Access Management, for access and permissions of operational software running on your clusters, and for granular application access to other AWS services with Amazon EKS Pod Identity. Amazon EKS is certified by multiple compliance programs for regulated and sensitive applications. Amazon EKS is compliant with SOC, PCI, ISO, FedRAMP-Moderate, IRAP, C5, K-ISMS, ENS High, OSPAR, HITRUST CSF, and is a HIPAA eligible service. See *Manage access* for more information.

Observability

Amazon EKS integrates with AWS Managed Service for Prometheus (AMP), Amazon CloudWatch, Amazon CloudTrail, and Amazon GuardDuty for monitoring, logging, and auditing capabilities. You can also view performance insights for your Amazon EKS clusters directly in the Amazon EKS console. You can use AMP agent-less scrapers or the AWS Distro for OpenTelemetry add-on to monitor and collect logs for your clusters, infrastructure, and applications. You can use Amazon CloudWatch Container Insights, the CloudWatch Observability Agent add-on, and Amazon EKS control plane logging to monitor, collect logs, and analyze issues with your clusters, infrastructure, and applications. Amazon EKS also integrates with Amazon CloudTrail for auditing cluster API activity, and Amazon GuardDuty for audit log threat analysis and runtime threat detection. See *Monitor clusters* for more information.

Storage

You can use a range of AWS storage services with Amazon EKS for the storage needs of your applications. Through an AWS-supported breadth of Container Storage Interface (CSI) drivers, you can easily use Amazon EBS, Amazon S3, Amazon EFS, Amazon FSX, and Amazon File Cache for the storage needs of your applications running on Amazon EKS. See *Store app data* for more information.

Add-ons

Amazon EKS offers a curated set of AWS-vended Kubernetes software, also known as Amazon EKS add-ons, that provide key operational capabilities for Kubernetes clusters and integration with various AWS services for cluster and pod networking, load balancing, storage, observability, and security. Amazon EKS provides a unified management experience for finding, selecting, installing, managing, and configuring third-party Kubernetes operational software (add-ons) from independent software vendors on Amazon EKS clusters. See the section called "Amazon EKS add-ons" for more information.

Management interfaces

Amazon EKS supports a range of interfaces to provision, manage, and maintain clusters including the Amazon EKS console, Amazon EKS API/SDKs, CDK, AWS CLI, eksctl CLI, AWS CloudFormation, and Terraform. You can also use AWS Controllers for Kubernetes (ACK) to provision and manage AWS services from within your Kubernetes environment using Kubernetes interfaces. ACK makes it simple to build scalable and highly available Kubernetes applications utilizing AWS services. See *Get started* for more information.

Operating systems

Amazon EKS supports a range of operating systems and you can use pre-built, Amazon EKS-optimized Amazon Machine Images (AMIs) for the base images of your compute nodes. Amazon EKS maintains optimized images for Amazon Linux 2, Amazon Linux 2023, Bottlerocket, Windows, and there are Ubuntu images maintained by Canonical. You can also use your own custom AMIs for other operating system variants. The Amazon EKS AMIs for Amazon Linux have built-in support for NVIDIA and AWS Neuron accelerated instance types. See the section called "Pre-built optimized AMIs" for more information.

Amazon EKS Pricing

Amazon EKS has per cluster pricing based on Kubernetes cluster version support, pricing for Amazon EKS Auto Mode, and per vCPU pricing for Amazon EKS Hybrid Nodes. When using Amazon EKS, you pay separately for the AWS resources you use to run your applications on Kubernetes worker nodes. For example, if you are running Kubernetes worker nodes as Amazon EC2 instances with Amazon EBS volumes and public IPv4 addresses, you are charged for the instance capacity through Amazon EC2, the volume capacity through Amazon EBS, and the IPv4 address through Amazon VPC. Visit the respective pricing pages of the AWS services you are using with your Kubernetes applications for detailed pricing information.

- For Amazon EKS cluster, Amazon EKS Auto Mode, and Amazon EKS Hybrid Nodes pricing, see Amazon EKS Pricing.
- For Amazon EC2 pricing, see Amazon EC2 On-Demand Pricing and Amazon EC2 Spot Pricing.
- For AWS Fargate pricing, see AWS Fargate Pricing.
- You can use your savings plans for compute used in Amazon EKS clusters. For more information, see Pricing with Savings Plans.

Common use cases in Amazon EKS

Amazon EKS offers robust managed Kubernetes services on AWS, designed to optimize containerized applications. The following are a few of the most common use cases of Amazon EKS, helping you leverage its strengths for your specific needs.

Deploying high-availability applications

Using Elastic Load Balancing, you can make sure that your applications are highly available across multiple Availability Zones.

Building microservices architectures

Use Kubernetes service discovery features with AWS Cloud Map or Amazon VPC Lattice to build resilient systems.

Automating software release process

Manage continuous integration and continuous deployment (CICD) pipelines that simplify the process of automated building, testing, and deployment of applications.

Running serverless applications

Use AWS Fargate with Amazon EKS to run serverless applications. This means you can focus solely on application development, while Amazon EKS and Fargate handle the underlying infrastructure.

Executing machine learning workloads

Amazon EKS is compatible with popular machine learning frameworks such as TensorFlow, MXNet, and PyTorch. With GPU support, you can handle even complex machine learning tasks effectively.

Deploying consistently on premises and in the cloud

To simplify running Kubernetes in on-premises environments, you can use the same Amazon EKS clusters, features, and tools to run self-managed nodes on AWS Outposts or can use Amazon EKS Hybrid Nodes with your own infrastructure. For self-contained, air-gapped environments, you can use Amazon EKS Anywhere to automate Kubernetes cluster lifecycle management on your own infrastructure.

Running cost-effective batch processing and big data workloads

Utilize Spot Instances to run your batch processing and big data workloads such as Apache Hadoop and Spark, at a fraction of the cost. This lets you take advantage of unused Amazon EC2 capacity at discounted prices.

Securing application and ensuring compliance

Implement strong security practices and maintain compliance with Amazon EKS, which integrates with AWS security services such as AWS Identity and Access Management (IAM), Amazon Virtual Private Cloud (Amazon VPC), and AWS Key Management Service (AWS KMS). This ensures data privacy and protection as per industry standards.

Amazon EKS architecture

Amazon EKS aligns with the general cluster architecture of Kubernetes. For more information, see Kubernetes Components in the Kubernetes documentation. The following sections summarize some extra architecture details for Amazon EKS.

Control plane

Amazon EKS ensures every cluster has its own unique Kubernetes control plane. This design keeps each cluster’s infrastructure separate, with no overlaps between clusters or AWS accounts. The setup includes:

Distributed components

The control plane positions at least two API server instances and three etcd instances across three AWS Availability Zones within an AWS Region.

Optimal performance

Amazon EKS actively monitors and adjusts control plane instances to maintain peak performance.

Resilience

If a control plane instance falters, Amazon EKS quickly replaces it, using different Availability Zone if needed.

Consistent uptime

By running clusters across multiple Availability Zones, a reliable API server endpoint availability Service Level Agreement (SLA) is achieved.

Amazon EKS uses Amazon Virtual Private Cloud (Amazon VPC) to limit traffic between control plane components within a single cluster. Cluster components can't view or receive communication from other clusters or AWS accounts, except when authorized by Kubernetes role-based access control (RBAC) policies.

Compute

In addition to the control plane, an Amazon EKS cluster has a set of worker machines called nodes. Selecting the appropriate Amazon EKS cluster node type is crucial for meeting your specific requirements and optimizing resource utilization. Amazon EKS offers the following primary node types:

EKS Auto Mode

EKS Auto Mode extends AWS management beyond the control plane to include the data plane, automating cluster infrastructure management. It integrates core Kubernetes capabilities as built-in components, including compute autoscaling, networking, load balancing, DNS, storage, and GPU support. EKS Auto Mode dynamically manages nodes based on workload demands, using immutable AMIs with enhanced security features. It automates updates and upgrades while respecting Pod Disruption Budgets, and includes managed components that would otherwise require add-on management. This option is ideal for users who want to leverage AWS expertise for day-to-day operations, minimize operational overhead, and focus on application development rather than infrastructure management.

AWS Fargate

Fargate is a serverless compute engine for containers that eliminates the need to manage the underlying instances. With Fargate, you specify your application's resource needs, and AWS automatically provisions, scales, and maintains the infrastructure. This option is ideal for users who prioritize ease-of-use and want to concentrate on application development and deployment rather than managing infrastructure.

Karpenter

Karpenter is a flexible, high-performance Kubernetes cluster autoscaler that helps improve application availability and cluster efficiency. Karpenter launches right-sized compute resources in response to changing application load. This option can provision just-in-time compute resources that meet the requirements of your workload.

Managed node groups

Managed node groups are a blend of automation and customization for managing a collection of Amazon EC2 instances within an Amazon EKS cluster. AWS takes care of tasks like patching, updating, and scaling nodes, easing operational aspects. In parallel, custom `kubelet` arguments are supported, opening up possibilities for advanced CPU and memory management policies. Moreover, they enhance security via AWS Identity and Access Management (IAM) roles for service accounts, while curbing the need for separate permissions per cluster.

Self-managed nodes

Self-managed nodes offer full control over your Amazon EC2 instances within an Amazon EKS cluster. You are in charge of managing, scaling, and maintaining the nodes, giving you total control over the underlying infrastructure. This option is suitable for users who need granular control and customization of their nodes and are ready to invest time in managing and maintaining their infrastructure.

Amazon EKS Hybrid Nodes

With Amazon EKS Hybrid Nodes, you can use your on-premises and edge infrastructure as nodes in Amazon EKS clusters. Amazon EKS Hybrid Nodes unifies Kubernetes management across environments and offloads Kubernetes control plane management to AWS for your on-premises and edge applications.

Kubernetes concepts

Amazon Elastic Kubernetes Service (Amazon EKS) is an AWS managed service based on the open source Kubernetes project. While there are things you need to know about how the Amazon EKS service integrates with AWS Cloud (particularly when you first create an Amazon EKS cluster), once it's up and running, you use your Amazon EKS cluster in much that same way as you would any other Kubernetes cluster. So to begin managing Kubernetes clusters and deploying workloads, you need at least a basic understanding of Kubernetes concepts.

This page divides Kubernetes concepts into three sections: the section called "Why Kubernetes?", the section called "Clusters", and the section called "Workloads". The first section describes the value of running a Kubernetes service, in particular as a managed service like Amazon EKS. The Workloads section covers how Kubernetes applications are built, stored, run, and managed. The Clusters section lays out the different components that make up Kubernetes clusters and what your responsibilities are for creating and maintaining Kubernetes clusters.

Topics

As you go through this content, links will lead you to further descriptions of Kubernetes concepts in both Amazon EKS and Kubernetes documentation, in case you want to take deep dives into any of the topics we cover here. For details about how Amazon EKS implements Kubernetes control plane and compute features, see the section called "Architecture".

Why Kubernetes?

Kubernetes was designed to improve availability and scalability when running mission-critical, production-quality containerized applications. Rather than just running Kubernetes on a single machine (although that is possible), Kubernetes achieves those goals by allowing you to run applications across sets of computers that can expand or contract to meet demand. Kubernetes includes features that make it easier for you to:

- Deploy applications on multiple machines (using containers deployed in Pods)
- Monitor container health and restart failed containers
- Scale containers up and down based on load
- Update containers with new versions
- Allocate resources between containers
- Balance traffic across machines

Having Kubernetes automate these types of complex tasks allows an application developer to focus on building and improving their application workloads, rather than worrying about infrastructure. The developer typically creates configuration files, formatted as YAML files, that describe the desired state of the application. This could include which containers to run, resource limits, number of Pod replicas, CPU/memory allocation, affinity rules, and more.

Attributes of Kubernetes

To achieve its goals, Kubernetes has the following attributes:

- **Containerized** — Kubernetes is a container orchestration tool. To use Kubernetes, you must first have your applications containerized. Depending on the type of application, this could be as a set of *microservices,* as batch jobs or in other forms. Then, your applications can take advantage of a Kubernetes workflow that encompasses a huge ecosystem of tools, where containers can be stored as images in a container registry, deployed to a Kubernetes cluster, and run on an available node. You can build and test individual containers on your local computer with Docker or another container runtime, before deploying them to your Kubernetes cluster.
- **Scalable** — If the demand for your applications exceeds the capacity of the running instances of those applications, Kubernetes is able to scale up. As needed, Kubernetes can tell if applications require more CPU or memory and respond by either automatically expanding available capacity or using more of existing capacity. Scaling can be done at the Pod level, if there is enough compute available to just run more instances of the application (horizontal Pod autoscaling), or at the node level, if more nodes need to be brought up to handle the increased capacity (Cluster Autoscaler or Karpenter). As capacity is no longer needed, these services can delete unnecessary Pods and shut down unneeded nodes.
- **Available** — If an application or node becomes unhealthy or unavailable, Kubernetes can move running workloads to another available node. You can force the issue by simply deleting a running instance of a workload or node that's running your workloads. The bottom line here is that workloads can be brought up in other locations if they can no longer run where they are.
- **Declarative** — Kubernetes uses active reconciliation to constantly check that the state that you declare for your cluster matches the actual state. By applying Kubernetes objects to a cluster, typically through YAML-formatted configuration files, you can, for example, ask to start up the workloads you want to run on your cluster. You can later change the configurations to do something like use a later version of a container or allocate more memory. Kubernetes will do what it needs to do to establish the desired state. This can include bringing nodes up or down, stopping and restarting workloads, or pulling updated containers.
- **Composable** — Because an application typically consists of multiple components, you want to be able to manage a set of these components (often represented by multiple containers) together. While Docker Compose offers a way to do this directly with Docker, the Kubernetes

Kompose command can help you do that with Kubernetes. See Translate a Docker Compose File to Kubernetes Resources for an example of how to do this.

- **Extensible** — Unlike proprietary software, the open source Kubernetes project is designed to be open to you extending Kubernetes any way that you like to meet your needs. APIs and configuration files are open to direct modifications. Third-parties are encouraged to write their own Controllers, to extend both infrastructure and end-user Kubernetes featues. Webhooks let you set up cluster rules to enforce policies and adapt to changing conditions. For more ideas on how to extend Kubernetes clusters, see Extending Kubernetes.
- **Portable** — Many organizations have standardized their operations on Kubernetes because it allows them to manage all of their application needs in the same way. Developers can use the same pipelines to build and store containerized applications. Those applications can then be deployed to Kubernetes clusters running on-premises, in clouds, on point-of-sales terminals in restaurants, or on IOT devices dispersed across company's remote sites. Its open source nature makes it possible for people to develop these special Kubernetes distributions, along will tools needed to manage them.

Managing Kubernetes

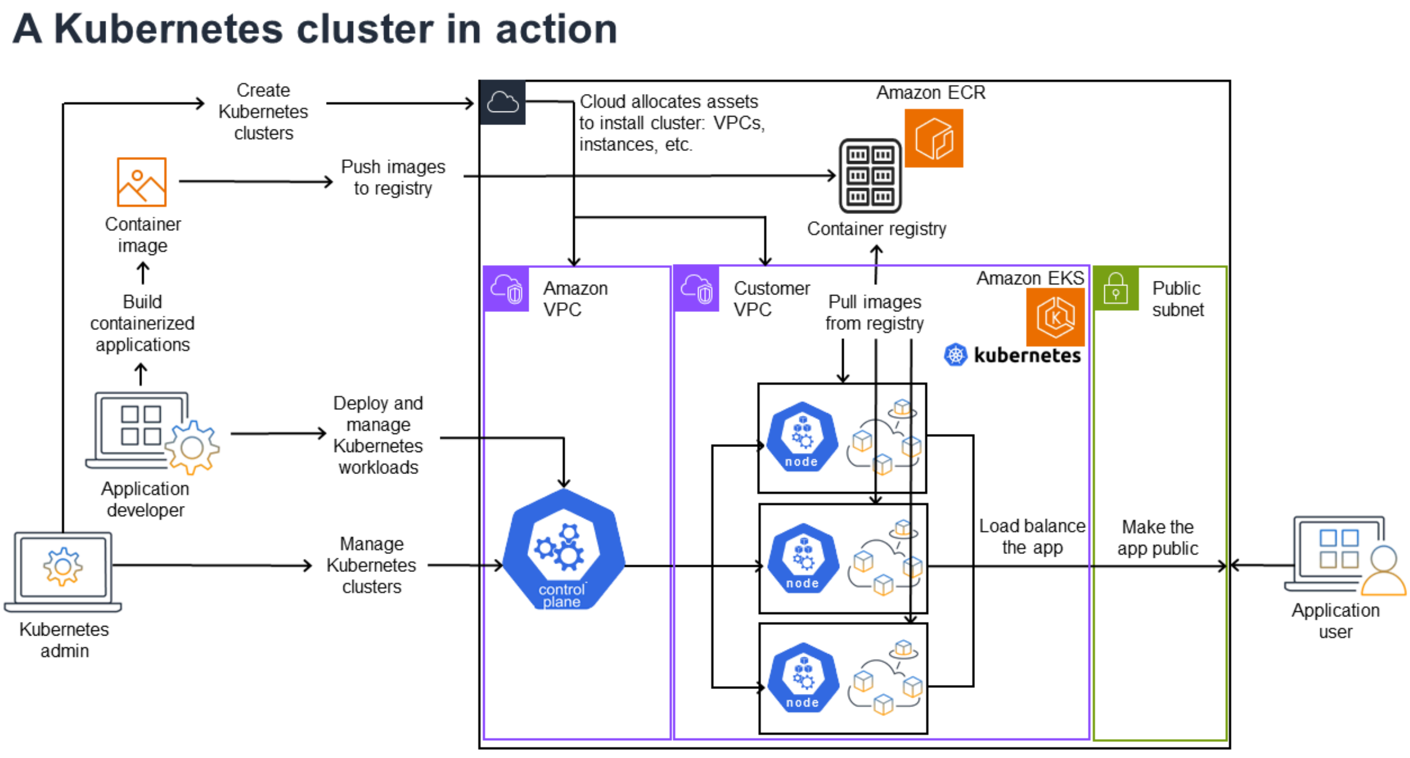
Kubernetes source code is freely available, so with your own equipment you could install and manage Kubernetes yourself. However, self-managing Kubernetes requires deep operational expertise and takes time and effort to maintain. For those reasons, most people deploying production workloads choose a cloud provider (such as Amazon EKS) or on-premises provider (such as Amazon EKS Anywhere) with its own tested Kubernetes distribution and support of Kubernetes experts. This allows you to offload much of the undifferentiated heavy lifting needed to maintain your clusters, including:

- **Hardware** — If you don't have hardware available to run Kubernetes per your requirements, a cloud provider such as AWS Amazon EKS can save you on upfront costs. With Amazon EKS, this means that you can consume the best cloud resources offered by AWS, including computer instances (Amazon Elastic Compute Cloud), your own private environment (Amazon VPC), central identity and permissions management (IAM), and storage (Amazon EBS). AWS manages the computers, networks, data centers, and all the other physical components needed to run Kubernetes. Likewise, you don't have to plan your datacenter to handle the maximum capacity on your highest-demand days. For Amazon EKS Anywhere, or other on premises Kubernetes clusters, you are responsible for managing the infrastructure used in your Kubernetes deployments, but you can still rely on AWS to help you keep Kubernetes up to date.

- **Control plane management** — Amazon EKS manages the security and availability of the AWS-hosted Kubernetes control plane, which is responsible for scheduling containers, managing the availability of applications, and other key tasks, so you can focus on your application workloads. If your cluster breaks, AWS should have the means to restore your cluster to a running state. For Amazon EKS Anywhere, you would manage the control plane yourself.
- **Tested upgrades** — When you upgrade your clusters, you can rely on Amazon EKS or Amazon EKS Anywhere to provide tested versions of their Kubernetes distributions.
- **Add-ons** — There are hundreds of projects built to extend and work with Kubernetes that you can add to your cluster's infrastructure or use to aid the running of your workloads. Instead of building and managing those add-ons yourself, AWS provides the section called "Amazon EKS add-ons" that you can use with your clusters. Amazon EKS Anywhere provides Curated Packages that include builds of many popular open source projects. So you don't have to build the software yourself or manage critical security patches, bug fixes, or upgrades. Likewise, if the defaults meet your needs, it's typical for very little configuration of those add-ons to be needed. See the section called "Extend Clusters" for details on extending your cluster with add-ons.

Kubernetes in action

The following diagram shows key activities you would do as a Kubernetes Admin or Application Developer to create and use a Kubernetes cluster. In the process, it illustrates how Kubernetes components interact with each other, using the AWS cloud as the example of the underlying cloud provider.



A Kubernetes Admin creates the Kubernetes cluster using a tool specific to the type of provider on which the cluster will be built. This example uses the AWS cloud as the provider, which offers the managed Kubernetes service called Amazon EKS. The managed service automatically allocates the resources needed to create the cluster, including creating two new Virtual Private Clouds (Amazon VPCs) for the cluster, setting up networking, and mapping Kubernetes permissions directly into the new VPCs for cloud asset management. The managed service also sees that the control plane services have places to run and allocates zero or more Amazon EC2 instances as Kubernetes nodes for running workloads. AWS manages one Amazon VPC itself for the control plane, while the other Amazon VPC contains the customer nodes that run workloads.

Many of the Kubernetes Admin’s tasks going forward are done using Kubernetes tools such as `kubectl`. That tool makes requests for services directly to the cluster’s control plane. The ways that queries and changes are made to the cluster are then very similar to the ways you would do them on any Kubernetes cluster.

An application developer wanting to deploy workloads to this cluster can perform several tasks. The developer needs to build the application into one or more container images, then push those images to a container registry that is accessible to the Kubernetes cluster. AWS offers the Amazon Elastic Container Registry (Amazon ECR) for that purpose.

To run the application, the developer can create YAML-formatted configuration files that tell the cluster how to run the application, including which containers to pull from the registry and how to wrap those containers in Pods. The control plane (scheduler) schedules the containers to one or more nodes and the container runtime on each node actually pulls and runs the needed containers. The developer can also set up an application load balancer to balance traffic to available containers running on each node and expose the application so it is available on a public network to the outside world. With that all done, someone wanting to use the application can connect to the application endpoint to access it.

The following sections go through details of each of these features, from the perspective of Kubernetes Clusters and Workloads.

Clusters

If your job is to start and manage Kubernetes clusters, you should know how Kubernetes clusters are created, enhanced, managed, and deleted. You should also know what the components are that make up a cluster and what you need to do to maintain those components.

Tools for managing clusters handle the overlap between the Kubernetes services and the underlying hardware provider. For that reason, automation of these tasks tend to be done by the Kubernetes provider (such as Amazon EKS or Amazon EKS Anywhere) using tools that are specific to the provider. For example, to start an Amazon EKS cluster you can use `eksctl create cluster`, while for Amazon EKS Anywhere you can use `eksctl anywhere create cluster`. Note that while these commands create a Kubernetes cluster, they are specific to the provider and are not part of the Kubernetes project itself.

Cluster creation and management tools

The Kubernetes project offers tools for creating a Kubernetes cluster manually. So if you want to install Kubernetes on a single machine, or run the control plane on a machine and add nodes manually, you can use CLI tools like kind, minikube, or kubeadm that are listed under Kubernetes Install Tools. To simplify and automate the full lifecycle of cluster creation and management, it is much easier to use tools supported by an established Kubernetes provider, such as Amazon EKS or Amazon EKS Anywhere.

In AWS Cloud, you can create Amazon EKS clusters using CLI tools, such as eksctl, or more declarative tools, such as Terraform (see Amazon EKS Blueprints for Terraform). You can also create a cluster from the AWS Management Console. See Amazon EKS features for a list what you get with Amazon EKS. Kubernetes responsibilities that Amazon EKS takes on for you include:

- **Managed control plane** — AWS makes sure that the Amazon EKS cluster is available and scalable because it manages the control plane for you and makes it available across AWS Availability Zones.
- **Node management** — Instead of manually adding nodes, you can have Amazon EKS create nodes automatically as needed, using Managed Node Groups (see the section called "Managed node groups") or Karpenter. Managed Node Groups have integrations with Kubernetes Cluster Autoscaling. Using node management tools, you can take advantage of cost savings, with things like Spot Instances and node consolidation, and availability, using Schedulingfeatures to set how workloads are deployed and nodes are selected.
- **Cluster networking** — Using CloudFormation templates, `eksctl` sets up networking between control plane and data plane (node) components in the Kubernetes cluster. It also sets up endpoints through which internal and external communications can take place. See De-mystifying cluster networking for Amazon EKS worker nodes for details. Communications between Pods in Amazon EKS is done using Amazon EKS Pod Identities (see the section called "Pod Identity"), which provides a means of letting Pods tap into AWS cloud methods of managing credentials and permissions.
- **Add-Ons** — Amazon EKS saves you from having to build and add software components that are commonly used to support Kubernetes clusters. For example, when you create an Amazon EKS cluster from the AWS Management Console, it automatically adds the Amazon EKS kube-proxy (the section called "kube-proxy"), Amazon VPC CNI plugin for Kubernetes (the section called "Amazon VPC CNI"), and CoreDNS (the section called "CoreDNS") add-ons. See the section called "Amazon EKS add-ons" for more on these add-ons, including a list of which are available.

To run your clusters on your own on-premises computers and networks, Amazon offers Amazon EKS Anywhere. Instead of the AWS Cloud being the provider, you have the choice of running Amazon EKS Anywhere on VMWare vSphere, bare metal (Tinkerbell provider), Snow, CloudStack, or Nutanix platforms using your own equipment.

Amazon EKS Anywhere is based on the same Amazon EKS Distro software that is used by Amazon EKS. However, Amazon EKS Anywhere relies on different implementations of the Kubernetes Cluster API (CAPI) interface to manage the full lifecycle of the machines in an Amazon EKS Anywhere cluster (such as CAPV for vSphere and CAPC for CloudStack). Because the entire cluster is running on your equipment, you take on the added responsibility of managing the control plane and backing up its data (see `etcd` later in this document).

Cluster components

Kubernetes cluster components are divided into two major areas: control plane and worker nodes. Control Plane Components manage the cluster and provide access to its APIs. Worker nodes (sometimes just referred to as Nodes) provide the places where the actual workloads are run. Node Components consist of services that run on each node to communicate with the control plane and run containers. The set of worker nodes for your cluster is referred to as the *Data Plane*.

Control plane

The control plane consists of a set of services that manage the cluster. These services may all be running on a single computer or may be spread across multiple computers. Internally, these are referred to as Control Plane Instances (CPIs). How CPIs are run depends on the size of the cluster and requirements for high availability. As demand increase in the cluster, a control plane service can scale to provide more instances of that service, with requests being load balanced between the instances.

Tasks that components of the Kubernetes control plane performs include:

- **Communicating with cluster components (API server)** — The API server (kube-apiserver) exposes the Kubernetes API so requests to the cluster can be made from both inside and outside of the cluster. In other words, requests to add or change a cluster's objects (Pods, Services, Nodes, and so on) can come from outside commands, such as requests from `kubectl` to run a Pod. Likewise, requests can be made from the API server to components within the cluster, such as a query to the `kubelet` service for the status of a Pod.
- **Store data about the cluster (`etcd` key value store)** — The `etcd` service provides the critical role of keeping track of the current state of the cluster. If the `etcd` service became inaccessible, you would be unable to update or query the status of the cluster, though workloads would continue to run for a while. For that reason, critical clusters typically have multiple, load-balanced instances of the `etcd` service running at a time and do periodic backups of the `etcd` key value store in case of data loss or corruption. Keep in mind that, in Amazon EKS, this is all handled for you automatically by default. Amazon EKS Anywhere provides instruction for etcd backup and restore. See the etcd Data Model to learn how `etcd` manages data.
- **Schedule Pods to nodes (Scheduler)** — Requests to start or stop a Pod in Kubernetes are directed to the Kubernetes Scheduler (kube-scheduler). Because a cluster could have multiple nodes that are capable of running the Pod, it is up to the Scheduler to choose which node (or nodes, in the case of replicas) the Pod should run on. If there is not enough available capacity to run the requested Pod on an existing node, the request will fail, unless you have made other

provisions. Those provisions could include enabling services such as Managed Node Groups (the section called "Managed node groups") or Karpenter that can automatically start up new nodes to handle the workloads.

- **Keep components in desired state (Controller Manager)** — The Kubernetes Controller Manager runs as a daemon process (kube-controller-manager) to watch the state of the cluster and make changes to the cluster to reestablish the expected states. In particular, there are several controllers that watch over different Kubernetes objects, which includes a `statefulset-controller`, `endpoint-controller`, `cronjob-controller`, `node-controller`, and others.
- **Manage cloud resources (Cloud Controller Manager)** — Interactions between Kubernetes and the cloud provider that carries out requests for the underlying data center resources are handled by the Cloud Controller Manager (cloud-controller-manager). Controllers managed by the Cloud Controller Manager can include a route controller (for setting up cloud network routes), service controller (for using cloud load balancing services), and node lifecycle controller (to keep nodes in sync with Kubernetes throughout their lifecycles).

Worker Nodes (data plane)

For a single-node Kubernetes cluster, workloads run on the same machine as the control plane. However, a more standard configuration is to have one or more separate computer systems (Nodes) that are dedicated to running Kubernetes workloads.

When you first create a Kubernetes cluster, some cluster creation tools allow you to configure a certain number nodes to be added to the cluster (either by identifying existing computer systems or by having the provider create new ones). Before any workloads are added to those systems, services are added to each node to implement these features:

- **Manage each node (`kubelet`)** — The API server communicates with the kubelet service running on each node to make sure that the node is properly registered and Pods requested by the Scheduler are running. The kubelet can read the Pod manifests and set up storage volumes or other features needed by the Pods on the local system. It can also check on the health of the locally running containers.
- **Run containers on a node (container runtime)** — The Container Runtime on each node manages the containers requested for each Pod assigned to the node. That means that it can pull container images from the appropriate registry, run the container, stop it, and responds to queries about the container. The default container runtime is containerd. As of Kubernetes 1.24, the special integration of Docker (`dockershim`) that could be used as the container runtime was

dropped from Kubernetes. While you can still use Docker to test and run containers on your local system, to use Docker with Kubernetes you would now have to Install Docker Engine on each node to use it with Kubernetes.

- **Manage networking between containers (`kube-proxy`)** — To be able to support communication between Pods, Kubernetes uses a feature referred to as a Service to set up Pod networks that track IP addresses and ports associated with those Pods. The kube-proxy service runs on every node to allow that communication between Pods to take place.

Extend Clusters

There are some services you can add to Kubernetes to support the cluster, but are not run in the control plane. These services often run directly on nodes in the kube-system namespace or in its own namespace (as is often done with third-party service providers). A common example is the CoreDNS service, which provides DNS services to the cluster. Refer to Discovering builtin services for information on how to see which cluster services are running in kube-system on your cluster.

There are different types of add-ons you can consider adding to your clusters. To keep your clusters healthy, you can add observability features (see *Monitor clusters*) that allow you to do things like logging, auditing, and metrics. With this information, you can troubleshoot problems that occur, often through the same observability interfaces. Examples of these types of services include Amazon GuardDuty, CloudWatch (see the section called "Amazon CloudWatch"), AWS Distro for OpenTelemetry, Amazon VPC CNI plugin for Kubernetes (see the section called "Amazon VPC CNI"), and Grafana Kubernetes Monitoring. For storage (see *Store app data*), add-ons to Amazon EKS include Amazon Elastic Block Store CSI Driver (see the section called "Amazon EBS"), Amazon Elastic File System CSI Driver (see the section called "Amazon EFS"), and several third-party storage add-ons such as Amazon FSx for NetApp ONTAP CSI driver the section called "Amazon FSx for NetApp ONTAP").

For a more complete list of available Amazon EKS add-ons, see the section called "Amazon EKS add-ons".

Workloads

Kubernetes defines a Workload as "an application running on Kubernetes." That application can consist of a set of microservices run as Containers in Pods, or could be run as a batch job or other type of applications. The job of Kubernetes is to make sure that the requests that you make for those objects to be set up or deployed are carried out. As someone deploying applications, you should learn about how containers are built, how Pods are defined, and what methods you can use for deploying them.

Containers

The most basic element of an application workload that you deploy and manage in Kubernetes is a *Pod* . A Pod represents a way of holding the components of an application as well as defining specifications that describe the Pod's attributes. Contrast this to something like an RPM or Deb package, which packages together software for a Linux system, but does not itself run as an entity.

Because the Pod is the smallest deployable unit, it typically holds a single container. However, multiple containers can be in a Pod in cases where the containers are tightly coupled. For example, a web server container might be packaged in a Pod with a sidecar type of container that may provide logging, monitoring, or other service that is closely tied to the web server container. In this case, being in the same Pod ensures that for each running instance of the Pod, both containers always run on the same node. Likewise, all containers in a Pod share the same environment, with the containers in a Pod running as though they are in the same isolated host. The effect of this is that the containers share a single IP address that provides access to the Pod and the containers can communicate with each other as though they were running on their own localhost.

Pod specifications (PodSpec) define the desired state of the Pod. You can deploy an individual Pod or multiple Pods by using workload resources to manage Pod Templates. Workload resources include Deployments (to manage multiple Pod Replicas), StatefulSets (to deploy Pods that need to be unique, such as database Pods), and DaemonSets (where a Pod needs to run continuously on every node). More on those later.

While a Pod is the smallest unit you deploy, a container is the smallest unit that you build and manage.

Building Containers

The Pod is really just a structure around one or more containers, with each container itself holding the file system, executables, configuration files, libraries, and other components to actually run the application. Because a company called Docker Inc. first popularized containers, some people refer to containers as Docker Containers. However, the Open Container Initiative has since defined container runtimes, images, and distribution methods for the industry. Add to that the fact that containers were created from many existing Linux features, others often refer to containers as OCI Containers, Linux Containers, or just Containers.

When you build a container, you typically start with a Dockerfile (literally named that). Inside that Dockerfile, you identify:

- **A base image** — A base container image is a container that is typically built from either a minimal version of an operating system's file system (such as Red Hat Enterprise Linux or Ubuntu) or a minimal system that is enhanced to provide software to run specific types of applications (such as a nodejs or python apps).
- **Application software** — You can add your application software to your container in much the same way you would add it to a Linux system. For example, in your Dockerfile you can run `npm` and `yarn` to install a Java application or `yum` and `dnf` to install RPM packages. In other words, using a RUN command in a Dockerfile, you can run any command that is available in the file system of your base image to install software or configure software inside of the resulting container image.
- **Instructions** — The Dockerfile reference describes the instructions you can add to a Dockerfile when you configure it. These include instructions used to build what is in the container itself (`ADD` or `COPY` files from the local system), identify commands to execute when the container is run (`CMD` or `ENTRYPOINT`), and connect the container to the system it runs on (by identifying the `USER` to run as, a local `VOLUME` to mount, or the ports to `EXPOSE`).

While the `docker` command and service have traditionally been used to build containers (`docker build`), other tools that are available to build container images include podman and nerdctl. See Building Better Container Images or Overview of Docker Build to learn about building containers.

Storing Containers

Once you've built your container image, you can store it in a container distribution registry on your workstation or on a public container registry. Running a private container registry on your workstation allows you to store container images locally, making them readily available to you.

To store container images in a more public manner, you can push them to a public container registry. Public container registries provide a central location for storing and distributing container images. Examples of public container registries include the Amazon Elastic Container Registry, Red Hat Quay registry, and Docker Hub registry.

When running containerized workloads on Amazon Elastic Kubernetes Service (Amazon EKS) we recommend pulling copies of Docker Official Images that are stored in Amazon Elastic Container Registry. Amazon ECR has been storing these images since 2021. You can search for popular container images in the [Amazon ECR Public Gallery], and specifically for the Docker Hub images, you can search the [Amazon ECR Docker Gallery].

Running containers

Because containers are built in a standard format, a container can run on any machine that can run a container runtime (such as Docker) and whose contents match the local machine's architecture (such as `x86_64` or `arm`). To test a container or just run it on your local desktop, you can use `docker run` or `podman run` commands to start up a container on the localhost. For Kubernetes, however, each worker node has a container runtime deployed and it is up to Kubernetes to request that a node run a container.

Once a container has been assigned to run on a node, the node looks to see if the requested version of the container image already exists on the node. If it doesn't, Kubernetes tells the container runtime to pull that container from the appropriate container registry, then run that container locally. Keep in mind that a *container image* refers to the software package that is moved around between your laptop, the container registry, and Kubernetes nodes. A *container* refers to a running instance of that image.

Pods

Once your containers are ready, working with Pods includes configuring, deploying, and making the Pods accessible.

Configuring Pods

When you define a Pod, you assign a set of attributes to it. Those attributes must include at least the Pod name and the container image to run. However, there are many other things you want to configure with your Pod definitions as well (see the [PodSpec] page for details on what can go into a Pod). These include:

- **Storage** — When a running container is stopped and deleted, data storage in that container will disappear, unless you set up more permanent storage. Kubernetes supports many different storage types and abstracts them under the umbrella of [Volumes]. Storage types include [CephFS], [NFS], [iSCSI], and others. You can even use a [local block device] from the local computer. With one of those storage types available from your cluster, you can mount the storage volume to a selected mount point in your container's file system. A [Persistent Volume] is one that continues to exist after the Pod is deleted, while an [Ephemeral Volume] is deleted when the Pod is deleted. If your cluster administrator created different [storage classes] for your cluster, you might have the option for choosing the attributes of the storage you use, such as whether the volume is deleted or reclaimed after use, whether it will expand if more space is needed, and even whether it meets certain performance requirements.

- **Secrets** — By making Secrets available to containers in Pod specs, you can provide the permissions those containers need to access file systems, data bases, or other protected assets. Keys, passwords, and tokens are among the items that can be stored as secrets. Using secrets makes it so you don't have to store this information in container images, but need only make the secrets available to running containers. Similar to Secrets are ConfigMaps. A `ConfigMap` tends to hold less critical information, such as key-value pairs for configuring a service.
- **Container resources** — Objects for further configuring containers can take the form of resource configuration. For each container, you can request the amount of memory and CPU that it can use, as well as place limits of the total amount of those resources that the container can use. See Resource Management for Pods and Containers for examples.
- **Disruptions** — Pods can be disrupted involuntarily (a node goes down) or voluntarily (an upgrade is desired). By configuring a Pod disruption budget, you can exert some control over how available your application remains when disruptions occur. See Specifying a Disruption Budget for your application for examples.
- **Namespaces** — Kubernetes provides different ways to isolate Kubernetes components and workloads from each other. Running all the Pods for a particular application in the same Namespace is a common way to secure and manage those Pods together. You can create your own namespaces to use or choose to not indicate a namespace (which causes Kubernetes to use the `default` namespace). Kubernetes control plane components typically run in the kube-system namespace.

The configuration just described is typically gathered together in a YAML file to be applied to the Kubernetes cluster. For personal Kubernetes clusters, you might just store these YAML files on your local system. However, with more critical clusters and workloads, GitOps is a popular way to automate storage and updates to both workload and Kubernetes infrastructure resources.

The objects used to gather together and deploy Pod information is defined by one of the following deployment methods.

Deploying Pods

The method you would choose for deploying Pods depends on the type of application you plan to run with those Pods. Here are some of your choices:

- **Stateless applications** — A stateless application doesn't save a client's session data, so another session doesn't need to refer back to what happened to a previous session. This makes is easier to just replace Pods with new ones if they become unhealthy or move them around without saving state. If you are running a stateless application (such as a web server), you can use a Deployment to deploy Podsand ReplicaSets. A ReplicaSet defines how many instances of a Pod that you want running concurrently. Although you can run a ReplicaSet directly, it is common to run replicas directly within a Deployment, to define how many replicas of a Pod should be running at a time.
- **Stateful applications** — A stateful application is one where the identity of the Pod and the order in which Pods are launched are important. These applications need persistent storage that is stable and need to be deployed and scaled in a consistent manner. To deploy a stateful application in Kubernetes, you can use StatefulSets. An example of an application that is typically run as a StatefulSet is a database. Within a StatefulSet, you could define replicas, the Pod and its containers, storage volumes to mount, and locations in the container where data are stored. See Run a Replicated Stateful Application for an example of a database being deployed as a ReplicaSet.
- **Per-node applications** — There are times when you want to run an application on every node in your Kubernetes cluster. For example, your data center might require that every computer run a monitoring application or a particular remote access service. For Kubernetes, you can use a DaemonSet to ensure that the selected application runs on every node in your cluster.
- **Applications run to completion** — There are some applications you want to run to complete a particular task. This could include one that runs monthly status reports or cleans out old data. A Job object can be used to set up an application to start up and run, then exit when the task is done. A CronJob object lets you set up an application to run at a specific hour, minute, day of the month, month, or day of the week, using a structure defined by the Linux crontab format.

Making applications accessible from the network

With applications often deployed as a set of microservices that moved around to different places, Kubernetes needed a way for those microservices to be able to find each other. Also, for others to access an application outside of the Kubernetes cluster, Kubernetes needed a way to expose that application on outside addresses and ports. These networking-related features are done with Service and Ingress objects, respectively:

- **Services** — Because a Pod can move around to different nodes and addresses, another Pod that needed to communicate with the first Pod could find it difficult to find where it is. To solve this problem, Kubernetes lets you represent an application as a Service. With a Service, you can identify a Pod or set of Pods with a particular name, then indicate what port exposes that application's service from the Pod and what ports another application could use to contact that service. Another Pod within a cluster can simply request a Service by name and Kubernetes will direct that request to the proper port for an instance of the Pod running that service.

- **Ingress** — Ingress is what can make applications represented by Kubernetes Services available to clients that are outside of the cluster. Basic features of Ingress include a load balancer (managed by Ingress), the Ingress controller, and rules for routing requests from the controller to the Service. There are several Ingress Controllers that you can choose from with Kubernetes.

Next steps

Understanding basic Kubernetes concepts and how they relate to Amazon EKS will help you navigate both the Amazon EKS documentation and Kubernetes documentation to find the information you need to manage Amazon EKS clusters and deploy workloads to those clusters. To begin using Amazon EKS, choose from the following:

- the section called "Create cluster (eksctl)"
- the section called "Create a cluster"
- the section called "Sample deployment (Linux)"
- *Cluster management*

Deploy Amazon EKS clusters across cloud and on-premises environments

Understand Amazon EKS deployment options

Amazon Elastic Kubernetes Service (Amazon EKS) is a fully managed Kubernetes service that enables you to run Kubernetes seamlessly in the cloud and in your on-premises environments.

In the cloud, Amazon EKS automates Kubernetes cluster infrastructure management for the Kubernetes control plane and nodes. This is essential for scheduling containers, managing application availability, dynamically scaling resources, optimizing compute, storing cluster data, and performing other critical functions. With Amazon EKS, you get the robust performance, scalability, reliability, and availability of AWS infrastructure, along with native integrations with AWS networking, security, storage, and observability services.

To simplify running Kubernetes in your on-premises environments, you can use the same Amazon EKS clusters, features, and tools to the section called "Nodes" or Amazon EKS Hybrid Nodes on your own infrastructure, or you can use Amazon EKS Anywhere for self-contained air-gapped environments.

Amazon EKS in the cloud

You can use Amazon EKS with compute in AWS Regions, AWS Local Zones, and AWS Wavelength Zones. With Amazon EKS in the cloud, the security, scalability, and availability of the Kubernetes control plane is fully managed by AWS in the AWS Region. When running applications with compute in AWS Regions, you get the full breadth of AWS and Amazon EKS features, including Amazon EKS Auto Mode, which fully automates Kubernetes cluster infrastructure management for compute, storage, and networking on AWS with a single click. When running applications with compute in AWS Local Zones and AWS Wavelength Zones, you can use Amazon EKS self-managed nodes to connect Amazon EC2 instances for your cluster compute and can use the other available AWS services in AWS Local Zones and AWS Wavelength Zones. For more information see AWS Local Zones features and AWS Wavelength Zones features.

	Amazon EKS in AWS Regions	**Amazon EKS in Local/Wavelength Zones**
Kuberenetes control plane management	AWS-managed	AWS-managed
Kubernetes control plane location	AWS Regions	AWS Regions
Kubernetes data plane	• Amazon EKS Auto Mode • Amazon EKS Managed Node Groups • Amazon EC2 self-managed nodes • AWS Fargate	• Amazon EKS Managed Node Groups (Local Zones only) • Amazon EC2 self-managed nodes
Kubernetes data plane location	AWS Regions	AWS Local or Wavelength Zones

Amazon EKS in your data center or edge environments

If you need to run applications in your own data centers or edge environments, you can use *Amazon EKS on AWS Outposts* or Amazon EKS Hybrid Nodes. You can use self-managed nodes with Amazon EC2 instances on AWS Outposts for your cluster compute, or you can use Amazon EKS Hybrid Nodes with your own on-premises or edge infrastructure for your cluster compute. AWS Outposts is AWS-managed infrastructure that you run in your data centers or co-location facilities, whereas Amazon EKS Hybrid Nodes runs on your physical or virtual machines that you manage in your on-premises or edge environments. Amazon EKS on AWS Outposts and Amazon EKS Hybrid Nodes require a reliable connection from your on-premises environments to an AWS Region, and you can use the same Amazon EKS clusters, features, and tools you use to run applications in the cloud. When running on AWS Outposts, you can alternatively deploy the entire Kubernetes cluster on AWS Outposts with Amazon EKS local clusters on AWS Outposts.

	Amazon EKS Hybrid Nodes	Amazon EKS on AWS Outposts
Kuberenetes control plane management	AWS-managed	AWS-managed
Kubernetes control plane location	AWS Regions	AWS Regions or AWS Outposts
Kubernetes data plane	Customer-managed physical or virtual machines	Amazon EC2 self-managed nodes
Kubernetes data plane location	Customer data center or edge environment	Customer data center or edge environment

Amazon EKS Anywhere for air-gapped environments

Amazon EKS Anywhere simplifies Kubernetes cluster management through the automation of undifferentiated heavy lifting such as infrastructure setup and Kubernetes cluster lifecycle operations in on-premises and edge environments. Unlike Amazon EKS, Amazon EKS Anywhere is a customer-managed product and customers are responsible for cluster lifecycle operations and maintenance of Amazon EKS Anywhere clusters. Amazon EKS Anywhere is built on the Kubernetes sub-project Cluster API (CAPI) and supports a range of infrastructure including VMware vSphere, bare metal, Nutanix, Apache CloudStack, and AWS Snow. Amazon EKS Anywhere can be run in air-gapped environments and offers optional integrations with regional AWS services for observability and identity management. To receive support for Amazon EKS Anywhere and access to AWS-vended Kubernetes add-ons, you can purchase Amazon EKS Anywhere Enterprise Subscriptions.

	Amazon EKS Anywhere
Kuberenetes control plane management	Customer-managed
Kubernetes control plane location	Customer data center or edge environment
Kubernetes data plane	Customer-managed physical or virtual machines
Kubernetes data plane location	Customer data center or edge environment

Amazon EKS tooling

You can use the Amazon EKS Connector to register and connect any conformant Kubernetes cluster to AWS and view it in the Amazon EKS console. After a cluster is connected, you can see the status, configuration, and workloads for that cluster in the Amazon EKS console. You can use this feature to view connected clusters in Amazon EKS console, but the Amazon EKS Connector does not enable management or mutating operations for your connected clusters through the Amazon EKS console.

Amazon EKS Distro is the AWS distribution of the underlying Kubernetes components that power all Amazon EKS offerings. It includes the core components required for a functioning Kubernetes cluster such as Kubernetes control plane components (etcd, kube-apiserver, kube-scheduler, kube-controller-manager) and networking components (CoreDNS, kube-proxy, CNI plugins). Amazon EKS Distro can be used to self-manage Kubernetes clusters with your choice of tooling. Amazon EKS Distro deployments are not covered by AWS Support Plans.

Set up to use Amazon EKS

To prepare for the command-line management of your Amazon EKS clusters, you need to install several tools. Use the following to set up credentials, create and modify clusters, and work with clusters once they are running:

- Set up AWS CLI – Get the AWS CLI to set up and manage the services you need to work with Amazon EKS clusters. In particular, you need AWS CLI to configure credentials, but you also need it with other AWS services.
- Set up kubectl and eksctl – The `eksctl` CLI interacts with AWS to create, modify, and delete Amazon EKS clusters. Once a cluster is up, use the open source `kubectl` command to manage Kubernetes objects within your Amazon EKS clusters.
- Set up a development environment (optional)– Consider adding the following tools:
  - **Local deployment tool** – If you're new to Kubernetes, consider installing a local deployment tool like minikube or kind. These tools allow you to have an Amazon EKS cluster on your local machine for testing applications.
  - **Package manager** – Helm is a popular package manager for Kubernetes that simplifies the installation and management of complex packages. With Helm, it's easier to install and manage packages like the AWS Load Balancer Controller on your Amazon EKS cluster.

Next steps

- Set up AWS CLI
- Set up kubectl and eksctl
- Quickstart: Deploy a web app and store data

Set up AWS CLI

The AWS CLI is a command line tool for working with AWS services, including Amazon EKS. It is also used to authenticate IAM users or roles for access to the Amazon EKS cluster and other AWS resources from your local machine. To provision resources in AWS from the command line, you need to obtain an AWS access key ID and secret key to use in the command line. Then you need to configure these credentials in the AWS CLI. If you haven't already installed the AWS CLI, see Install or update the latest version of the AWS CLI in the *AWS Command Line Interface User Guide*.

To create an access key

1. Sign into the AWS Management Console.
2. For single-user or multiple-user accounts:
   - **Single-user account –**:: In the top right, choose your AWS user name to open the navigation menu. For example, choose **`webadmin`** .
   - **Multiple-user account –**:: Choose IAM from the list of services. From the IAM Dashboard, select **Users**, and choose the name of the user.
3. Choose **Security credentials**.
4. Under **Access keys**, choose **Create access key**.
5. Choose **Command Line Interface (CLI)**, then choose **Next**.
6. Choose **Create access key**.
7. Choose **Download .csv file**.

To configure the AWS CLI

After installing the AWS CLI, do the following steps to configure it. For more information, see Configure the AWS CLI in the *AWS Command Line Interface User Guide*.

1. In a terminal window, enter the following command:

   ```
   aws configure
   ```

   Optionally, you can configure a named profile, such as `--profile cluster-admin`. If you configure a named profile in the AWS CLI, you must **always** pass this flag in subsequent commands.
2. Enter your AWS credentials. For example:

   ```
   Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
   Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   Default region name [None]: region-code
   Default output format [None]: json
   ```

To get a security token

If needed, run the following command to get a new security token for the AWS CLI. For more information, see get-session-token in the *AWS CLI Command Reference*.

By default, the token is valid for 15 minutes. To change the default session timeout, pass the `--duration-seconds` flag. For example:

```
aws sts get-session-token --duration-seconds 3600
```

This command returns the temporary security credentials for an AWS CLI session. You should see the following response output:

```
{
    "Credentials": {
        "AccessKeyId": "ASIA5FTRU3LOEXAMPLE",
        "SecretAccessKey": "JnKgvwfqUD9mNsPoi9IbxAYEXAMPLE",
        "SessionToken": "VERYLONGSESSIONTOKENSTRING",
        "Expiration": "2023-02-17T03:14:24+00:00"
    }
}
```

To verify the user identity

If needed, run the following command to verify the AWS credentials for your IAM user identity (such as *`ClusterAdmin`*) for the terminal session.

```
aws sts get-caller-identity
```

This command returns the Amazon Resource Name (ARN) of the IAM entity that's configured for the AWS CLI. You should see the following example response output:

```
{
    "UserId": "AKIAIOSFODNN7EXAMPLE",
    "Account": "01234567890",
    "Arn": "arn:aws:iam::01234567890:user/ClusterAdmin"
}
```

Next steps

- Set up kubectl and eksctl
- Quickstart: Deploy a web app and store data

Set up `kubectl` and `eksctl`

`Kubectl` is a command line tool that you use to communicate with the Kubernetes API server. The `kubectl` binary is available in many operating system package managers. Using a package manager for your installation is often easier than a manual download and install process. The `eksctl` command lets you create and modify Amazon EKS clusters.

Topics on this page help you install and set up these tools:

- Install or update kubectl
- Install eksctl

Install or update `kubectl`

This topic helps you to download and install, or update, the `kubectl` binary on your device. The binary is identical to the upstream community versions. The binary is not unique to Amazon EKS or AWS. Use the steps below to get the specific version of `kubectl` that you need, although many builders simply run `brew install kubectl` to install it.

> **Note**
>
> You must use a `kubectl` version that is within one minor version difference of your Amazon EKS cluster control plane. For example, a `1.31` `kubectl` client works with Kubernetes `1.30`, `1.31`, and `1.32` clusters.

Step 1: Check if `kubectl` is installed

Determine whether you already have `kubectl` installed on your device.

```
kubectl version --client
```

If you have `kubectl` installed in the path of your device, the example output includes information similar to the following. If you want to update the version that you currently have installed with a later version, complete the next step, making sure to install the new version in the same location that your current version is in.

```
Client Version: v1.31.X-eks-1234567
```

If you receive no output, then you either don't have `kubectl` installed, or it's not installed in a location that's in your device's path.

Step 2: Install or update kubectl

Install or update `kubectl` on one of the following operating systems:

- the section called "macOS"
- the section called "Linux (amd64)"
- the section called "Linux (arm64)"
- the section called "Windows"

macOS

1. Download the binary for your cluster's Kubernetes version from Amazon S3.

   - Kubernetes `1.32`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.0/2025-01-10/bin/darwin/amd64/kubectl
     ```

   - Kubernetes `1.31`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.4/2025-01-10/bin/darwin/amd64/kubectl
     ```

   - Kubernetes `1.30`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.8/2025-01-10/bin/darwin/amd64/kubectl
     ```

   - Kubernetes `1.29`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.12/2025-01-10/bin/
darwin/amd64/kubectl
```

- Kubernetes `1.28`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-01-10/bin/
darwin/amd64/kubectl
```

- Kubernetes `1.27`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2025-01-10/bin/
darwin/amd64/kubectl
```

- Kubernetes `1.26`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/
darwin/amd64/kubectl
```

- Kubernetes `1.25`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.25.16/2024-12-12/bin/
darwin/amd64/kubectl
```

- Kubernetes `1.24`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.24.17/2024-12-12/bin/
darwin/amd64/kubectl
```

- Kubernetes `1.23`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.23.17/2024-09-11/bin/
darwin/amd64/kubectl
```

- Kubernetes `1.22`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.17/2024-09-11/bin/
darwin/amd64/kubectl
```

- Kubernetes `1.21`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.21.14/2024-09-11/bin/
darwin/amd64/kubectl
```

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

   a. Download the SHA-256 checksum for your cluster's Kubernetes version.

      - Kubernetes 1.32

        ```
        curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.0/2025-01-10/bin/
        darwin/amd64/kubectl.sha256
        ```

      - Kubernetes 1.31

        ```
        curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.4/2025-01-10/bin/
        darwin/amd64/kubectl.sha256
        ```

      - Kubernetes 1.30

        ```
        curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.8/2025-01-10/bin/
        darwin/amd64/kubectl.sha256
        ```

      - Kubernetes 1.29

        ```
        curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.12/2025-01-10/bin/
        darwin/amd64/kubectl.sha256
        ```

      - Kubernetes 1.28

        ```
        curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-01-10/bin/
        darwin/amd64/kubectl.sha256
        ```

      - Kubernetes 1.27

        ```
        curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2025-01-10/bin/
        darwin/amd64/kubectl.sha256
        ```

      - Kubernetes 1.26

        ```
        curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/
        darwin/amd64/kubectl.sha256
        ```

- Kubernetes 1.25

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.25.16/2024-12-12/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes 1.24

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.24.17/2024-12-12/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes 1.23

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.23.17/2024-09-11/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes 1.22

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.17/2024-09-11/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes 1.21

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.21.14/2024-09-11/bin/
darwin/amd64/kubectl.sha256
```

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

```
openssl sha1 -sha256 kubectl
```

c. Make sure that the generated checksum in the output matches in the checksum in the downloaded kubectl.sha256 file.

3. Apply execute permissions to the binary.

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

4. Copy the binary to a folder in your PATH. If you have already installed a version of kubectl, then we recommend creating a $HOME/bin/kubectl and ensuring that $HOME/bin comes first in your $PATH.

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

5. (Optional) Add the `$HOME/bin` path to your shell initialization file so that it is configured when you open a shell.

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

Linux (amd64)

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

   - Kubernetes `1.32`

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.0/2024-12-20/bin/linux/amd64/kubectl
   ```

   - Kubernetes `1.31`

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.3/2024-12-12/bin/linux/amd64/kubectl
   ```

   - Kubernetes `1.30`

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.7/2024-12-12/bin/linux/amd64/kubectl
   ```

   - Kubernetes `1.29`

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.10/2024-12-12/bin/linux/amd64/kubectl
   ```

   - Kubernetes `1.28`

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2024-12-12/bin/linux/amd64/kubectl
   ```

   - Kubernetes `1.27`

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/linux/amd64/kubectl
   ```

   - Kubernetes `1.26`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/linux/amd64/kubectl
```

- Kubernetes 1.25

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.25.16/2024-12-12/bin/linux/amd64/kubectl
```

- Kubernetes 1.24

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.24.17/2024-12-12/bin/linux/amd64/kubectl
```

- Kubernetes 1.23

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.23.17/2024-09-11/bin/linux/amd64/kubectl
```

- Kubernetes 1.22

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.17/2024-09-11/bin/linux/amd64/kubectl
```

- Kubernetes 1.21

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.21.14/2024-09-11/bin/linux/amd64/kubectl
```

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

   a. Download the SHA-256 checksum for your cluster's Kubernetes version from Amazon S3using the command for your device's hardware platform.

      - Kubernetes 1.32

      ```
      curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.0/2024-12-20/bin/linux/amd64/kubectl.sha256
      ```

      - Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.3/2024-12-12/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.30`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.7/2024-12-12/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.29`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.10/2024-12-12/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.28`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2024-12-12/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.27`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.26`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.25`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.25.16/2024-12-12/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.24`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.24.17/2024-12-12/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.23`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.23.17/2024-09-11/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.22`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.17/2024-09-11/bin/
linux/amd64/kubectl.sha256
```

- Kubernetes `1.21`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.21.14/2024-09-11/bin/
linux/amd64/kubectl.sha256
```

b. Check the `SHA-256` checksum for your downloaded binary with one of the following commands.

```
sha256sum -c kubectl.sha256
```

or

```
openssl sha1 -sha256 kubectl
```

c. For the first, you should see `kubectl: OK`, for the second, you can check that the generated checksum in the output matches in the checksum in the downloaded `kubectl.sha256` file.

3. Apply execute permissions to the binary.

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

4. Copy the binary to a folder in your `PATH`. If you have already installed a version of `kubectl`, then we recommend creating a `$HOME/bin/kubectl` and ensuring that `$HOME/bin` comes first in your `$PATH`.

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

5. (Optional) Add the `$HOME/bin` path to your shell initialization file so that it is configured when you open a shell.

> **Note**
>
> This step assumes you are using the Bash shell; if you are using another shell, change the command to use your specific shell initialization file.

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

Linux (arm64)

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

   - Kubernetes `1.32`

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.0/2024-12-20/bin/linux/arm64/kubectl
   ```

   - Kubernetes `1.31`

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.3/2024-12-12/bin/linux/arm64/kubectl
   ```

   - Kubernetes `1.30`

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.7/2024-12-12/bin/linux/arm64/kubectl
   ```

   - Kubernetes `1.29`

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.10/2024-12-12/bin/linux/arm64/kubectl
   ```

   - Kubernetes `1.28`

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2024-12-12/bin/linux/arm64/kubectl
   ```

   - Kubernetes `1.27`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/linux/
arm64/kubectl
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/linux/
arm64/kubectl
```

- Kubernetes 1.25

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.25.16/2024-12-12/bin/linux/
arm64/kubectl
```

- Kubernetes 1.24

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.24.17/2024-12-12/bin/linux/
arm64/kubectl
```

- Kubernetes 1.23

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.23.17/2024-09-11/bin/linux/
arm64/kubectl
```

- Kubernetes 1.22

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.17/2024-09-11/bin/linux/
arm64/kubectl
```

- Kubernetes 1.21

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.21.14/2024-09-11/bin/linux/
arm64/kubectl
```

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

   a. Download the SHA-256 checksum for your cluster's Kubernetes version from Amazon S3using the command for your device's hardware platform.

      - Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.0/2024-12-20/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes `1.31`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.3/2024-12-12/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes `1.30`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.7/2024-12-12/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes `1.29`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.10/2024-12-12/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes `1.28`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2024-12-12/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes `1.27`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes `1.26`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes `1.25`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.25.16/2024-12-12/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes `1.24`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.24.17/2024-12-12/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes `1.23`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.23.17/2024-09-11/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes `1.22`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.17/2024-09-11/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes `1.21`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.21.14/2024-09-11/bin/
linux/arm64/kubectl.sha256
```

b. Check the `SHA-256` checksum for your downloaded binary with one of the following commands.

```
sha256sum -c kubectl.sha256
```

or

```
openssl sha1 -sha256 kubectl
```

c. For the first, you should see `kubectl: OK`, for the second, you can check that the generated checksum in the output matches in the checksum in the downloaded `kubectl.sha256` file.

3. Apply execute permissions to the binary.

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

4. Copy the binary to a folder in your `PATH`. If you have already installed a version of `kubectl`, then we recommend creating a `$HOME/bin/kubectl` and ensuring that `$HOME/bin` comes first in your `$PATH`.

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

5. (Optional) Add the `$HOME/bin` path to your shell initialization file so that it is configured when you open a shell.

> **Note**
>
> This step assumes you are using the Bash shell; if you are using another shell, change the command to use your specific shell initialization file.

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

Windows

1. Open a PowerShell terminal.
2. Download the `kubectl` binary for your cluster's Kubernetes version from Amazon S3.
   - Kubernetes `1.32`

   ```
   curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.0/2024-12-20/bin/
   windows/amd64/kubectl.exe
   ```

   - Kubernetes `1.31`

   ```
   curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.3/2024-12-12/bin/
   windows/amd64/kubectl.exe
   ```

   - Kubernetes `1.30`

   ```
   curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.7/2024-12-12/bin/
   windows/amd64/kubectl.exe
   ```

   - Kubernetes `1.29`

   ```
   curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.10/2024-12-12/bin/
   windows/amd64/kubectl.exe
   ```

   - Kubernetes `1.28`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2024-12-12/bin/
windows/amd64/kubectl.exe
```

- Kubernetes 1.27

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/
windows/amd64/kubectl.exe
```

- Kubernetes 1.26

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/
windows/amd64/kubectl.exe
```

- Kubernetes 1.25

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.25.16/2024-12-12/bin/
windows/amd64/kubectl.exe
```

- Kubernetes 1.24

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.24.17/2024-12-12/bin/
windows/amd64/kubectl.exe
```

- Kubernetes 1.23

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.23.17/2024-09-11/bin/
windows/amd64/kubectl.exe
```

- Kubernetes 1.22

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.17/2024-09-11/bin/
windows/amd64/kubectl.exe
```

- Kubernetes 1.21

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.21.14/2024-09-11/bin/
windows/amd64/kubectl.exe
```

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

   a. Download the SHA-256 checksum for your cluster's Kubernetes version for Windows.

- Kubernetes `1.32`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.0/2024-12-20/bin/
windows/amd64/kubectl.exe.sha256
```

- Kubernetes `1.31`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.3/2024-12-12/bin/
windows/amd64/kubectl.exe.sha256
```

- Kubernetes `1.30`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.7/2024-12-12/bin/
windows/amd64/kubectl.exe.sha256
```

- Kubernetes `1.29`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.10/2024-12-12/
bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes `1.28`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2024-12-12/
bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes `1.27`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/
bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes `1.26`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/
bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes `1.25`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.25.16/2024-12-12/
bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes `1.24`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.24.17/2024-12-12/
bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.23

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.23.17/2024-09-11/
bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.22

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.17/2024-09-11/
bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.21

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.21.14/2024-09-11/
bin/windows/amd64/kubectl.exe.sha256
```

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

```
Get-FileHash kubectl.exe
```

c. Make sure that the generated checksum in the output matches in the checksum in the downloaded `kubectl.sha256` file. The PowerShell output should be an uppercase equivalent string of characters.

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.

```
kubectl version --client
```

6. When first installing `kubectl`, it isn't yet configured to communicate with any server. We will cover this configuration as needed in other procedures. If you ever need to update

the configuration to communicate with a particular cluster, you can run the following command.Replace *region-code* with the AWS Region that your cluster is in. Replace *my-cluster* with the name of your cluster.

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

7. Consider configuring auto completion, which lets you use the tab key to complete `kubectl` subcommands after typing the first few letters. See Kubectl autocomplete in the Kubernetes documentation for details.

Install `eksctl`

The `eksctl` CLI is used to work with EKS clusters. It automates many individual tasks. See Installation in the `eksctl` documentation for instructions on installing `eksctl`.

When using `eksctl` the IAM security principal that you're using must have permissions to work with Amazon EKS IAM roles, service linked roles, AWS CloudFormation, a VPC, and related resources. For more information, see Actions and Using service-linked roles in the IAM User Guide. You must complete all steps in this guide as the same user. To check the current user, run the following command:

```
aws sts get-caller-identity
```

Next steps

- Quickstart: Deploy a web app and store data

Quickstart: Deploy a web app and store data

Deploy a game application and persist its data on Amazon EKS

This quickstart tutorial shows the steps to deploy the 2048 game sample application and persist its data on an Amazon EKS Auto Mode cluster using eksctl. Amazon EKS Auto Mode automates routine tasks for cluster block storage, networking, load balancing, and compute autoscaling.

As we progress, we'll walk you through the cluster setup process. Amazon EKS Auto Mode will automate tasks for creating a node using an EC2 managed instance, creating an application load balancer, and creating an EBS volume.

Overall, you'll deploy a sample workload with the custom annotations required to fully integrate with AWS services.

In this tutorial

Using the `eksctl` cluster template that follows, you'll build a cluster with EKS Auto Mode for automated node provisioning.

VPC Configuration When using the eksctl cluster template that follows, eksctl automatically creates an IPv4 Virtual Private Cloud (VPC) for the cluster. By default, eksctl configures a VPC that addresses all networking requirements, in addition to creating both public and private endpoints.

Instance Management EKS Auto Mode dynamically adds or removes nodes in your EKS cluster based on the demands of your Kubernetes applications.

Data Persistence Use the block storage capability of EKS Auto Mode to ensure the persistence of application data, even in scenarios involving pod restarts or failures.

External App Access Use the load balancing capability of EKS Auto Mode to dynamically provision an Application Load Balancer (ALB).

Prerequisites

Before you begin, ensure you have the following prerequisites set up to use Amazon EKS:

- Set up AWS CLI and configure credentials

- Install eksctl
- Install kubectl

For more information, see *Set up*.

Configure the cluster

In this section, you'll create a cluster using EKS Auto Mode for dynamic node provisioning.

Create a `cluster-config.yaml` file and paste the following contents into it. Replace `region-code` with a valid Region, such as `us-east-1`:

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

metadata:
  name: web-quickstart
  region: <region-code>

autoModeConfig:
  enabled: true
```

Now, we're ready to create the cluster.

Create the Amazon EKS cluster:

```
eksctl create cluster -f cluster-config.yaml
```

> **Important**
>
> If you do not use eksctl to create the cluster, you need to manually tag the VPC subnets.

Create IngressClass

Create a Kubernetes `IngressClass` for EKS Auto Mode. The IngressClass defines how EKS Auto Mode handles Ingress resources. This step configures the load balancing capability of EKS Auto Mode. When you create Ingress resources for your applications, EKS Auto Mode uses this

IngressClass to automatically provision and manage load balancers, integrating your Kubernetes applications with AWS load balancing services.

Save the following yaml file as `ingressclass.yaml`:

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: alb
  annotations:
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  controller: eks.amazonaws.com/alb
```

Apply the IngressClass to your cluster:

```
kubectl apply -f ingressclass.yaml
```

Deploy the 2048 game sample application

In this section, we walk you through the steps to deploy the popular "2048 game" as a sample application within the cluster. The provided manifest includes custom annotations for the Application Load Balancer (ALB). These annotations integrate with and instruct the EKS to handle incoming HTTP traffic as "internet-facing" and route it to the appropriate service in the 'game-2048' namespace using the target type "ip".

1. Create a Kubernetes namespace called `game-2048` with the `--save-config` flag.

   ```
   kubectl create namespace game-2048 --save-config
   ```

   You should see the following response output:

   ```
   namespace/game-2048 created
   ```

2. Deploy the 2048 Game Sample application.

   ```
   kubectl apply -n game-2048 -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.8.0/docs/examples/2048/2048_full.yaml
   ```

This manifest sets up a Kubernetes Deployment, Service, and Ingress for the `game-2048` namespace, creating the necessary resources to deploy and expose the `game-2048` application within the cluster. It includes the creation of a service named `service-2048` that exposes the deployment on port `80`, and an Ingress resource named `ingress-2048` that defines routing rules for incoming HTTP traffic and annotations for an internet-facing Application Load Balancer (ALB). You should see the following response output:

```
namespace/game-2048 configured
deployment.apps/deployment-2048 created
service/service-2048 created
ingress.networking.k8s.io/ingress-2048 created
```

3. Run the following command to get the Ingress resource for the `game-2048` namespace.

```
kubectl get ingress -n game-2048
```

You should see the following response output:

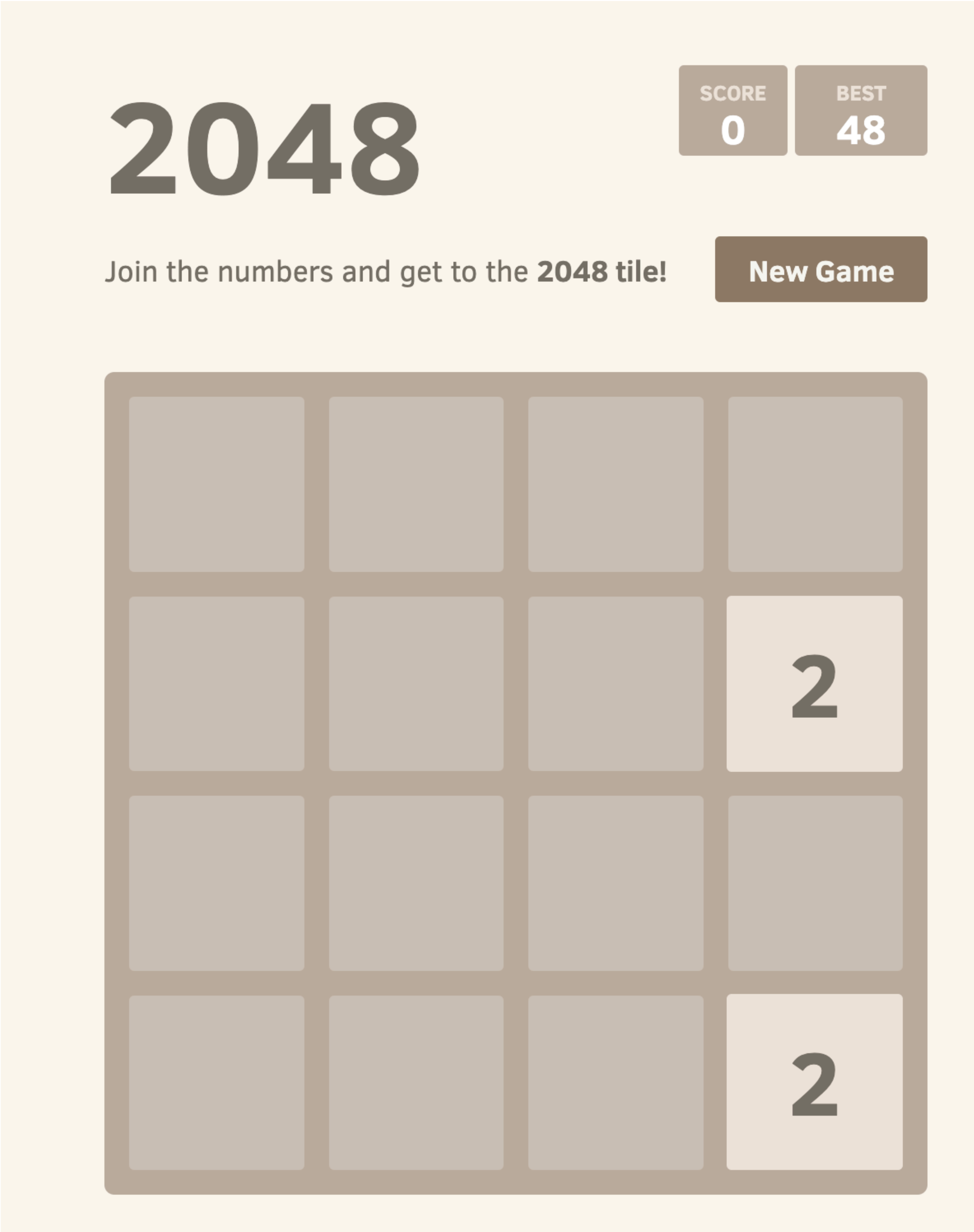
```
NAME           CLASS   HOSTS   ADDRESS
                     PORTS   AGE
ingress-2048   alb     *       k8s-game2048-ingress2-eb379a0f83-378466616.region-
code.elb.amazonaws.com   80      31s
```

You'll need to wait several minutes for the Application Load Balancer (ALB) to provision before you begin the following steps.

4. Open a web browser and enter the `ADDRESS` from the previous step to access the web application. For example:

```
k8s-game2048-ingress2-eb379a0f83-378466616.region-code.elb.amazonaws.com
```

You should see the 2048 game in your browser. Play!


2048
SCORE
0
BEST
48
Join the numbers and get to the 2048 tile!
New Game
2
2

Persist Data using Amazon EKS Auto Mode

Now that the 2048 game is up and running on your Amazon EKS cluster, it's time to ensure that your game data is safely persisted using the block storage capability of Amazon EKS Auto Mode.

1. Create a file named `storage-class.yaml`:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: auto-ebs-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
```

2. Apply the `StorageClass`:

```
kubectl apply -f storage-class.yaml
```

3. Create a Persistent Volume Claim (PVC) to request storage for your game data. Create a file named `ebs-pvc.yaml` and add the following content to it:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: game-data-pvc
  namespace: game-2048
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: auto-ebs-sc
```

4. Apply the PVC to your cluster:

```
kubectl apply -f ebs-pvc.yaml
```

You should see the following response output:

```
persistentvolumeclaim/game-data-pvc created
```

5. Now, you need to update your 2048 game deployment to use this PVC for storing data. The following deployment is configured to use the PVC for storing game data. Create a file named `ebs-deployment.yaml` and add the following contents to it:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: game-2048
  name: deployment-2048
spec:
  replicas: 3  # Adjust the number of replicas as needed
  selector:
    matchLabels:
      app.kubernetes.io/name: app-2048
  template:
    metadata:
      labels:
        app.kubernetes.io/name: app-2048
    spec:
      containers:
        - name: app-2048
          image: public.ecr.aws/l6m2t8p7/docker-2048:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 80
          volumeMounts:
            - name: game-data
              mountPath: /var/lib/2048
      volumes:
        - name: game-data
          persistentVolumeClaim:
            claimName: game-data-pvc
```

6. Apply the updated deployment:

```
kubectl apply -f ebs-deployment.yaml
```

You should see the following response output:

```
deployment.apps/deployment-2048 configured
```

With these steps, your 2048 game on the cluster is now set up to persist data using the block storage capability of Amazon EKS Auto Mode. This ensures that your game progress and data are safe even in the event of pod or node failures.

If you liked this tutorial, let us know by providing feedback so we're able to provide you with more use case-specific quickstart tutorials like this one.

Delete your cluster and nodes

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

```
eksctl delete cluster -f ./cluster-config.yaml
```

EKS will automatically clean up any nodes it provisioned when the cluster is deleted.

Learn Amazon EKS by example

Overview

This Amazon EKS User Guide contains general-purpose procedures to create your first EKS cluster from the command line or AWS Management Console and a solid reference for all major Amazon EKS components. However, as an Amazon EKS cluster administrator or developer, you can gain a deeper understanding of Amazon EKS by following learning paths that exist in sites outside of this guide. These sites can help you:

- **Set up specific types of clusters**. Specific cluster types can be based on your workload types or security requirements. For example, you may want to tune a cluster to run batch, machine learning, or compute-intensive workloads.
- **Enhance your clusters**. You can add advanced features to your cluster to provide things like observability,flexible storage, autoscaling, or specialized cluster networking.
- **Automate updates**. Using features like GitOps, you can set up to provision cluster infrastructure and workloads automatically, based on changes that occur to those components in your Git repositories.
- **Use advanced cluster setup tools**. While `eksctl` provides a quick way to create a cluster, there are other tools that can make it easier to configure and upgrade more complex clusters. These include tools like Terraform and CloudFormation.

To start out on your Amazon EKS learning path, I recommend that you visit some of the sites described on this page. If you run into problems along the way, there are also resources to help you get through them. For example, the Re:post Knowledge Center lets you search the support database for Amazon EKS-related support issues. Also the Amazon EKS Best Practices Guide offers tips on the best ways to set up your production-grade clusters.

Amazon EKS Workshop

Starting with a basic understanding of Kubernetes and containers, the Amazon EKS workshop is a learning platform for walking a cluster administrator through important features of Amazon EKS. Here are ways you can engage with the Amazon EKS workshop:

- **Amazon EKS Basics**: Watch the video on the Introduction page to learn about how Amazon EKS implements Kubernetes features on the AWS cloud. If you need an even more basic understanding of Kubernetes, watch the What is Kubernetes video.
- **Amazon EKS Setup**: If you have an AWS account, the Setup section helps you set up a CloudShell environment to you for creating a cluster. It offers a choice of eksctl (a simple cluster creation command line) and Terraform (a more infrastructure-as-code approach to creating a cluster) for creating your Amazon EKS cluster.
- **Amazon EKS Getting started**: Try out a simple web store from the Sample application section. You can use this throughout the other exercises. In this section, you can also learn about packaging container images and how microservices are managed using Kubernetes Pods, Deployments, Services, StatefulSets and Namespaces. Then use Kustomize to deploy changes to Kubernetes manifests.
- **Amazon EKS Fundamentals**: Using AWS features such as the AWS Load Balancer Controller, the workshop shows you how to expose your applications to the outside world. For storage, the workshop showcases how to use Amazon EBS for block storage, Amazon EFS for filesystem storage, and Amazon FSx for NetApp ONTAP to manage ONTAP file systems in AWS. For node management, the workshop helps you set up Managed Node Groups.
- **Amazon EKS advanced features**: More advanced features offered through the Amazon EKS workshop include labs for setting up:
  - Autoscaling: This includes node autoscaling (with Cluster Autoscaler or Karpenter) and workload autoscaling (with Horizontal Pod Autoscaler and Cluster Proportional Autoscaler).
  - Observability: Learn about Logging, OpenSearch, Container Insights on Amazon EKS, and Cost Visibility with Kubecost in a set of Observability labs.
  - Security: This set of Security labs let you explore Secrets Management, Amazon GuardDuty, Pod Security Standards, and Kyverno policy management.
  - Networking: Learn networking features for Amazon EKS from Networking labs that include Amazon VPC CNI (supporting network plugins) and Amazon VPC Lattice (for configuring clusters across VC and user accounts).
  - Automation: Labs on Automation step you through GitOps methods of managing your clusters and projects like AWS Controllers for Kubernetes and Crossplane for managing Amazon EKS control planes.

Amazon EKS hands-on cluster setup tutorials

A set of Amazon EKS Cluster Setup tutorials on the AWS Community site can help you create special-purpose Amazon EKS clusters and enhance those clusters in various ways. The tutorials are divided into three different types:

Building clusters

These tutorials help you build clusters that can be used for special purposes. These special purposes include the ability to run:

- Globally scalable applications based on IPv6
- Asynchronous batch tasks
- High traffic microservices
- Autoscaling with Karpenter on Fargate
- Financial workloads
- Windows Managed Node Groups

Enhancing clusters

Once you have an existing cluster, you can extend and enhance that cluster in ways that allow it to run specialized workloads and otherwise enhance the clusters. These tutorials include ways to:

- Provide storage solutions with EFS CSI
- Provide dynamic database storage with EBS CSI
- Expose applications on IPv4 clusters using the AWS Load Balancer Controller
- Expose applications on IPv6 clusters using the AWS Load Balancer Controller

Optimizing AWS services

Using these tutorials, you can better integrate your clusters with AWS services. These tutorials include those that help you:

- Manage DNS records for microservices with ExternalDNS
- Monitor applications with CloudWatch
- Manage asynchronous tasks with SQS and EFS storage

- Consume AWS Secrets Manager Secrets from workloads
- Set up mTLS with Fargate, NGINX, and ACM PCA

Amazon EKS Samples

The Amazon EKS Samples repository stores manifests to use with Amazon EKS. These manifests give you the opportunity to try out different kinds of applications in Amazon EKS or create specific types of Amazon EKS clusters. Samples include manifests to:

- Create an AWS Amazon EKS Fargate cluster
- Create a cluster with an existing IAM role
- Add and Ubuntu Managed Node Group to a cluster
- Backup and restore Pod storage with volume snapshots
- Recover EBS volumes mounted as PVCs with multiple accounts
- Enable proxy protocol for NGINX Ingress Controller with Classic Load Balancers
- Configure Logging on Fargate to AWS OpenSearch
- Run Python SDK with a web federated identity provider
- Deploy a sample app on an NFS CSI controller
- Use volume snapshots for StatefulSets
- Deploy pods across nodes on different availability zones

Keep in mind that these samples are for learning and testing purposes only and are not intended to be used in production.

AWS Tutorials

The AWS Tutorials site publishes a few Amazon EKS tutorials, but also offers a search tool to find other tutorials published on AWS sites (such as the AWS Community site). Amazon EKS tutorials published directly on this site include:

- Deploy a Container Web App on Amazon EKS
- Run Kubernetes clusters for less (Amazon EKS and Spot instances)
- How to cost optimize Jenkins jobs on Kubernetes

Developers Workshop

If you are a software developer, looking to create or refactor applications to run on Amazon EKS, the Amazon EKS Developers workshopis a good place to start. The workshop not only helps you build containerized applications, but also helps you deploy those containers to a container registry (ECR) and from there to an Amazon EKS cluster.

Start with the Amazon EKS Python Workshop to go through the process of refactoring a python application, then set up your development environment to prepare for deploying the application. Step through sections on Containers, Kubernetes, and Amazon EKS to prepare to run your containerized applications in those environments.

Terraform Workshop

While `eksctl` is a simple tool for creating a cluster, for more complex infrastructure-as-code types of Amazon EKS deployments, Terraform is a popular Amazon EKS cluster creation and management tool. The Terraform Amazon EKS Workshop teaches how to use Terraform to build an AWS VPC, create Amazon EKS clusters, and add optional enhancements to your cluster. In particular, there is a section for creating a private Amazon EKS cluster

AWS Amazon EKS Training

AWS offers formal training for learning about Amazon EKS. A three-day training course entitled Running Containers on Amazon Elastic Kubernetes Service teaches:

- Kubernetes and Amazon EKS fundamentals
- How to build Amazon EKS clusters
- Securing Amazon EKS with AWS IAM and Kubernetes RBAC authorization
- GitOps automation tools
- Monitoring tools
- Techniques for improving cost, efficiency, and resiliency

Get started with Amazon EKS

Make sure that you are set up to use Amazon EKS before going through the getting started guides. For more information, see *Set up*.

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

- Get started with Amazon EKS – eksctl – This getting started guide helps you to install all of the required resources to get started with Amazon EKS using `eksctl`, a simple command line utility for creating and managing Kubernetes clusters on Amazon EKS. At the end of the tutorial, you will have a running Amazon EKS cluster that you can deploy applications to. This is the fastest and simplest way to get started with Amazon EKS.
- Get started with Amazon EKS – AWS Management Console and AWS CLI – This getting started guide helps you to create all of the required resources to get started with Amazon EKS using the AWS Management Console and AWS CLI. At the end of the tutorial, you will have a running Amazon EKS cluster that you can deploy applications to. In this guide, you manually create each resource required for an Amazon EKS cluster. The procedures give you visibility into how each resource is created and how they interact with each other.

We also offer the following references:

- For a collection of hands-on tutorials, see EKS Cluster Setup on *AWS Community*.
- For code examples, see Code examples for Amazon EKS using AWS SDKs.

Get started with Amazon EKS – EKS Auto Mode

Like other EKS getting started experiences, creating your first cluster with EKS Auto Mode delegates the management of the cluster itself to AWS. However, EKS Auto Mode extends EKS automation by handing responsibility of many essential services needed to set up workload infrastructure (nodes, networks, and various services), making it easier to manage nodes and scale up to meet workload demands.

Choose from one of the following ways to create a cluster with EKS Auto Mode:

- the section called “ AWS CLI”: Use the `aws` command line interface to create a cluster.

- the section called "Management console": Use the AWS Management Console to create a cluster.
- the section called "eksctl CLI": Use the `eksctl` command line interface to create a cluster.

If you are comparing different approaches to creating your first EKS cluster, you should know that EKS Auto Mode has AWS take over additional cluster management responsibilities that include setting up components to:

- Start up and scale nodes as workload demand increases and decreases.
- Regularly upgrade the cluster itself (control plane), node operating systems, and services running on nodes.
- Choose default settings that determine things like the size and speed of node storage and Pod network configuration.

For details on what you get with EKS Auto Mode clusters, see *EKS Auto Mode*.

Get started with Amazon EKS – `eksctl`

> **Note**
>
> This topic covers getting started **without** EKS Auto Mode.
> EKS Auto Mode automates routine tasks for cluster compute, storage, and networking.
> Learn how to get started with Amazon EKS Auto Mode.

This guide helps you to create all of the required resources to get started with Amazon Elastic Kubernetes Service (Amazon EKS) using `eksctl`, a simple command line utility for creating and managing Kubernetes clusters on Amazon EKS. At the end of this tutorial, you will have a running Amazon EKS cluster that you can deploy applications to.

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

Prerequisites

Before starting this tutorial, you must install and configure the AWS CLI, kubectl, and eksctl tools as described in Set up to use Amazon EKS.

Step 1: Create your Amazon EKS cluster and nodes

> **Important**
>
> To get started as simply and quickly as possible, this topic includes steps to create a cluster and nodes with default settings. Before creating a cluster and nodes for production use, we recommend that you familiarize yourself with all settings and deploy a cluster and nodes with the settings that meet your requirements. For more information, see the section called "Create a cluster" and *Manage compute*. Some settings can only be enabled when creating your cluster and nodes.

You can create a cluster with one of the following node types. To learn more about each type, see *Manage compute*. After your cluster is deployed, you can add other node types.

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

Create your Amazon EKS cluster with the following command. You can replace *`my-cluster`* with your own value. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in. Replace *`region-code`* with any AWS Region that is supported by Amazon EKS. For a list of AWS Regions, see Amazon EKS endpoints and quotas in the AWS General Reference guide.

Example

Fargate - Linux

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

Managed nodes - Linux

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

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

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

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

After cluster creation is complete, view the AWS CloudFormation stack named `eksctl-my-cluster-cluster` in the AWS CloudFormation console to see all of the resources that were created.

Step 2: View Kubernetes resources

1. View your cluster nodes.

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

   An example output is as follows.

   Example

   Fargate - Linux

   ```
   NAME                                                STATUS   ROLES    AGE
    VERSION              INTERNAL-IP   EXTERNAL-IP   OS-IMAGE        KERNEL-VERSION
                    CONTAINER-RUNTIME
   ```

```
fargate-ip-192-0-2-0.region-code.compute.internal   Ready    <none>
 8m3s    v1.2.3-eks-1234567   192.0.2.0     <none>         Amazon Linux 2
 1.23.456-789.012.amzn2.x86_64   containerd://1.2.3
fargate-ip-192-0-2-1.region-code.compute.internal   Ready    <none>
 7m30s   v1.2.3-eks-1234567   192-0-2-1     <none>         Amazon Linux 2
 1.23.456-789.012.amzn2.x86_64   containerd://1.2.3
```

Managed nodes - Linux

```
NAME                                            STATUS   ROLES    AGE    VERSION
        INTERNAL-IP   EXTERNAL-IP   OS-IMAGE         KERNEL-VERSION
   CONTAINER-RUNTIME
ip-192-0-2-0.region-code.compute.internal   Ready    <none>   6m7s
  v1.2.3-eks-1234567   192.0.2.0     192.0.2.2     Amazon Linux 2
 1.23.456-789.012.amzn2.x86_64   containerd://1.2.3
ip-192-0-2-1.region-code.compute.internal   Ready    <none>   6m4s
  v1.2.3-eks-1234567   192.0.2.1     192.0.2.3     Amazon Linux 2
 1.23.456-789.012.amzn2.x86_64   containerd://1.2.3
```

For more information about what you see in the output, see the section called “Access cluster resources”.

2. View the workloads running on your cluster.

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

An example output is as follows.

Example

Fargate - Linux

```
NAMESPACE     NAME                       READY   STATUS    RESTARTS   AGE   IP
          NODE                                                NOMINATED NODE
 READINESS GATES
kube-system   coredns-1234567890-abcde   1/1     Running   0          18m
 192.0.2.0   fargate-ip-192-0-2-0.region-code.compute.internal   <none>
 <none>
kube-system   coredns-1234567890-12345   1/1     Running   0          18m
 192.0.2.1   fargate-ip-192-0-2-1.region-code.compute.internal   <none>
 <none>
```

Managed nodes - Linux

```
NAMESPACE     NAME                          READY   STATUS    RESTARTS   AGE     IP
         NODE                                         NOMINATED NODE   READINESS
 GATES
kube-system   aws-node-12345                1/1     Running   0          7m43s
 192.0.2.1   ip-192-0-2-1.region-code.compute.internal   <none>           <none>
kube-system   aws-node-67890                1/1     Running   0          7m46s
 192.0.2.0   ip-192-0-2-0.region-code.compute.internal   <none>           <none>
kube-system   coredns-1234567890-abcde      1/1     Running   0          14m
 192.0.2.3   ip-192-0-2-3.region-code.compute.internal   <none>           <none>
kube-system   coredns-1234567890-12345      1/1     Running   0          14m
 192.0.2.4   ip-192-0-2-4.region-code.compute.internal   <none>           <none>
kube-system   kube-proxy-12345              1/1     Running   0          7m46s
 192.0.2.0   ip-192-0-2-0.region-code.compute.internal   <none>           <none>
kube-system   kube-proxy-67890              1/1     Running   0          7m43s
 192.0.2.1   ip-192-0-2-1.region-code.compute.internal   <none>           <none>
```

For more information about what you see in the output, see the section called “Access cluster resources”.

Step 3: Delete your cluster and nodes

After you’ve finished with the cluster and nodes that you created for this tutorial, you should clean up by deleting the cluster and nodes with the following command. If you want to do more with this cluster before you clean up, see the section called “Next steps”.

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

Next steps

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

- Deploy a sample application to your cluster.
- The IAM principal that created the cluster is the only principal that can make calls to the Kubernetes API server with `kubectl` or the AWS Management Console. If you want other IAM principals to have access to your cluster, then you need to add them. For more information, see the section called “Kubernetes API access” and the section called “Required permissions”.

- Before deploying a cluster for production use, we recommend familiarizing yourself with all of the settings for clusters and nodes. Some settings (such as enabling SSH access to Amazon EC2 nodes) must be made when the cluster is created.
- To increase security for your cluster, configure the Amazon VPC Container Networking Interface plugin to use IAM roles for service accounts.

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

> **Note**
>
> This topic covers getting started **without** EKS Auto Mode. It uses Managed Node Groups to deploy nodes.
> EKS Auto Mode automates routine tasks for cluster compute, storage, and networking. Learn how to get started with Amazon EKS Auto Mode. EKS Auto Mode is the preferred method of deploying nodes.

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

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

Prerequisites

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

- **AWS CLI** – A command line tool for working with AWS services, including Amazon EKS. For more information, see Installing in the AWS Command Line Interface User Guide. After installing the AWS CLI, we recommend that you also configure it. For more information, see Quick configuration with aws configure in the AWS Command Line Interface User Guide. Note that AWS CLI v2 is required to use the **update-kubeconfig** option shown in this page.

- **`kubectl`** – A command line tool for working with Kubernetes clusters. For more information, see the section called "Set up kubectl and eksctl".
- **Required IAM permissions** – The IAM security principal that you're using must have permissions to work with Amazon EKS IAM roles, service linked roles, AWS CloudFormation, a VPC, and related resources. For more information, see Actions and Using service-linked roles in the IAM User Guide. You must complete all steps in this guide as the same user. To check the current user, run the following command:

```
aws sts get-caller-identity
```

We recommend that you complete the steps in this topic in a Bash shell. If you aren't using a Bash shell, some script commands such as line continuation characters and the way variables are set and used require adjustment for your shell. Additionally, the quoting and escaping rules for your shell might be different. For more information, see Using quotation marks with strings in the AWS CLI in the AWS Command Line Interface User Guide.

Step 1: Create your Amazon EKS cluster

> **Important**
>
> To get started as simply and quickly as possible, this topic includes steps to create a cluster with default settings. Before creating a cluster for production use, we recommend that you familiarize yourself with all settings and deploy a cluster with the settings that meet your requirements. For more information, see the section called "Create a cluster". Some settings can only be enabled when creating your cluster.

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

```
aws cloudformation create-stack \
  --region region-code \
```

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

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

2. Create a cluster IAM role and attach the required Amazon EKS IAM managed policy to it. Kubernetes clusters managed by Amazon EKS make calls to other AWS services on your behalf to manage the resources that you use with the service.

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

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

   b. Create the role.

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

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

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

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

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

4. Choose **Create cluster**. If you don't see this option, then choose **Clusters** in the left navigation pane first.
5. On the **Configure cluster** page, do the following:
   a. Select **Custom configuration** and disable **Use EKS Auto Mode**. (If you prefer an EKS Auto Mode cluster, refer instead to the section called "Management console".)
   b. Enter a **Name** for your cluster, such as *my-cluster*. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.
   c. For **Cluster Service Role**, choose *myAmazonEKSClusterRole*.
   d. Leave the remaining settings at their default values and choose **Next**.
6. On the **Specify networking** page, do the following:
   a. Choose the ID of the VPC that you created in a previous step from the **VPC** dropdown list. It is something like *vpc-00x0000x000x0x000* | *my-eks-vpc-stack-VPC*.
   b. Leave the remaining settings at their default values and choose **Next**.
7. On the **Configure observability** page, choose **Next**.
8. On the **Select add-ons** page, choose **Next**.

   For more information on add-ons, see the section called "Amazon EKS add-ons".
9. On the **Configure selected add-ons settings** page, choose **Next**.
10. On the **Review and create** page, choose **Create**.

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

> **Note**
>
> You might receive an error that one of the Availability Zones in your request doesn't have sufficient capacity to create an Amazon EKS cluster. If this happens, the error output contains the Availability Zones that can support a new cluster. Retry creating your cluster with at least two subnets that are located in the supported Availability

Zones for your account. For more information, see the section called "Insufficient capacity".

Step 2: Configure your computer to communicate with your cluster

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

Before proceeding, be sure that your cluster creation completed successfully in Step 1.

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

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

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

2. Test your configuration.

```
kubectl get svc
```

> **Note**
>
> If you receive any authorization or resource type errors, see the section called "Unauthorized or access denied (`kubectl`)" in the troubleshooting topic.

   An example output is as follows.

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

Step 3: Create nodes

> **Important**
>
> To get started as simply and quickly as possible, this topic includes steps to create nodes with mostly default settings. Before creating nodes for production use, we recommend that you familiarize yourself with all settings and deploy nodes with the settings that meet your requirements. For more information, see *Manage compute*. Some settings can only be enabled when creating your nodes.

This procedure configures your cluster to use Managed node groups to create nodes, specifying the subnets and node IAM role that you created in previous steps. It lets you run Amazon Linux applications on Amazon EC2 instances.

To learn more about different ways to configure nodes in EKS, see *Manage compute*. After your cluster is deployed, you can add other node types. Though not covered in this guide, you can also add Windows self-managed and Bottlerocket nodes to your cluster.

To create your EC2 Linux managed node group

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

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

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

   b. Create the node IAM role.

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

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

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

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

e. Choose the name of the cluster that you created in [Step 1: Create your Amazon EKS cluster](), such as *`my-cluster`*.

f. On the ***`my-cluster`*** page, do the following:

g. Choose the **Compute** tab.

h. Choose **Add Node Group**.

2. On the **Configure Node Group** page, do the following:

   a. For **Name**, enter a unique name for your managed node group, such as *`my-nodegroup`*. The node group name can't be longer than 63 characters. It must start with letter or digit, but can also include hyphens and underscores for the remaining characters.

   b. For **Node IAM role name**, choose *`myAmazonEKSNodeRole`* role that you created in a previous step. We recommend that each node group use its own unique IAM role.

   c. Choose **Next**.

3. On the **Set compute and scaling configuration** page, accept the default values and choose **Next**.

4. On the **Specify networking** page, accept the default values and choose **Next**.

5. On the **Review and create** page, review your managed node group configuration and choose **Create**.

6. After several minutes, the **Status** in the **Node Group configuration** section will change from **Creating** to **Active**. Don't continue to the next step until the status is **Active**.

Step 4: View resources

You can view your nodes and Kubernetes workloads.

1. In the left navigation pane, choose **Clusters**. In the list of **Clusters**, choose the name of the cluster that you created, such as *my-cluster*.
2. On the ***my-cluster*** page, choose the following:
   a. **Compute** tab – You see the list of **Nodes** that were deployed for the cluster. You can choose the name of a node to see more information about it.
   b. **Resources** tab – You see all of the Kubernetes resources that are deployed by default to an Amazon EKS cluster. Select any resource type in the console to learn more about it.

Step 5: Delete resources

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

1. Delete any node groups profiles that you created.
   a. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
   b. In the left navigation pane, choose **Clusters**. In the list of clusters, choose *my-cluster*.
   c. Choose the **Compute** tab.
   d. If you created a node group, choose the *my-nodegroup* node group and then choose **Delete**. Enter *my-nodegroup*, and then choose **Delete**.
   e. Don't continue until the node group profiles are deleted.
2. Delete the cluster.
   a. In the left navigation pane, choose **Clusters**. In the list of clusters, choose *my-cluster*.
   b. Choose **Delete cluster**.
   c. Enter *my-cluster* and then choose **Delete**. Don't continue until the cluster is deleted.
3. Delete the VPC AWS CloudFormation stack that you created.
   a. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation/.
   b. Choose the *my-eks-vpc-stack* stack, and then choose **Delete**.
   c. In the **Delete *my-eks-vpc-stack*** confirmation dialog box, choose **Delete stack**.
4. Delete the IAM roles that you created.

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

b. In the left navigation pane, choose **Roles**.

c. Select each role you created from the list (***myAmazonEKSClusterRole*** , as well as *myAmazonEKSNodeRole*). Choose **Delete**, enter the requested confirmation text, then choose **Delete**.

Next steps

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

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

Automate cluster infrastructure with EKS Auto Mode

EKS Auto Mode extends AWS management of Kubernetes clusters beyond the cluster itself, to allow AWS to also set up and manage the infrastructure that enables the smooth operation of your workloads. You can delegate key infrastructure decisions and leverage the expertise of AWS for day-to-day operations. Cluster infrastructure managed by AWS includes many Kubernetes capabilities as core components, as opposed to add-ons, such as compute autoscaling, pod and service networking, application load balancing, cluster DNS, block storage, and GPU support.

To get started, you can deploy a new EKS Auto Mode cluster or enable EKS Auto Mode on an existing cluster. You can deploy, upgrade, or modify your EKS Auto Mode clusters using eksctl, the AWS CLI, the AWS Management Console, EKS APIs, or your preferred infrastructure-as-code tools.

With EKS Auto Mode, you can continue using your preferred Kubernetes-compatible tools. EKS Auto Mode integrates with AWS services like Amazon EC2, Amazon EBS, and ELB, leveraging AWS cloud resources that follow best practices. These resources are automatically scaled, cost-optimized, and regularly updated to help minimize operational costs and overhead.

Features

EKS Auto Mode provides the following high-level features:

Streamline Kubernetes Cluster Management: EKS Auto Mode streamlines EKS management by providing production-ready clusters with minimal operational overhead. With EKS Auto Mode, you can run demanding, dynamic workloads confidently, without requiring deep EKS expertise.

Application Availability: EKS Auto Mode dynamically adds or removes nodes in your EKS cluster based on the demands of your Kubernetes applications. This minimizes the need for manual capacity planning and ensures application availability.

Efficiency: EKS Auto Mode is designed to optimize compute costs while adhering to the flexibility defined by your NodePool and workload requirements. It also terminates unused instances and consolidates workloads onto other nodes to improve cost efficiency.

Security: EKS Auto Mode uses AMIs that are treated as immutable for your nodes. These AMIs enforce locked-down software, enable SELinux mandatory access controls, and provide read-only root file systems. Additionally, nodes launched by EKS Auto Mode have a maximum lifetime of 21 days (which you can reduce), after which they are automatically replaced with new nodes. This approach enhances your security posture by regularly cycling nodes, aligning with best practices already adopted by many customers.

Automated Upgrades: EKS Auto Mode keeps your Kubernetes cluster, nodes, and related components up to date with the latest patches, while respecting your configured Pod Disruption Budgets (PDBs) and NodePool Disruption Budgets (NDBs). Up to the 21-day maximum lifetime, intervention might be required if blocking PDBs or other configurations prevent updates.

Managed Components: EKS Auto Mode includes Kubernetes and AWS cloud features as core components that would otherwise have to be managed as add-ons. This includes built-in support for Pod IP address assignments, Pod network policies, local DNS services, GPU plug-ins, health checkers, and EBS CSI storage.

Customizable NodePools and NodeClasses: If your workload requires changes to storage, compute, or networking configurations, you can create custom NodePools and NodeClasses using EKS Auto Mode. While default NodePools and NodeClasses can't be edited, you can add new custom NodePools or NodeClasses alongside the default configurations to meet your specific requirements.

Automated Components

EKS Auto Mode streamlines the operation of your Amazon EKS clusters by automating key infrastructure components. Enabling EKS Auto Mode further reduces the tasks to manage your EKS clusters.

The following is a list of data plane components that are automated:

- **Compute**: For many workloads, with EKS Auto Mode you can forget about many aspects of compute for your EKS clusters. These include:
  - **Nodes**: EKS Auto Mode nodes are designed to be treated like appliances. EKS Auto Mode does the following:
    - Chooses an appropriate AMI that's configured with many services needed to run your workloads without intervention.
    - Locks down those features using SELinux enforcing mode and a read-only root file system.
    - Prevents direct access to the nodes by disallowing SSH or SSM access.
    - Includes GPU support, with separate kernel drivers and plugins for NVIDIA and Neuron GPUs, enabling high-performance workloads.

  - **Auto scaling**: Relying on Karpenter auto scaling, EKS Auto Mode monitors for unschedulable Pods and makes it possible for new nodes to be deployed to run those pods. As workloads are terminated, EKS Auto Mode dynamically disrupts and terminates nodes when they are no longer needed, optimizing resource usage.
  - **Upgrades**: Taking control of your nodes streamlines EKS Auto Mode's ability to provide security patches and operating system and component upgrades as needed. Those upgrades are designed to provide minimal disruption of your workloads. EKS Auto Mode enforces a 21-day maximum node lifetime to ensure up-to-date software and APIs.
- **Load balancing**: EKS Auto Mode streamlines load balancing by integrating with Amazon's Elastic Load Balancing service, automating the provisioning and configuration of load balancers for Kubernetes Services and Ingress resources. It supports advanced features for both Application and Network Load Balancers, manages their lifecycle, and scales them to match cluster demands. This integration provides a production-ready load balancing solution adhering to AWS best practices, allowing you to focus on applications rather than infrastructure management.
- **Storage**: EKS Auto Mode configures ephemeral storage for you by setting up volume types, volume sizes, encryption policies, and deletion policies upon node termination.
- **Networking**: EKS Auto Mode automates critical networking tasks for Pod and service connectivity. This includes IPv4/IPv6 support and the use of secondary CIDR blocks for extending IP address spaces.
- **Identity and Access Management**: You do not have to install the EKS Pod Identity Agent on EKS Auto Mode clusters.

For more information about these components, see the section called "How it works".

Configuration

While EKS Auto Mode will effectively manage most of your data plane services without your intervention, there might be times when you want to change the behavior of some of those services. You can modify the configuration of your EKS Auto Mode clusters in the following ways:

- **Kubernetes DaemonSets**: Rather than modify services installed on your nodes, you can instead use Kubernetes daemonsets. Daemonsets are designed to be managed by Kubernetes, but run on every node in the cluster. In this way, you can add special services for monitoring or otherwise watching over your nodes.

- **Custom NodePools and NodeClasses**: Default NodePools and NodeClasses are configured by EKS Auto Mode and can't be edited. To customize node behavior, you can create additional NodePools or NodeClasses for use cases such as:
  - Selecting specific instance types (for example, accelerated processors or EC2 Spot instances).
  - Isolating workloads for security or cost-tracking purposes.
  - Configuring ephemeral storage settings like IOPS, size, and throughput.
- **Load Balancing**: Some services, such as load balancing, that EKS Auto Mode runs as Kubernetes objects, can be configured directly on your EKS Auto Mode clusters.

For more information about options for configuring EKS Auto Mode, see the section called "Configure".

Create a cluster with Amazon EKS Auto Mode

This chapter explains how to create an Amazon EKS cluster with Auto Mode enabled using various tools and interfaces. Auto Mode simplifies cluster creation by automatically configuring and managing the cluster's compute, networking, and storage infrastructure. You'll learn how to create an Auto Mode cluster using the AWS CLI, AWS Management Console, or the eksctl command line tool.

> **Note**
>
> EKS Auto Mode requires Kubernetes version 1.29 or greater. EKS Auto Mode isn't available in the `ap-southeast-7` or `mx-central-1` AWS Regions.

Choose your preferred tool based on your needs: The AWS Management Console provides a visual interface ideal for learning about EKS Auto Mode features and creating individual clusters. The AWS CLI is best suited for scripting and automation tasks, particularly when integrating cluster creation into existing workflows or CI/CD pipelines. The eksctl CLI offers a Kubernetes-native experience and is recommended for users familiar with Kubernetes tooling who want simplified command line operations with sensible defaults.

Before you begin, ensure you have the necessary prerequisites installed and configured, including appropriate IAM permissions to create EKS clusters. To learn how to install CLI tools such as `kubectl`, `aws`, and `eksctl`, see *Set up*.

You can use the AWS CLI, AWS Management Console, or eksctl CLI to create a cluster with Amazon EKS Auto Mode.

Topics

Create an EKS Auto Mode Cluster with the eksctl CLI

This topic shows you how to create an Amazon EKS Auto Mode cluster using the eksctl command line interface (CLI). You can create an Auto Mode cluster either by running a single CLI command or by applying a YAML configuration file. Both methods provide the same functionality, with the YAML approach offering more granular control over cluster settings.

The eksctl CLI simplifies the process of creating and managing EKS Auto Mode clusters by handling the underlying AWS resource creation and configuration. Before proceeding, ensure you have the necessary AWS credentials and permissions configured on your local machine. This guide assumes you're familiar with basic Amazon EKS concepts and have already installed the required CLI tools.

> **Note**
>
> You must install version `0.195.0` or greater of eksctl. For more information, see eksctl releases on GitHub.

Create an EKS Auto Mode cluster with a CLI command

You must have the `aws` and `eksctl` tools installed. You must be logged into the AWS CLI with sufficent permissions to manage AWS resources including: EC2 instances, EC2 networking, EKS clusters, and IAM roles. For more information, see *Set up*.

Run the following command to create a new EKS Auto Mode cluster with

```
eksctl create cluster --name=<cluster-name> --enable-auto-mode
```

Create an EKS Auto Mode cluster with a YAML file

You must have the `aws` and `eksctl` tools installed. You must be logged into the AWS CLI with sufficent permissions to manage AWS resources including: EC2 instances, EC2 networking, EKS clusters, and IAM roles. For more information, see *Set up*.

Review the EKS Auto Mode configuration options in the sample ClusterConfig resource below. For the full ClusterConfig specification, see the eksctl documentation.

AWS suggests enabling EKS Auto Mode. If this is your first time creating an EKS Auto Mode cluster, leave the `nodeRoleARN` unspecified to create a Node IAM Role for EKS Auto Mode. If you already have a Node IAM Role in your AWS account, AWS suggests reusing it.

AWS suggests not specifying any value for `nodePools`. EKS Auto Mode will create default node pools. You can use the Kubernetes API to create additional node pools.

```
# cluster.yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: <cluster-name>
  region: <aws-region>

iam:
  # ARN of the Cluster IAM Role
  # optional, eksctl creates a new role if not supplied
  # suggested to use one Cluster IAM Role per account
  serviceRoleARN: <arn-cluster-iam-role>

autoModeConfig:
  # defaults to false
  enabled: boolean
  # optional, defaults to [general-purpose, system].
  # suggested to leave unspecified
  # To disable creation of nodePools, set it to the empty array ([]).
  nodePools: []string
  # optional, eksctl creates a new role if this is not supplied
  # and nodePools are present.
  nodeRoleARN: string
```

Save the `ClusterConfig` file as `cluster.yaml`, and use the following command to create the cluster:

```
eksctl create cluster -f cluster.yaml
```

Create an EKS Auto Mode Cluster with the AWS CLI

EKS Auto Mode Clusters automate routine cluster management tasks for compute, storage, and networking. For example, EKS Auto Mode Clusters automatically detect when additional nodes are required and provision new EC2 instances to meet workload demands.

This topic guides you through creating a new EKS Auto Mode Cluster using the AWS CLI and optionally deploying a sample workload.

Prerequisites

- The latest version of the AWS Command Line Interface (AWS CLI) installed and configured on your device. To check your current version, use `aws --version`. To install the latest version, see Installing and Quick configuration with aws configure in the AWS Command Line Interface User Guide.
  - Login to the CLI with sufficent IAM permissions to create AWS resources including IAM Policies, IAM Roles, and EKS Clusters.
- The kubectl command line tool installed on your device. AWS suggests you use the same kubectl version as the Kubernetes version of your EKS Cluster. To install or upgrade kubectl, see the section called "Set up kubectl and eksctl".

Specify VPC subnets

Amazon EKS Auto Mode deploy nodes to VPC subnets. When creating an EKS cluster, you must specify the VPC subnets where the nodes will be deployed. You can use the default VPC subnets in your AWS account or create a dedicated VPC for critical workloads.

- AWS suggests creating a dedicated VPC for your cluster. Learn how to the section called "Create a VPC".
- The EKS Console assists with creating a new VPC. Learn how to the section called "Management console".

- Alternatively, you can use the default VPC of your AWS account. Use the following instructions to find the Subnet IDs.

To find the Subnet IDs of your default VPC

Using the AWS CLI:

1. Run the following command to list the default VPC and its subnets:

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$(aws ec2 describe-vpcs
 --query 'Vpcs[?IsDefault==`true`].VpcId' --output text)" --query 'Subnets[*].
{ID:SubnetId,AZ:AvailabilityZone}' --output table
```

2. Save the output and note the **Subnet IDs**.

   Sample output:

```
----------------------------------------
|             DescribeSubnets          |
----------------------------------------
|   SubnetId        |   AvailabilityZone  |
|--------------------|---------------------|
|   subnet-012345678 |   us-west-2a        |
|   subnet-234567890 |   us-west-2b        |
|   subnet-345678901 |   us-west-2c        |
----------------------------------------
```

IAM Roles for EKS Auto Mode Clusters

Cluster IAM Role

EKS Auto Mode requires a Cluster IAM Role to perform actions in your AWS account, such as provisioning new EC2 instances. You must create this role to grant EKS the necessary permissions. AWS recommends attaching the following AWS managed policies to the Cluster IAM Role:

- AmazonEKSComputePolicy
- AmazonEKSBlockStoragePolicy
- AmazonEKSLoadBalancingPolicy
- AmazonEKSNetworkingPolicy

- AmazonEKSClusterPolicy

Node IAM Role

When you create an EKS Auto Mode cluster, you specify a Node IAM Role. When EKS Auto Mode creates nodes to process pending workloads, each new EC2 instance node is assigned the Node IAM Role. This role allows the node to communicate with EKS but is generally not accessed by workloads running on the node.

If you want to grant permissions to workloads running on a node, use EKS Pod Identity. For more information, see the section called “Pod Identity”.

You must create this role and attach the following AWS managed policy:

- AmazonEKSWorkerNodeMinimalPolicy
- AmazonEC2ContainerRegistryPullOnly

Service-Linked Role

EKS Auto Mode also requires a Service-Linked Role, which is automatically created and configured by AWS. For more information, see AWSServiceRoleForAmazonEKS.

Create an EKS Auto Mode Cluster IAM Role

Step 1: Create the Trust Policy

Create a trust policy that allows the Amazon EKS service to assume the role. Save the policy as `trust-policy.json`:

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

```
      ]
    }
  ]
}
```

Step 2: Create the IAM Role

Use the trust policy to create the Cluster IAM Role:

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

Step 3: Note the Role ARN

Retrieve and save the ARN of the new role for use in subsequent steps:

```
aws iam get-role --role-name AmazonEKSAutoClusterRole --query "Role.Arn" --output text
```

Step 4: Attach Required Policies

Attach the following AWS managed policies to the Cluster IAM Role to grant the necessary permissions:

AmazonEKSClusterPolicy:

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

AmazonEKSComputePolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSComputePolicy
```

AmazonEKSBlockStoragePolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
```

```
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSBlockStoragePolicy
```

AmazonEKSLoadBalancingPolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSLoadBalancingPolicy
```

AmazonEKSNetworkingPolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSNetworkingPolicy
```

Create an EKS Auto Mode Node IAM Role

Step 1: Create the Trust Policy

Create a trust policy that allows the Amazon EKS service to assume the role. Save the policy as `node-trust-policy.json`:

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

Step 2: Create the Node IAM Role

Use the **node-trust-policy.json** file from the previous step to define which entities can assume the role. Run the following command to create the Node IAM Role:

```
aws iam create-role \
```

```
    --role-name AmazonEKSAutoNodeRole \
    --assume-role-policy-document file://node-trust-policy.json
```

Step 3: Note the Role ARN

After creating the role, retrieve and save the ARN of the Node IAM Role. You will need this ARN in subsequent steps. Use the following command to get the ARN:

```
aws iam get-role --role-name AmazonEKSAutoNodeRole --query "Role.Arn" --output text
```

Step 4: Attach Required Policies

Attach the following AWS managed policies to the Node IAM Role to provide the necessary permissions:

AmazonEKSWorkerNodeMinimalPolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodeMinimalPolicy
```

AmazonEC2ContainerRegistryPullOnly:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

Create an EKS Auto Mode Cluster

Overview

To create an EKS Auto Mode Cluster using the AWS CLI, you will need the following parameters:

- `cluster-name`: The name of the cluster.
- `k8s-version`: The Kubernetes version (e.g., 1.31).
- `subnet-ids`: Subnet IDs identified in the previous steps.
- `cluster-role-arn`: ARN of the Cluster IAM Role.
- `node-role-arn`: ARN of the Node IAM Role.

Default Cluster Configurations

Review these default values and features before creating the cluster:

- `nodePools`: EKS Auto Mode includes general-purpose and system default Node Pools. Learn more about [Node Pools](#).

Note: Node Pools in EKS Auto Mode differ from Amazon EKS Managed Node Groups but can coexist in the same cluster.

- `computeConfig.enabled`: Automates routine compute tasks, such as creating and deleting EC2 instances.
- `kubernetesNetworkConfig.elasticLoadBalancing.enabled`: Automates load balancing tasks, including creating and deleting Elastic Load Balancers.
- `storageConfig.blockStorage.enabled`: Automates storage tasks, such as creating and deleting Amazon EBS volumes.
- `accessConfig.authenticationMode`: Requires EKS access entries. Learn more about [EKS authentication modes](#).

Run the Command

Use the following command to create the cluster:

```
aws eks create-cluster \
  --region ${AWS_REGION} \
  --cli-input-json \
  "{
      \"name\": \"${CLUSTER_NAME}\",
      \"version\": \"${K8S_VERSION}\",
      \"roleArn\": \"${CLUSTER_ROLE_ARN}\",
      \"resourcesVpcConfig\": {
        \"subnetIds\": ${SUBNETS_JSON},
        \"endpointPublicAccess\": true,
        \"endpointPrivateAccess\": true
      },
      \"computeConfig\": {
        \"enabled\": true,
        \"nodeRoleArn\":\"${NODE_ROLE_ARN}\",
        \"nodePools\": [\"general-purpose\", \"system\"]
      },
```

```
      \"kubernetesNetworkConfig\": {
        \"elasticLoadBalancing\": {
          \"enabled\": true
        }
      },
      \"storageConfig\": {
        \"blockStorage\": {
          \"enabled\": true
        }
      },
      \"accessConfig\": {
        \"authenticationMode\": \"API\"
      }
    }
```

Check Cluster Status

Step 1: Verify Cluster Creation

Run the following command to check the status of your cluster. Cluster creation typically takes about 15 minutes:

```
aws eks describe-cluster --name "${CLUSTER_NAME}" --output json
```

Step 2: Update kubeconfig

Once the cluster is ready, update your local kubeconfig file to enable `kubectl` to communicate with the cluster. This configuration uses the AWS CLI for authentication.

```
aws eks update-kubeconfig --name "${CLUSTER_NAME}"
```

Step 3: Verify Node Pools

List the Node Pools in your cluster using the following command:

```
kubectl get nodepools
```

Next Steps

- Learn how to deploy a sample workload to your new EKS Auto Mode cluster.

Create an EKS Auto Mode Cluster with the AWS Management Console

Creating an EKS Auto Mode cluster in the AWS Management Console requires less configuration than other options. EKS integrates with AWS IAM and VPC Networking to help you create the resources associated with an EKS cluster.

You have two options to create a cluster in the console:

- Quick configuration (with EKS Auto Mode)
- Custom configuration

In this topic, you will learn how to create an EKS Auto Mode cluster using the Quick configuration option.

Create an EKS Auto Mode using the quick configuration option

You must be logged into the AWS Management Console with sufficent permissions to manage AWS resources including: EC2 instances, EC2 networking, EKS clusters, and IAM roles.

1. Navigate to the EKS Console
2. Click **Create cluster**
3. Confirm the **Quick configuration** option is selected
4. Determine the following values, or use the defaults for a test cluster.
   - Cluster **Name**
   - Kubernetes Version
5. Select the Cluster IAM Role. If this is your first time creating an EKS Auto Mode cluster, use the **Create recommended role** option.
   - Optionally, you can reuse a single Cluster IAM Role in your AWS account for all EKS Auto Mode clusters.
   - The Cluster IAM Role includes required permissions for EKS Auto Mode to manage resources including EC2 instances, EBS volumes, and EC2 load balancers.
   - The **Create recommended role** option pre-fills all fields with recommended values. Select **Next** and then **Create**. The role will use the suggested `AmazonEKSAutoClusterRole` name.
   - If you recently created a new role, use the **Refresh** icon to reload the role selection dropdown.
6. Select the Node IAM Role. If this is your first time creating an EKS Auto Mode cluster, use the **Create recommended role** option.

- Optionally, you can reuse a single Node IAM Role in your AWS account for all EKS Auto Mode clusters.
- The Node IAM Role includes required permissions for Auto Mode nodes to connect to the cluster. The Node IAM Role must include permissions to retrieve ECR images for your containers.
- The **Create recommended role** option pre-fills all fields with recommended values. Select **Next** and then **Create**. The role will use the suggested `AmazonEKSAutoNodeRole` name.
- If you recently created a new role, use the **Refresh** icon to reload the role selection dropdown.

7. Select the VPC for your EKS Auto Mode cluster. Choose the **Create VPC** to create a new VPC for EKS, or choose a VPC you previously created for EKS.
   - If you use the VPC Console to create a new VPC, AWS suggests you create at least one NAT Gateway per Availability Zone. Otherwise, you can use all other defaults.
   - For more information and details of IPv6 cluster requirements, see the section called “Create a VPC”.
8. (optional) EKS Auto Mode automatically populates the private subnets for your selected VPC. You can remove unwanted subnets.
   - EKS automatically selects private subnets from the VPC following best practices. You can optionally select additional subnets from the VPC, such as public subnets.
9. (optional) Select **View quick configuration defaults** to review all configuration values for the new cluster. The table indicates some values are not editable after the cluster is created.

10Select **Create cluster** . Note it may take fifteen minutes for cluster creation to complete.

Next Steps

- Learn how to Deploy a Sample Workload to your EKS Auto Mode cluster

Enable EKS Auto Mode on existing EKS clusters

You can enable EKS Auto Mode on existing EKS Clusters.

Note

EKS Auto Mode requires Kubernetes version 1.29 or greater. EKS Auto Mode isn't available in the `ap-southeast-7` or `mx-central-1` AWS Regions.

AWS supports the following migrations:

- Migrating from Karpenter to EKS Auto Mode Nodes
  - Learn how to the section called "Migrate from Karpenter"
- Migrating from EKS Managed Node Groups to EKS Auto Mode Nodes
  - Learn how to the section called "Migrate from MNGs"
- Migrating from EKS Fargate to EKS Auto Mode Nodes

AWS does not support the following migrations:

- Migrating volumes from the EBS CSI Controller to EKS Auto Mode Block Storage
  - You can install the EBS CSI controller on an Amazon EKS Auto Mode cluster. Use a `StorageClass` to associate volumes with either the EBS CSI Controller or EKS Auto Mode.
- Migrating load balancers from the AWS Load Balancer Controller to EKS Auto Mode
  - You can install the AWS Load Balancer Controller on an Amazon EKS Auto Mode cluster. Use the `IngressClass` or `loadBalancerClass` options to associate Service and Ingress resources with either the Load Balancer Controller or EKS Auto Mode.
- Migrating EKS Clusters with alternative CNIs or other unsupported networking configurations

Migration Reference

Use the following migration reference to configure Kubernetes Resources to be owned by either self-managed controllers or EKS Auto Mode.

Capability	Resource	Field	Self Managed	EKS Auto Mode
Block Storage	`StorageClass`	`provisioner`	`kubernetes.io/aws-ebs`	`ebs.csi.eks.amazonaws.com`
Load Balancing	`Service`	`loadBalancerClass`	`service.k8s.aws/nlb`	`eks.amazonaws.com/nlb`

Capability	Resource	Field	Self Managed	EKS Auto Mode
Load Balancing	`IngressClass`	controller	`ingress.k8s.aws/alb`	`eks.amazonaws.com/alb`
Load Balancing	`IngressClassParams`	`apiversion`	`elbv2.k8s.aws/v1beta1`	`eks.amazonaws.com/v1`
Load Balancing	`TargetGroupBinding`	`apiversion`	`elbv2.k8s.aws/v1beta1`	`eks.amazonaws.com/v1`
Compute	NodeClass	`apiVersion`	`karpenter.sh/v1alpha5`	`eks.amazonaws.com/v1`

Load Balancer Migration

You cannot directly transfer existing load balancers from the self-managed AWS load balancer controller to EKS Auto Mode. Instead, you must implement a blue-green deployment strategy. This involves maintaining your existing load balancer configuration while creating new load balancers under the managed controller.

To minimize service disruption, we recommend a DNS-based traffic shifting approach. First, create new load balancers using EKS Auto Mode while keeping your existing configuration operational. Then, use DNS routing (such as Route 53) to gradually shift traffic from the old load balancers to the new ones. Once traffic has been successfully migrated and you've verified the new configuration, you can decommission the old load balancers and self-managed controller.

Enable EKS Auto Mode on an existing cluster

This topic describes how to enable Amazon EKS Auto Mode on your existing Amazon EKS clusters. Enabling Auto Mode on an existing cluster requires updating IAM permissions and configuring core EKS Auto Mode settings. Once enabled, you can begin migrating your existing compute workloads to take advantage of Auto Mode's simplified operations and automated infrastructure management.

> **Important**
>
> Verify you have the minimum required version of certain Amazon EKS Add-ons installed before enabling EKS Auto Mode. For more information, see the section called "Required Add-on Versions".

Before you begin, ensure you have administrator access to your Amazon EKS cluster and permissions to modify IAM roles. The steps in this topic guide you through enabling Auto Mode using either the AWS Management Console or AWS CLI.

AWS Management Console

You must be logged into the AWS console with permission to manage IAM, EKS, and EC2 resources.

> **Note**
>
> The Cluster IAM role of an EKS Cluster cannot be changed after the cluster is created. EKS Auto Mode requires additional permissions on this role. You must attach additional policies to the current role.

Update Cluster IAM Role

1. Open your cluster overview page in the AWS Management Console.
2. Under **Cluster IAM role ARN**, select **View in IAM**.
3. From the **Add Permissions** dropdown, select **Attach Policies**.
4. Use the **Search** box to find and select the following policies:
   - `AmazonEKSComputePolicy`
   - `AmazonEKSBlockStoragePolicy`
   - `AmazonEKSLoadBalancingPolicy`
   - `AmazonEKSNetworkingPolicy`
   - `AmazonEKSClusterPolicy`
5. Select **Add permissions**
6. From the **Trust relationships** tab, select **Edit trust policy**
7. Insert the following Cluster IAM Role trust policy, and select **Update policy**

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

Enable EKS Auto Mode

1. Open your cluster overview page in the AWS Management Console.
2. Under **EKS Auto Mode** select **Manage**
3. Toggle **EKS Auto Mode** to on.
4. From the **EKS Node Pool** dropdown, select the default node pools you want to create.
   - Learn more about Node Pools in EKS Auto Mode. For more information, see the section called “Create node pool”.
5. If you have previously created an EKS Auto Mode Node IAM role this AWS account, select it in the **Node IAM Role** dropdown. If you have not created this role before, select **Create recommended Role** and follow the steps.

AWS CLI

Prerequisites

- The Cluster IAM Role of the existing EKS Cluster must include sufficent permissiosn for EKS Auto Mode, such as the following policies:
  - `AmazonEKSComputePolicy`
  - `AmazonEKSBlockStoragePolicy`
  - `AmazonEKSLoadBalancingPolicy`

  - `AmazonEKSNetworkingPolicy`
  - `AmazonEKSClusterPolicy`
- The Cluster IAM Role must have an updated trust policy including the `sts:TagSession` action. For more information on creating a Cluster IAM Role, see the section called " AWS CLI".
- `aws` CLI installed, logged in, and a sufficent version. You must have permission to manage IAM, EKS, and EC2 resources. For more information, see *Set up*.

Procedure

Use the following commands to enable EKS Auto Mode on an existing cluster.

> **Note**
>
> The compute, block storage, and load balancing capabilities must all be enabled or disabled in the same request.

```
aws eks update-cluster-config \
 --name $CLUSTER_NAME \
 --compute-config enabled=true \
 --kubernetes-network-config '{"elasticLoadBalancing":{"enabled": true}}' \
 --storage-config '{"blockStorage":{"enabled": true}}'
```

Required Add-on Versions

If you're planning to enable EKS Auto Mode on an existing cluster, you may need to update certain add-ons. Please note:

- This applies only to existing clusters transitioning to EKS Auto Mode.
- New clusters created with EKS Auto Mode enabled don't require these updates.

If you have any of the following add-ons installed, ensure they are at least at the specified minimum version:

Add-on Name	Minimum Required Version
Amazon VPC CNI plugin for Kubernetes	v1.19.0-eksbuild.1

Add-on Name	Minimum Required Version
Kube-proxy	• v1.25.16-eksbuild.22 • v1.26.15-eksbuild.19 • v1.27.16-eksbuild.14 • v1.28.15-eksbuild.4 • v1.29.10-eksbuild.3 • v1.30.6-eksbuild.3 • v1.31.2-eksbuild.3
Amazon EBS CSI driver	v1.37.0-eksbuild.1
CSI snapshot controller	v8.1.0-eksbuild.2
EKS Pod Identity Agent	v1.3.4-eksbuild.1

For more information, see the section called “Update an add-on”.

Next Steps

- To migrate Manage Node Group workloads, see the section called “Migrate from MNGs”.
- To migrate from Self-Managed Karpenter, see the section called “Migrate from Karpenter”.

Migrate from Karpenter to EKS Auto Mode using kubectl

This topic walks you through the process of migrating workloads from Karpenter to Amazon EKS Auto Mode using kubectl. The migration can be performed gradually, allowing you to move workloads at your own pace while maintaining cluster stability and application availability throughout the transition.

The step-by-step approach outlined below enables you to run Karpenter and EKS Auto Mode side by side during the migration period. This dual-operation strategy helps ensure a smooth transition by allowing you to validate workload behavior on EKS Auto Mode before completely decommissioning Karpenter. You can migrate applications individually or in groups, providing flexibility to accommodate your specific operational requirements and risk tolerance.

Prerequisites

Before beginning the migration, ensure you have:

- Karpenter v1.1 or later installed on your cluster. For more information, see Upgrading to 1.1.0+ in the Karpenter docs.
- `kubectl` installed and connected to your cluster. For more information, see *Set up*.

This topic assumes you are familiar with Karpenter and NodePools. For more information, see the Karpenter Documentation.

Step 1: Enable EKS Auto Mode on the cluster

Enable EKS Auto Mode on your existing cluster using the AWS CLI or Management Console. For more information, see the section called "Enable on cluster".

> **Note**
>
> While enabling EKS Auto Mode, don't enable the `general purpose` nodepool at this stage during transition. This node pool is not selective.
> For more information, see the section called "Review built-in node pools".

Step 2: Create a tainted EKS Auto Mode NodePool

Create a new NodePool for EKS Auto Mode with a taint. This ensures that existing pods won't automatically schedule on the new EKS Auto Mode nodes. This node pool uses the `default` `NodeClass` built into EKS Auto Mode. For more information, see the section called "Create node class".

Example node pool with taint:

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: eks-auto-mode
spec:
  template:
    spec:
      requirements:
```

```
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m", "r"]
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      taints:
        - key: "eks-auto-mode"
          effect: "NoSchedule"
```

Update the requirements for the node pool to match the Karpenter configuration you are migrating from. You need at least one requirement.

Step 3: Update workloads for migration

Identify and update the workloads you want to migrate to EKS Auto Mode. Add both tolerations and node selectors to these workloads:

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      tolerations:
      - key: "eks-auto-mode"
        effect: "NoSchedule"
      nodeSelector:
        eks.amazonaws.com/compute-type: auto
```

This change allows the workload to be scheduled on the new EKS Auto Mode nodes.

EKS Auto Mode uses different labels than Karpenter. Labels related to EC2 managed instances start with `eks.amazonaws.com`. For more information, see the section called “Create node pool”.

Step 4: Gradually migrate workloads

Repeat Step 3 for each workload you want to migrate. This allows you to move workloads individually or in groups, based on your requirements and risk tolerance.

Step 5: Remove the original Karpenter NodePool

Once all workloads have been migrated, you can remove the original Karpenter NodePool:

```
kubectl delete nodepool <original-nodepool-name>
```

Step 6: Remove taint from EKS Auto Mode NodePool (Optional)

If you want EKS Auto Mode to become the default for new workloads, you can remove the taint from the EKS Auto Mode NodePool:

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: eks-auto-mode
spec:
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      # Remove the taints section
```

Step 7: Remove node selectors from workloads (Optional)

If you've removed the taint from the EKS Auto Mode NodePool, you can optionally remove the node selectors from your workloads, as EKS Auto Mode is now the default:

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      # Remove the nodeSelector section
      tolerations:
      - key: "eks-auto-mode"
        effect: "NoSchedule"
```

Step 8: Uninstall Karpenter from your cluster

The steps to remove Karpenter depend on how you installed it. For more information, see the Karpenter install instructions and the Helm Uninstall command.

Migrate from EKS Managed Node Groups to EKS Auto Mode

When transitioning your Amazon EKS cluster to use EKS auto mode, you can smoothly migrate your existing workloads from managed node groups (MNGs) using the eksctl CLI tool. This process ensures continuous application availability while EKS auto mode optimizes your compute resources. The migration can be performed with minimal disruption to your running applications.

This topic walks you through the steps to safely drain pods from your existing managed node groups and allow EKS auto mode to reschedule them on newly provisioned instances. By following this procedure, you can take advantage of EKS auto mode's intelligent workload consolidation while maintaining your application's availability throughout the migration.

Prerequisites

- Cluster with EKS Auto Mode enabled
- `eksctl` CLI installed and connected to your cluster. For more information, see *Set up*.
- Karpenter is not installed on the cluster.

Procedure

Use the following `eksctl` CLI command to initiate draining pods from the existing managed node group instances. EKS Auto Mode will create new nodes to back the displaced pods.

```
eksctl update auto-mode-config --drain-all-nodegroups
```

Run sample workloads in EKS Auto Mode clusters

This chapter provides examples of how to deploy different types of workloads to Amazon EKS clusters running in Auto Mode. The examples demonstrate key workload patterns including sample applications, load-balanced web applications, stateful workloads using persistent storage, and workloads with specific node placement requirements. Each example includes complete manifests and step-by-step deployment instructions that you can use as templates for your own applications.

Before proceeding with the examples, ensure that you have an EKS cluster running in Auto Mode and that you have installed the AWS CLI and kubectl. For more information, see *Set up*. The examples assume basic familiarity with Kubernetes concepts and kubectl commands.

You can use these use case-based samples to run workloads in EKS Auto Mode clusters.

the section called "Deploy inflate workload"

Shows how to deploy a sample workload to an EKS Auto Mode cluster using `kubectl` commands.

the section called "Deploy load balancer"

Shows how to deploy a containerized version of the 2048 game on Amazon EKS.

the section called "Deploy stateful workload"

Shows how to deploy a sample stateful application to an EKS Auto Mode cluster.

the section called "Control deployment"

Shows how to use an annotation to control if a workload is deployed to nodes managed by EKS Auto Mode.

Deploy a sample inflate workload to an Amazon EKS Auto Mode cluster

In this tutorial, you'll learn how to deploy a sample workload to an EKS Auto Mode cluster and observe how it automatically provisions the required compute resources. You'll use `kubectl` commands to watch the cluster's behavior and see firsthand how Auto Mode simplifies Kubernetes operations on AWS. By the end of this tutorial, you'll understand how EKS Auto Mode responds to workload deployments by automatically managing the underlying compute resources, without requiring manual node group configuration.

Prerequisites

- An Amazon EKS Auto Mode cluster with the compute capability enabled. Note the name and AWS region of the cluster.
- An IAM principal, such as a user or role, with sufficent permissions to manage networking, compute, and EKS resources.
  - For more information, see Creating roles and attaching policies in the IAM User Guide in the IAM User Guide.
- `aws` CLI installed and configured with an IAM identity.
- `kubectl` CLI installed and connected to cluster.
  - For more information, see *Set up*.

Step 1: Review existing compute resources (optional)

First, use `kubectl` to list the node pools on your cluster.

```
kubectl get nodepools
```

Sample Output:

```
general-purpose
```

In this tutorial, we will deploy a workload configured to use the `general-purpose` node pool. This node pool is built into EKS Auto Mode, and includes reasonable defaults for general workloads, such as microservices and web apps. You can create your own node pool. For more information, see the section called “Create node pool”.

Second, use `kubectl` to list the nodes connected to your cluster.

```
kubectl get nodes
```

If you just created an EKS Auto Mode cluster, you will have no nodes.

In this tutorial you will deploy a sample workload. If you have no nodes, or the workload cannot fit on existing nodes, EKS Auto Mode will provision a new node.

Step 2: Deploy a sample application to the cluster

Review the following Kubernetes Deployment and save it as `inflate.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: inflate
spec:
  replicas: 1
  selector:
    matchLabels:
      app: inflate
  template:
    metadata:
      labels:
```

```
        app: inflate
    spec:
      terminationGracePeriodSeconds: 0
      nodeSelector:
        eks.amazonaws.com/compute-type: auto
      securityContext:
        runAsUser: 1000
        runAsGroup: 3000
        fsGroup: 2000
      containers:
        - name: inflate
          image: public.ecr.aws/eks-distro/kubernetes/pause:3.7
          resources:
            requests:
              cpu: 1
          securityContext:
            allowPrivilegeEscalation: false
```

Note the `eks.amazonaws.com/compute-type: auto` selector requires the workload be deployed on an Amazon EKS Auto Mode node.

Apply the Deployment to your cluster.

```
kubectl apply -f inflate.yaml
```

Step 3: Watch Kubernetes Events

Use the following command to watch Kubernetes events, including creating a new node. Use `ctrl +c` to stop watching events.

```
kubectl get events -w --sort-by '.lastTimestamp'
```

Use `kubectl` to list the nodes connected to your cluster again. Note the newly created node.

```
kubectl get nodes
```

Step 4: View nodes and instances in the AWS console

You can view EKS Auto Mode Nodes in the EKS console, and the associated EC2 instances in the EC2 console.

EC2 Instances deployed by EKS Auto Mode are restricted. You cannot run arbitrary commands on EKS Auto Mode nodes.

Step 5: Delete the deployment

Use `kubectl` to delete the sample deployment

```
kubectl delete -f inflate.yaml
```

If you have no other workloads deployed to your cluster, the node created by EKS Auto Mode will be empty.

In the default configration, EKS Auto Mode detects nodes that have been empty for thirty seconds, and terminates them.

Use `kubectl` or the EC2 console to confirm the associated instance has been deleted.

Deploy a Sample Load Balancer Workload to EKS Auto Mode

This guide walks you through deploying a containerized version of the 2048 game on Amazon EKS, complete with load balancing and internet accessibility.

Prerequisites

- An EKS Auto Mode cluster
- `kubectl` configured to interact with your cluster
- Appropriate IAM permissions for creating ALB resources

Step 1: Create the Namespace

First, create a dedicated namespace for the 2048 game application.

Create a file named `01-namespace.yaml`:

```
apiVersion: v1
kind: Namespace
metadata:
  name: game-2048
```

Apply the namespace configuration:

```
kubectl apply -f 01-namespace.yaml
```

Step 2: Deploy the Application

The application runs multiple replicas of the 2048 game container.

Create a file named `02-deployment.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: game-2048
  name: deployment-2048
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: app-2048
  replicas: 5
  template:
    metadata:
      labels:
        app.kubernetes.io/name: app-2048
    spec:
      containers:
        - image: public.ecr.aws/l6m2t8p7/docker-2048:latest
          imagePullPolicy: Always
          name: app-2048
          ports:
            - containerPort: 80
          resources:
            requests:
              cpu: "0.5"
```

> **Note**
>
> If you receive an error loading the image `public.ecr.aws/l6m2t8p7/docker-2048:latest`, confirm your Node IAM role has sufficent permissions to pull images from ECR. For more information, see the section called "Node IAM role".

Key components:

- Deploys 5 replicas of the application
- Uses a public ECR image
- Requests 0.5 CPU cores per pod
- Exposes port 80 for HTTP traffic

Apply the deployment:

```
kubectl apply -f 02-deployment.yaml
```

Step 3: Create the Service

The service exposes the deployment to the cluster network.

Create a file named `03-service.yaml`:

```
apiVersion: v1
kind: Service
metadata:
  namespace: game-2048
  name: service-2048
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: NodePort
  selector:
    app.kubernetes.io/name: app-2048
```

Key components:

- Creates a NodePort service
- Maps port 80 to the container’s port 80
- Uses label selector to find pods

Apply the service:

```
kubectl apply -f 03-service.yaml
```

Step 4: Configure Load Balancing

You will set up an ingress to expose the application to the internet.

First, create the `IngressClass`. Create a file named `04-ingressclass.yaml`:

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  namespace: game-2048
  labels:
    app.kubernetes.io/name: LoadBalancerController
  name: alb
spec:
  controller: eks.amazonaws.com/alb
```

> **Note**
>
> EKS Auto Mode requires subnet tags to identify public and private subnets.
> If you created your cluster with `eksctl`, you already have these tags.
> Learn how to the section called "Tag subnets".

Then create the Ingress resource. Create a file named `05-ingress.yaml`:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  namespace: game-2048
  name: ingress-2048
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
```

```
            pathType: Prefix
            backend:
              service:
                name: service-2048
                port:
                  number: 80
```

Key components:

- Creates an internet-facing ALB
- Uses IP target type for direct pod routing
- Routes all traffic (/) to the game service

Apply the ingress configurations:

```
kubectl apply -f 04-ingressclass.yaml
kubectl apply -f 05-ingress.yaml
```

Step 5: Verify the Deployment

1. Check that all pods are running:

   ```
   kubectl get pods -n game-2048
   ```

2. Verify the service is created:

   ```
   kubectl get svc -n game-2048
   ```

3. Get the ALB endpoint:

   ```
   kubectl get ingress -n game-2048
   ```

The ADDRESS field in the ingress output will show your ALB endpoint. Wait 2-3 minutes for the ALB to provision and register all targets.

Step 6: Access the Game

Open your web browser and browse to the ALB endpoint URL from the earlier step. You should see the 2048 game interface.

Step 7: Cleanup

To remove all resources created in this tutorial:

```
kubectl delete namespace game-2048
```

This will delete all resources in the namespace, including the deployment, service, and ingress resources.

What's Happening Behind the Scenes

1. The deployment creates 5 pods running the 2048 game
2. The service provides stable network access to these pods
3. EKS Auto Mode:
   - Creates an Application Load Balancer in AWS
   - Configures target groups for the pods
   - Sets up routing rules to direct traffic to the service

Troubleshooting

If the game doesn't load:

- Ensure all pods are running: `kubectl get pods -n game-2048`
- Check ingress status: `kubectl describe ingress -n game-2048`
- Verify ALB health checks: Check the target group health in AWS Console

Deploy a sample stateful workload to EKS Auto Mode

This tutorial will guide you through deploying a sample stateful application to your EKS Auto Mode cluster. The application writes timestamps to a persistent volume, demonstrating EKS Auto Mode's automatic EBS volume provisioning and persistence capabilities.

Prerequisites

- An EKS Auto Mode cluster
- The AWS CLI configured with appropriate permissions
- `kubectl` installed and configured

- For more information, see *Set up*.

Step 1: Configure your environment

1. Set your environment variables:

```
export CLUSTER_NAME=my-auto-cluster
export AWS_REGION="us-west-2"
```

2. Update your kubeconfig:

```
aws eks update-kubeconfig --name "${CLUSTER_NAME}"
```

Step 2: Create the storage class

The `StorageClass` defines how EKS Auto Mode will provision EBS volumes.

EKS Auto Mode does not create a `StorageClass` for you. You must create a `StorageClass` referencing `ebs.csi.eks.amazonaws.com` to use the storage capability of EKS Auto Mode.

1. Create a file named `storage-class.yaml`:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: auto-ebs-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
```

2. Apply the `StorageClass`:

```
kubectl apply -f storage-class.yaml
```

Key components:

- `provisioner: ebs.csi.eks.amazonaws.com` - Uses EKS Auto Mode
- `volumeBindingMode: WaitForFirstConsumer` - Delays volume creation until a pod needs it
- `type: gp3` - Specifies the EBS volume type
- `encrypted: "true"` - EBS will use the default `aws/ebs` key to encrypt volumes created with this class. This is optional, but recommended.
- `storageclass.kubernetes.io/is-default-class: "true"` - Kubernetes will use this storage class by default, unless you specify a different volume class on a persistent volume claim. Use caution when setting this value if you are migrating from another storage controller. (optional)

Step 3: Create the persistent volume claim

The PVC requests storage from the `StorageClass`.

1. Create a file named `pvc.yaml`:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: auto-ebs-claim
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: auto-ebs-sc
  resources:
    requests:
      storage: 8Gi
```

2. Apply the PVC:

```
kubectl apply -f pvc.yaml
```

Key components:

- `accessModes: ReadWriteOnce` - Volume can be mounted by one node at a time
- `storage: 8Gi` - Requests an 8 GiB volume
- `storageClassName: auto-ebs-sc` - References the `StorageClass` we created

Step 4: Deploy the Application

The Deployment runs a container that writes timestamps to the persistent volume.

1. Create a file named `deployment.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: inflate-stateful
spec:
  replicas: 1
  selector:
    matchLabels:
      app: inflate-stateful
  template:
    metadata:
      labels:
        app: inflate-stateful
    spec:
      terminationGracePeriodSeconds: 0
      nodeSelector:
        eks.amazonaws.com/compute-type: auto
      containers:
        - name: bash
          image: public.ecr.aws/docker/library/bash:4.4
          command: ["/usr/local/bin/bash"]
          args: ["-c", "while true; do echo $(date -u) >> /data/out.txt; sleep 60;
 done"]
          resources:
            requests:
              cpu: "1"
          volumeMounts:
            - name: persistent-storage
              mountPath: /data
      volumes:
        - name: persistent-storage
          persistentVolumeClaim:
            claimName: auto-ebs-claim
```

2. Apply the Deployment:

```
kubectl apply -f deployment.yaml
```

Key components:

- Simple bash container that writes timestamps to a file
- Mounts the PVC at `/data`
- Requests 1 CPU core
- Uses node selector for EKS managed nodes

Step 5: Verify the Setup

1. Check that the pod is running:

```
kubectl get pods -l app=inflate-stateful
```

2. Verify the PVC is bound:

```
kubectl get pvc auto-ebs-claim
```

3. Check the EBS volume:

```
# Get the PV name
PV_NAME=$(kubectl get pvc auto-ebs-claim -o jsonpath='{.spec.volumeName}')
# Describe the EBS volume
aws ec2 describe-volumes \
  --filters Name=tag:CSIVolumeName,Values=${PV_NAME}
```

4. Verify data is being written:

```
kubectl exec "$(kubectl get pods -l app=inflate-stateful \
  -o=jsonpath='{.items[0].metadata.name}')" -- \
  cat /data/out.txt
```

Step 6: Cleanup

Run the following command to remove all resources created in this tutorial:

```
# Delete all resources in one command
kubectl delete deployment/inflate-stateful pvc/auto-ebs-claim storageclass/auto-ebs-sc
```

What's Happening Behind the Scenes

1. The PVC requests storage from the `StorageClass`
2. When the Pod is scheduled:
   a. EKS Auto Mode provisions an EBS volume
   b. Creates a PersistentVolume
   c. Attaches the volume to the node
3. The Pod mounts the volume and begins writing timestamps

Snapshot Controller

EKS Auto Mode is compatible with the Kubernetes CSI Snapshotter, also known as the snapshot controller. However, EKS Auto Mode does not include the snapshot controller. You are responsible for installing and configuring the snapshot controller. For more information, see the section called "CSI snapshot controller".

Review the following `VolumeSnapshotClass` that references the storage capability of EKS Auto Mode.

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: auto-ebs-vsclass
driver: ebs.csi.eks.amazonaws.com
deletionPolicy: Delete
```

Learn more about the Kubernetes CSI Snapshotter.

Configure EKS Auto Mode settings

This chapter describes how to configure specific aspects of your Amazon Elastic Kubernetes Service (EKS) Auto Mode clusters. While EKS Auto Mode manages most infrastructure components automatically, you can customize certain features to meet your workload requirements.

Using the configuration options described in this topic, you can modify networking settings, compute resources, and load balancing behaviors while maintaining the benefits of automated infrastructure management. Before making any configuration changes, review the available options in the following sections to determine which approach best suits your needs.

What features do you want to configure?	Configuration option
Node networking and storage • Configure node placement across public and private subnets • Define custom security groups for node access control • Customize network address translation (SNAT) policies • Enable detailed network policy logging and monitoring • Set ephemeral storage parameters (size, IOPS, throughput) • Configure encrypted ephemeral storage with custom KMS keys	the section called "Create node class"
Node compute resources • Select specific EC2 instance types and families • Define CPU architectures (x86_64, ARM64) • Configure capacity types (On-Demand, Spot) • Specify Availability Zones • Configure node taints and labels • Set minimum and maximum node counts	the section called "Create node pool"
Application Load Balancer settings • Deploy internal or internet-facing load balancers • Configure cross-zone load balancing • Set idle timeout periods • Enable HTTP/2 and WebSocket support	the section called "Create ingress class"

What features do you want to configure?	Configuration option
• Configure health check parameters • Specify TLS certificate settings • Define target group attributes • Set IP address type (IPv4, dual-stack)	
Network Load Balancer settings • Configure direct pod IP routing • Enable cross-zone load balancing • Set connection idle timeout • Configure health check parameters • Specify subnet placement • Set IP address type (IPv4, dual-stack) • Configure preserve client source IP • Define target group attributes	the section called "Create service"
Storage Class settings • Define EBS volume types (gp3, io1, io2, etc.) • Configure volume encryption and KMS key usage • Set IOPS and throughput parameters • Set as default storage class • Define custom tags for provisioned volumes	the section called "Create StorageClass"

Create a Node Class for Amazon EKS

Amazon EKS Node Classes provide granular control over the configuration of your EKS Auto Mode managed nodes. A Node Class defines infrastructure-level settings that apply to groups of nodes in your EKS cluster, including network configuration, storage settings, and resource tagging. This topic explains how to create and configure a Node Class to meet your specific operational requirements.

When you need to customize how EKS Auto Mode provisions and configures EC2 instances beyond the default settings, creating a Node Class gives you precise control over critical infrastructure parameters. For example, you can specify private subnet placement for enhanced security, configure instance ephemeral storage for performance-sensitive workloads, or apply custom tagging for cost allocation.

Create a Node Class

To create a Node Class, follow these steps:

1. Create a YAML file (for example, `nodeclass.yaml`) with your Node Class configuration
2. Apply the configuration to your cluster using `kubectl`
3. Reference the Node Class in your Node Pool configuration. For more information, see the section called "Create node pool".

You need `kubectl` installed and configured. For more information, see *Set up*.

Basic Node Class Example

Here's an example Node Class:

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: private-compute
spec:
  ephemeralStorage:
    size: "160Gi"
```

This NodeClass increases the amount of ephemeral storage on the node.

Apply this configuration using:

```
kubectl apply -f nodeclass.yaml
```

Next, reference the Node Class in your Node Pool configuration. For more information, see the section called "Create node pool".

Create node class access entry

If you create a custom node class, you need to create an EKS Access Entry to permit the nodes to join the cluster. EKS automatically creates access entries when you use the built-in node class and node pools.

For information about how Access Entries work, see the section called "Grant permissions".

When creating access entries for EKS Auto Mode node classes, you need to use the EC2 access entry type.

Create access entry with CLI

To create an access entry for EC2 nodes and associate the EKS Auto Node Policy:

Update the following CLI commands with your cluster name, and node role ARN. The node role ARN is specified in the node class YAML.

```
# Create the access entry for EC2 nodes
aws eks create-access-entry \
  --cluster-name <cluster-name> \
  --principal-arn <node-role-arn> \
  --type EC2

# Associate the auto node policy
aws eks associate-access-policy \
  --cluster-name <cluster-name> \
  --principal-arn <node-role-arn> \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy \
  --access-scope type=cluster
```

Create access entry with CloudFormation

To create an access entry for EC2 nodes and associate the EKS Auto Node Policy:

Update the following CloudFormation with your cluster name, and node role ARN. The node role ARN is specified in the node class YAML.

```
EKSAutoNodeRoleAccessEntry:
  Type: AWS::EKS::AccessEntry
  Properties:
    ClusterName: <cluster-name>
```

```
    PrincipalArn: <node-role-arn>
    Type: "EC2"
    AccessPolicies:
      - AccessScope:
          Type: cluster
        PolicyArn: arn:aws:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy
  DependsOn: [ <cluster-name> ] # previously defined in CloudFormation
```

For information about deploying CloudFormation stacks, see Getting started with CloudFormation

Node Class Specification

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: default
spec:

  # Required: Name of IAM Role for Nodes
  role: "MyNodeRole"

  # Required: Subnet selection for node placement
  subnetSelectorTerms:
    - tags:
        Name: "<tag-name>"
        kubernetes.io/role/internal-elb: "1"
    # Alternative using direct subnet ID
    # - id: "subnet-0123456789abcdef0"

  # Required: Security group selection for nodes
  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-node-sg"
    # Alternative approaches:
    # - id: "sg-0123456789abcdef0"
    # - name: "eks-cluster-node-security-group"

  # Optional: Configure SNAT policy (defaults to Random)
  snatPolicy: Random  # or Disabled

  # Optional: Network policy configuration (defaults to DefaultAllow)
  networkPolicy: DefaultAllow  # or DefaultDeny
```

```
# Optional: Network policy event logging (defaults to Disabled)
networkPolicyEventLogs: Disabled  # or Enabled

# Optional: Configure ephemeral storage (shown with default values)
ephemeralStorage:
  size: "80Gi"    # Range: 1-59000Gi or 1-64000G or 1-58Ti or 1-64T
  iops: 3000      # Range: 3000-16000
  throughput: 125 # Range: 125-1000

# IAM role to use for EC2 instance role
# If unspecified, EKS will create a role
# If specified, role requires access entry described above
role: arn:aws:iam::123456789012:role/MyNodeRole

# Optional: Additional EC2 tags
tags:
  Environment: "production"
  Team: "platform"
```

Considerations:

- If you change the Node IAM Role associated with a NodeClass, you will need to create a new Access Entry. EKS automatically creates an Access Entry for the Node IAM Role during cluster creation. The Node IAM Role requires the `AmazonEKSAutoNodePolicy` EKS Access Policy. For more information, see the section called "Grant permissions".
- EKS limits the maximum number of pods on a node to 110. This limit is applied after the existing max pods calculation. For more information, see the section called "Amazon EC2 instance types".
- If you want to propagate tags from Kubernetes to EC2, you need to configure additional IAM permissions. For more information, see the section called "Identity and access".

Create a Node Pool for EKS Auto Mode

Amazon EKS node pools provide a flexible way to manage compute resources in your Kubernetes cluster. This topic demonstrates how to create and configure node pools using Karpenter, a node provisioning tool that helps optimize cluster scaling and resource utilization. With Karpenter's NodePool resource, you can define specific requirements for your compute resources, including instance types, availability zones, architectures, and capacity types.

You cannot modify the built in `system` and `general-purpose` node pools. You can only enable or disable them. For more information, see the section called "Review built-in node pools".

The NodePool specification allows for fine-grained control over your EKS cluster's compute resources through various supported labels and requirements. These include options for specifying EC2 instance categories, CPU configurations, availability zones, architectures (ARM64/AMD64), and capacity types (spot/on-demand). You can also set resource limits for CPU and memory usage, ensuring your cluster stays within desired operational boundaries.

EKS Auto Mode leverages well-known Kubernetes labels to provide consistent and standardized ways of identifying node characteristics. These labels, such as `topology.kubernetes.io/zone` for availability zones and `kubernetes.io/arch` for CPU architecture, follow established Kubernetes conventions. Additionally, EKS-specific labels (prefixed with `eks.amazonaws.com/`) extend this functionality with AWS-specific attributes like instance types, CPU manufacturers, GPU capabilities, and networking specifications. This standardized labeling system enables seamless integration with existing Kubernetes tooling while providing deep AWS infrastructure integration.

Create a NodePool

Follow these steps to create a NodePool for your Amazon EKS cluster:

1. Create a YAML file named `nodepool.yaml` with your desired NodePool configuration. You can use the sample configuration below.
2. Apply the NodePool to your cluster:

```
kubectl apply -f nodepool.yaml
```

3. Verify that the NodePool was created successfully:

```
kubectl get nodepools
```

4. (Optional) Monitor the NodePool status:

```
kubectl describe nodepool default
```

Ensure that your NodePool references a valid NodeClass that exists in your cluster. The NodeClass defines AWS-specific configurations for your compute resources. For more information, see the section called "Create node class".

Sample NodePool

```
apiVersion: karpenter.sh/v1
```

```
kind: NodePool
metadata:
  name: my-node-pool
spec:
  template:
    metadata:
      labels:
        billing-team: my-team
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default

      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m", "r"]
        - key: "eks.amazonaws.com/instance-cpu"
          operator: In
          values: ["4", "8", "16", "32"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-west-2a", "us-west-2b"]
        - key: "kubernetes.io/arch"
          operator: In
          values: ["arm64", "amd64"]

  limits:
    cpu: "1000"
    memory: 1000Gi
```

EKS Auto Mode Supported Labels

EKS Auto Mode supports the following well known labels.

Label	Example	Description
topology.kubernetes.io/zone	us-east-2a	AWS region

Label	Example	Description
node.kubernetes.io/instance-type	g4dn.8xlarge	AWS instance type
kubernetes.io/arch	amd64	Architectures are defined by [GOARCH values](#) on the instance
karpenter.sh/capacity-type	spot	Capacity types include `spot`, `on-demand`
eks.amazonaws.com/instance-hypervisor	nitro	Instance types that use a specific hypervisor
eks.amazonaws.com/compute-type	auto	Identifies EKS Auto Mode managed nodes
eks.amazonaws.com/instance-encryption-in-transit-supported	true	Instance types that support (or not) in-transit encryption
eks.amazonaws.com/instance-category	g	Instance types of the same category, usually the string before the generation number
eks.amazonaws.com/instance-generation	4	Instance type generation number within an instance category
eks.amazonaws.com/instance-family	g4dn	Instance types of similar properties but different resource quantities
eks.amazonaws.com/instance-size	8xlarge	Instance types of similar resource quantities but different properties
eks.amazonaws.com/instance-cpu	32	Number of CPUs on the instance
eks.amazonaws.com/instance-cpu-manufacturer	aws	Name of the CPU manufacturer
eks.amazonaws.com/instance-memory	131072	Number of mebibytes of memory on the instance

Label	Example	Description
eks.amazonaws.com/instance-ebs-bandwidth	9500	Number of maximum megabits of EBS available on the instance
eks.amazonaws.com/instance-network-bandwidth	131072	Number of baseline megabits available on the instance
eks.amazonaws.com/instance-gpu-name	t4	Name of the GPU on the instance, if available
eks.amazonaws.com/instance-gpu-manufacturer	nvidia	Name of the GPU manufacturer
eks.amazonaws.com/instance-gpu-count	1	Number of GPUs on the instance
eks.amazonaws.com/instance-gpu-memory	16384	Number of mebibytes of memory on the GPU
eks.amazonaws.com/instance-local-nvme	900	Number of gibibytes of local nvme storage on the instance

Note

EKS Auto Mode only supports certain instances, and has minimum size requirements. For more information, see the section called “EKS Auto Mode supported instance reference”.

EKS Auto Mode Not Supported Labels

EKS Auto Mode does not support the following labels.

- EKS Auto Mode only supports Linux
  - `node.kubernetes.io/windows-build`
  - `kubernetes.io/os`

Disable built-in node pools

If you create custom node pools, you can disable the built-in node pools. For more information, see the section called "Review built-in node pools".

Cluster without built-in node pools

You can create a cluster without the built-in node pools. This is helpful when your organization has created customized node pools.

Overview:

1. Create an EKS cluster with the both `nodePools` and `nodeRoleArn` values empty.
   - Sample eksctl `autoModeConfig`:

   ```
   autoModeConfig:
     enabled: true
     nodePools: []
     # Do not set a nodeRoleARN
   ```

   For more information, see the section called "eksctl CLI"
2. Create a custom node class with a node role ARN
   - For more information, see the section called "Create node class"
3. Create an access entry for the custom node class
   - For more information, see the section called "Create node class access entry"
4. Create a custom node pool, as described above.

Disruption

You can configure EKS Auto Mode to disrupt Nodes through your NodePool in multiple ways. You can use `spec.disruption.consolidationPolicy`, `spec.disruption.consolidateAfter`, or `spec.template.spec.expireAfter`. You can also rate limit EKS Auto Mode's disruption through the NodePool's `spec.disruption.budgets`. You can also control the time windows and number of simultaneous Nodes disrupted. For instructions on configuring this behavior, see Disruption in the Karpenter Documentation.

You can configure disruption for node pools to:

- Identify when instances are underutilized, and consolidate workloads.
- Create a node pool disruption budget to rate limit node terminations due to drift, emptiness, and consolidation.

By default, EKS Auto Mode:

- Consolidates underutilized instances.
- Terminates instances after 720 hours.
- Sets a single disruption budget of 10% of nodes.

Create an IngressClass to configure an Application Load Balancer

EKS Auto Mode automates routine tasks for load balancing, including exposing cluster apps to the internet.

AWS suggests using Application Load Balancers (ALB) to serve HTTP and HTTPS traffic. Application Load Balancers can route requests based on the content of the request. For more information on Application Load Balancers, see [What is Elastic Load Balancing?](#)

EKS Auto Mode creates and configures Application Load Balancers (ALBs). For example, EKS Auto Mode creates a load balancer when you create an `Ingress` Kubernetes objects and configures it to route traffic to your cluster workload.

Overview

1. Create an `IngressClassParams` resource, specifying AWS specific configuration values such as the certificate to use for SSL/TLS and VPC Subnets.
2. Create an `IngressClass` resource, specifying that EKS Auto Mode will be the controller for the resource.
3. Create an `Ingress` resource that associates a HTTP path and port with a cluster workload.
4. EKS Auto Mode will create an Application Load Balancer that points to the workload specified in the `Ingress` resource, using the load balancer configuration specified in the `IngressClassParams` resource.

Prerequisites

- EKS Auto Mode Enabled on an Amazon EKS Cluster

- Kubectl configured to connect to your cluster
  - You can use `kubectl apply -f <filename>` to apply the sample configuration YAML files below to your cluster.

> **Note**
>
> EKS Auto Mode requires subnet tags to identify public and private subnets.
> If you created your cluster with `eksctl`, you already have these tags.
> Learn how to the section called “Tag subnets”.

Step 1: Create IngressClassParams

Create an `IngressClassParams` object to specify AWS specific configuration options for the Application Load Balancer. Use the reference below to update the sample YAML file.

Note the name you set for the `IngressClassParams` resource, you will need it in the next step.

```
apiVersion: eks.amazonaws.com/v1
kind: IngressClassParams
metadata:
  name: alb
spec:
  scheme: internet-facing
```

Step 2: Create IngressClass

Create an `IngressClass` that references the AWS specific configuration values set in the `IngressClassParams` resource. Note the name of the `IngressClass` . In this example, both the `IngressClass` and `IngressClassParams` are named `alb`.

Use the `is-default-class` annotation to control if `Ingress` resources should use this class by default.

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: alb
  annotations:
```

```
    # Use this annotation to set an IngressClass as Default
    # If an Ingress doesn't specify a class, it will use the Default
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  # Configures the IngressClass to use EKS Auto Mode
  controller: eks.amazonaws.com/alb
  parameters:
    apiGroup: eks.amazonaws.com
    kind: IngressClassParams
    # Use the name of the IngressClassParams set in the previous step
    name: alb
```

For more information on configuration options, see the section called “IngressClassParams Reference”.

Step 3: Create Ingress

Create an `Ingress` resource. The purpose of this resource is to associate paths and ports on the Application Load Balancer with workloads in your cluster.

For more information about configuring this resource, see Ingress in the Kubernetes Documentation.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: 2048-ingress
spec:
  # this matches the name of IngressClass.
  # this can be omitted if you have a default ingressClass in cluster: the one with
 ingressclass.kubernetes.io/is-default-class: "true"  annotation
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /*
            pathType: ImplementationSpecific
            backend:
              service:
                name: <your-service>
                port:
                  number: 80
```

Step 4: Check Status

Use `kubectl` to find the status of the `Ingress`. It can take a few minutes for the load balancer to become available.

Use the name of the `Ingress` resource you set in the previous step.

```
kubectl get ingress <ingress-name>
```

Once the resource is ready, retrieve the domain name of the load balancer.

```
kubectl get ingress api-ingress -o
 jsonpath='{.status.loadBalancer.ingress[0].hostname}'
```

To view the service in a web browser, review the port and path specified in the `Ingress` rescue.

Step 5: Cleanup

To clean up the load balancer, use the following command:

```
kubectl delete ingress <ingress-name>
```

EKS Auto Mode will automatically delete the associated load balancer in your AWS account.

IngressClassParams Reference

The table below is a quick reference for commonly used configuration options.

Field	Description	Example Value
`scheme`	Defines whether the ALB is internal or internet-facing	`internet-facing`
`namespaceSelector`	Restricts which namespaces can use this IngressClass	`environment: prod`
`group.name`	Groups multiple Ingresses to share a single ALB	`retail-apps`

Field	Description	Example Value
`ipAddressType`	Sets IP address type for the ALB	`dualstack`
`subnets.ids`	List of subnet IDs for ALB deployment	`subnet-xxxx, subnet-yyyy`
`subnets.tags`	Tag filters to select subnets for ALB	`Environment: prod`
`certificateARNs`	ARNs of SSL certificates to use	`arn:aws:acm:region:account:certificate/id`
`tags`	Custom tags for AWS resources	`Environment: prod, Team: platform`
`loadBalancerAttributes`	Load balancer specific attributes	`idle_timeout.timeout_seconds: 60`

Considerations

- You cannot use Annotations on an IngressClass to configure load balancers with EKS Auto Mode.
- You must update the Cluster IAM Role to enable tag propagation from Kubernetes to AWS Load Balancer resources. For more information, see the section called “Custom AWS tags for EKS Auto resources”.
- For information about associating resources with either EKS Auto Mode or the self-managed AWS Load Balancer Controller, see the section called “Migration Reference”.
- For information about fixing issues with load balancers, see the section called “Troubleshoot”.
- For more considerations about using the load balancing capability of EKS Auto Mode, see the section called “Load balancing”.

The following tables provide a detailed comparison of changes in IngressClassParams, Ingress annotations, and TargetGroupBinding configurations for EKS Auto Mode. These tables highlight the key differences between the load balancing capability of EKS Auto Mode and the open source load balancer controller, including API version changes, deprecated features, and updated parameter names.

IngressClassParams

Previous	New	Description
`elbv2.k8s.aws/v1beta1`	`eks.amazonaws.com/v1`	API version change
`spec.certificateArn`	`spec.certificateARNs`	Support for multiple certificate ARNs
`spec.subnets.tags`	`spec.subnets.matchTags`	Changed subnet matching schema
`spec.listeners.listenerAttributes`	`spec.listeners.attributes`	Simplified attribute naming

Ingress annotations

Previous	New	Description
`kubernetes.io/ingress.class`	Not supported	Use `spec.ingressClassName` on Ingress objects
`alb.ingress.kubernetes.io/group.name`	Not supported	Specify groups in IngressClass only
`alb.ingress.kubernetes.io/waf-acl-id`	Not supported	Use WAF v2 instead
`alb.ingress.kubernetes.io/web-acl-id`	Not supported	Use WAF v2 instead
`alb.ingress.kubernetes.io/shield-advanced-protection`	Not supported	Shield integration disabled

TargetGroupBinding

Previous	New	Description
`elbv2.k8s.aws/v1beta1`	`eks.amazonaws.com/v1`	API version change
`spec.targetType` optional	`spec.targetType` required	Explicit target type specification
`spec.networking.ingress.from`	Not supported	No longer supports NLB without security groups

Use Service Annotations to configure Network Load Balancers

Learn how to configure Network Load Balancers (NLB) in Amazon EKS using Kubernetes service annotations. This topic explains the annotations supported by EKS Auto Mode for customizing NLB behavior, including internet accessibility, health checks, SSL/TLS termination, and IP targeting modes.

When you create a Kubernetes service of type `LoadBalancer` in EKS Auto Mode, EKS automatically provisions and configures an AWS Network Load Balancer based on the annotations you specify. This declarative approach allows you to manage load balancer configurations directly through your Kubernetes manifests, maintaining infrastructure as code practices.

EKS Auto Mode handles Network Load Balancer provisioning by default for all services of type LoadBalancer - no additional controller installation or configuration is required. The `loadBalancerClass: eks.amazonaws.com/nlb` specification is automatically set as the cluster default, streamlining the deployment process while maintaining compatibility with existing Kubernetes workloads.

> **Note**
>
> EKS Auto Mode requires subnet tags to identify public and private subnets.
> If you created your cluster with `eksctl`, you already have these tags.
> Learn how to the section called “Tag subnets”.

Sample Service

For more information about the Kubernetes `Service` resource, see the Kubernetes Documentation.

Review the sample `Service` resource below:

```
apiVersion: v1
kind: Service
metadata:
  name: echoserver
  annotations:
    # Specify the load balancer scheme as internet-facing to create a public-facing
 Network Load Balancer (NLB)
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  selector:
    app: echoserver
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
  # Specify the new load balancer class for NLB as part of EKS Auto Mode feature
  # For clusters with Auto Mode enabled, this field can be omitted as it's the default
  loadBalancerClass: eks.amazonaws.com/nlb
```

Commonly used annotations

The following table lists commonly used annotations supported by EKS Auto Mode. Note that EKS Auto Mode may not support all annotations.

> **Tip**
>
> All of the following annotations need to be prefixed with `service.beta.kubernetes.io/`

Field	Description	Example
`aws-load-balancer-type`	Specifies the load balancer type. Use `external` for new deployments.	`external`
`aws-load-balancer-nlb-target-type`	Specifies whether to route traffic to node instances or directly to pod IPs. Use `instance` for standard deployments or `ip` for direct pod routing.	`instance`
`aws-load-balancer-scheme`	Controls whether the load balancer is internal or internet-facing.	`internet-facing`
`aws-load-balancer-healthcheck-protocol`	Health check protocol for target group. Common options are `TCP` (default) or `HTTP`.	`HTTP`
`aws-load-balancer-healthcheck-path`	The HTTP path for health checks when using HTTP/HTTPS protocol.	`/healthz`
`aws-load-balancer-healthcheck-port`	Port used for health checks. Can be a specific port number or `traffic-port` .	`traffic-port`
`aws-load-balancer-subnets`	Specifies which subnets to create the load balancer in. Can use subnet IDs or names.	`subnet-xxxx, subnet-yyyy`
`aws-load-balancer-ssl-cert`	ARN of the SSL certificate from AWS Certificate Manager for HTTPS/TLS.	`arn:aws:acm:region:account:certificate/cert-id`

Field	Description	Example
aws-load-balancer-ssl-ports	Specifies which ports should use SSL/TLS.	443, 8443
load-balancer-source-ranges	CIDR ranges allowed to access the load balancer.	10.0.0.0/24, 192.168.1.0/24
aws-load-balancer-additional-resource-tags	Additional AWS tags to apply to the load balancer and related resources.	Environment=prod,Team=platform
aws-load-balancer-ip-address-type	Specifies whether the load balancer uses IPv4 or dual-stack (IPv4 + IPv6).	ipv4 or dualstack

Considerations

- You must update the Cluster IAM Role to enable tag propagation from Kubernetes to AWS Load Balancer resources. For more information, see the section called “Custom AWS tags for EKS Auto resources”.
- For information about associating resources with either EKS Auto Mode or the self-managed AWS Load Balancer Controller, see the section called “Migration Reference”.
- For information about fixing issues with load balancers, see the section called “Troubleshoot”.
- For more considerations about using the load balancing capability of EKS Auto Mode, see the section called “Load balancing”.

When migrating to EKS Auto Mode for load balancing, several changes in service annotations and resource configurations are necessary. The following tables outline key differences between previous and new implementations, including unsupported options and recommended alternatives.

Service annotations

Previous	New	Description
`service.beta.kubernetes.io/load-balancer-source-ranges`	Not supported	Use `spec.loadBalancerSourceRanges` on Service
`service.beta.kubernetes.io/aws-load-balancer-type`	Not supported	Use `spec.loadBalancerClass` on Service
`service.beta.kubernetes.io/aws-load-balancer-internal`	Not supported	Use `service.beta.kubernetes.io/aws-load-balancer-scheme`
Various load balancer attributes	Not supported	Use `service.beta.kubernetes.io/aws-load-balancer-attributes`
`service.beta.kubernetes.io/aws-load-balancer-proxy-protocol`	Not supported	Use `service.beta.kubernetes.io/aws-load-balancer-attributes` instead
`service.beta.kubernetes.io/aws-load-balancer-access-log-enabled`	Not supported	Use `service.beta.kubernetes.io/aws-load-balancer-attributes` instead
`service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-name`	Not supported	Use `service.beta.kubernetes.io/aws-load-balancer-attributes` instead
`service.beta.kubernetes.io/aws-load-`	Not supported	Use `service.beta.kubernetes.io/aws-load-`

Previous	New	Description
balancer-access-log-s3-bucket-prefix		balancer-attributes instead
service.beta.kubernetes.io/aws-load-balancer-cross-zone-load-balancing-enabled	Not supported	Use service.beta.kubernetes.io/aws-load-balancer-attributes instead

To migrate from deprecated load balancer attribute annotations, consolidate these settings into the `service.beta.kubernetes.io/aws-load-balancer-attributes` annotation. This annotation accepts a comma-separated list of key-value pairs for various load balancer attributes. For example, to specify proxy protocol, access logging, and cross-zone load balancing, use the following format:

```
service.beta.kubernetes.io/aws-load-balancer-attributes: |
  access_logs.s3.enabled=true
  access_logs.s3.bucket=my-bucket
  access_logs.s3.prefix=my-prefix
  load_balancing.cross_zone.enabled=true
```

```
service.beta.kubernetes.io/aws-load-balancer-target-group-attributes: |
  proxy_protocol_v2.enabled=true
```

This consolidated format provides a more consistent and flexible way to configure load balancer attributes while reducing the number of individual annotations needed. Review your existing Service configurations and update them to use this consolidated format.

TargetGroupBinding

Previous	New	Description
elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1	API version change

Previous	New	Description
`spec.targetType` optional	`spec.targetType` required	Explicit target type specifica tion
`spec.networking.in gress.from`	Not supported	No longer supports NLB without security groups

Create a storage class

A `StorageClass` in Amazon EKS Auto Mode defines how Amazon EBS volumes are automatically provisioned when applications request persistent storage. This page explains how to create and configure a `StorageClass` that works with the Amazon EKS Auto Mode to provision EBS volumes.

By configuring a `StorageClass`, you can specify default settings for your EBS volumes including volume type, encryption, IOPS, and other storage parameters. You can also configure the `StorageClass` to use AWS KMS keys for encryption management.

EKS Auto Mode does not create a `StorageClass` for you. You must create a `StorageClass` referencing `ebs.csi.eks.amazonaws.com` to use the storage capability of EKS Auto Mode.

First, create a file named `storage-class.yaml`:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: auto-ebs-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
```

Second, apply the storage class to your cluster.

```
kubectl apply -f storage-class.yaml
```

Key components:

- `provisioner: ebs.csi.eks.amazonaws.com` - Uses EKS Auto Mode
- `volumeBindingMode: WaitForFirstConsumer` - Delays volume creation until a pod needs it
- `type: gp3` - Specifies the EBS volume type
- `encrypted: "true"` - EBS will encrypt any volumes created using the `StorageClass`. EBS will use the default `aws/ebs` key alias. For more information, see How Amazon EBS encryption works in the Amazon EBS User Guide. This value is optional but suggested.
- `storageclass.kubernetes.io/is-default-class: "true"` - Kubernetes will use this storage class by default, unless you specify a different volume class on a persistent volume claim. This value is optional. Use caution when setting this value if you are migrating from a different storage controller.

Use self-managed KMS key to encrypt EBS volumes

To use a self-managed KMS key to encrypt EBS volumes automated by EKS Auto Mode, you need to:

1. Create a self-managed KMS key.
   - For more information, see Create a symmetric encryption KMS key or How Amazon Elastic Block Store (Amazon EBS) uses KMS in the KMS User Guide.
2. Create a new policy that permits access to the KMS key.
   - Use the sample IAM policy below to create the policy. Insert the ARN of the new self-managed KMS key. For more information, see Creating roles and attaching policies (console) in the AWS IAM User Guide.
3. Attach the policy to the EKS Cluster Role.
   - Use the AWS console to find the ARN of the EKS Cluster Role. The role information is visible in the **Overview** section. For more information, see the section called “Cluster IAM role”.
4. Update the `StorageClass` to reference the KMS Key ID at the `parameters.kmsKeyId` field.

Sample self-managed KMS IAM Policy

Update the following values in the policy below:

- `<account-id>` – Your AWS account ID, such as `111122223333`
- `<aws-region>` – The AWS region of your cluster, such as `us-west-2`

```
{
  "Version": "2012-10-17",
  "Id": "key-auto-policy-3",
  "Statement": [
      {
          "Sid": "Enable IAM User Permissions",
          "Effect": "Allow",
          "Principal": {
              "AWS": "arn:aws:iam::<account-id>:root"
          },
          "Action": "kms:*",
          "Resource": "*"
      },
      {
        "Sid": "Allow access through EBS for all principals in the account that are
 authorized to use EBS",
        "Effect": "Allow",
        "Principal": {
            "AWS": "*"
        },
        "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:CreateGrant",
            "kms:DescribeKey"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "kms:CallerAccount": "<account-id>",
                "kms:ViaService": "ec2.<aws-region>.amazonaws.com"
            }
        }
    }
  ]
}
```

Sample self-managed KMS StorageClass

```
parameters:
  type: gp3
```

```
  encrypted: "true"
  kmsKeyId: <custom-key-arn>
```

`StorageClass` Parameters Reference

For general information on the Kubernetes `StorageClass` resources, see [Storage Classes](#) in the Kubernetes Documentation.

THe `parameters` section of the `StorageClass` resource is specific to AWS. Use the following table to review available options.

Parameters	Values	Default	Description
"csi.storage.k8s.io/fstype"	xfs, ext2, ext3, ext4	ext4	File system type that will be formatted during volume creation. This parameter is case sensitive!
"type"	io1, io2, gp2, gp3, sc1, st1, standard, sbp1, sbg1	gp3	EBS volume type.
"iopsPerGB"			I/O operations per second per GiB. Can be specified for IO1, IO2, and GP3 volumes.
"allowAutoIOPSPerGBIncrease"	true, false	false	When `"true"`, the CSI driver increases IOPS for a volume when `iopsPerGB * <volume size>` is too low to fit into IOPS range supported by AWS. This allows dynamic

Parameters	Values	Default	Description
			provisioning to always succeed, even when user specifies too small PVC capacity or `iopsPerGB` value. On the other hand, it may introduce additional costs, as such volumes have higher IOPS than requested in `iopsPerGB` .
"iops"			I/O operations per second. Can be specified for IO1, IO2, and GP3 volumes.
"throughput"		125	Throughput in MiB/s. Only effective when gp3 volume type is specified.
"encrypted"	true, false	false	Whether the volume should be encrypted or not. Valid values are "true" or "false".
"blockExpress"	true, false	false	Enables the creation of io2 Block Express volumes.

Parameters	Values	Default	Description
"kmsKeyId"			The full ARN of the key to use when encrypting the volume. If not specified, AWS will use the default KMS key for the region the volume is in. This will be an auto-generated key called `/aws/ebs` if not changed.
"blockSize"			The block size to use when formatting the underlying filesystem. Only supported on linux nodes and with fstype `ext2`, `ext3`, `ext4`, or `xfs`.
"inodeSize"			The inode size to use when formatting the underlying filesystem. Only supported on linux nodes and with fstype `ext2`, `ext3`, `ext4`, or `xfs`.

Parameters	Values	Default	Description
"bytesPerInode"			The `bytes-per-inode` to use when formatting the underlying filesystem. Only supported on linux nodes and with fstype `ext2`, `ext3`, `ext4`.
"numberOfInodes"			The `number-of-inodes` to use when formatting the underlying filesystem. Only supported on linux nodes and with fstype `ext2`, `ext3`, `ext4`.
"ext4BigAlloc"	true, false	false	Changes the `ext4` filesystem to use clustered block allocation by enabling the `bigalloc` formatting option. Warning: `bigalloc` may not be fully supported with your node's Linux kernel.

Parameters	Values	Default	Description
"ext4ClusterSize"			The cluster size to use when formatting an `ext4` filesystem when the `bigalloc` feature is enabled. Note: The `ext4BigAlloc` parameter must be set to true.

For more information, see the AWS EBS CSI Driver on GitHub.

Considerations

The block storage capability of EKS Auto Mode is different from the EBS CSI Driver.

- Static Provisioning
  - If you want to use externally-created EBS volumes with EKS Auto Mode, you need to manually add an AWS tag with the key `eks:eks-cluster-name` and the value of the cluster name.
- Node Startup Taint
  - You cannot use the node startup taint feature to prevent pod scheduling before storage capability readiness
- Custom Tags on Dynamically Provisioned Volumes
  - You cannot use the extra-tag CLI flag to configure custom tags on dynamically provisioned EBS volumes
  - You can use `StorageClass` tagging to add custom tags. EKS Auto Mode will add tags to the associated AWS resources. You will need to update the Cluster IAM Role for custom tags. For more information, see the section called "Custom AWS tags for EKS Auto resources".
- EBS Detailed Performance Metrics
  - You cannot access Prometheus metrics for EBS detailed performance

Install CSI Snapshot Controller add-on

EKS Auto Mode is compatible with the CSI Snapshot Controller Amazon EKS add-on.

AWS suggests you configure this add-on to run on the built-in `system` node pool.

For more information, see:

- the section called "Run critical add-ons"
- the section called "Review built-in node pools"
- the section called "CSI snapshot controller"

To install snapshot controller in system node pool

1. Open your EKS cluster in the AWS console
2. From the **Add-ons** tab, select **Get more add-ons**
3. Select the **CSI Snapshot Controller** and then **Next**
4. On the **Configure selected add-ons settings** page, select **Optional configuration settings** to view the **Add-on configuration schema**
   a. Insert the following yaml to associate the snapshot controller with the `system` node pool. The snapshot controller includes a toleration for the `CriticalAddonsOnly` taint.

   ```
   {
           "nodeSelector": {
               "karpenter.sh/nodepool": "system"
           }
   }
   ```

   b. Select **Next**
5. Review the add-on configuration and then select **Create**

Disable EKS Auto Mode

You can disable EKS Auto Mode on an existing EKS Cluster. This is a destructive operation.

- EKS will terminate all EC2 instances operated by EKS Auto Mode.
- EKS will delete all Load Balancers operated by EKS Auto Mode.

- EKS will **not** delete EBS volumes provisioned by EKS Auto Mode.

EKS Auto Mode is designed to fully manage the resources that it creates. Manual interventions could result in EKS Auto Mode failing to completely clean up those resources when it is disabled. For example, if you referred to a managed Security Group from external Security Group rules, and forget to remove that reference before you disable EKS Auto Mode for a cluster, the managed Security Group will leak (not be deleted). Steps below describe how to remove a leaked Security Group if that should happen.

Disable EKS Auto Mode (AWS Console)

1. Open your cluster overview page in the AWS Management Console.
2. Under **EKS Auto Mode** select **Manage**
3. Toggle **EKS Auto Mode** to `off`.

If any managed Security Group is not deleted at the end of this process, you can delete it manually using descriptions from [Delete a security group](#).

Disable EKS Auto Mode (AWS CLI)

Use the following command to disable EKS Auto Mode on an existing cluster.

You need to have the `aws` CLI installed, and be logged in with sufficent permissions to manage EKS clusters. For more information, see *Set up*.

> **Note**
>
> The compute, block storage, and load balancing capabilities must all be enabled or disabled in the same request.

```
aws eks update-cluster-config \
 --name $CLUSTER_NAME \
 --compute-config enabled=false \
 --kubernetes-network-config '{"elasticLoadBalancing":{"enabled": false}}' \
 --storage-config '{"blockStorage":{"enabled": false}}'
```

You can check if a leaked EKS Auto Mode Security Group failed to be deleted after disabling EKS Auto Mode as follows:

```
aws ec2 describe-security-groups \
    --filters Name=tag:eks:eks-cluster-name,Values=<cluster-Name> Name=tag-
key,Values=ingress.eks.amazonaws.com/resource,service.eks.amazonaws.com/resource --
query "SecurityGroups[*].[GroupName]"
```

To then delete the Security Group:

```
aws ec2 delete-security-group --group-name=<sg-name>
```

Update the Kubernetes Version of an EKS Auto Mode cluster

This topic explains how to update the Kubernetes version of your Auto Mode cluster. Auto Mode simplifies the version update process by handling the coordination of control plane updates with node replacements, while maintaining workload availability through pod disruption budgets.

When upgrading an Auto Mode cluster, many components that traditionally required manual updates are now managed as part of the service. Understanding the automated aspects of the upgrade process and your responsibilities helps ensure a smooth version transition for your cluster.

Learn about updates with EKS Auto Mode

After you initiate a control plane upgrade, EKS Auto Mode begins replacing nodes in your cluster. The new nodes have the corresponding new Kubernetes version. EKS Auto Mode observes pod disruption budgets when upgrading nodes.

Additionally, you no longer need to update components like:

- CoreDNS
- KubeProxy
- AWS Load Balancer Controller
- Karpenter
- AWS EBS CSI Driver

EKS Auto Mode replaces these components with service functionality.

You are still responsible for updating:

- Apps and workloads deployed to your cluster
- Self-managed add-ons and controllers
- Amazon EKS Add-ons
  - Learn how to the section called “Update an add-on”

Learn Best Practices for Cluster Upgrades

Start Cluster Update

To start a cluster update, see the section called “Update Kubernetes version”.

Enable or Disable Built-in NodePools

EKS Auto Mode has two built-in NodePools. You can enable or disable these NodePools using the AWS console, CLI, or API.

Built-in NodePool Reference

- `system`
  - This NodePool has a `CriticalAddonsOnly` taint. Many EKS addons, such as CoreDNS, tolerate this taint. Use this system node pool to segregate cluster-critical applications.
  - Supports both `amd64` and `arm64` architectures.
- `general-purpose`
  - This NodePool provides support for launching nodes for general purpose workloads in your cluster.
  - Uses only `amd64` architecture.

Both built-in NodePools:

- Use the default EKS NodeClass
- Use only on-demand EC2 capacity
- Use the C, M, and R EC2 instance families
- Require generation 5 or newer EC2 instances

Procedure

Prerequisites

- The latest version of the AWS Command Line Interface (AWS CLI) installed and configured on your device. To check your current version, use `aws --version`. To install the latest version, see [Installing] and [Quick configuration] with aws configure in the AWS Command Line Interface User Guide.
  - Login to the CLI with sufficent IAM permissions to create AWS resources including IAM Policies, IAM Roles, and EKS Clusters.

Enable with AWS CLI

Use the following command to enable both built-in NodePools:

```
aws eks update-cluster-config \
  --name <cluster-name> \
  --compute-config '{
    "nodeRoleArn": "<node-role-arn>",
    "nodePools": ["general-purpose", "system"]
  "enabled": true,
  "nodeRoleArn":"<node-role-arn>",
  "nodePools": ["general-purpose", "system"]
  }' \
  --kubernetes-network-config '{
  "elasticLoadBalancing":{"enabled": true}
  }' \
  --storage-config '{
  "blockStorage":{"enabled": true}
  }'
```

You can modify the command to selectively enable the NodePools.

Disable with AWS CLI

Use the following command to disable both built-in NodePools:

```
aws eks update-cluster-config \
  --name <cluster-name> \
  --compute-config '{"nodePools": []}'
  --compute-config '{
```

```
  "enabled": true,
  "nodePools": []
  }' \
  --kubernetes-network-config '{
  "elasticLoadBalancing":{"enabled": true}}' \
  --storage-config '{
  "blockStorage":{"enabled": true}
  }'
```

Control if a workload is deployed on EKS Auto Mode nodes

When running workloads in an EKS cluster with EKS Auto Mode, you might need to control whether specific workloads run on EKS Auto Mode nodes or other compute types. This topic describes how to use node selectors and affinity rules to ensure your workloads are scheduled on the intended compute infrastructure.

The examples in this topic demonstrate how to use the `eks.amazonaws.com/compute-type` label to either require or prevent workload deployment on EKS Auto Mode nodes. This is particularly useful in mixed-mode clusters where you're running both EKS Auto Mode and other compute types, such as self-managed Karpenter provisioners or EKS Managed Node Groups.

EKS Auto Mode nodes have set the value of the label `eks.amazonaws.com/compute-type` to `auto`. You can use this label to control if a workload is deployed to nodes managed by EKS Auto Mode.

Require a workload is deployed to EKS Auto Mode nodes

> **Note**
>
> This `nodeSelector` value is not required for EKS Auto Mode. This `nodeSelector` value is only relevant if you are running a cluster in a mixed mode, node types not managed by EKS Auto Mode. For example, you may have static compute capacity deployed to your cluster with EKS Managed Node Groups, and have dynamic compute capacity managed by EKS Auto Mode.

You can add this `nodeSelector` to Deployments or other workloads to require Kubernetes schedule them onto EKS Auto Mode nodes.

```
apiVersion: apps/v1
```

```
kind: Deployment
spec:
  template:
    nodeSelector:
      eks.amazonaws.com/compute-type: auto
```

Require a workload is not deployed to EKS Auto Mode nodes

You can add this `nodeAffinity` to Deployments or other workloads to require Kubernetes **not** schedule them onto EKS Auto Mode nodes.

```
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - auto
```

Run critical add-ons on dedicated instances

In this topic, you will learn how to deploy a workload with a `CriticalAddonsOnly` toleration so EKS Auto Mode will schedule it onto the `system` node pool.

EKS Auto Mode's built-in `system` node pool is designed for running critical add-ons on dedicated instances. This segregation ensures essential components have dedicated resources and are isolated from general workloads, enhancing overall cluster stability and performance.

This guide demonstrates how to deploy add-ons to the `system` node pool by utilizing the `CriticalAddonsOnly` toleration and appropriate node selectors. By following these steps, you can ensure that your critical applications are scheduled onto the dedicated `system` nodes, leveraging the isolation and resource allocation benefits provided by EKS Auto Mode's specialized node pool structure.

EKS Auto Mode has two built-in node pools: `general-purpose` and `system`. For more information, see the section called "Review built-in node pools".

The purpose of the `system` node pool is to segregate critical add-ons onto different nodes. Nodes provisioned by the `system` node pool have a `CriticalAddonsOnly` Kubernetes taint.

Kubernetes will only schedule pods onto these nodes if they have a corresponding toleration. For more information, see Taints and Tolerations in the Kubernetes documentation.

Prerequisites

- EKS Auto Mode Cluster with the built-in `system` node pool enabled. For more information, see the section called "Review built-in node pools"
- `kubectl` installed and configured. For more information, see *Set up*.

Procedure

Review the example yaml below. Note the following configurations:

- `nodeSelector` — This associates the workload with the built-in `system` node pool. This node pool must be enabled with the AWS API. For more information, see the section called "Review built-in node pools".
- `tolerations` — This toleration overcomes the `CriticalAddonsOnly` taint on nodes in the `system` node pool.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      nodeSelector:
        karpenter.sh/nodepool: system
      tolerations:
      - key: "CriticalAddonsOnly"
        operator: "Exists"
      containers:
      - name: app
```

```
        image: nginx:latest
        resources:
          requests:
            cpu: "500m"
            memory: "512Mi"
```

To update a workload to run on the `system` node pool, you need to:

1. Update the existing workload to add the following configurations described above:
   - `nodeSelector`
   - `tolerations`
2. Deploy the updated workload to your cluster with `kubectl apply`

After updating the workload, it will run on dedicated nodes.

Use Network Policies with EKS Auto Mode

Network policies allow you to control traffic flow at the IP address or port level within your Amazon EKS cluster. This topic explains how to enable and use network policies with EKS Auto Mode.

Prerequisites

- An Amazon EKS cluster with EKS Auto Mode enabled
- kubectl configured to connect to your cluster

Step 1: Enable Network Policy Controller

To use network policies with EKS Auto Mode, you first need to enable the Network Policy Controller by applying a ConfigMap to your cluster.

1. Create a file named `enable-network-policy.yaml` with the following content:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: amazon-vpc-cni
  namespace: kube-system
```

```
data:
  enable-network-policy-controller: "true"
```

2. Apply the ConfigMap to your cluster:

```
kubectl apply -f enable-network-policy.yaml
```

Step 2: Enable Network Policies in Node Class

Before you can use network policies, you need to ensure that your Node Class is configured to support them. Follow these steps:

1. Create or edit a Node Class YAML file (e.g., `nodeclass-network-policy.yaml`) with the following content:

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: network-policy-enabled
spec:
  # Enables network policy support
  networkPolicy: DefaultAllow
  # Optional: Enables logging for network policy events
  networkPolicyEventLogs: Enabled
  # Include other Node Class configurations as needed
```

2. Apply the Node Class configuration to your cluster:

```
kubectl apply -f nodeclass-network-policy.yaml
```

3. Verify that the Node Class has been created:

```
kubectl get nodeclass network-policy-enabled
```

4. Update your Node Pool to use this Node Class. For more information, see the section called “Create node pool”.

Once your nodes are using this Node Class, they will be able to enforce network policies. You can now proceed to create and apply network policies to control traffic within your cluster. For all the node class configuration options, see the section called “Create node class”.

Step 3: Create and test network policies

Your EKS Auto Mode cluster is now configured to support Kubernetes network policies. You can test this with the the section called "Stars policy demo".

Tag subnets for EKS Auto Mode

If you use the load balancing capability of EKS Auto Mode, you need to add AWS tags to your VPC subnets.

Background

These tags identify subnets as associated with the cluster, and more importantly if the subnet is public or private.

Public subnets have direct internet access via an internet gateway. They are used for resources that need to be publicly accessible such as load balancers.

Private subnets do not have direct internet access and use NAT gateways for outbound traffic. They are used for internal resources such as EKS nodes that don't need public IPs.

To learn more about NAT gateways and Internet gateways, see Connect your VPC to other networks in the Amazon Virtual Private Cloud (VPC) User Guide.

Requirement

At this time, subnets used for load balancing by EKS Auto Mode are required to have one of the following tags.

Public subnets

Public subnets are used for internet-facing load balancers. These subnets must have the following tags:

Key
Value
`kubernetes.io/role/elb`
1 or ``

Private subnets

Private subnets are used for internal load balancers. These subnets must have the following tags:

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

Procedure

Before you begin, identify which subnets are public (with Internet Gateway access) and which are private (using NAT Gateway). You'll need permissions to modify VPC resources.

AWS Management Console

1. Open the Amazon VPC console and navigate to Subnets
2. Select the subnet to tag
3. Choose the Tags tab and select Add tag
4. Add the appropriate tag:
   - For public subnets: Key=`kubernetes.io/role/elb`
   - For private subnets: Key=`kubernetes.io/role/internal-elb`
5. Set Value to 1 or leave empty
6. Save and repeat for remaining subnets

AWS CLI

For public subnets:

```
aws ec2 create-tags \
    --resources subnet-ID \
    --tags Key=kubernetes.io/role/elb,Value=1
```

For private subnets:

```
aws ec2 create-tags \
    --resources subnet-ID \
    --tags Key=kubernetes.io/role/internal-elb,Value=1
```

Replace `subnet-ID` with your actual subnet ID.

Learn how EKS Auto Mode works

Use this chapter to learn how the components of Amazon EKS Auto Mode clusters work.

Topics

Learn about Amazon EKS Auto Mode Managed instances

This topic explains how Amazon EKS Auto Mode manages Amazon EC2 instances in your EKS cluster. When you enable EKS Auto Mode, your cluster's compute resources are automatically provisioned and managed by EKS, changing how you interact with the EC2 instances that serve as nodes in your cluster.

Understanding how Amazon EKS Auto Mode manages instances is essential for planning your workload deployment strategy and operational procedures. Unlike traditional EC2 instances or managed node groups, these instances follow a different lifecycle model where EKS assumes responsibility for many operational aspects, while restricting certain types of access and customization.

Amazon EKS Auto Mode automates routine tasks for creating new EC2 Instances, and attaches them as nodes to your EKS cluster. EKS Auto Mode detects when a workload can't fit onto existing nodes, and creates a new EC2 Instance.

Amazon EKS Auto Mode is responsible for creating, deleting, and patching EC2 Instances. You are responsible for the containers and pods deployed on the instance.

EC2 Instances created by EKS Auto Mode are different from other EC2 Instances, they are managed instances. These managed instances are owned by EKS and are more restricted. You can't directly access or install software on instances managed by EKS Auto Mode.

AWS suggests running either EKS Auto Mode or self-managed Karpenter. You can install both during a migration or in an advanced configuration. If you have both installed, configure your node pools so that workloads are associated with either Karpenter or EKS Auto Mode.

For more information, see Amazon EC2 managed instances in the Amazon EC2 user guide.

Comparison table

Standard EC2 Instance	EKS Auto Mode managed instance
You are responsible for patching and updating the instance.	AWS automatically patches and updates the instance.
EKS is not responsible for the software on the instance.	EKS is responsible for certain software on the instance, such as `kubelet`, the container runtime, and the operating system.
You can delete the EC2 Instance using the EC2 API.	EKS determines the number of instances deployed in your account. If you delete a workload, EKS will reduce the number of instances in your account.
You can use SSH to access the EC2 Instance.	You can deploy pods and containers to the managed instance.
You determine the operating system and image (AMI).	AWS determines the operating system and image.
You can deploy workloads that rely on Windows or Ubuntu functionality.	You can deploy containers based on Linux, but without specific OS dependencies.
You determine what instance type and family to launch.	AWS determines what instance type and family to launch. You can use a Node Pool to limit the instance types EKS Auto Mode selects from.

The following functionality works for both Managed instances and Standard EC2 instances:

- You can view the instance in the AWS console.

- You can use instance storage as ephemeral storage for workloads.

AMI Support

With EKS Auto Mode, AWS determines the image (AMI) used for your compute nodes. AWS monitors the rollout of new EKS Auto Mode AMI versions. If you experience workload issues related to an AMI version, create a support case. For more information, see Creating support cases and case management in the AWS Support User Guide.

EKS Auto Mode supported instance reference

EKS Auto Mode only creates instances of supported types, and that meet a minimum size requirement.

EKS Auto Mode supports the following instance types:

Family	Instance Types
Compute Optimized (C)	c8g, c7a, c7g, c7gn, c7gd, c7i, c7i-flex, c6a, c6g, c6i, c6gn, c6id, c6in, c6gd, c5, c5a, c5d, c5ad, c5n, c4
General Purpose (M)	m8g, m7i, m7a, m7g, m7gd, m7i-flex, m6a, m6i, m6in, m6g, m6idn, m6id, m6gd, m5, m5a, m5ad, m5n, m5dn, m5d, m5zn, m4
Memory Optimized (R)	r8g, r7a, r7iz, r7gd, r7i, r7g, r6a, r6i, r6id, r6in, r6idn, r6g, r6gd, r5, r5n, r5a, r5dn, r5b, r5ad, r5d, r4
Burstable (T)	t4g, t3, t3a, t2
High Memory (Z/X)	z1d, x8g, x2gd
Storage Optimized (I/D)	i4g, i4i, i3, i3en, is4gen, d3, d3en, im4gn
Accelerated Computing (P/G/Inf/Trn)	p5, p4d, p3, p3dn, gr6, g6, g6e, g5g, g5, g4dn, inf2, inf1, trn1, trn1n

Family	Instance Types
High Performance Computing (X2)	x2iezn, x2iedn, x2idn

Additionally, EKS Auto Mode will only create EC2 instances that meet the following requirements:

- More than 1 CPU
- Instance size is not nano, micro or small

For more information, see Amazon EC2 instance type naming conventions.

Considerations

- EKS Auto Mode automatically formats and configures NVMe local storage on supported instance types. For nodes with multiple NVMe drives, EKS sets up a RAID 0 array. This automation eliminates the need for manual formatting and RAID configuration of local NVMe storage in EKS clusters.
- Amazon EKS Auto Mode does not support AWS Fault Injection Service. For more information, see Managing Fault Injection Service experiments in the AWS Resilience Hub User Guide.
- You do not need to install the `Neuron Device Plugin` on EKS Auto Mode nodes.
  - If you have other types of nodes in your cluster, you need to configure the Neuron Device plugin to not run on auto mode nodes. For more information, see the section called "Control deployment".

Learn about identity and access in EKS Auto Mode

This topic describes the Identity and Access Management (IAM) roles and permissions required to use EKS Auto Mode. EKS Auto Mode uses two primary IAM roles: a Cluster IAM Role and a Node IAM Role. These roles work in conjunction with EKS Pod Identity and EKS access entries to provide comprehensive access management for your EKS clusters.

When you configure EKS Auto Mode, you will need to set up these IAM roles with specific permissions that allow AWS services to interact with your cluster resources. This includes permissions for managing compute resources, storage volumes, load balancers, and networking components. Understanding these role configurations is essential for proper cluster operation and security.

In EKS Auto Mode, AWS IAM roles are automatically mapped to Kubernetes permissions through EKS access entries, removing the need for manual configuration of `aws-auth` ConfigMaps or custom bindings. When you create a new auto mode cluster, EKS automatically creates the corresponding Kubernetes permissions using Access entries, ensuring that AWS services and cluster components have the appropriate access levels within both the AWS and Kubernetes authorization systems. This automated integration reduces configuration complexity and helps prevent permission-related issues that commonly occur when managing EKS clusters.

Cluster IAM role

The Cluster IAM role is an AWS Identity and Access Management (IAM) role used by Amazon EKS to manage permissions for Kubernetes clusters. This role grants Amazon EKS the necessary permissions to interact with other AWS services on behalf of your cluster, and is automatically configured with Kubernetes permissions using EKS access entries.

- You must attach AWS IAM policies to this role.
- EKS Auto Mode attaches Kubernetes permissions to this role automatically using EKS access entries.
- With EKS Auto Mode, AWS suggests creating a single Cluster IAM Role per AWS account.
- AWS suggests naming this role `AmazonEKSAutoClusterRole`.
- This role requires permissions for multiple AWS services to manage resources including EBS volumes, Elastic Load Balancers, and EC2 instances.
- The suggested configuration for this role includes multiple AWS managed IAM policies, related to the different capabilities of EKS Auto Mode.
  - `AmazonEKSComputePolicy`
  - `AmazonEKSBlockStoragePolicy`
  - `AmazonEKSLoadBalancingPolicy`
  - `AmazonEKSNetworkingPolicy`
  - `AmazonEKSClusterPolicy`

For more information about the Cluster IAM Role and AWS managed IAM policies, see:

- the section called " AWS managed policies"

- the section called "Cluster IAM role"

For more information about Kubernetes access, see:

- the section called "Review access policies"

Node IAM role

The Node IAM role is an AWS Identity and Access Management (IAM) role used by Amazon EKS to manage permissions for worker nodes in Kubernetes clusters. This role grants EC2 instances running as Kubernetes nodes the necessary permissions to interact with AWS services and resources, and is automatically configured with Kubernetes RBAC permissions using EKS access entries.

- You must attach AWS IAM policies to this role.
- EKS Auto Mode attaches Kubernetes RBAC permissions to this role automatically using EKS access entries.
- AWS suggests naming this role `AmazonEKSAutoNodeRole`.
- With EKS Auto Mode, AWS suggests creating a single Node IAM Role per AWS account.
- This role has limited permissions. The key permissions include assuming a Pod Identity Role, and pulling images from ECR.
- AWS suggests the following AWS managed IAM policies:
  - `AmazonEKSWorkerNodeMinimalPolicy`
  - `AmazonEC2ContainerRegistryPullOnly`

For more information about the Cluster IAM Role and AWS managed IAM policies, see:

- the section called " AWS managed policies"
- the section called "Node IAM role"

For more information about Kubernetes access, see:

- the section called "Review access policies"

Service-linked role

Amazon EKS uses a service-linked role (SLR) for certain operations. A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

AWS automatically creates and configures the SLR. You can delete an SLR only after first deleting their related resources. This protects your Amazon EKS resources because you can't inadvertently remove permission to access the resources.

The SLR policy grants Amazon EKS permissions to observe and delete core infrastructure components: EC2 resources (instances, network interfaces, security groups), ELB resources (load balancers, target groups), CloudWatch capabilities (logging and metrics), and IAM roles with "eks" prefix. It also enables private endpoint networking through VPC/hosted zone association and includes permissions for EventBridge monitoring and cleanup of EKS-tagged resources.

For more information, see:

- the section called "AWS managed policy: AmazonEKSServiceRolePolicy"
- the section called "Service-linked role permissions for Amazon EKS"

Custom AWS tags for EKS Auto resources

By default, the managed policies related to EKS Auto Mode do not permit applying user defined tags to Auto Mode provisioned AWS resources. If you want to apply user defined tags to AWS resources, you must attach additional permissions to the Cluster IAM Role with sufficient permissions to create and modify tags on AWS resources. Below is an example of a policy that will allow unrestricted tagging access:

View custom tag policy example

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Compute",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateFleet",
```

```
                "ec2:RunInstances",
                "ec2:CreateLaunchTemplate"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                },
                "StringLike": {
                    "aws:RequestTag/eks:kubernetes-node-class-name": "*",
                    "aws:RequestTag/eks:kubernetes-node-pool-name": "*"
                }
            }
        },
        {
            "Sid": "Storage",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateVolume",
                "ec2:CreateSnapshot"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:volume/*",
                "arn:aws:ec2:*:*:snapshot/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                }
            }
        },
        {
            "Sid": "Networking",
            "Effect": "Allow",
            "Action": "ec2:CreateNetworkInterface",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                },
                "StringLike": {
```

```
                    "aws:RequestTag/eks:kubernetes-cni-node-name": "*"
                }
            }
        },
        {
            "Sid": "LoadBalancer",
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:CreateLoadBalancer",
                "elasticloadbalancing:CreateTargetGroup",
                "elasticloadbalancing:CreateListener",
                "elasticloadbalancing:CreateRule",
                "ec2:CreateSecurityGroup"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                }
            }
        },
        {
            "Sid": "ShieldProtection",
            "Effect": "Allow",
            "Action": [
                "shield:CreateProtection"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                }
            }
        },
        {
            "Sid": "ShieldTagResource",
            "Effect": "Allow",
            "Action": [
                "shield:TagResource"
            ],
            "Resource": "arn:aws:shield::*:protection/*",
            "Condition": {
```

```
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                }
            }
        }
    ]
}
```

Access Policy Reference

For more information about the Kubernetes permissions used by EKS Auto Mode, see the section called "Review access policies".

Learn about VPC Networking and Load Balancing in EKS Auto Mode

This topic explains how to configure Virtual Private Cloud (VPC) networking and load balancing features in EKS Auto Mode. While EKS Auto Mode manages most networking components automatically, you can still customize certain aspects of your cluster's networking configuration through NodeClass resources and load balancer annotations.

When you use EKS Auto Mode, AWS manages the VPC Container Network Interface (CNI) configuration and load balancer provisioning for your cluster. You can influence networking behaviors by defining NodeClass objects and applying specific annotations to your Service and Ingress resources, while maintaining the automated operational model that EKS Auto Mode provides.

Networking capability

EKS Auto Mode has a new networking capability that handles node and pod networking. You can configure it by creating a `NodeClass` Kubernetes object.

Configuration options for the previous AWS VPC CNI will not apply to EKS Auto Mode.

Configure networking with NodeClass

The NodeClass resource in EKS Auto Mode allows you to customize certain aspects of the networking capability. Through NodeClass, you can specify security group selections, control node placement across VPC subnets, set SNAT policies, configure network policies, and enable network event logging. This approach maintains the automated operational model of EKS Auto Mode while providing flexibility for network customization.

You can use a NodeClass to:

- Select a Security Group for Nodes
- Control how nodes are placed on VPC Subnets
- Set the Node SNAT Policy to `random` or `disabled`
- Set the Network Policy to Default Deny or Default Allow
- Enable Network Event Logging to a file.

Learn how to Create an Amazon EKS NodeClass.

Considerations

EKS Auto Mode supports:

- EKS Network Policies.
- The `HostPort` and `HostNetwork` options for Kubernetes Pods.
- Pods in public or private subnets.

EKS Auto Mode does **not** support:

- Security Groups per Pod (SGPP).
- Custom Networking. The IP Addresses of Pods and Nodes must be from the same CIDR Block.
- Warm IP, warm prefix, and warm ENI configurations.
- Minimum IP targets configuration.
- Enabling or disabling prefix delegation.
- Other configurations supported by the open-source AWS CNI.
- Network Policy configurations such as conntrack timer customization (default is 300s).
- Exporting network event logs to CloudWatch.

Load balancing

You configure AWS Elastic Load Balancers provisioned by EKS Auto Mode using annotations on Service and Ingress resources.

For more information, see the section called “Create ingress class” or the section called “Create service”.

Considerations for load balancing with EKS Auto Mode

- The default targeting mode is IP Mode, not Instance Mode.
- EKS Auto Mode only supports Security Group Mode for Network Load Balancers.
- AWS does not support migrating load balancers from the self managed AWS load balancer controller to management by EKS Auto Mode.
- The `networking.ingress.ipBlock` field in `TargetGroupBinding` spec is not supported.
- If your worker nodes use custom security groups (not `0` naming pattern), your cluster role needs additional IAM permissions. The default EKS-managed policy only allows EKS to modify security groups named `1`. Without permission to modify your custom security groups, EKS cannot add the required ingress rules that allow ALB/NLB traffic to reach your pods.
- You cannot bring your own target groups.

Troubleshoot EKS Auto Mode

With EKS Auto Mode, AWS assumes more responsibility for EC2 Instances in your AWS account. EKS assumes responsibility for the container runtime on nodes, the operating system on the nodes, and certain controllers. This includes a block storage controller, a load balancing controller, and a compute controller.

You must use AWS and Kubernetes APIs to troubleshoot nodes. You can:

- Use a Kubernetes `NodeDiagnostic` resource to retrieve node logs by using the the section called "Node monitoring agent". For more steps, see the section called "Get node logs".
- Use the AWS EC2 CLI command `get-console-output` to retrieve console output from nodes. For more steps, see the section called "Get console output from an EC2 managed instance by using the AWS EC2 CLI".
- Use Kubernetes *debugging containers* to retrieve node logs. For more steps, see the section called "Get node logs by using *debug containers* and the `kubectl` CLI".

> **Note**
>
> EKS Auto Mode uses EC2 managed instances. You cannot directly access EC2 managed instances, including by SSH.

You might have the following problems that have solutions specific to EKS Auto Mode components:

- Pods stuck in the `Pending` state, that aren't being scheduled onto Auto Mode nodes. For solutions see the section called "Troubleshoot Pod failing to schedule onto Auto Mode node".
- EC2 managed instances that don't join the cluster as Kubernetes nodes. For solutions see the section called "Troubleshoot node not joining the cluster".
- Errors and issues with the `NodePools`, `PersistentVolumes`, and `Services` that use the controllers that are included in EKS Auto Mode. For solutions see the section called "Troubleshoot included controllers in Auto Mode".
- Enhanced Pod security prevents sharing volumes across Pods. For solutions see the section called "Sharing Volumes Across Pods".

You can use the following methods to troubleshoot EKS Auto Mode components:

- the section called "Get console output from an EC2 managed instance by using the AWS EC2 CLI"
- the section called "Get node logs by using *debug containers* and the `kubectl` CLI"
- the section called "View resources associated with EKS Auto Mode in the AWS Console"
- the section called "View IAM Errors in your AWS account"
- the section called "Detect node connectivity issues with the `VPC Reachability Analyzer`"

Node monitoring agent

EKS Auto Mode includes the Amazon EKS node monitoring agent. You can use this agent to view troubleshooting and debugging information about nodes. The node monitoring agent publishes Kubernetes `events` and node `conditions`. For more information, see the section called "Node health".

Get console output from an EC2 managed instance by using the AWS EC2 CLI

This procedure helps with troubleshooting boot-time or kernel-level issues.

First, you need to determine the EC2 Instance ID of the instance associated with your workload. Second, use the AWS CLI to retrieve the console output.

1. Confirm you have `kubectl` installed and connected to your cluster
2. (Optional) Use the name of a Kubernetes Deployment to list the associated pods.

```
kubectl get pods -l app=<deployment-name>
```

3. Use the name of the Kubernetes Pod to determine the EC2 instance ID of the associated node.

```
kubectl get pod <pod-name> -o wide
```

4. Use the EC2 instance ID to retrieve the console output.

```
aws ec2 get-console-output --instance-id <instance id> --latest --output text
```

Get node logs by using *debug containers* and the `kubectl` CLI

The recommended way of retrieving logs from an EKS Auto Mode node is to use `NodeDiagnostic` resource. For these steps, see the section called “Get node logs”.

However, you can stream logs live from an instance by using the `kubectl debug node` command. This command launches a new Pod on the node that you want to debug which you can then interactively use.

1. Launch a debug container. The following command uses `i-01234567890123456` for the instance ID of the node, `-it` allocates a `tty` and attach `stdin` for interactive usage, and uses the `sysadmin` profile from the kubeconfig file.

```
kubectl debug node/i-01234567890123456 -it --profile=sysadmin --image=public.ecr.aws/
amazonlinux/amazonlinux:2023
```

   An example output is as follows.

```
Creating debugging pod node-debugger-i-01234567890123456-nxb9c with container
 debugger on node i-01234567890123456.
If you don't see a command prompt, try pressing enter.
bash-5.2#
```

2. From the shell, you can now install `util-linux-core` which provides the `nsenter` command. Use `nsenter` to enter the mount namespace of PID 1 (`init`) on the host, and run the `journalctl` command to stream logs from the `kubelet`:

```
yum install -y util-linux-core
nsenter -t 1 -m journalctl -f -u kubelet
```

For security, the Amazon Linux container image doesn't install many binaries by default. You can use the `yum whatprovides` command to identify the package that must be installed to provide a given binary.

```
yum whatprovides ps
```

```
Last metadata expiration check: 0:03:36 ago on Thu Jan 16 14:49:17 2025.
procps-ng-3.3.17-1.amzn2023.0.2.x86_64 : System and process monitoring utilities
Repo        : @System
Matched from:
Filename    : /usr/bin/ps
Provide    : /bin/ps

procps-ng-3.3.17-1.amzn2023.0.2.x86_64 : System and process monitoring utilities
Repo        : amazonlinux
Matched from:
Filename    : /usr/bin/ps
Provide    : /bin/ps
```

View resources associated with EKS Auto Mode in the AWS Console

You can use the AWS console to view the status of resources associated with your EKS Auto Mode cluster.

- EBS Volumes
  - View EKS Auto Mode volumes by searching for the tag key `eks:eks-cluster-name`
- Load Balancers
  - View EKS Auto Mode load balancers by searching for the tag key `eks:eks-cluster-name`
- EC2 Instances
  - View EKS Auto Mode instances by searching for the tag key `eks:eks-cluster-name`

View IAM Errors in your AWS account

1. Navigate to CloudTrail console
2. Select "Event History" from the left navigation pane
3. Apply error code filters:
   - AccessDenied
   - UnauthorizedOperation
   - InvalidClientTokenId

Look for errors related to your EKS cluster. Use the error messages to update your EKS access entries, cluster IAM role, or node IAM role. You might need to attach a new policy to these roles with permissions for EKS Auto Mode.

Troubleshoot Pod failing to schedule onto Auto Mode node

If pods staying in the `Pending` state and aren't being scheduled onto an auto mode node, verify if your pod or deployment manifest has a `nodeSelector`. If a `nodeSelector` is present, ensure that it is using `eks.amazonaws.com/compute-type: auto` to be scheduled on nodes that are made by EKS Auto Mode. For more information about the node labels that are used by EKS Auto Mode, see the section called "Control deployment".

Troubleshoot node not joining the cluster

EKS Auto Mode automatically configures new EC2 instances with the correct information to join the cluster, including the cluster endpoint and cluster certificate authority (CA). However, these instances can still fail to join the EKS cluster as a node. Run the following commands to identify instances that didn't join the cluster:

1. Run `kubectl get nodeclaim` to check for `NodeClaims` that are `Ready = False`.

```
kubectl get nodeclaim
```

2. Run `kubectl describe nodeclaim <node_claim>` and look under **Status** to find any issues preventing the node from joining the cluster.

```
kubectl describe nodeclaim <node_claim>
```

Common error messages:

`Error getting launch template configs`

You might receive this error if you are setting custom tags in the `NodeClass` with the default cluster IAM role permissions. See the section called "Identity and access".

`Error creating fleet`

There might be some authorization issue with calling the `RunInstances` call from the EC2 API. Check AWS CloudTrail for errors and see the section called "Auto Mode cluster IAM role" for the required IAM permissions.

Detect node connectivity issues with the `VPC Reachability Analyzer`

Note

You are charged for each analysis that is run the VPC Reachability Analyzer. For pricing details, see Amazon VPC Pricing.

One reason that an instance didn't join the cluster is a network connectivity issue that prevents them from reaching the API server. To diagnose this issue, you can use the VPC Reachability Analyzer to perform an analysis of the connectivity between a node that is failing to join the cluster and the API server. You will need two pieces of information:

- **instance ID** of a node that can't join the cluster
- IP address of the **Kubernetes API server endpoint**

To get the **instance ID**, you will need to create a workload on the cluster to cause EKS Auto Mode to launch an EC2 instance. This also creates a `NodeClaim` object in your cluster that will have the instance ID. Run `kubectl get nodeclaim -o yaml` to print all of the `NodeClaims` in your cluster. Each `NodeClaim` contains the instance ID as a field and again in the providerID:

```
kubectl get nodeclaim -o yaml
```

An example output is as follows.

```
    nodeName: i-01234567890123456
```

```
    providerID: aws:///us-west-2a/i-01234567890123456
```

You can determine your **Kubernetes API server endpoint** by running `kubectl get endpoint kubernetes -o yaml`. The addresses are in the addresses field:

```
kubectl get endpoints kubernetes -o yaml
```

An example output is as follows.

```
apiVersion: v1
kind: Endpoints
metadata:
  name: kubernetes
  namespace: default
subsets:
- addresses:
  - ip: 10.0.143.233
  - ip: 10.0.152.17
  ports:
  - name: https
    port: 443
    protocol: TCP
```

With these two pieces of information, you can perform the s analysis. First navigate to the VPC Reachability Analyzer in theAWS Management Console.

1. Click "Create and Analyze Path"
2. Provide a name for the analysis (e.g. "Node Join Failure")
3. For the "Source Type" select "Instances"
4. Enter the instance ID of the failing Node as the "Source"
5. For the "Path Destination" select "IP Address"
6. Enter one of the IP addresses for the API server as the "Destination Address"
7. Expand the "Additional Packet Header Configuration Section"
8. Enter a "Destination Port" of 443
9. Select "Protocol" as TCP if it is not already selected
10. Click "Create and Analyze Path"

11. The analysis might take a few minutes to complete. If the analysis results indicates failed reachability, it will indicate where the failure was in the network path so you can resolve the issue.

Sharing Volumes Across Pods

EKS Auto Mode Nodes are configured with SELinux in enforcing mode which provides more isolation between Pods that are running on the same Node. When SELinux is enabled, most non-privileged pods will automatically have their own multi-category security (MCS) label applied to them. This MCS label is unique per Pod, and is designed to ensure that a process in one Pod cannot manipulate a process in any other Pod or on the host. Even if a labeled Pod runs as root and has access to the host filesystem, it will be unable to manipulate files, make sensitive system calls on the host, access the container runtime, or obtain kubelet's secret key material.

Due to this, you may experience issues when trying to share data between Pods. For example, a `PersistentVolumeClaim` with an access mode of `ReadWriteOnce` will still not allow multiple Pods to access the volume concurrently.

To enable this sharing between Pods, you can use the Pod's `seLinuxOptions` to configure the same MCS label on those Pods. In this example, we assign the three categories `c123,c456,c789` to the Pod. This will not conflict with any categories assigned to Pods on the node automatically, as they will only be assigned two categories.

```
securityContext:
  seLinuxOptions:
    level: "s0:c123,c456,c789"
```

Troubleshoot included controllers in Auto Mode

If you have a problem with a controller, you should research:

- If the resources associated with that controller are properly formatted and valid.
- If the AWS IAM and Kubernetes RBAC resources are properly configured for your cluster. For more information, see the section called "Identity and access".

Organize workloads with Amazon EKS clusters

An Amazon EKS cluster consists of two primary components:

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

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

All of the data stored by the `etcd` nodes and associated Amazon EBS volumes is encrypted using AWS KMS. The cluster control plane is provisioned across multiple Availability Zones and fronted by an Elastic Load Balancing Network Load Balancer. Amazon EKS also provisions elastic network interfaces in your VPC subnets to provide connectivity from the control plane instances to the nodes (for example, to support `kubectl exec` `logs` `proxy` data flows).

Important

In the Amazon EKS environment, `etcd` storage is limited to 8 GiB as per upstream guidance. You can monitor a metric for the current database size by running the following command. If your cluster has a Kubernetes version below `1.28`, replace *`apiserver_storage_size_bytes`* with the following:

- Kubernetes version `1.27` and `1.26` – `apiserver_storage_db_total_size_in_bytes`
- Kubernetes version `1.25` and below – `etcd_db_total_size_in_bytes`

```
kubectl get --raw=/metrics | grep "apiserver_storage_size_bytes"
```

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

> **Note**
>
> - You can find out how the different components of Amazon EKS work in Configure networking for Amazon EKS clusters.
> - For connected clusters, see *Amazon EKS Connector*.

Topics

Create an Amazon EKS Auto Mode cluster

This topic provides detailed instructions for creating an Amazon EKS Auto Mode cluster using advanced configuration options. It covers prerequisites, networking options, and add-on configurations. The process includes setting up IAM roles, configuring cluster settings, specifying networking parameters, and selecting add-ons. Users can create clusters using either the AWS Management Console or the AWS CLI, with step-by-step guidance provided for both methods.

For users seeking a less complex setup process, refer to the following for simplified cluster creation steps:

- the section called “eksctl CLI”
- the section called “ AWS CLI”
- the section called “Management console”

This advanced configuration guide is intended for users who require more granular control over their EKS Auto Mode cluster setup and are familiar with Amazon EKS concepts and requirements. Before proceeding with the advanced configuration, ensure you have met all prerequisites and have a thorough understanding of the networking and IAM requirements for EKS Auto Mode clusters.

EKS Auto Mode requires additional IAM permissions. For more information, see:

- the section called “IAM Roles for EKS Auto Mode Clusters”
- the section called “Identity and access”

> **Note**
>
> If you want to create a cluster without EKS Auto Mode, see the section called “Create a cluster”.
> This topic covers advanced configuration. If you are looking to get started with EKS Auto Mode, see the section called “Create cluster”.

Prerequisites

- An existing VPC and subnets that meet Amazon EKS requirements. Before you deploy a cluster for production use, we recommend that you have a thorough understanding of the VPC and subnet requirements. If you don’t have a VPC and subnets, you can create them using an Amazon EKS provided AWS CloudFormation template.
- The `kubectl` command line tool is installed on your device or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. For example, if your cluster version is `1.29`, you can use `kubectl` version `1.28`, `1.29`, or `1.30` with it. To install or upgrade `kubectl`, see the section called “Set up kubectl and eksctl”.
- Version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version, use `aws --version`. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*.

- An IAM principal with permissions to create and modify EKS and IAM resources.

Create cluster - AWS console

1. Open the Amazon EKS console.
2. Choose **Add cluster** and then choose **Create**.
3. Under *Configuration options*, select **Custom configuration**.
    - This topic covers custom configuration. For information about Quick configuration, see the section called “Management console”.
4. Confirm **Use EKS Auto Mode** is enabled.
    - This topic covers creating clusters with EKS Auto Mode. For more information about creating clusters without EKS Auto Mode, see the section called “Create a cluster”.
5. On the **Configure cluster** page, enter the following fields:
    - **Name –** A name for your cluster. The name can contain only alphanumeric characters (case-sensitive), hyphens, and underscores. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.
    - **Cluster IAM role –** Choose the Amazon EKS cluster IAM role that you created to allow the Kubernetes control plane to manage AWS resources on your behalf. If you haven't previously created a Cluster IAM role for EKS Auto Mode, select the **Create recommended role** button to create the role with the required permissions in the IAM console.
    - **Kubernetes version –** The version of Kubernetes to use for your cluster. We recommend selecting the latest version, unless you need an earlier version.
    - **Upgrade policy** — The Kubernetes version policy you would like to set for your cluster. If you want your cluster to only run on a standard support version, you can choose **Standard**. If you want your cluster to enter extended support at the end of standard support for a version, you can choose **Extended**. If you select a Kubernetes version that is currently in extended support, you can not select standard support as an option.
6. In the **Auto Mode Compute** section of the configure cluster page, enter the following fields:
    - **Node pools** — Determine if you want to use the build in node pools. For more information, see the section called “Review built-in node pools”.
    - **Node IAM role** — If you enable any of the built-in node pools, you need to select a Node IAM Role. EKS Auto Mode will assign this role to new nodes. You cannot change this value after the cluster is created. If you haven't previously created a Node IAM role for EKS Auto Mode,

select the Create recommended role button to create the role with the required permissions. For more information about this role, see the section called "Identity and access".

7. In the **Cluster access** section of the configure cluster page, enter the following fields:
   - **Bootstrap cluster administrator access** — The cluster creator is automatically a Kubernetes administrator. If you want to disable this, select **Disallow cluster administrator access**.
   - **Cluster authentication mode** — EKS Auto Mode requires EKS access entries, the EKS API authentication mode. You can optionally enable the `ConfigMap` authentication mode by selecting **EKS API and ConfigMap**.
8. Enter the remaining fields on the configure cluster page:
   - **Secrets encryption** – (Optional) Choose to enable secrets encryption of Kubernetes secrets using a KMS key. You can also enable this after you create your cluster. Before you enable this capability, make sure that you're familiar with the information in the section called "Enable secret encryption".
   - **ARC Zonal shift** — EKS Auto Mode does not support Arc Zonal shift.
   - **Tags** – (Optional) Add any tags to your cluster. For more information, see the section called "Tagging your resources".

   When you're done with this page, choose **Next**.
9. On the **Specify networking** page, select values for the following fields:
   - **VPC** – Choose an existing VPC that meets Amazon EKS VPC requirements to create your cluster in. Before choosing a VPC, we recommend that you're familiar with all of the requirements and considerations in the section called "VPC and subnet requirements". You can't change which VPC you want to use after cluster creation. If no VPCs are listed, then you need to create one first. For more information, see the section called "Create a VPC".
   - **Subnets** – By default, all available subnets in the VPC specified in the previous field are preselected. You must select at least two.

     The subnets that you choose must meet the Amazon EKS subnet requirements. Before selecting subnets, we recommend that you're familiar with all of the Amazon EKS VPC and subnet requirements and considerations.

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

     Whether you choose any security groups or not, Amazon EKS creates a security group that enables communication between your cluster and your VPC. Amazon EKS associates this security group, and any that you choose, to the network interfaces that it creates. For more information about the cluster security group that Amazon EKS creates, see the section called "Security group requirements". You can modify the rules in the cluster security group that Amazon EKS creates.

- **Choose cluster IP address family** – You can choose either **IPv4** and **IPv6**.

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

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

  The CIDR block must meet the following requirements:

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

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

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

  When you're done with this page, choose **Next**.

10. (Optional) On the **Configure observability** page, choose which **Metrics** and **Control plane logging** options to turn on. By default, each log type is turned off.

- For more information about the Prometheus metrics option, see the section called "Step 1: Turn on Prometheus metrics".
- For more information about the **Control plane logging** options, see the section called "Control plane logs".
- When you're done with this page, choose **Next**.

11. On the **Select add-ons** page, choose the add-ons that you want to add to your cluster. You can choose as many **Amazon EKS add-ons** and **AWS Marketplace add-ons** as you require. If the **AWS Marketplace add-ons** that you want to install isn't listed, you can click the page numbering to view additional page results or search for available **AWS Marketplace add-ons** by entering text in the search box. You can also filter by **category**, **vendor**, or **pricing model** and then choose the add-ons from the search results. When creating a cluster, you can view, select, and install any add-on that supports EKS Pod Identities as detailed in the section called "Pod Identity".
    - EKS Auto Mode automates the functionality of certain add-ons. If you plan to deploy EKS Managed Node Groups to your EKS Auto Mode Cluster, select **Additional Amazon EKS Add-ons** and review the options. You may need to install add-ons such as CoreDNS and kube-proxy. EKS will only install the add-ons in this section on self-managed nodes and node groups.
    - When you're done with this page, choose **Next**.
12. On the **Configure selected add-ons settings** page, select the version that you want to install. You can always update to a later version after cluster creation.

    For add-ons that support EKS Pod Identities, you can use the console to automatically generate the role with the name, AWS managed policy, and trust policy prepopulated specifically for the add-on. You can re-use existing roles or create new roles for supported add-ons. For the steps to use the console to create roles for add-ons that support EKS Pod Identities, see the section called "Create add-on (AWS Console)". If an add-on does not support EKS Pod Identity, a message displays with instructions to use the wizard to create the IAM roles for service accounts (IRSA) after the cluster is created.

    You can update the configuration of each add-on after cluster creation. For more information about configuring add-ons, see the section called "Update an add-on". When you're done with this page, choose **Next**.
13. On the **Review and create** page, review the information that you entered or selected on the previous pages. If you need to make changes, choose **Edit**. When you're satisfied, choose **Create**. The **Status** field shows **CREATING** while the cluster is provisioned.

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

Cluster provisioning takes several minutes.

Create cluster - AWS CLI

The following CLI instructions cover creating IAM resources and creating the cluster.

Create an EKS Auto Mode Cluster IAM Role

Step 1: Create the Trust Policy

Create a trust policy that allows the Amazon EKS service to assume the role. Save the policy as `trust-policy.json`:

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

Step 2: Create the IAM Role

Use the trust policy to create the Cluster IAM Role:

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

Step 3: Note the Role ARN

Retrieve and save the ARN of the new role for use in subsequent steps:

```
aws iam get-role --role-name AmazonEKSAutoClusterRole --query "Role.Arn" --output text
```

Step 4: Attach Required Policies

Attach the following AWS managed policies to the Cluster IAM Role to grant the necessary permissions:

AmazonEKSClusterPolicy:

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

AmazonEKSComputePolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSComputePolicy
```

AmazonEKSBlockStoragePolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSBlockStoragePolicy
```

AmazonEKSLoadBalancingPolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
```

```
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSLoadBalancingPolicy
```

AmazonEKSNetworkingPolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSNetworkingPolicy
```

Create an EKS Auto Mode Node IAM Role

Step 1: Create the Trust Policy

Create a trust policy that allows the Amazon EKS service to assume the role. Save the policy as `node-trust-policy.json`:

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

Step 2: Create the Node IAM Role

Use the **node-trust-policy.json** file from the previous step to define which entities can assume the role. Run the following command to create the Node IAM Role:

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

Step 3: Note the Role ARN

After creating the role, retrieve and save the ARN of the Node IAM Role. You will need this ARN in subsequent steps. Use the following command to get the ARN:

```
aws iam get-role --role-name AmazonEKSAutoNodeRole --query "Role.Arn" --output text
```

Step 4: Attach Required Policies

Attach the following AWS managed policies to the Node IAM Role to provide the necessary permissions:

AmazonEKSWorkerNodeMinimalPolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodeMinimalPolicy
```

AmazonEC2ContainerRegistryPullOnly:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

Create cluster

1. Create your cluster with the command that follows. Before running the command, make the following replacements:
   - Replace *`region-code`* with the AWS Region that you want to create your cluster in.
   - Replace *`my-cluster`* with a name for your cluster. The name can contain only alphanumeric characters (case-sensitive), hyphens, and underscores. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.
   - Replace *`1.30`* with any Amazon EKS supported version.
   - Replace *`111122223333`* with your account ID
   - If you have created differently named IAM Roles for the Cluster and Node roles, replace the ARNs.
   - Replace the values for `subnetIds` with your own. You can also add additional IDs. You must specify at least two subnet IDs.

     The subnets that you choose must meet the Amazon EKS subnet requirements. Before selecting subnets, we recommend that you're familiar with all of the Amazon EKS VPC and subnet requirements and considerations.

- If you don't want to specify a security group ID, remove `,securityGroupIds=sg-<ExampleID1>` from the command. If you want to specify one or more security group IDs, replace the values for `securityGroupIds` with your own. You can also add additional IDs.

  Whether you choose any security groups or not, Amazon EKS creates a security group that enables communication between your cluster and your VPC. Amazon EKS associates this security group, and any that you choose, to the network interfaces that it creates. For more information about the cluster security group that Amazon EKS creates, see the section called "Security group requirements". You can modify the rules in the cluster security group that Amazon EKS creates.

```
aws eks create-cluster \
  --region region-code \
  --name my-cluster \
  --kubernetes-version 1.30 \
  --role-arn arn:aws:iam::111122223333:role/AmazonEKSAutoClusterRole \
  --resources-vpc-config '{"subnetIds": ["subnet-ExampleID1","subnet-ExampleID2"],
 "securityGroupIds": ["sg-ExampleID1"], "endpointPublicAccess": true,
 "endpointPrivateAccess": true}' \
  --compute-config '{"enabled": true, "nodeRoleArn":
 "arn:aws:iam::111122223333:role/AmazonEKSAutoNodeRole", "nodePools": ["general-
purpose", "system"]}' \
  --kubernetes-network-config '{"elasticLoadBalancing": {"enabled": true}}' \
  --storage-config '{"blockStorage": {"enabled": true}}' \
  --access-config '{"authenticationMode": "API"}'
```

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

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

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

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

  The CIDR block must meet the following requirements:

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

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

- If you're creating a cluster and want the cluster to assign IPv6 addresses to Pods and services instead of IPv4 addresses, add `--kubernetes-network-config ipFamily=ipv6` to the following command.

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

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

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

Next steps

- the section called "Access cluster with kubectl"
- the section called "Grant permissions"
- Enable secrets encryption for your cluster.
- Configure logging for your cluster.
- Add nodes to your cluster.

Create an Amazon EKS cluster

> **Note**
>
> This topic covers creating EKS clusters without EKS Auto Mode.
> For detailed instructions on creating an EKS Auto Mode cluster, see the section called "Create auto cluster".
> To get started with EKS Auto Mode, see the section called "Create cluster (EKS Auto Mode)".

This topic provides an overview of the available options and describes what to consider when you create an Amazon EKS cluster. If you need to create a cluster with your on-premises infrastructure as the compute for nodes, see the section called "Create cluster". If this is your first time creating an Amazon EKS cluster, we recommend that you follow one of our guides in *Get started*. These guides help you to create a simple, default cluster without expanding into all of the available options.

Prerequisites

- An existing VPC and subnets that meet Amazon EKS requirements. Before you deploy a cluster for production use, we recommend that you have a thorough understanding of the VPC and subnet requirements. If you don't have a VPC and subnets, you can create them using an Amazon EKS provided AWS CloudFormation template.
- The `kubectl` command line tool is installed on your device or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. To install or upgrade `kubectl`, see the section called "Set up kubectl and eksctl".

- Version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version, use `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*. The AWS CLI version that is installed in AWS CloudShell might also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the *AWS CloudShell User Guide*.
- An IAM principal with permissions to `create` and `describe` an Amazon EKS cluster. For more information, see the section called "Create a local Kubernetes cluster on an Outpost" and the section called "List or describe all clusters".

Step 1: Create cluster IAM role

1. If you already have a cluster IAM role, or you're going to create your cluster with `eksctl`, then you can skip this step. By default, `eksctl` creates a role for you.
2. Run the following command to create an IAM trust policy JSON file.

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

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

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

4. You can assign either the Amazon EKS managed policy or create your own custom policy. For the minimum permissions that you must use in your custom policy, see the section called “Cluster IAM role”.

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

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

Service Linked Role

Amazon EKS automatically creates a service linked role called `AWSServiceRoleForAmazonEKS`.

This is in addition to the cluster IAM role. A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. The role allows Amazon EKS to manage clusters in your account. For more information, see the section called “Cluster role”.

The IAM Identity you use to create the EKS cluster must have permission to create the service-linked role. This includes the `iam:CreateServiceLinkedRole` permission.

If the service linked role doesn’t already exist, and your current IAM role doesn’t have sufficient permissions to create it, the cluster create operation will fail.

Step 2: Create cluster

You can create a cluster by using:

- `eksctl`
- the AWS Management Console
- the AWS CLI

Create cluster - eksctl

1. You need version `0.204.0` or later of the `eksctl` command line tool installed on your device or AWS CloudShell. To install or update `eksctl`, see Installation in the `eksctl` documentation.
2. Create an Amazon EKS `IPv4` cluster with the Amazon EKS default Kubernetes version in your default AWS Region. Before running command, make the following replacements:
3. Replace *`region-code`* with the AWS Region that you want to create your cluster in.
4. Replace *`my-cluster`* with a name for your cluster. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.
5. Replace *`1.32`* with any Amazon EKS supported version.
6. Change the values for `vpc-private-subnets` to meet your requirements. You can also add additional IDs. You must specify at least two subnet IDs. If you'd rather specify public subnets, you can change `--vpc-private-subnets` to `--vpc-public-subnets`. Public subnets have an associated route table with a route to an internet gateway, but private subnets don't have an associated route table. We recommend using private subnets whenever possible.

   The subnets that you choose must meet the Amazon EKS subnet requirements. Before selecting subnets, we recommend that you're familiar with all of the Amazon EKS VPC and subnet requirements and considerations.
7. Run the following command:

   ```
   eksctl create cluster --name my-cluster --region region-code --version {k8s-n} --vpc-private-subnets subnet-ExampleID1,subnet-ExampleID2 --without-nodegroup
   ```

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

   ```
   [#]  EKS cluster "my-cluster" in "region-code" region is ready
   ```
8. Continue with the section called “Step 3: Update kubeconfig”

Optional Settings

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

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

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

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

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

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

    You can only specify this option when using the `IPv4` address family and only at cluster creation. If you don’t specify this, then Kubernetes assigns service IP addresses from either the `10.100.0.0/16` or `172.20.0.0/16` CIDR blocks.
- If you’re creating cluster and want the cluster to assign `IPv6` addresses to Pods and services instead of `IPv4` addresses, specify the ipFamily option.

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

Create cluster - AWS console

1. Open the Amazon EKS console.
2. Choose **Add cluster** and then choose **Create**.
3. Under **Configuration options** select **Custom configuration**
   - For information about quickly creating a cluster wih EKS Auto Mode, see the section called "Management console".
4. Under **EKS Auto Mode**, toggle **Use EKS Auto Mode** off.
   - For information about creating an EKS Auto Mode cluster with custom configuration, see the section called "Create auto cluster".
5. On the **Configure cluster** page, enter the following fields:
   - **Name** – A name for your cluster. The name can contain only alphanumeric characters (case-sensitive), hyphens, and underscores. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.
   - **Cluster IAM role** – Choose the Amazon EKS cluster IAM role that you created to allow the Kubernetes control plane to manage AWS resources on your behalf.
   - **Kubernetes version** – The version of Kubernetes to use for your cluster. We recommend selecting the latest version, unless you need an earlier version.
   - **Support type** — The Kubernetes version policy you would like to set for your cluster. If you want your cluster to only run on a standard support version, you can choose **Standard support**. If you want your cluster to enter extended support at the end of standard support for a version, you can choose **Extended support**. If you select a Kubernetes version that is currently in extended support, you can not select standard support as an option.

- **Secrets encryption** – (Optional) Choose to enable secrets encryption of Kubernetes secrets using a KMS key. You can also enable this after you create your cluster. Before you enable this capability, make sure that you're familiar with the information in the section called "Enable secret encryption".
- **Tags** – (Optional) Add any tags to your cluster. For more information, see the section called "Tagging your resources".
- **ARC Zonal shift** - (Optional) You can use Route53 Application Recovery controller to mitigate impaired availability zones. For more information, see the section called "Learn about Zonal Shift".

6. In the **Cluster access** section of the configure cluster page, enter the following fields:
   - **Bootstrap cluster administrator access** — The cluster creator is automatically a Kubernetes administrator. If you want to disable this, select **Disallow cluster administrator access**.
   - **Cluster authentication mode** — Determine how you want to grant IAM users and roles access to Kubernetes APIs. For more information, see the section called "Set Cluster Authentication Mode".

   When you're done with this page, choose **Next**.
7. On the **Specify networking** page, select values for the following fields:
   - **VPC** – Choose an existing VPC that meets Amazon EKS VPC requirements to create your cluster in. Before choosing a VPC, we recommend that you're familiar with all of the requirements and considerations in the section called "VPC and subnet requirements". You can't change which VPC you want to use after cluster creation. If no VPCs are listed, then you need to create one first. For more information, see the section called "Create a VPC".
   - **Subnets** – By default, all available subnets in the VPC specified in the previous field are preselected. You must select at least two.

     The subnets that you choose must meet the Amazon EKS subnet requirements. Before selecting subnets, we recommend that you're familiar with all of the Amazon EKS VPC and subnet requirements and considerations.

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

     Whether you choose any security groups or not, Amazon EKS creates a security group that enables communication between your cluster and your VPC. Amazon EKS associates this security group, and any that you choose, to the network interfaces that it creates. For more information about the cluster security group that Amazon EKS creates, see the section called "Security group requirements". You can modify the rules in the cluster security group that Amazon EKS creates.

- **Choose cluster IP address family** – You can choose either **IPv4** and **IPv6**.

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

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

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

  The CIDR block must meet the following requirements:

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

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

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

  When you're done with this page, choose **Next**.

8. (Optional) On the **Configure observability** page, choose which **Metrics** and **Control plane logging** options to turn on. By default, each log type is turned off.

- For more information about the Prometheus metrics option, see the section called "Step 1: Turn on Prometheus metrics".
- For more information about the **Control plane logging** options, see the section called "Control plane logs".

When you're done with this page, choose **Next**.

9. On the **Select add-ons** page, choose the add-ons that you want to add to your cluster. Certain add-ons are pre-selected. You can choose as many **Amazon EKS add-ons** and **AWS Marketplace add-ons** as you require. If the **AWS Marketplace add-ons** that you want to install isn't listed, you can click the page numbering to view additional page results or search for available **AWS Marketplace add-ons** by entering text in the search box. You can also filter by **category**, **vendor**, or **pricing model** and then choose the add-ons from the search results. When creating a cluster, you can view, select, and install any add-on that supports EKS Pod Identities as detailed in the section called "Pod Identity".

   When you're done with this page, choose **Next**.

   Some add-ons, such as Amazon VPC CNI, CoreDNS, and kube-proxy, are installed by default. If you disable any of the default add-ons, this may affect your ability to run Kubernetes applications.

10. On the **Configure selected add-ons settings** page, select the version that you want to install. You can always update to a later version after cluster creation.

    For add-ons that support EKS Pod Identities, you can use the console to automatically generate the role with the name, AWS managed policy, and trust policy prepopulated specifically for the add-on. You can re-use existing roles or create new roles for supported add-ons. For the steps to use the console to create roles for add-ons that support EKS Pod Identities, see the section called "Create add-on (AWS Console)". If an add-on does not support EKS Pod Identity, a message displays with instructions to use the wizard to create the IAM roles for service accounts (IRSA) after the cluster is created.

    You can update the configuration of each add-on after cluster creation. For more information about configuring add-ons, see the section called "Update an add-on". When you're done with this page, choose **Next**.

11. On the **Review and create** page, review the information that you entered or selected on the previous pages. If you need to make changes, choose **Edit**. When you're satisfied, choose **Create**. The **Status** field shows **CREATING** while the cluster is provisioned.

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

Cluster provisioning takes several minutes.

12. Continue with the section called "Step 3: Update kubeconfig"

Create cluster - AWS CLI

1. Create your cluster with the command that follows. Before running the command, make the following replacements:

   - Replace *region-code* with the AWS Region that you want to create your cluster in.
   - Replace *my-cluster* with a name for your cluster. The name can contain only alphanumeric characters (case-sensitive), hyphens, and underscores. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.
   - Replace *1.32* with any Amazon EKS supported version.
   - Replace *111122223333* with your account ID and *myAmazonEKSClusterRole* with the name of your cluster IAM role.
   - Replace the values for `subnetIds` with your own. You can also add additional IDs. You must specify at least two subnet IDs.

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

Whether you choose any security groups or not, Amazon EKS creates a security group that enables communication between your cluster and your VPC. Amazon EKS associates this security group, and any that you choose, to the network interfaces that it creates. For more information about the cluster security group that Amazon EKS creates, see the section called “Security group requirements”. You can modify the rules in the cluster security group that Amazon EKS creates.

```
aws eks create-cluster --region region-code --name my-cluster --kubernetes-version
 1.32 \
   --role-arn arn:aws:iam::111122223333:role/myAmazonEKSClusterRole \
   --resources-vpc-config subnetIds=subnet-ExampleID1,subnet-
ExampleID2,securityGroupIds=sg-ExampleID1
```

Note

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

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

- By default, EKS installs multiple networking add-ons during cluster creation. This includes the Amazon VPC CNI, CoreDNS, and kube-proxy.

  If you’d like to disable the installation of these default networking add-ons, use the parameter below. This may be used for alternate CNIs, such as Cilium. Review the EKS API reference for more information.

  ```
  aws eks create-cluster --bootstrapSelfManagedAddons false
  ```

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

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

The CIDR block must meet the following requirements:

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

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

- If you're creating a cluster and want the cluster to assign `IPv6` addresses to Pods and services instead of `IPv4` addresses, add `--kubernetes-network-config ipFamily=ipv6` to the following command.

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

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

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

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

3. Continue with the section called "Step 3: Update kubeconfig"

Step 3: Update kubeconfig

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

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

   An example output is as follows.

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

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

   ```
   kubectl get svc
   ```

   An example output is as follows.

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

Step 4: Cluster setup

1. (Recommended) To use some Amazon EKS add-ons, or to enable individual Kubernetes workloads to have specific AWS Identity and Access Management (IAM) permissions, create an IAM OpenID Connect (OIDC) provider for your cluster. You only need to create an IAM OIDC provider for your cluster once. To learn more about Amazon EKS add-ons, see the section called "Amazon EKS add-ons". To learn more about assigning specific IAM permissions to your workloads, see the section called "Credentials with IRSA".

2. (Recommended) Configure your cluster for the Amazon VPC CNI plugin for Kubernetes plugin before deploying Amazon EC2 nodes to your cluster. By default, the plugin was installed with your cluster. When you add Amazon EC2 nodes to your cluster, the plugin is automatically deployed to each Amazon EC2 node that you add. The plugin requires you to attach one of the following IAM policies to an IAM role. If your cluster uses the `IPv4` family, use the

AmazonEKS_CNI_Policy managed IAM policy. If your cluster uses the `IPv6` family, use an IAM policy that you create.

The IAM role that you attach the policy to can be the node IAM role, or a dedicated role used only for the plugin. We recommend attaching the policy to this role. For more information about creating the role, see the section called “Configure for IRSA” or Amazon EKS node IAM role.

3. If you deployed your cluster using the AWS Management Console, you can skip this step. The AWS Management Console deploys the Amazon VPC CNI plugin for Kubernetes, CoreDNS, and `kube-proxy` Amazon EKS add-ons, by default.

   If you deploy your cluster using either `eksctl` or the AWS CLI, then the Amazon VPC CNI plugin for Kubernetes, CoreDNS, and `kube-proxy` self-managed add-ons are deployed. You can migrate the Amazon VPC CNI plugin for Kubernetes, CoreDNS, and `kube-proxy` self-managed add-ons that are deployed with your cluster to Amazon EKS add-ons. For more information, see the section called “Amazon EKS add-ons”.
4. (Optional) If you haven’t already done so, you can enable Prometheus metrics for your cluster. For more information, see Create a scraper in the *Amazon Managed Service for Prometheus User Guide*.
5. If you plan to deploy workloads to your cluster that use Amazon EBS volumes , and you created a `1.23` or later cluster, then you must install the Amazon EBS CSI to your cluster before deploying the workloads.

Next steps

- The IAM principal that created the cluster is the only principal that has access to the cluster. Grant permissions to other IAM principals so they can access your cluster.
- If the IAM principal that created the cluster only has the minimum IAM permissions referenced in the prerequisites, then you might want to add additional Amazon EKS permissions for that principal. For more information about granting Amazon EKS permissions to IAM principals, see the section called “IAM Reference”.
- If you want the IAM principal that created the cluster, or any other principals to view Kubernetes resources in the Amazon EKS console, grant the Required permissions to the entities.
- If you want nodes and IAM principals to access your cluster from within your VPC, enable the private endpoint for your cluster. The public endpoint is enabled by default. You can disable the public endpoint once you’ve enabled the private endpoint, if desired. For more information, see the section called “Configure endpoint access”.

- Enable secrets encryption for your cluster.
- Configure logging for your cluster.
- Add nodes to your cluster.

Prepare for Kubernetes version upgrades with cluster insights

Amazon EKS cluster insights provide recommendations to help you follow Amazon EKS and Kubernetes best practices. Every Amazon EKS cluster undergoes automatic, recurring checks against an Amazon EKS curated list of insights. These insight checks are fully managed by Amazon EKS and offer recommendations on how to address any findings.

- Before updating your cluster Kubernetes version, check the **Cluster insights** tab of the observability dashboard in the Amazon EKS console.
- If your cluster has identified issues, review them and make appropriate fixes. The issues include links to Amazon EKS and Kubernetes.
- After fixing issues, wait for the cluster insights to refresh. If all issues have been resolved, update your cluster.

Amazon EKS returns insights related to Kubernetes version upgrade readiness. Upgrade insights identify possible issues that could impact Kubernetes cluster upgrades. This minimizes the effort that administrators spend preparing for upgrades and increases the reliability of applications on newer Kubernetes versions. Clusters are automatically scanned by Amazon EKS against a list of possible Kubernetes version upgrade impacting issues. Amazon EKS frequently updates the list of insight checks based on reviews of changes made in each Kubernetes version release.

Amazon EKS upgrade insights speed up the testing and verification process for new versions. They also allow cluster administrators and application developers to leverage the newest Kubernetes capabilities by highlighting concerns and offering remediation advice. To see the list of insight checks performed and any relevant issues that Amazon EKS has identified, you can call the Amazon EKS `ListInsights` API operation or look in the Amazon EKS console.

Cluster insights update periodically. You cannot manually refresh cluster insights. If you fix a cluster issue, it will take some time for cluster insights to update. To determine if a fix was successful, compare the time the change deployed to the "last refresh time" of the cluster insight.

Amazon EKS automatically creates a cluster access entry. This entry gives EKS permission to view information about your cluster. This information is used to generate the insights. For more information, see the section called "AmazonEKSClusterInsightsPolicy".

View cluster insights (Console)

1. Open the Amazon EKS console.
2. From the cluster list, choose the name of the Amazon EKS cluster for which you want to see the insights.
3. Choose **View dashboard**.
4. Choose the **Cluster Insights** tab.
5. In the **Upgrade Insights** table, you will see the following columns:
   - **Name –** The check that was performed by Amazon EKS against the cluster.
   - **Insight status –** An insight with a status of "Error" typically means the impacted Kubernetes version is N+1 of the current cluster version, while a status of "Warning" means the insight applies to a future Kubernetes version N+2 or more. An insight with status of "Passing" means Amazon EKS has not found any issues associated with this insight check in your cluster. An insight status of "Unknown" means Amazon EKS is unable to determine if your cluster is impacted by this insight check.
   - **Version –** The Kubernetes version that the insight checked for possible issues.
   - **Last refresh time –** The time the status of the insight was last refreshed for this cluster.
   - **Last transition time –** The time the status of this insight last changed.
   - **Description –** Information from the insight check, which includes the alert and recommended actions for remediation.

View cluster insights (AWS CLI)

1. Determine which cluster you would like to check for insights. The following command lists the insights for a specified cluster. Make the following modifications to the command as needed and then run the modified command:
   - Replace *region-code* with the code for your AWS Region.
   - Replace *my-cluster* with the name of your cluster.

```
aws eks list-insights --region region-code --cluster-name my-cluster
```

An example output is as follows.


```
{
"insights":
    [
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "name": "Deprecated APIs removed in Kubernetes vX.XX",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557315.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for usage of deprecated APIs that are scheduled
 for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
 updated APIs supported by vX.XX could cause application impact.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "No deprecated API usage detected within the last 30
 days.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
            "name": "Kubelet version skew",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557309.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for kubelet versions of worker nodes in the
 cluster to see if upgrade would cause non compliance with supported Kubernetes
 kubelet version skew policy.",
            "insightStatus":
            {
                "status": "UNKNOWN",
                "reason": "Unable to determine status of node kubelet versions.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
            "name": "Deprecated APIs removed in Kubernetes vX.XX",
```

```
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557315.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for usage of deprecated APIs that are scheduled
for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
updated APIs supported by vX.XX could cause application impact.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "No deprecated API usage detected within the last 30
days.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
            "name": "Cluster health issues",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557314.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for any cluster health issues that prevent
successful upgrade to the next Kubernetes version on EKS.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "No cluster health issues detected.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
            "name": "EKS add-on version compatibility",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557314.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks version of installed EKS add-ons to ensure they
are compatible with the next version of Kubernetes. ",
            "insightStatus": { "status": "PASSING", "reason": "All installed EKS
add-on versions are compatible with next Kubernetes version."},
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEccccc",
            "name": "kube-proxy version skew",
```

```
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557314.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks version of kube-proxy in cluster to see if
 upgrade would cause non compliance with supported Kubernetes kube-proxy version
 skew policy.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "kube-proxy versions match the cluster control plane
 version.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEddddd",
            "name": "Deprecated APIs removed in Kubernetes vX.XX",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557315.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for usage of deprecated APIs that are scheduled
 for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
 updated APIs supported by vX.XX could cause application impact.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "No deprecated API usage detected within the last 30
 days.",
            },
        },
    ],
"nextToken": null,
}
```


2. For descriptive information about the insight, run the following command. Make the following modifications to the command as needed and then run the modified command:

   - Replace *region-code* with the code for your AWS Region.
   - Replace *a1b2c3d4-5678-90ab-cdef-EXAMPLE22222* with an insight ID retrieved from listing the cluster insights.
   - Replace *my-cluster* with the name of your cluster.

```
aws eks describe-insight --region region-code --id a1b2c3d4-5678-90ab-cdef-
EXAMPLE22222 --cluster-name my-cluster
```

An example output is as follows.

```
{
  "insight":
    {
      "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
      "name": "Kubelet version skew",
      "category": "UPGRADE_READINESS",
      "kubernetesVersion": "1.27",
      "lastRefreshTime": 1734557309.000,
      "lastTransitionTime": 1734557309.000,
      "description": "Checks for kubelet versions of worker nodes in the cluster
 to see if upgrade would cause non compliance with supported Kubernetes kubelet
 version skew policy.",
      "insightStatus":
        {
          "status": "UNKNOWN",
          "reason": "Unable to determine status of node kubelet versions.",
        },
      "recommendation": "Upgrade your worker nodes to match the Kubernetes version
 of your cluster control plane.",
      "additionalInfo":
        {
          "Kubelet version skew policy": "https://kubernetes.io/releases/version-
skew-policy/#kubelet",
          "Updating a managed node group": "https://docs.aws.amazon.com/eks/latest/
userguide/update-managed-node-group.html",
        },
      "resources": [],
      "categorySpecificSummary":
        { "deprecationDetails": [], "addonCompatibilityDetails": [] },
    },
}
```

Update existing cluster to new Kubernetes version

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

> **Important**
>
> Once you upgrade a cluster, you can't downgrade to a previous version. We recommend that, before you update to a new Kubernetes version, you review the information in Understand the Kubernetes version lifecycle on EKS and also review in the update steps in this topic.

New Kubernetes versions sometimes introduce significant changes. Therefore, we recommend that you test the behavior of your applications against a new Kubernetes version before you update your production clusters. You can do this by building a continuous integration workflow to test your application behavior before moving to a new Kubernetes version.

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

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

To ensure that the API server endpoint for your cluster is always accessible, Amazon EKS provides a highly available Kubernetes control plane and performs rolling updates of API server instances during update operations. In order to account for changing IP addresses of API server instances supporting your Kubernetes API server endpoint, you must ensure that your API server clients manage reconnects effectively. Recent versions of `kubectl` and the Kubernetes client libraries that are officially supported, perform this reconnect process transparently.

> **Note**
>
> To learn more about what goes into a cluster update, see Best Practices for Cluster Upgrades in the EKS Best Practices Guide. This resource helps you plan an upgrade, and understand the strategy of upgrading a cluster.

Considerations for Amazon EKS Auto Mode

- The compute capability of Amazon EKS Auto Mode controls the Kubernetes version of nodes. After you upgrade the control plane, EKS Auto Mode will begin incrementally updating managed nodes. EKS Auto Mode respects pod disruption budgets.
- You do not have to manually upgrade the capabilities of Amazon EKS Auto Mode, including the compute autoscaling, block storage, and load balancing capabilities.

Step 1: Prepare for upgrade

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

     ```
     kubectl version
     ```

   - Get the Kubernetes version of your nodes. This command returns all self-managed and managed Amazon EC2, Fargate, and hybrid nodes. Each Fargate Pod is listed as its own node.

     ```
     kubectl get nodes
     ```

   Before updating your control plane to a new Kubernetes version, make sure that the Kubernetes minor version of both the managed nodes and Fargate nodes in your cluster are the same as your control plane's version. For example, if your control plane is running version `1.29` and one of your nodes is running version `1.28`, then you must update your nodes to version `1.29` before updating your control plane to 1.30. We also recommend that you update your self-managed nodes and hybrid nodes to the same version as your control plane before updating the control plane. For more information, see the section called "Update", the section called "Update methods", and the section called "Upgrade hybrid nodes". If you have Fargate nodes with a minor version lower than the control plane version, first delete the Pod that's represented by the node. Then update your control plane. Any remaining Pods will update to the new version after you redeploy them.

2. If the Kubernetes version that you originally deployed your cluster with was Kubernetes `1.25` or later, skip this step.

   By default, the Pod security policy admission controller is enabled on Amazon EKS clusters. Before updating your cluster, ensure that the proper Pod security policies are in place. This is to avoid potential security issues. You can check for the default policy with the `kubectl get psp eks.privileged` command.

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

   If you receive the following error, see the section called "Amazon EKS default Pod security policy" before proceeding.

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

3. If the Kubernetes version that you originally deployed your cluster with was Kubernetes `1.18` or later, skip this step.

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

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

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

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

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

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

Step 2: Review upgrade considerations

Review the [Deprecated API Migration Guide] in the Kubernetes docs.

- If you're updating to version `1.23` and use Amazon EBS volumes in your cluster, then you must install the Amazon EBS CSI driver in your cluster before updating your cluster to version `1.23` to avoid workload disruptions. For more information, see the section called "Kubernetes 1.23" and the section called "Amazon EBS".
- Kubernetes `1.24` and later use `containerd` as the default container runtime. If you're switching to the `containerd` runtime and already have Fluentd configured for Container Insights, then you must migrate Fluentd to Fluent Bit before updating your cluster. The Fluentd parsers are configured to only parse log messages in JSON format. Unlike `dockerd`, the `containerd` container runtime has log messages that aren't in JSON format. If you don't migrate to Fluent Bit, some of the configured Fluentd's parsers will generate a massive amount of errors inside the Fluentd container. For more information on migrating, see Set up Fluent Bit as a DaemonSet to send logs to CloudWatch Logs.
  - Because Amazon EKS runs a highly available control plane, you can update only one minor version at a time. For more information about this requirement, see Kubernetes Version and Version Skew Support Policy. Assume that your current cluster version is version `1.28` and you want to update it to version `1.30`. You must first update your version `1.28` cluster to version `1.29` and then update your version `1.29` cluster to version `1.30`.
- Review the version skew between the Kubernetes `kube-apiserver` and the `kubelet` on your nodes.
  - Starting from Kubernetes version `1.28`, `kubelet` may be up to three minor versions older than `kube-apiserver`. See Kubernetes upstream version skew policy.
  - If the `kubelet` on your managed and Fargate nodes is on Kubernetes version `1.25` or newer, you can update your cluster up to three versions ahead without updating the `kubelet` version. For example, if the `kubelet` is on version `1.25`, you can update your Amazon EKS cluster version from `1.25` to `1.26`, to `1.27`, and to `1.28` while the `kubelet` remains on version `1.25`.

  - If the kubelet on your managed and Fargate nodes is on Kubernetes version 1.24 or older, it may only be up to two minor versions older than the kube-apiserver. In other words, if the kubelet is version 1.24 or older, you can only update your cluster up to two versions ahead. For example, if the kubelet is on version 1.21, you can update your Amazon EKS cluster version from 1.21 to 1.22, and to 1.23, but you will not be able to update the cluster to 1.24 while the kubelet remains on 1.21.
- As a best practice before starting an update, make sure that the kubelet on your nodes is at the same Kubernetes version as your control plane.
- If your cluster is configured with a version of the Amazon VPC CNI plugin for Kubernetes that is earlier than 1.8.0, then we recommend that you update the plugin to the latest version before updating your cluster. To update the plugin, see the section called "Amazon VPC CNI".
- If you're updating your cluster to version 1.25 or later and have the AWS Load Balancer Controller deployed in your cluster, then update the controller to version 2.4.7 or later *before* updating your cluster version to 1.25. For more information, see the Kubernetes 1.25 release notes.

Step 3: Update cluster control plane

You can submit the request to upgrade your EKS control plane version using:

- eksctl
- the AWS console
- the AWS cli

Update cluster - eksctl

This procedure requires eksctl version 0.204.0 or later. You can check your version with the following command:

```
eksctl version
```

For instructions on how to install and update eksctl, see Installation in the eksctl documentation.

Update the Kubernetes version of your Amazon EKS control plane. Replace *my-cluster* with your cluster name. Replace *1.30* with the Amazon EKS supported version number that you want to update your cluster to. For a list of supported version numbers, see the section called "Kubernetes versions".

```
eksctl upgrade cluster --name my-cluster --version 1.30 --approve
```

The update takes several minutes to complete.

Continue to the section called "Step 4: Update cluster components"

Update cluster - AWS console

1. Open the Amazon EKS console.
2. Choose the name of the Amazon EKS 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**, enter the name of your cluster and choose **Confirm**.

   The update takes several minutes to complete.
5. Continue to the section called "Step 4: Update cluster components"

Update cluster - AWS CLI

1. Update your Amazon EKS cluster with the following AWS CLI command. Replace the *example values* with your own. Replace *1.30* with the Amazon EKS supported version number that you want to update your cluster to. For a list of supported version numbers, see the section called "Kubernetes versions".

   ```
   aws eks update-cluster-version --region region-code --name my-cluster --kubernetes-
   version 1.30
   ```

   An example output is as follows.

   ```
   {
       "update": {
           "id": "b5f0ba18-9a87-4450-b5a0-825e6e84496f",
           "status": "InProgress",
           "type": "VersionUpdate",
           "params": [
               {
   ```

```
                "type": "Version",
                "value": "1.30"
            },
            {
                "type": "PlatformVersion",
                "value": "eks.1"
            }
        ],
[...]
        "errors": []
    }
```

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

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

An example output is as follows.

```
{
    "update": {
        "id": "b5f0ba18-9a87-4450-b5a0-825e6e84496f",
        "status": "Successful",
        "type": "VersionUpdate",
        "params": [
            {
                "type": "Version",
                "value": "1.30"
            },
            {
                "type": "PlatformVersion",
                "value": "eks.1"
            }
        ],
[...]
        "errors": []
    }
```

3. Continue to the section called “Step 4: Update cluster components”

Step 4: Update cluster components

1. After your cluster update is complete, update your nodes to the same Kubernetes minor version as your updated cluster. For more information, see the section called "Update methods", the section called "Update", and the section called "Upgrade hybrid nodes". Any new Pods that are launched on Fargate have a `kubelet` version that matches your cluster version. Existing Fargate Pods aren't changed.
2. (Optional) If you deployed the Kubernetes Cluster Autoscaler to your cluster before updating the cluster, update the Cluster Autoscaler to the latest version that matches the Kubernetes major and minor version that you updated to.
   a. Open the Cluster Autoscaler releases page in a web browser and find the latest Cluster Autoscaler version that matches your cluster's Kubernetes major and minor version. For example, if your cluster's Kubernetes version is `1.30` find the latest Cluster Autoscaler release that begins with `1.30`. Record the semantic version number (`1.30.n`, for example) for that release to use in the next step.
   b. Set the Cluster Autoscaler image tag to the version that you recorded in the previous step with the following command. If necessary, replace *`1.30.n`* with your own value.

   ```
   kubectl -n kube-system set image deployment.apps/cluster-autoscaler cluster-
   autoscaler=registry.k8s.io/autoscaling/cluster-autoscaler:v1.30.n
   ```
3. (Clusters with GPU nodes only) If your cluster has node groups with GPU support (for example, `p3.2xlarge`), you must update the NVIDIA device plugin for KubernetesDaemonSet on your cluster. Replace *`vX.X.X`* with your desired NVIDIA/k8s-device-plugin version before running the following command.

   ```
   kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
   deployments/static/nvidia-device-plugin.yml
   ```
4. Update the Amazon VPC CNI plugin for Kubernetes, CoreDNS, and `kube-proxy` add-ons. We recommend updating the add-ons to the minimum versions listed in Service account tokens.
   - If you are using Amazon EKS add-ons, select **Clusters** in the Amazon EKS console, then select the name of the cluster that you updated in the left navigation pane. Notifications appear in the console. They inform you that a new version is available for each add-on that has an available update. To update an add-on, select the **Add-ons** tab. In one of the boxes for an add-on that has an update available, select **Update now**, select an available version, and then select **Update**.

- Alternately, you can use the AWS CLI or `eksctl` to update add-ons. For more information, see the section called "Update an add-on".

5. If necessary, update your version of `kubectl`. You must use a `kubectl` version that is within one minor version difference of your Amazon EKS cluster control plane.

Downgrade the Kubernetes version for an Amazon EKS cluster

You cannot downgrade the Kubernetes of an Amazon EKS cluster. Instead, create a new cluster on a previous Amazon EKS version and migrate the workloads.

Delete a cluster

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

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

Considerations

- If you have active services in your cluster that are associated with a load balancer, you must delete those services before deleting the cluster so that the load balancers are deleted properly. Otherwise, you can have orphaned resources in your VPC that prevent you from being able to delete the VPC.
- If you receive an error because the cluster creator has been removed, see this article to resolve.
- Amazon Managed Service for Prometheus resources are outside of the cluster lifecycle and need to be maintained independent of the cluster. When you delete your cluster, make sure to also delete any applicable scrapers to stop applicable costs. For more information, see Find and delete scrapers in the *Amazon Managed Service for Prometheus User Guide*.
- To remove a connected cluster, see the section called "Deregister a cluster"

Considerations for EKS Auto Mode

- Any EKS Auto Mode Nodes will be deleted, including the EC2 managed instances
- All load balancers will be deleted

For more information, see the section called "Disable EKS Auto Mode".

Delete cluster (eksctl)

This procedure requires `eksctl` version `0.204.0` or later. You can check your version with the following command:

```
eksctl version
```

For instructions on how to install or upgrade `eksctl`, see Installation in the `eksctl` documentation.

1. List all services running in your cluster.

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

   a. 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
      ```

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

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

   Output:

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

Delete cluster (AWS console)

1. List all services running in your cluster.

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

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

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

3. Delete all node groups and Fargate profiles.
   a. Open the Amazon EKS console.
   b. In the left navigation pane, choose Amazon EKS **Clusters**, and then in the tabbed list of clusters, choose the name of the cluster that you want to delete.
   c. Choose the **Compute** tab and choose a node group to delete. Choose **Delete**, enter the name of the node group, and then choose **Delete**. Delete all node groups in the cluster.

   > **Note**
   >
   > The node groups listed are managed node groups only.

   d. Choose a **Fargate Profile** to delete, select **Delete**, enter the name of the profile, and then choose **Delete**. Delete all Fargate profiles in the cluster.
4. Delete all self-managed node AWS CloudFormation stacks.
   a. Open the AWS CloudFormation console.
   b. Choose the node stack to delete, and then choose **Delete**.
   c. In the **Delete stack** confirmation dialog box, choose **Delete stack**. Delete all self-managed node stacks in the cluster.
5. Delete the cluster.
   a. Open the Amazon EKS console.
   b. choose the cluster to delete and choose **Delete**.
   c. On the delete cluster confirmation screen, choose **Delete**.
6. (Optional) Delete the VPC AWS CloudFormation stack.
   a. Open the AWS CloudFormation console.
   b. Select the VPC stack to delete, and then choose **Delete**.
   c. In the **Delete stack** confirmation dialog box, choose **Delete stack**.

Delete cluster (AWS CLI)

1. List all services running in your cluster.

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

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

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

3. Delete all node groups and Fargate profiles.

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

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

   > **Note**
   >
   > The node groups listed are managed node groups only.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Control network access to cluster API server endpoint

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

When you create a new cluster, Amazon EKS creates an endpoint for the managed Kubernetes API server that you use to communicate with your cluster (using Kubernetes management tools such as `kubectl`). By default, this API server endpoint is public to the internet, and access to the API server is secured using a combination of AWS Identity and Access Management (IAM) and native Kubernetes Role Based Access Control (RBAC). This endpoint is known as the *cluster public endpoint*. Also there is a *cluster private endpoint*. For more information about the cluster private endpoint, see the following section the section called "Cluster private endpoint".

IPv6 cluster endpoint format

EKS creates a unique dual-stack endpoint in the following format for new `IPv6` clusters that are made after October 2024. An *IPv6 cluster* is a cluster that you select `IPv6` in the IP family (`ipFamily`) setting of the cluster.

Example

AWS

EKS cluster public/private endpoint: `eks-cluster.`*`region`*`.api.aws`

AWS GovCloud (US)

EKS cluster public/private endpoint: `eks-cluster.`*`region`*`.api.aws`

Amazon Web Services in China

EKS cluster public/private endpoint: `eks-cluster.`*`region`*`.api.amazonwebservices.com.cn`

> **Note**
>
> The dual-stack cluster endpoint was introduced in October 2024. For more information about `IPv6` clusters, see the section called "IPv6". Clusters made before October 2024, use following endpoint format instead.

IPv4 cluster endpoint format

EKS creates a unique endpoint in the following format for each cluster that select `IPv4` in the IP family (ipFamily) setting of the cluster:

Example

AWS

EKS cluster public/private endpoint `eks-cluster.`*`region`*`.eks.amazonaws.com`

AWS GovCloud (US)

EKS cluster public/private endpoint `eks-cluster.`*`region`*`.eks.amazonaws.com`

Amazon Web Services in China

EKS cluster public/private endpoint `eks-cluster.*region*.api.amazonwebservices.com.cn`

> **Note**
>
> Before October 2024, `IPv6` clusters used this endpoint format also. For those clusters, both the public endpoint and the private endpoint have only `IPv4` addresses resolve from this endpoint.

Cluster private endpoint

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

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

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

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

Modifying cluster endpoint access

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

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

Endpoint public access	Endpoint private access	Behavior
		(such as node to control plane communication) use the private VPC endpoint. • Your cluster API server is accessible from the internet. You can, optionall y, limit the CIDR blocks that can access the public endpoint. • If you are using hybrid nodes with your Amazon EKS cluster, it is not recommended to have both Public and Private cluster endpoint access enabled. Because your hybrid nodes are running outside of your VPC, they will resolve the cluster endpoint to the public IP addresses. It is recommended to use either Public or Private cluster endpoint access for clusters with hybrid nodes.

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

You can modify your cluster API server endpoint access using the AWS Management Console or AWS CLI.

Configure endpoint access - AWS console

1. Open the Amazon EKS console.
2. Choose the name of the cluster to display your cluster information.
3. Choose the **Networking** tab and choose **Manage endpoint access**.
4. For **Private access**, choose whether to enable or disable private access for your cluster's Kubernetes API server endpoint. If you enable private access, Kubernetes API requests that originate from within your cluster's VPC use the private VPC endpoint. You must enable private access to disable public access.
5. For **Public access**, choose whether to enable or disable public access for your cluster's Kubernetes API server endpoint. If you disable public access, your cluster's Kubernetes API server can only receive requests from within the cluster VPC.
6. (Optional) If you've enabled **Public access**, you can specify which addresses from the internet can communicate to the public endpoint. Select **Advanced Settings**. Enter a CIDR block, such as *`203.0.113.5/32`*. The block cannot include reserved addresses. You can enter additional blocks by selecting **Add Source**. There is a maximum number of CIDR blocks that you can specify. For more information, see the section called "Service quotas". If you specify no blocks, then the public API server endpoint receives requests from all (`0.0.0.0/0`) IP addresses. If you restrict access to your public endpoint using CIDR blocks, it is recommended that you also enable private endpoint access so that nodes and Fargate Pods (if you use them) can communicate with the cluster. Without the private endpoint enabled, your public access endpoint CIDR sources must include the egress sources from your VPC. For example, if you have a node in a private subnet that communicates to the internet through a NAT Gateway, you will need to add the outbound IP address of the NAT gateway as part of an allowed CIDR block on your public endpoint.
7. Choose **Update** to finish.

Configure endpoint access - AWS CLI

Complete the following steps using the AWS CLI version `1.27.160` or later. You can check your current version with `aws --version`. To install or upgrade the AWS CLI, see Installing the AWS CLI.

1. Update your cluster API server endpoint access with the following AWS CLI command. Substitute your cluster name and desired endpoint access values. If you set `endpointPublicAccess=true`, then you can (optionally) enter single CIDR block, or a comma-separated list of CIDR blocks for `publicAccessCidrs`. The blocks cannot include reserved addresses. If you specify CIDR blocks, then the public API server endpoint will only receive requests from the listed blocks. There is a maximum number of CIDR blocks that you can specify. For more information, see the section called "Service quotas". If you restrict access to your public endpoint using CIDR blocks, it is recommended that you also enable private endpoint access so that nodes and Fargate Pods (if you use them) can communicate with the cluster. Without the private endpoint enabled, your public access endpoint CIDR sources must include the egress sources from your VPC. For example, if you have a node in a private subnet that communicates to the internet through a NAT Gateway, you will need to add the outbound IP address of the NAT gateway as part of an allowed CIDR block on your public endpoint. If you specify no CIDR blocks, then the public API server endpoint receives requests from all (0.0.0.0/0) IP addresses.

   > **Note**
   >
   > The following command enables private access and public access from a single IP address for the API server endpoint. Replace *`203.0.113.5/32`* with a single CIDR block, or a comma-separated list of CIDR blocks that you want to restrict network access to.

   ```
   aws eks update-cluster-config \
       --region region-code \
       --name my-cluster \
       --resources-vpc-config
    endpointPublicAccess=true,publicAccessCidrs="203.0.113.5/32",endpointPrivateAccess=true
   ```

   An example output is as follows.

   ```
   {
       "update": {
           "id": "e6f0905f-a5d4-4a2a-8c49-EXAMPLE00000",
           "status": "InProgress",
           "type": "EndpointAccessUpdate",
           "params": [
               {
   ```

```
                "type": "EndpointPublicAccess",
                "value": "true"
            },
            {
                "type": "EndpointPrivateAccess",
                "value": "true"
            },
            {
                "type": "publicAccessCidrs",
                "value": "[\203.0.113.5/32\"]"
            }
        ],
        "createdAt": 1576874258.137,
        "errors": []
    }
}
```

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

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

An example output is as follows.

```
{
    "update": {
        "id": "e6f0905f-a5d4-4a2a-8c49-EXAMPLE00000",
        "status": "Successful",
        "type": "EndpointAccessUpdate",
        "params": [
            {
                "type": "EndpointPublicAccess",
                "value": "true"
            },
            {
                "type": "EndpointPrivateAccess",
                "value": "true"
            },
```

```
            {
                "type": "publicAccessCidrs",
                "value": "[\203.0.113.5/32\"]"
            }
        ],
        "createdAt": 1576874258.137,
        "errors": []
    }
}
```

Accessing a private only API server

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

Connected network

Connect your network to the VPC with an AWS transit gateway or other connectivity option and then use a computer in the connected network. You must ensure that your Amazon EKS control plane security group contains rules to allow ingress traffic on port 443 from your connected network.

Amazon EC2 bastion host

You can launch an Amazon EC2 instance into a public subnet in your cluster's VPC and then log in via SSH into that instance to run `kubectl` commands. For more information, see Linux bastion hosts on AWS. You must ensure that your Amazon EKS control plane security group contains rules to allow ingress traffic on port 443 from your bastion host. For more information, see the section called "Security group requirements".

When you configure `kubectl` for your bastion host, be sure to use AWS credentials that are already mapped to your cluster's RBAC configuration, or add the IAM principal that your bastion will use to the RBAC configuration before you remove endpoint public access. For more information, see the section called "Kubernetes API access" and the section called "Unauthorized or access denied (`kubectl`)".

AWS Cloud9 IDE

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

When you configure `kubectl` for your AWS Cloud9 IDE, be sure to use AWS credentials that are already mapped to your cluster's RBAC configuration, or add the IAM principal that your IDE will use to the RBAC configuration before you remove endpoint public access. For more information, see the section called "Kubernetes API access" and the section called "Unauthorized or access denied (`kubectl`)".

Deploy Windows nodes on EKS clusters

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

- EKS Auto Mode does not support Windows nodes
- You can use host networking on Windows nodes using `HostProcess` Pods. For more information, see Create a Windows HostProcessPod in the Kubernetes documentation.
- Amazon EKS clusters must contain one or more Linux or Fargate nodes to run core system Pods that only run on Linux, such as CoreDNS.
- The `kubelet` and `kube-proxy` event logs are redirected to the `EKS Windows` Event Log and are set to a 200 MB limit.
- You can't use Assign security groups to individual pods with Pods running on Windows nodes.
- You can't use custom networking with Windows nodes.
- You can't use `IPv6` with Windows nodes.
- Windows nodes support one elastic network interface per node. By default, the number of Pods that you can run per Windows node is equal to the number of IP addresses available per elastic network interface for the node's instance type, minus one. For more information, see IP addresses per network interface per instance type in the *Amazon EC2 User Guide*.
- In an Amazon EKS cluster, a single service with a load balancer can support up to 1024 back-end Pods. Each Pod has its own unique IP address. The previous limit of 64 Pods is no longer the case, after a Windows Server update starting with OS Build 17763.2746.
- Windows containers aren't supported for Amazon EKS Pods on Fargate.
- You can't use Amazon EKS Hybrid Nodes with Windows as the operating system for the host.

- You can't retrieve logs from the `vpc-resource-controller` Pod. You previously could when you deployed the controller to the data plane.
- There is a cool down period before an `IPv4` address is assigned to a new Pod. This prevents traffic from flowing to an older Pod with the same `IPv4` address due to stale `kube-proxy` rules.
- The source for the controller is managed on GitHub. To contribute to, or file issues against the controller, visit the project on GitHub.
- When specifying a custom AMI ID for Windows managed node groups, add `eks:kube-proxy-windows` to your AWS IAM Authenticator configuration map. For more information, see the section called "Limits and conditions when specifying an AMI ID".
- If preserving your available IPv4 addresses is crucial for your subnet, refer to EKS Best Practices Guide - Windows Networking IP Address Management for guidance.
- An existing cluster.
- Your cluster must have at least one (we recommend at least two) Linux node or Fargate Pod to run CoreDNS. If you enable legacy Windows support, you must use a Linux node (you can't use a Fargate Pod) to run CoreDNS.
- An existing Amazon EKS cluster IAM role.

Enable Windows support

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

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

   An example output is as follows.

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

```
        }
    ]
}
```

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

2. Attach the **AmazonEKSVPCResourceController** managed policy to your Amazon EKS cluster IAM role. Replace *eksClusterRole* with your cluster role name.

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

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

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

4. Apply the `ConfigMap` to your cluster.

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

5. Verify that your `aws-auth` `ConfigMap` contains a mapping for the instance role of the Windows node to include the `eks:kube-proxy-windows` RBAC permission group. You can verify by running the following command.

```
kubectl get configmap aws-auth -n kube-system -o yaml
```

An example output is as follows.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
```

```
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      - eks:kube-proxy-windows # This group is required for Windows DNS resolution to
 work
      rolearn: arn:aws:iam::111122223333:role/eksNodeRole
      username: system:node:{{EC2PrivateDNSName}}
[...]
```

You should see `eks:kube-proxy-windows` listed under groups. If the group isn't specified, you need to update your `ConfigMap` or create it to include the required group. For more information about the `aws-auth` `ConfigMap`, see the section called "Apply the `aws-auth` `ConfigMap` to your cluster".

Deploy Windows Pods

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

For Linux Pods, use the following node selector text in your manifests.

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

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

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

You can deploy a sample application to see the node selectors in use.

Support higher Pod density on Windows nodes

In Amazon EKS, each Pod is allocated an `IPv4` address from your VPC. Due to this, the number of Pods that you can deploy to a node is constrained by the available IP addresses, even if there are sufficient resources to run more Pods on the node. Since only one elastic network interface is supported by a Windows node, by default, the maximum number of available IP addresses on a Windows node is equal to:

```
Number of private IPv4 addresses for each interface on the node - 1
```

One IP address is used as the primary IP address of the network interface, so it can't be allocated to Pods.

You can enable higher Pod density on Windows nodes by enabling IP prefix delegation. This feature enables you to assign a `/28` `IPv4` prefix to the primary network interface, instead of assigning secondary `IPv4` addresses. Assigning an IP prefix increases the maximum available `IPv4` addresses on the node to:

```
(Number of private IPv4 addresses assigned to the interface attached to the node - 1) *
 16
```

With this significantly larger number of available IP addresses, available IP addresses shouldn't limit your ability to scale the number of Pods on your nodes. For more information, see the section called "Increase IP addresses".

Disable Windows support

1. If your cluster contains Amazon Linux nodes and you use security groups for Pods with them, then skip this step.

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

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

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

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

Deploy private clusters with limited internet access

This topic describes how to deploy an Amazon EKS cluster that is deployed on the AWS Cloud, but doesn't have outbound internet access. If you have a local cluster on AWS Outposts, see the section called "Nodes", instead of this topic.

If you're not familiar with Amazon EKS networking, see De-mystifying cluster networking for Amazon EKS worker nodes. If your cluster doesn't have outbound internet access, then it must meet the following requirements:

- Your cluster must pull images from a container registry that's in your VPC. You can create an Amazon Elastic Container Registry in your VPC and copy container images to it for your nodes to pull from. For more information, see the section called "Copy an image to a repository".
- Your cluster must have endpoint private access enabled. This is required for nodes to register with the cluster endpoint. Endpoint public access is optional. For more information, see the section called "Configure endpoint access".
- Self-managed Linux and Windows nodes must include the following bootstrap arguments before they're launched. These arguments bypass Amazon EKS introspection and don't require access to the Amazon EKS API from within the VPC.

  a. Determine the value of your cluster's endpoint with the following command. Replace *my-cluster* with the name of your cluster.

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

  An example output is as follows.

  ```
  https://EXAMPLE108C897D9B2F1B21D5EXAMPLE.sk1.region-code.eks.amazonaws.com
  ```

  b. Determine the value of your cluster's certificate authority with the following command. Replace *my-cluster* with the name of your cluster.

  ```
  aws eks describe-cluster --name my-cluster --query cluster.certificateAuthority --
  output text
  ```

  The returned output is a long string.

  c. Replace *cluster-endpoint* and *certificate-authority* in the following commands with the values returned in the output from the previous commands. For more information about specifying bootstrap arguments when launching self-managed nodes, see the section called “Amazon Linux” and the section called “Windows”.

- For Linux nodes:

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

  For additional arguments, see the bootstrap script on GitHub.

- For Windows nodes:

> **Note**
>
> If you’re using custom service CIDR, then you need to specify it using the `-ServiceCIDR` parameter. Otherwise, the DNS resolution for Pods in the cluster will fail.

```
-APIServerEndpoint cluster-endpoint -Base64ClusterCA certificate-authority
```

  For additional arguments, see the section called “Bootstrap script configuration parameters”.

- Your cluster’s `aws-auth` `ConfigMap` must be created from within your VPC. For more information about creating and adding entries to the `aws-auth` `ConfigMap`, enter `eksctl create iamidentitymapping --help` in your terminal. If the `ConfigMap` doesn’t exist on your server, `eksctl` will create it when you use the command to add an identity mapping.
- Pods configured with IAM roles for service accounts acquire credentials from an AWS Security Token Service (AWS STS) API call. If there is no outbound internet access, you must create and use an AWS STS VPC endpoint in your VPC. Most AWS `v1` SDKs use the global AWS STS endpoint by default (`sts.amazonaws.com`), which doesn’t use the AWS STS VPC endpoint. To use the AWS STS VPC endpoint, you might need to configure your SDK to use the regional AWS STS endpoint (`sts.`*`region-code`*`.amazonaws.com`). For more information, see the section called “STS endpoints”.
- Your cluster’s VPC subnets must have a VPC interface endpoint for any AWS services that your Pods need access to. For more information, see Access an AWS service using an interface VPC endpoint. Some commonly-used services and endpoints are listed in the following table. For a

complete list of endpoints, see AWS services that integrate with AWS PrivateLink in the AWS PrivateLink Guide.

We recommend that you enable private DNS names for your VPC endpoints, that way workloads can continue using public AWS service endpoints without issues.

Service	Endpoint
Amazon EC2	com.amazonaws.*region-code* .ec2
Amazon Elastic Container Registry (for pulling container images)	com.amazonaws.*region-code* .ecr.api, com.amazonaws.*region-code* .ecr.dkr, and com.amazonaws.*region-code* .s3
Application Load Balancers and Network Load Balancers	com.amazonaws.*region-code* .elasticloadbalancing
AWS X-Ray	com.amazonaws.*region-code* .xray
Amazon CloudWatch Logs	com.amazonaws.*region-code* .logs
AWS Security Token Service (required when using IAM roles for service accounts)	com.amazonaws.*region-code* .sts
Amazon EKS Auth (required when using Pod Identity associations)	com.amazonaws.*region-code* .eks-auth
Amazon EKS	com.amazonaws.*region-code* .eks

- Any self-managed nodes must be deployed to subnets that have the VPC interface endpoints that you require. If you create a managed node group, the VPC interface endpoint security group must allow the CIDR for the subnets, or you must add the created node security group to the VPC interface endpoint security group.
- If your Pods use Amazon EFS volumes, then before deploying the Store an elastic file system with Amazon EFS, the driver's kustomization.yaml file must be changed to set the container images to use the same AWS Region as the Amazon EKS cluster.
- Route53 does not support AWS PrivateLink. You cannot manage Route53 DNS records from a private Amazon EKS cluster. This impacts Kubernetes external-dns.

- If you use the EKS Optimized AMI, you should enable the `ec2` endpoint in the table above. Alternatively, you can manually set the Node DNS name. The optimized AMI uses EC2 APIs to set the node DNS name automatically.
- You can use the AWS Load Balancer Controller to deploy AWS Application Load Balancers (ALB) and Network Load Balancers to your private cluster. When deploying it, you should use command line flags to set `enable-shield`, `enable-waf`, and `enable-wafv2` to false. Certificate discovery with hostnames from Ingress objects isn't supported. This is because the controller needs to reach AWS Certificate Manager, which doesn't have a VPC interface endpoint.

  The controller supports network load balancers with IP targets, which are required for use with Fargate. For more information, see the section called "Application load balancing" and the section called "Create a network load balancer".
- Cluster Autoscaler is supported. When deploying Cluster Autoscaler Pods, make sure that the command line includes `--aws-use-static-instance-list=true`. For more information, see Use Static Instance List on GitHub. The worker node VPC must also include the AWS STS VPC endpoint and autoscaling VPC endpoint.
- Some container software products use API calls that access the AWS Marketplace Metering Service to monitor usage. Private clusters do not allow these calls, so you can't use these container types in private clusters.

Understand the Kubernetes version lifecycle on EKS

Kubernetes rapidly evolves with new features, design updates, and bug fixes. The community releases new Kubernetes minor versions (such as `1.32`) on average once every four months. Amazon EKS follows the upstream release and deprecation cycle for minor versions. As new Kubernetes versions become available in Amazon EKS, we recommend that you proactively update your clusters to use the latest available version.

A minor version is under standard support in Amazon EKS for the first 14 months after it's released. Once a version is past the end of standard support date, it enters extended support for the next 12 months. Extended support allows you to stay at a specific Kubernetes version for longer at an additional cost per cluster hour. If you haven't updated your cluster before the extended support period ends, your cluster is auto-upgraded to the oldest currently supported extended version.

Extended support is enabled by default. the section called "Disable extended support"

We recommend that you create your cluster with the latest available Kubernetes version supported by Amazon EKS. If your application requires a specific version of Kubernetes, you can select older versions. You can create new Amazon EKS clusters on any version offered in standard or extended support.

Available versions on standard support

The following Kubernetes versions are currently available in Amazon EKS standard support:

- `1.32`
- `1.31`
- `1.30`
- `1.29`

For important changes to be aware of for each version in standard support, see the section called "Standard support versions".

Available versions on extended support

The following Kubernetes versions are currently available in Amazon EKS extended support:

- `1.28`
- `1.27`
- `1.26`
- `1.25`
- `1.24`

For important changes to be aware of for each version in extended support, see the section called "Extended support versions".

Amazon EKS Kubernetes release calendar

The following table shows important release and support dates to consider for each Kubernetes version. Billing for extended support starts at the beginning of the day that the version reaches end of standard support.

Note

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

Kubernetes version	Upstream release	Amazon EKS release	End of standard support	End of extended support
`1.32`	December 11, 2024	January 23, 2025	March 23, 2026	March 23, 2027
`1.31`	August 13, 2024	September 26, 2024	November 26, 2025	November 26, 2026
`1.30`	April 17, 2024	May 23, 2024	July 23, 2025	July 23, 2026
`1.29`	December 13, 2023	January 23, 2024	March 23, 2025	March 23, 2026
`1.28`	August 15, 2023	September 26, 2023	November 26, 2024	November 26, 2025
`1.27`	April 11, 2023	May 24, 2023	July 24, 2024	July 24, 2025
`1.26`	December 9, 2022	April 11, 2023	June 11, 2024	June 11, 2025
`1.25`	August 23, 2022	February 22, 2023	May 1, 2024	May 1, 2025
`1.24`	May 3, 2022	November 15, 2022	January 31, 2024	January 31, 2025

Get version information with AWS CLI

You can use the AWS CLI to get information about Kubernetes versions available on EKS, such as the end date of Standard Support.

To retrieve information about available Kubernetes versions on EKS using the AWS CLI

1. Open your terminal.
2. Ensure you have the AWS CLI installed and configured. For more information, see Installing or updating to the latest version of the CLI.
3. Run the following command:

```
aws eks describe-cluster-versions
```

4. The command will return a JSON output with details about the available cluster versions. Here's an example of the output:

```
{
    "clusterVersions": [
        {
            "clusterVersion": "1.31",
            "clusterType": "eks",
            "defaultPlatformVersion": "eks.21",
            "defaultVersion": true,
            "releaseDate": "2024-09-25T17:00:00-07:00",
            "endOfStandardSupportDate": "2025-11-25T16:00:00-08:00",
            "endOfExtendedSupportDate": "2026-11-25T16:00:00-08:00",
            "status": "STANDARD_SUPPORT",
            "kubernetesPatchVersion": "1.31.3"
        }
    ]
}
```

The output provides the following information for each cluster version:

- `clusterVersion`: The Kubernetes version of the EKS cluster
- `clusterType`: The type of cluster (e.g., "eks")
- `defaultPlatformVersion`: The default EKS platform version
- `defaultVersion`: Whether this is the default version
- `releaseDate`: The date when this version was released
- `endOfStandardSupportDate`: The date when standard support ends

- `endOfExtendedSupportDate`: The date when extended support ends
- `status`: The current support status of the version, such as `STANDARD_SUPPORT` or `EXTENDED_SUPPORT`
- `kubernetesPatchVersion`: The specific Kubernetes patch version

Amazon EKS version FAQs

How many Kubernetes versions are available in standard support?

In line with the Kubernetes community support for Kubernetes versions, Amazon EKS is committed to offering support for three Kubernetes versions at any given time. We will announce the end of standard support date of a given Kubernetes minor version at least 60 days in advance. Because of the Amazon EKS qualification and release process for new Kubernetes versions, the end of standard support date of a Kubernetes version on Amazon EKS will be after the date that the Kubernetes project stops supporting the version upstream.

How long does a Kubernetes receive standard support by Amazon EKS?

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

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

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

Which Kubernetes features are supported by Amazon EKS?

Amazon EKS supports all generally available (GA) features of the Kubernetes API. Starting with Kubernetes version `1.24`, new beta APIs aren’t enabled in clusters by default. However, previously existing beta APIs and new versions of existing beta APIs continue to be enabled by default. Alpha features aren’t supported.

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

No. A managed node group creates Amazon EC2 instances in your account. These instances aren't automatically upgraded when you or Amazon EKS update your control plane. For more information, see the section called "Update". We recommend maintaining the same Kubernetes version on your control plane and nodes.

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

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

The Kubernetes project tests compatibility between the control plane and nodes for up to three minor versions. For example, `1.29` nodes continue to operate when orchestrated by a `1.32` control plane. However, running a cluster with nodes that are persistently three minor versions behind the control plane isn't recommended. For more information, see Kubernetes version and version skew support policy in the Kubernetes documentation. We recommend maintaining the same Kubernetes version on your control plane and nodes.

Are Pods running on Fargate automatically upgraded with an automatic cluster control plane version upgrade?

No. We strongly recommend running Fargate Pods as part of a replication controller, such as a Kubernetes deployment. Then do a rolling restart of all Fargate Pods. The new version of the Fargate Pod is deployed with a `kubelet` version that's the same version as your updated cluster control plane version. For more information, see Deployments in the Kubernetes documentation.

> **Important**
>
> If you update the control plane, you must still update the Fargate nodes yourself. To update Fargate nodes, delete the Fargate Pod represented by the node and redeploy the

Pod. The new Pod is deployed with a `kubelet` version that's the same version as your cluster.

What Kubernetes versions are supported for hybrid nodes?

Amazon EKS Hybrid Nodes supports the same Kubernetes versions as Amazon EKS clusters with other node compute types, including standard and extended Kubernetes version support. Hybrid nodes are not automatically upgraded when you upgrade your control plane version and you are responsible for upgrading your hybrid nodes. For more information, see the section called "Upgrade hybrid nodes".

Amazon EKS extended support FAQs

The standard support and extended support terminology is new to me. What do those terms mean?

Standard support for a Kubernetes version in Amazon EKS begins when a Kubernetes version is released on Amazon EKS, and will end 14 months after the release date. Extended support for a Kubernetes version will begin immediately after the end of standard support, and will end after the next 12 months. For example, standard support for version `1.23` in Amazon EKS ended on October 11, 2023. Extended support for version `1.23` began on October 12, 2023 and ended on October 11, 2024.

What do I need to do to get extended support for Amazon EKS clusters?

You will need to enable extended support (see the section called "Enable extended support") for your cluster by changing the cluster upgrade policy to EXTENDED. By default, for all new and existing clusters, the upgrade policy is set to EXTENDED, unless specified otherwise. See the section called "View upgrade policy" to view the upgrade policy for your cluster. Standard support will begin when a Kubernetes version is released on Amazon EKS, and will end 14 months after the release date. Extended support for a Kubernetes version will begin immediately after the end of standard support, and will end after the next 12 months.

For which Kubernetes versions can I get extended support?

Extended support is available for Kubernetes versions `1.23` and higher. You can run clusters on any version for up to 12 months after the end of standard support for that version. This means that each version will be supported for 26 months in Amazon EKS (14 months of standard support plus 12 months of extended support).

What if I don't want to use extended support?

If you don't want to be automatically enrolled in extended support, you can upgrade your cluster to a Kubernetes version that's in standard Amazon EKS support. See the section called "Disable extended support" to learn how to disable extended support. Note: If you disable extended support, your cluster will be auto upgraded at the end of standard support.

What will happen at the end of 12 months of extended support?

Clusters running on a Kubernetes version that has completed its 26-month lifecycle (14 months of standard support plus 12 months of extended support) will be auto-upgraded to the next version. The auto-upgrade includes only the Kubernetes control plane. If you have EKS Auto Mode nodes, they may automatically update. Self managed nodes and EKS Managed Node Groups will remain on the previous version.

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

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

Amazon EKS can't provide specific time frames. Automatic updates can happen at any time after the end of extended support date. You won't receive any notification before the update. We recommend that you proactively update your control plane without relying on the Amazon EKS automatic update process. For more information, see the section called "Update Kubernetes version".

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

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

Is there additional cost to get extended support?

Yes, there is additional cost for Amazon EKS clusters running in extended support. For pricing details, see Amazon EKS extended support for Kubernetes version pricing on the AWS blog or our pricing page.

What is included in extended support?

Amazon EKS clusters in Extended Support receive ongoing security patches for the Kubernetes control plane. Additionally, Amazon EKS will release patches for the Amazon VPC CNI, `kube-proxy`, and CoreDNS add-ons for Extended Support versions. Amazon EKS will also release patches for AWS-published Amazon EKS optimized AMIs for Amazon Linux, Bottlerocket, and Windows, as well as Amazon EKS Fargate nodes for those versions. All clusters in Extended Support will continue to get access to technical support from AWS.

> **Note**
>
> Extended Support for Amazon EKS optimized Windows AMIs that are published by AWS isn't available for Kubernetes version `1.23` but is available for Kubernetes version `1.24` and higher.

Are there any limitations to patches for non-Kubernetes components in extended support?

While Extended Support covers all of the Kubernetes specific components from AWS, it will only provide support for AWS-published Amazon EKS optimized AMIs for Amazon Linux, Bottlerocket, and Windows at all times. This means, you will potentially have newer components (such as OS or kernel) on your Amazon EKS optimized AMI while using Extended Support. For example, once Amazon Linux 2 reaches the end of its lifecycle in 2025, the Amazon EKS optimized Amazon Linux AMIs will be built using a newer Amazon Linux OS. Amazon EKS will announce and document important support lifecycle discrepancies such as this for each Kubernetes version.

Can I create new clusters using a version on extended support?

Yes.

Review release notes for Kubernetes versions on standard support

This topic gives important changes to be aware of for each Kubernetes version in standard support. When upgrading, carefully review the changes that have occurred between the old and new versions for your cluster.

> **Note**
>
> For `1.24` and later clusters, officially published Amazon EKS AMIs include `containerd` as the only runtime. Kubernetes versions earlier than `1.24` use Docker as the default runtime. These versions have a bootstrap flag option that you can use to test out your workloads on any supported cluster with `containerd`. For more information, see the section called "Dockershim deprecation".

Kubernetes 1.32

Kubernetes `1.32` is now available in Amazon EKS. For more information about Kubernetes `1.32`, see the official release announcement.

> **Important**
>
> - The `flowcontrol.apiserver.k8s.io/v1beta3` API version of FlowSchema and PriorityLevelConfiguration has been removed in `v1.32`. If you are using these APIs, you must update your configurations to use the latest supported version before upgrading.
> - ServiceAccount `metadata.annotations[kubernetes.io/enforce-mountable-secrets]` has been deprecated in `v1.32` and will be removed in a future Kubernetes minor version release. It is recommended to use separate namespaces to isolate access to mounted secrets.
> - Kubernetes version 1.32 is the last version for which Amazon EKS will release Amazon Linux 2 (AL2) AMIs. From v1.33 onwards, Amazon EKS will continue to release Amazon Linux 2023 (AL2023) and Bottlerocket based AMIs.

- The Memory Manager feature has graduated to Generally Available (GA) status in Kubernetes `v1.32`. This enhancement provides more efficient and predictable memory allocation for containerized applications, particularly beneficial for workloads with specific memory requirements.

- PersistentVolumeClaims (PVCs) created by StatefulSets now include automatic cleanup functionality. When PVCs are no longer needed, they will be automatically deleted while maintaining data persistence during StatefulSet updates and node maintenance operations. This feature simplifies storage management and helps prevent orphaned PVCs in your cluster.
- Custom Resource Field Selector functionality has been introduced, allowing developers to add field selectors to custom resources. This feature provides the same filtering capabilities available for built-in Kubernetes objects to custom resources, enabling more precise and efficient resource filtering and promoting better API design practices.

For the complete Kubernetes `1.32` changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.32.md

Anonymous authentication changes

Starting with Amazon EKS `1.32`, anonymous authentication is restricted to the following API server health check endpoints:

- `/healthz`
- `/livez`
- `/readyz`

Requests to any other endpoint using the `system:unauthenticated` user will receive a `401 Unauthorized` HTTP response. This security enhancement helps prevent unintended cluster access that could occur due to misconfigured RBAC policies.

> **Note**
>
> The `public-info-viewer` RBAC role continues to apply for the health check endpoints listed above.

Amazon Linux 2 AMI deprecation

For Kubernetes versions 1.33 and later, EKS will not provide pre-built optimized Amazon Linux 2 (AL2) Amazon Machine Images (AMIs).

AWS suggests adopting EKS Auto Mode, or migrating to a more recent operating system, such as Amazon Linux 2023 (AL2023) or Bottlerocket.

- the section called “Migrate from MNGs”
- the section called “Upgrade to AL2023”
- the section called “Bottlerocket”

> **Note**
>
> This update applies to EKS-optimized AL2 AMIs. For more information about the operating system itself, see Amazon Linux 2 FAQs.

Kubernetes 1.31

Kubernetes `1.31` is now available in Amazon EKS. For more information about Kubernetes `1.31`, see the official release announcement.

> **Important**
>
> - The kubelet flag `--keep-terminated-pod-volumes` deprecated since 2017 has been removed as part of the `v1.31` release. This change impacts how terminated pod volumes are handled by the kubelet. If you are using this flag in your node configurations, you must update your bootstrap scripts and launch templates to remove it before upgrading.

- The beta `VolumeAttributesClass` feature gate and API resource is enabled in Amazon EKS `v1.31`. This feature allows cluster operators to modify mutable properties of Persistent Volumes (PVs) managed by compatible CSI Drivers, including the Amazon EBS CSI Driver. To leverage this feature, ensure that your CSI Driver supports the `VolumeAttributesClass` feature (for the Amazon EBS CSI Driver, upgrade to version `v1.35.0` or later to automatically enable the feature). You will be able to create `VolumeAttributesClass` objects to define the desired volume attributes, such as volume type and throughput, and associate them with your Persistent Volume Claims (PVCs). See the official Kubernetes documentation as well as the documentation of your CSI driver for more information.
  - For more information about the Amazon EBS CSI Driver, see the section called “Amazon EBS”.

- Kubernetes support for AppArmor has graduated to stable and is now generally available for public use. This feature allows you to protect your containers with AppArmor by setting the `appArmorProfile.type` field in the container's `securityContext`. Prior to Kubernetes `v1.30`, AppArmor was controlled by annotations. Starting with `v1.30`, it is controlled using fields. To leverage this feature, we recommend migrating away from annotations and using the `appArmorProfile.type` field to ensure that your workloads are compatible.
- The PersistentVolume last phase transition time feature has graduated to stable and is now generally available for public use in Kubernetes `v1.31`. This feature introduces a new field, `.status.lastTransitionTime`, in the PersistentVolumeStatus, which provides a timestamp of when a PersistentVolume last transitioned to a different phase. This enhancement allows for better tracking and management of PersistentVolumes, particularly in scenarios where understanding the lifecycle of volumes is important.

For the complete Kubernetes `1.31` changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.31.md

Kubernetes 1.30

Kubernetes `1.30` is now available in Amazon EKS. For more information about Kubernetes `1.30`, see the official release announcement.

> **Important**
>
> - Starting with Amazon EKS version `1.30` or newer, any newly created managed node groups will automatically default to using Amazon Linux 2023 (AL2023) as the node operating system. Previously, new node groups would default to Amazon Linux 2 (AL2). You can continue to use AL2 by choosing it as the AMI type when creating a new node group.
>   - For information about migrating from AL2 to AL2023, see the section called "Upgrade to AL2023".
>   - For more information about Amazon Linux, see Comparing AL2 and AL2023 in the Amazon Linux User Guide.
>   - For more information about specifiying the operating system for a managed node group, see the section called "Create".

- With Amazon EKS `1.30`, the `topology.k8s.aws/zone-id` label is added to worker nodes. You can use Availability Zone IDs (AZ IDs) to determine the location of resources in one account relative to the resources in another account. For more information, see Availability Zone IDs for your AWS resources in the *AWS RAM User Guide*.
- Starting with `1.30`, Amazon EKS no longer includes the `default` annotation on the `gp2 StorageClass` resource applied to newly created clusters. This has no impact if you are referencing this storage class by name. You must take action if you were relying on having a default `StorageClass` in the cluster. You should reference the `StorageClass` by the name `gp2`. Alternatively, you can deploy the Amazon EBS recommended default storage class by setting the `defaultStorageClass.enabled` parameter to true when installing `v1.31.0` or later of the `aws-ebs-csi-driver add-on`.
- The minimum required IAM policy for the Amazon EKS cluster IAM role has changed. The action `ec2:DescribeAvailabilityZones` is required. For more information, see the section called "Cluster IAM role".

For the complete Kubernetes `1.30` changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.30.md.

Kubernetes 1.29

Kubernetes `1.29` is now available in Amazon EKS. For more information about Kubernetes `1.29`, see the official release announcement.

> **Important**
>
> - The deprecated `flowcontrol.apiserver.k8s.io/v1beta2` API version of `FlowSchema` and `PriorityLevelConfiguration` are no longer served in Kubernetes `v1.29`. If you have manifests or client software that uses the deprecated beta API group, you should change these before you upgrade to `v1.29`.

- The `.status.kubeProxyVersion` field for node objects is now deprecated, and the Kubernetes project is proposing to remove that field in a future release. The deprecated field is not accurate and has historically been managed by `kubelet` - which does not actually know the `kube-proxy` version, or even whether `kube-proxy` is running. If you've been using this field in client software, stop - the information isn't reliable and the field is now deprecated.

- In Kubernetes 1.29 to reduce potential attack surface, the LegacyServiceAccountTokenCleanUp feature labels legacy auto-generated secret-based tokens as invalid if they have not been used for a long time (1 year by default), and automatically removes them if use is not attempted for a long time after being marked as invalid (1 additional year by default). To identify such tokens, a you can run:

```
kubectl get cm kube-apiserver-legacy-service-account-token-tracking -n kube-system
```

For the complete Kubernetes 1.29 changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.29.md#changelog-since-v1280.

Review release notes for Kubernetes versions on extended support

This topic gives important changes to be aware of for each Kubernetes version in extended support. When upgrading, carefully review the changes that have occurred between the old and new versions for your cluster.

Kubernetes 1.28

Kubernetes 1.28 is now available in Amazon EKS. For more information about Kubernetes 1.28, see the official release announcement.

- Kubernetes v1.28 expanded the supported skew between core node and control plane components by one minor version, from n-2 to n-3, so that node components (kubelet and kube-proxy) for the oldest supported minor version can work with control plane components (kube-apiserver, kube-scheduler, kube-controller-manager, cloud-controller-manager) for the newest supported minor version.
- Metrics force_delete_pods_total and force_delete_pod_errors_total in the Pod GC Controller are enhanced to account for all forceful pods deletion. A reason is added to the metric to indicate whether the pod is forcefully deleted because it's terminated, orphaned, terminating with the out-of-service taint, or terminating and unscheduled.
- The PersistentVolume (PV) controller has been modified to automatically assign a default StorageClass to any unbound PersistentVolumeClaim with the storageClassName not set. Additionally, the PersistentVolumeClaim admission validation mechanism within the API server has been adjusted to allow changing values from an unset state to an actual StorageClass name.

For the complete Kubernetes 1.28 changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.28.md#changelog-since-v1270.

Kubernetes 1.27

Kubernetes 1.27 is now available in Amazon EKS. For more information about Kubernetes 1.27, see the official release announcement.

> **Important**
>
> - The support for the alpha seccomp annotations seccomp.security.alpha.kubernetes.io/pod and container.seccomp.security.alpha.kubernetes.io annotations was removed. The alpha seccomp annotations was deprecated in 1.19, and with their removal in 1.27, seccomp fields will no longer auto-populate for Pods with seccomp annotations. Instead, use the securityContext.seccompProfile field for Pods or containers to configure seccomp profiles. To check whether you are using the deprecated alpha seccomp annotations in your cluster, run the following command:
>
> ```
> kubectl get pods --all-namespaces -o json | grep
>  -E 'seccomp.security.alpha.kubernetes.io/pod|
> container.seccomp.security.alpha.kubernetes.io'
> ```
>
> - The --container-runtime command line argument for the kubelet was removed. The default container runtime for Amazon EKS has been containerd since 1.24, which eliminates the need to specify the container runtime. From 1.27 onwards, Amazon EKS will ignore the --container-runtime argument passed to any bootstrap scripts. It is important that you don't pass this argument to --kubelet-extra-args in order to prevent errors during the node bootstrap process. You must remove the --container-runtime argument from all of your node creation workflows and build scripts.

- The kubelet in Kubernetes 1.27 increased the default kubeAPIQPS to 50 and kubeAPIBurst to 100. These enhancements allow the kubelet to handle a higher volume of API queries, improving response times and performance. When the demands for Pods increase, due to scaling requirements, the revised defaults ensure that the kubelet can efficiently manage the increased workload. As a result, Pod launches are quicker and cluster operations are more effective.

- You can use more fine grained Pod topology to spread policies such as minDomain. This parameter gives you the ability to specify the minimum number of domains your Pods should be spread across. nodeAffinityPolicy and nodeTaintPolicy allow for an extra level of granularity in governing Pod distribution. This is in accordance to node affinities, taints, and the matchLabelKeys field in the topologySpreadConstraints of your Pod's specification. This permits the selection of Pods for spreading calculations following a rolling upgrade.
- Kubernetes 1.27 promoted to beta a new policy mechanism for StatefulSets that controls the lifetime of their PersistentVolumeClaims(PVCs). The new PVC retention policy lets you specify if the PVCs generated from the StatefulSet spec template will be automatically deleted or retained when the StatefulSet is deleted or replicas in the StatefulSet are scaled down.
- The goaway-chance option in the Kubernetes API server helps prevent HTTP/2 client connections from being stuck on a single API server instance, by randomly closing a connection. When the connection is closed, the client will try to reconnect, and will likely land on a different API server as a result of load balancing. Amazon EKS version 1.27 has enabled goaway-chance flag. If your workload running on Amazon EKS cluster uses a client that is not compatible with HTTP GOAWAY, we recommend that you update your client to handle GOAWAY by reconnecting on connection termination.

For the complete Kubernetes 1.27 changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.27.md#changelog-since-v1260.

Kubernetes 1.26

Kubernetes 1.26 is now available in Amazon EKS. For more information about Kubernetes 1.26, see the official release announcement.

> **Important**
>
> Kubernetes 1.26 no longer supports CRI v1alpha2. This results in the kubelet no longer registering the node if the container runtime doesn't support CRI v1. This also means that Kubernetes 1.26 doesn't support containerd minor version 1.5 and earlier. If you're using containerd, you need to upgrade to containerd version 1.6.0 or later before you upgrade any nodes to Kubernetes 1.26. You also need to upgrade any other container runtimes that only support the v1alpha2. For more information, defer to the container runtime vendor. By default, Amazon Linux and Bottlerocket AMIs include containerd version 1.6.6.

- Before you upgrade to Kubernetes 1.26, upgrade your Amazon VPC CNI plugin for Kubernetes to version 1.12 or later. If you don't upgrade to Amazon VPC CNI plugin for Kubernetes version 1.12 or later, the Amazon VPC CNI plugin for Kubernetes will crash. For more information, see the section called "Amazon VPC CNI".
- The goaway-chance option in the Kubernetes API server helps prevent HTTP/2 client connections from being stuck on a single API server instance, by randomly closing a connection. When the connection is closed, the client will try to reconnect, and will likely land on a different API server as a result of load balancing. Amazon EKS version 1.26 has enabled goaway-chance flag. If your workload running on Amazon EKS cluster uses a client that is not compatible with HTTP GOAWAY, we recommend that you update your client to handle GOAWAY by reconnecting on connection termination.

For the complete Kubernetes 1.26 changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.26.md#changelog-since-v1250.

Kubernetes 1.25

Kubernetes 1.25 is now available in Amazon EKS. For more information about Kubernetes 1.25, see the official release announcement.

Important

- Amazon EC2 P2 instances aren't supported on Amazon EKS because they require NVIDIA driver version 470 or earlier.
- PodSecurityPolicy (PSP) is removed in Kubernetes 1.25. PSPs are replaced with Pod Security Admission (PSA) and Pod Security Standards (PSS). PSA is a built-in admission controller that implements the security controls outlined in the PSS. PSA and PSS are graduated to stable in Kubernetes 1.25 and are enabled in Amazon EKS by default. If you have PSPs in your cluster, make sure to migrate from PSP to the built-in Kubernetes PSS or to a policy-as-code solution before upgrading your cluster to version 1.25. If you don't migrate from PSP, you might encounter interruptions to your workloads. For more information, see the the section called "Migrate from legacy PSP".
- Kubernetes version 1.25 contains changes that alter the behavior of an existing feature known as API Priority and Fairness (APF). APF serves to shield the API server from potential overload during periods of heightened request volumes. It does this by placing restrictions on the number of concurrent requests that can be processed at

any given time. This is achieved through the application of distinct priority levels and limits to requests originating from various workloads or users. This approach ensures that critical applications or high-priority requests receive preferential treatment, while simultaneously preventing lower priority requests from overwhelming the API server. For more information, see API Priority and Fairness in the Kubernetes documentation or API Priority and Fairness in the EKS Best Practices Guide.

These updates were introduced in PR #10352 and PR #118601. Previously, APF treated all types of requests uniformly, with each request consuming a single unit of the concurrent request limit. The APF behavior change assigns higher units of concurrency to `LIST` requests due to the exceptionally heavy burden put on the API server by these requests. The API server estimates the number of objects that will be returned by a `LIST` request. It assigns a unit of concurrency that is proportional to the number of objects returned.

Upon upgrading to Amazon EKS version `1.25` or higher, this updated behavior might cause workloads with heavy `LIST` requests (that previously functioned without issue) to encounter rate limiting. This would be indicated by an HTTP 429 response code. To avoid potential workload disruption due to `LIST` requests being rate limited, we strongly encourage you to restructure your workloads to reduce the rate of these requests. Alternatively, you can address this issue by adjusting the APF settings to allocate more capacity for essential requests while reducing the capacity allocated to non-essential ones. For more information about these mitigation techniques, see Preventing Dropped Requests in the EKS Best Practices Guide.

- Amazon EKS `1.25` includes enhancements to cluster authentication that contain updated YAML libraries. If a YAML value in the `aws-auth` `ConfigMap` found in the `kube-system` namespace starts with a macro, where the first character is a curly brace, you should add quotation marks (`" "`) before and after the curly braces (`{ }`). This is required to ensure that `aws-iam-authenticator` version `v0.6.3` accurately parses the `aws-auth` `ConfigMap` in Amazon EKS `1.25`.
- The beta API version (`discovery.k8s.io/v1beta1`) of `EndpointSlice` was deprecated in Kubernetes `1.21` and is no longer served as of Kubernetes `1.25`. This API has been updated to `discovery.k8s.io/v1`. For more information, see EndpointSlice in the Kubernetes documentation. The AWS Load Balancer Controller `v2.4.6` and earlier used the `v1beta1` endpoint to communicate with `EndpointSlices`. If you're using the `EndpointSlices` configuration for the AWS Load Balancer Controller, you must upgrade to AWS Load Balancer Controller `v2.4.7` *before* upgrading your Amazon EKS

cluster to 1.25. If you upgrade to 1.25 while using the EndpointSlices configuration for the AWS Load Balancer Controller, the controller will crash and result in interruptions to your workloads. To upgrade the controller, see the section called " AWS Load Balancer Controller".

- The beta API version (autoscaling/v2beta1) of HorizontalPodAutoscaler is no longer served as of Kubernetes 1.25. This API was deprecated in version 1.23. Migrate manifests and API clients to use the autoscaling/v2 HorizontalPodAutoscaler API version. For more information, see the Kubernetes documentation.

- SeccompDefault is promoted to beta in Kubernetes 1.25. By setting the --seccomp-default flag when you configure kubelet, the container runtime uses its RuntimeDefaultseccomp profile, rather than the unconfined (seccomp disabled) mode. The default profiles provide a strong set of security defaults, while preserving the functionality of the workload. Although this flag is available, Amazon EKS doesn't enable this flag by default, so Amazon EKS behavior is effectively unchanged. If you want to, you can start enabling this on your nodes. For more details, see the tutorial Restrict a Container's Syscalls with seccomp in the Kubernetes documentation.
- Support for the Container Runtime Interface (CRI) for Docker (also known as dockershim) was removed from Kubernetes 1.24 and later. The only container runtime in Amazon EKS official AMIs for Kubernetes 1.24 and later clusters is containerd. Before upgrading to Amazon EKS 1.24 or later, remove any reference to bootstrap script flags that aren't supported anymore. For more information, see the section called "Dockershim deprecation".
- The support for wildcard queries was deprecated in CoreDNS 1.8.7 and removed in CoreDNS 1.9. This was done as a security measure. Wildcard queries no longer work and return NXDOMAIN instead of an IP address.
- The goaway-chance option in the Kubernetes API server helps prevent HTTP/2 client connections from being stuck on a single API server instance, by randomly closing a connection. When the connection is closed, the client will try to reconnect, and will likely land on a different API server as a result of load balancing. Amazon EKS version 1.25 has enabled goaway-chance flag. If your workload running on Amazon EKS cluster uses a client that is not compatible with HTTP GOAWAY, we recommend that you update your client to handle GOAWAY by reconnecting on connection termination.

For the complete Kubernetes `1.25` changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.25.md#changelog-since-v1240.

Kubernetes 1.24

Kubernetes `1.24` is now available in Amazon EKS. For more information about Kubernetes `1.24`, see the official release announcement.

> **Important**
>
> - Starting with Kubernetes `1.24`, new beta APIs aren't enabled in clusters by default. By default, existing beta APIs and new versions of existing beta APIs continue to be enabled. Amazon EKS follows the same behavior as upstream Kubernetes `1.24`. The feature gates that control new features for both new and existing API operations are enabled by default. This is in alignment with upstream Kubernetes. For more information, see KEP-3136: Beta APIs Are Off by Default on GitHub.
> - Support for Container Runtime Interface (CRI) for Docker (also known as `dockershim`) is removed from Kubernetes `1.24`. Amazon EKS official AMIs have containerd as the only runtime. Before moving to Amazon EKS `1.24` or higher, you must remove any reference to bootstrap script flags that aren't supported anymore. You must also make sure that IP forwarding is enabled for your worker nodes. For more information, see the section called "Dockershim deprecation".
> - If you already have Fluentd configured for Container Insights, then you must migrate Fluentd to Fluent Bit before updating your cluster. The Fluentd parsers are configured to only parse log messages in JSON format. Unlike `dockerd`, the `containerd` container runtime has log messages that aren't in JSON format. If you don't migrate to Fluent Bit, some of the configured Fluentd's parsers will generate a massive amount of errors inside the Fluentd container. For more information on migrating, see Set up Fluent Bit as a DaemonSet to send logs to CloudWatch Logs.
> - In Kubernetes `1.23` and earlier, `kubelet` serving certificates with unverifiable IP and DNS Subject Alternative Names (SANs) are automatically issued with unverifiable SANs. These unverifiable SANs are omitted from the provisioned certificate. In version `1.24` and later clusters, `kubelet` serving certificates aren't issued if any SAN can't be verified. This prevents `kubectl` exec and `kubectl` logs commands from working. For more information, see the section called "Certificate signing considerations before upgrading your cluster to Kubernetes 1.24".

- When upgrading an Amazon EKS 1.23 cluster that uses Fluent Bit, you must make sure that it's running k8s/1.3.12 or later. You can do this by reapplying the latest applicable Fluent Bit YAML file from GitHub. For more information, see Setting up Fluent Bit in the *Amazon CloudWatch User Guide*.

- You can use Topology Aware Hints to indicate your preference for keeping traffic in zone when cluster worker nodes are deployed across multiple availability zones. Routing traffic within a zone can help reduce costs and improve network performance. By default, Topology Aware Hints are enabled in Amazon EKS 1.24. For more information, see Topology Aware Hints in the Kubernetes documentation.
- The `PodSecurityPolicy` (PSP) is scheduled for removal in Kubernetes 1.25. PSPs are being replaced with Pod Security Admission (PSA). PSA is a built-in admission controller that uses the security controls that are outlined in the Pod Security Standards (PSS). PSA and PSS are both beta features and are enabled in Amazon EKS by default. To address the removal of PSP in version 1.25, we recommend that you implement PSS in Amazon EKS. For more information, see Implementing Pod Security Standards in Amazon EKS on the AWS blog.
- The `client.authentication.k8s.io/v1alpha1` ExecCredential is removed in Kubernetes 1.24. The ExecCredential API was generally available in Kubernetes 1.22. If you use a client-go credential plugin that relies on the `v1alpha1` API, contact the distributor of your plugin on how to migrate to the `v1` API.
- For Kubernetes 1.24, we contributed a feature to the upstream Cluster Autoscaler project that simplifies scaling Amazon EKS managed node groups to and from zero nodes. Previously, for the Cluster Autoscaler to understand the resources, labels, and taints of a managed node group that was scaled to zero nodes, you needed to tag the underlying Amazon EC2 Auto Scaling group with the details of the nodes that it was responsible for. Now, when there are no running nodes in the managed node group, the Cluster Autoscaler calls the Amazon EKS `DescribeNodegroup` API operation. This API operation provides the information that the Cluster Autoscaler requires of the managed node group's resources, labels, and taints. This feature requires that you add the `eks:DescribeNodegroup` permission to the Cluster Autoscaler service account IAM policy. When the value of a Cluster Autoscaler tag on the Auto Scaling group powering an Amazon EKS managed node group conflicts with the node group itself, the Cluster Autoscaler prefers the value of the Auto Scaling group tag. This is so that you can override values as needed. For more information, see Cluster Autoscaler.

- If you intend to use Inferentia or Trainium instance types with Amazon EKS 1.24, you must upgrade to the AWS Neuron device plugin version 1.9.3.0 or later. For more information, see Neuron K8 release [1.9.3.0] in the AWS Neuron Documentation.
- Containerd has IPv6 enabled for Pods, by default. It applies node kernel settings to Pod network namespaces. Because of this, containers in a Pod bind to both IPv4 (127.0.0.1) and IPv6 (::1) loopback addresses. IPv6 is the default protocol for communication. Before updating your cluster to version 1.24, we recommend that you test your multi-container Pods. Modify apps so that they can bind to all IP addresses on loopback interfaces. The majority of libraries enable IPv6 binding, which is backward compatible with IPv4. When it's not possible to modify your application code, you have two options:
  - Run an init container and set disable ipv6 to true (sysctl -w net.ipv6.conf.all.disable_ipv6=1).
  - Configure a mutating admission webhook to inject an init container alongside your application Pods.

  If you need to block IPv6 for all Pods across all nodes, you might have to disable IPv6 on your instances.
- The goaway-chance option in the Kubernetes API server helps prevent HTTP/2 client connections from being stuck on a single API server instance, by randomly closing a connection. When the connection is closed, the client will try to reconnect, and will likely land on a different API server as a result of load balancing. Amazon EKS version 1.24 has enabled goaway-chance flag. If your workload running on Amazon EKS cluster uses a client that is not compatible with HTTP GOAWAY, we recommend that you update your client to handle GOAWAY by reconnecting on connection termination.

For the complete Kubernetes 1.24 changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.24.md#changelog-since-v1230.

Kubernetes 1.23

Kubernetes 1.23 is now available in Amazon EKS. For more information about Kubernetes 1.23, see the official release announcement.

> **Important**
>
> - The Kubernetes in-tree to container storage interface (CSI) volume migration feature is enabled. This feature enables the replacement of existing Kubernetes in-tree storage plugins for Amazon EBS with a corresponding Amazon EBS CSI driver. For more information, see Kubernetes 1.17 Feature: Kubernetes In-Tree to CSI Volume Migration Moves to Beta on the Kubernetes blog.
>
>   The feature translates in-tree APIs to equivalent CSI APIs and delegates operations to a replacement CSI driver. With this feature, if you use existing `StorageClass`, `PersistentVolume`, and `PersistentVolumeClaim` objects that belong to these workloads, there likely won't be any noticeable change. The feature enables Kubernetes to delegate all storage management operations from the in-tree plugin to the CSI driver. If you use Amazon EBS volumes in an existing cluster, install the Amazon EBS CSI driver in your cluster before you update your cluster to version `1.23`. If you don't install the driver before updating an existing cluster, interruptions to your workloads might occur. If you plan to deploy workloads that use Amazon EBS volumes in a new `1.23` cluster, install the Amazon EBS CSI driver in your cluster before deploying the workloads your cluster. For instructions on how to install the Amazon EBS CSI driver on your cluster, see the section called "Amazon EBS".
> - Extended Support for Amazon EKS optimized Windows AMIs that are published by AWS isn't available for Kubernetes version `1.23` but is available for Kubernetes version `1.24` and higher.

- Kubernetes stopped supporting `dockershim` in version `1.20` and removed `dockershim` in version `1.24`. For more information, see Kubernetes is Moving on From Dockershim: Commitments and Next Steps in the Kubernetes blog. Amazon EKS will end support for `dockershim` starting in Amazon EKS version `1.24`. Starting with Amazon EKS version `1.24`, Amazon EKS official AMIs will have `containerd` as the only runtime.

  Even though Amazon EKS version `1.23` continues to support `dockershim`, we recommend that you start testing your applications now to identify and remove any Docker dependencies. This way, you are prepared to update your cluster to version `1.24`. For more information about `dockershim` removal, see the section called "Dockershim deprecation".

- Kubernetes graduated IPv4/IPv6 dual-stack networking for Pods, services, and nodes to general availability. However, Amazon EKS and the Amazon VPC CNI plugin for Kubernetes don't support dual-stack networking. Your clusters can assign IPv4 or IPv6 addresses to Pods and services, but can't assign both address types.
- Kubernetes graduated the Pod Security Admission (PSA) feature to beta. The feature is enabled by default. For more information, see Pod Security Admission in the Kubernetes documentation. PSA replaces the Pod Security Policy (PSP) admission controller. The PSP admission controller isn't supported and is scheduled for removal in Kubernetes version 1.25.

  The PSP admission controller enforces Pod security standards on Pods in a namespace based on specific namespace labels that set the enforcement level. For more information, see Pod Security Standards (PSS) and Pod Security Admission (PSA) in the Amazon EKS best practices guide.
- The kube-proxy image deployed with clusters is now the minimal base image maintained by Amazon EKS Distro (EKS-D). The image contains minimal packages and doesn't have shells or package managers.
- Kubernetes graduated ephemeral containers to beta. Ephemeral containers are temporary containers that run in the same namespace as an existing Pod. You can use them to observe the state of Pods and containers for troubleshooting and debugging purposes. This is especially useful for interactive troubleshooting when kubectl exec is insufficient because either a container has crashed or a container image doesn't include debugging utilities. An example of a container that includes a debugging utility is distroless images. For more information, see Debugging with an ephemeral debug container in the Kubernetes documentation.
- Kubernetes graduated the HorizontalPodAutoscaler autoscaling/v2 stable API to general availability. The HorizontalPodAutoscaler autoscaling/v2beta2 API is deprecated. It will be unavailable in 1.26.
- The goaway-chance option in the Kubernetes API server helps prevent HTTP/2 client connections from being stuck on a single API server instance, by randomly closing a connection. When the connection is closed, the client will try to reconnect, and will likely land on a different API server as a result of load balancing. Amazon EKS version 1.23 has enabled goaway-chance flag. If your workload running on Amazon EKS cluster uses a client that is not compatible with HTTP GOAWAY, we recommend that you update your client to handle GOAWAY by reconnecting on connection termination.

For the complete Kubernetes 1.23 changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.23.md#changelog-since-v1220.

View current cluster support period

The **cluster support period** section of the AWS console indicates if your cluster is *currently* on standard or extended support. If your cluster support period is **Extended support**, you are being charged for EKS extended support.

For more information about standard and extended support, see the section called “Kubernetes versions”.

1. Navigate to the **Clusters** page in the EKS section of the AWS Console. Confirm the console is set to the same AWS region as the cluster you want to review.
2. Review the **Support Period** column. If the value is **Standard support until...**, you are not currently being charged for extended support. You are within the standard support period. If the value is **Extended support...** this cluster is currently being charged for extended support.

> **Note**
>
> The **Support Period** cannot be retrieved with the AWS API or CLI.

View current cluster upgrade policy

The **cluster upgrade policy** determines what happens to your cluster when it leaves the standard support period. If your upgrade policy is `EXTENDED`, the cluster will not be automatically upgraded, and will enter extended support. If your upgrade policy is `STANDARD`, it will be automatically upgraded.

Amazon EKS controls for Kubernetes version policy allows you to choose the end of standard support behavior for your EKS clusters. With these controls you can decide which clusters should enter extended support and which clusters should be automatically upgraded at the end of standard support for a Kubernetes version.

A minor version is under standard support in Amazon EKS for the first 14 months after it’s released. Once a version is past the end of standard support date, it enters extended support for the next 12 months. Extended support allows you to stay at a specific Kubernetes version for longer at an additional cost per cluster hour. You can enable or disable extended support for an EKS Cluster. If you disable extended support, AWS will automatically upgrade your cluster to the next version at the end of standard support. If you enable extended support, you can stay at the current version for an additional cost for a limited period of time. Plan to regularly upgrade your Kubernetes cluster, even if you use extended support.

You can set the version policy for both new and existing clusters, using the `supportType` property. There are two options that can be used to set the version support policy:

- **`STANDARD`** — Your EKS cluster eligible for automatic upgrade at the end of standard support. You will not incur extended support charges with this setting but you EKS cluster will automatically upgrade to the next supported Kubernetes version in standard support.
- **`EXTENDED`** — Your EKS cluster will enter into extended support once the Kubernetes version reaches end of standard support. You will incur extended support charges with this setting. You can upgrade your cluster to a standard supported Kubernetes version to stop incurring extended support charges. Clusters running on extended support will be eligible for automatic upgrade at the end of extended support.

Extended support is enabled by default for new clusters, and existing clusters. You can view if extended support is enabled for a cluster in the AWS Management Console, or by using the AWS CLI.

> **Important**
>
> If you want your cluster to stay on its current Kubernetes version to take advantage of the extended support period, you must enable the extended support upgrade policy before the end of standard support period.

You can only set the version support policy for your clusters while its running on Kubernetes version in standard support. Once the version enters extended support, you will not be able to change this setting until you are running on a version in standard support.

For example, if you have set your version support policy as `standard` then you will not be able to change this setting after the Kubernetes version running on your cluster reaches the end of standard support. If you have set your version support policy as `extended` then you will not be able to change this setting after the Kubernetes version running on your cluster reaches end of standard support. In order to change the version support policy setting, your cluster must be running on a standard supported Kubernetes version.

View cluster upgrade policy (AWS Console)

1. Navigate to the **Clusters** page in the EKS section of the AWS Console. Confirm the console is set to the same AWS region as the cluster you want to review.
2. Review the **Upgrade Policy** column. If the value is **Standard Support**, your cluster will not enter extended support. If the value is **Extended Support**, your cluster will enter extended support.

View cluster upgrade policy (AWS CLI)

1. Verify the AWS CLI is installed and you are logged in. Learn how to update and install the AWS CLI.
2. Determine the name of your EKS cluster. Set the CLI to the same AWS region as your EKS cluster.
3. Run the following command:

```
aws eks describe-cluster \
--name <cluster-name> \
--query "cluster.upgradePolicy.supportType"
```

4. If the value is `STANDARD`, your cluster will not enter extended support. If the value is `EXTENDED`, your cluster will enter extended support.

Add flexibility to plan Kubernetes version upgrades by enabling EKS extended support

This topic describes how to set the *upgrade policy* of an EKS cluster to enable extended support. The upgrade policy of an EKS cluster determines what happens when a cluster reaches the end of the standard *support period*. If a cluster upgrade policy has extended support enabled, it will enter the extended support period at the end of the standard support period. The cluster will not be automatically upgraded at the end of the standard support period.

Clusters actually in the *extended support period* incur higher costs. If a cluster merely has the upgrade policy set to enable extended support, and is otherwise in the *standard support period*, it incurs standard costs.

If you create a cluster in the AWS console, it will have the upgrade policy set to disable extended support. If you create a cluster in another way, it will have the upgrade policy set to enable extended support. For example, clusters created with the AWS API have extended support enabled.

For more information about upgrade policies, see the section called “View upgrade policy”.

> **Important**
>
> If you want your cluster to stay on its current Kubernetes version to take advantage of the extended support period, you must enable the extended support upgrade policy before the end of standard support period.
> If you do not enable extended support, your cluster will be automatically upgraded.

Enable EKS extended support (AWS Console)

1. Navigate to your EKS cluster in the AWS Console. Select the **Overview** tab on the **Cluster Info** page.
2. In the **Kubernetes version settings** section, select **Manage**.
3. Select **Extended support** and then **Save changes**.

Enable EKS extended support (AWS CLI)

1. Verify the AWS CLI is installed and you are logged in. Learn how to update and install the AWS CLI.
2. Determine the name of your EKS cluster.
3. Run the following command:

```
aws eks update-cluster-config \
--name <cluster-name> \
--upgrade-policy supportType=EXTENDED
```

Prevent increased cluster costs by disabling EKS extended support

This topic describes how to set the *upgrade policy* of an EKS cluster to disable extended support. The upgrade policy of an EKS cluster determines what happens when a cluster reaches the end of the standard *support period*. If a cluster upgrade policy has extended support disabled, it will be automatically upgraded to the next Kubernetes version.

For more information about upgrade policies, see the section called “View upgrade policy”.

> **Important**
>
> You cannot disable extended support once your cluster has entered it. You can only disable extended support for clusters on standard support.
>
> AWS recommends upgrading your cluster to a version in the standard support period.

Disable EKS extended support (AWS Console)

1. Navigate to your EKS cluster in the AWS Console. Select the **Overview** tab on the **Cluster Info** page.
2. In the **Kubernetes version setting** section, select **Manage**.
3. Select **Standard support** and then **Save changes**.

Disable EKS extended support (AWS CLI)

1. Verify the AWS CLI is installed and you are logged in. Learn how to update and install the AWS CLI.
2. Determine the name of your EKS cluster.
3. Run the following command:

```
aws eks update-cluster-config \
--name <cluster-name> \
--upgrade-policy supportType=STANDARD
```

View Amazon EKS platform versions for each Kubernetes version

Amazon EKS platform versions represent the capabilities of the Amazon EKS cluster control plane, such as which Kubernetes API server flags are enabled, as well as the current Kubernetes patch version. Each Kubernetes minor version has one or more associated Amazon EKS platform versions. The platform versions for different Kubernetes minor versions are independent. You can retrieve your cluster's current platform version using the AWS CLI or AWS Management Console. If you have a local cluster on AWS Outposts, see the section called "EKS platform versions" instead of this topic.

When a new Kubernetes minor version is available in Amazon EKS, such as 1.32, the initial Amazon EKS platform version for that Kubernetes minor version starts at `eks.1`. However, Amazon EKS releases new platform versions periodically to enable new Kubernetes control plane settings and to provide security fixes.

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

- The Amazon EKS platform version number is incremented (`eks.<n+1>`).
- Amazon EKS automatically upgrades all existing clusters to the latest Amazon EKS platform version for their corresponding Kubernetes minor version. Automatic upgrades of existing Amazon EKS platform versions are rolled out incrementally. The roll-out process might take some time. If you need the latest Amazon EKS platform version features immediately, you should create a new Amazon EKS cluster.

  If your cluster is more than two platform versions behind the current platform version, then it's possible that Amazon EKS wasn't able to automatically update your cluster. For details of what may cause this, see the section called "Amazon EKS platform version is more than two versions behind the current platform version".
- Amazon EKS might publish a new node AMI with a corresponding patch version. However, all patch versions are compatible between the EKS control plane and node AMIs for a given Kubernetes minor version.

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

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

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

> **Note**
>
> AWS recently disabled some platform versions published in June 2024. The platform versions had stability issues. No action is needed.

Kubernetes version 1.32

The following admission controllers are enabled for all 1.32 platform versions: NodeRestriction, ExtendedResourceToleration, NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, PodSecurity, Priority, DefaultTolerationSeconds, DefaultStorageClass, StorageObjectInUseProtection, PersistentVolumeClaimResize, RuntimeClass, CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota.

Kubernetes version	EKS platform version	Release notes	Release date
1.32.0	eks.2	Initial release of Kubernetes version 1.32 for EKS. For more information, see the section called "Kubernetes 1.32".	January 2025

Kubernetes version 1.31

The following admission controllers are enabled for all 1.31 platform versions: NodeRestriction, ExtendedResourceToleration, NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, PodSecurity, Priority, DefaultTolerationSeconds, DefaultStorageClass, StorageObjectInUseProtection, PersistentVolumeClaimResize, RuntimeClass, CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota, ObjectCount.

Kubernetes version	EKS platform version	Release notes	Release date
`1.31.4`	`eks.17`	New platform version with security fixes and enhancements.	January 17, 2025
`1.31.2`	`eks.12`	New platform version with Amazon EKS Hybrid Nodes support and enhancements to control plane observability. See the section called "Hybrid nodes" and see Amazon EKS enhances performance observability, respectively.	November 15, 2024
`1.31.1`	`eks.6`	New platform version with security fixes and enhancements.	October 21, 2024
`1.31.0`	`eks.4`	Initial release of Kubernetes version `1.31` for EKS. For more information, see the section called "Kubernetes 1.31".	September 26, 2024

Kubernetes version `1.30`

The following admission controllers are enabled for all `1.30` platform versions: `NodeRestriction`, `ExtendedResourceToleration`, `NamespaceLifecycle`, `LimitRanger`, `ServiceAccount`, `TaintNodesByCondition`, `PodSecurity`, `Priority`, `DefaultTolerationSeconds`, `DefaultStorageClass`, `StorageObjectInUseProtection`, `PersistentVolumeClaimResize`, `RuntimeClass`, `CertificateApproval`, `CertificateSigning`, `CertificateSubjectRestriction`, `DefaultIngressClass`, `MutatingAdmissionWebhook`, `ValidatingAdmissionWebhook`, `ResourceQuota`.

Kubernetes version	EKS platform version	Release notes	Release date
`1.30.8`	`eks.25`	New platform version with security fixes and enhancements.	January 17, 2025
`1.30.6`	`eks.20`	New platform version with Amazon EKS Hybrid Nodes support and enhancements to control plane observability. See the section called "Hybrid nodes" and see Amazon EKS enhances performance observability, respectively.	November 15, 2024
`1.30.5`	`eks.12`	New platform version with security fixes and enhancements.	October 21, 2024
`1.30.4`	`eks.8`	New platform version with security fixes and enhancements.	September 3, 2024

Kubernetes version	EKS platform version	Release notes	Release date
`1.30.3`	`eks.7`	New platform version with security fixes and enhancements.	August 28, 2024
`1.30.3`	`eks.6`	New platform version with security fixes and enhancements.	August 9, 2024
`1.30.2`	`eks.5`	New platform version with security fixes and enhancements.	July 2, 2024
`1.30.0`	`eks.2`	Initial release of Kubernetes version `1.30` for EKS. For more information, see the section called "Kubernetes 1.30".	May 23, 2024

Kubernetes version `1.29`

The following admission controllers are enabled for all `1.29` platform versions: `NodeRestriction`, `ExtendedResourceToleration`, `NamespaceLifecycle`, `LimitRanger`, `ServiceAccount`, `TaintNodesByCondition`, `PodSecurity`, `Priority`, `DefaultTolerationSeconds`, `DefaultStorageClass`, `StorageObjectInUseProtection`, `PersistentVolumeClaimResize`, `RuntimeClass`, `CertificateApproval`, `CertificateSigning`, `CertificateSubjectRestriction`, `DefaultIngressClass`, `MutatingAdmissionWebhook`, `ValidatingAdmissionWebhook`, `ResourceQuota`.

Kubernetes version	EKS platform version	Release notes	Release date
1.29.12	eks.28	New platform version with security fixes and enhancements.	January 17, 2025
1.29.10	eks.23	New platform version with Amazon EKS Hybrid Nodes support and enhancements to control plane observability. See the section called "Hybrid nodes" and see Amazon EKS enhances performance observability, respectively.	November 15, 2024
1.29.9	eks.17	New platform version with security fixes and enhancements.	October 21, 2024
1.29.8	eks.13	New platform version with security fixes and enhancements.	September 3, 2024
1.29.7	eks.12	New platform version with security fixes and enhancements.	August 28, 2024
1.29.7	eks.11	New platform version with security fixes and enhancements.	August 9, 2024

Kubernetes version	EKS platform version	Release notes	Release date
`1.29.6`	`eks.10`	New platform version with security fixes and enhancements.	July 2, 2024
`1.29.4`	`eks.7`	New platform version with CoreDNS autoscaling, security fixes and enhancements. For more information about CoreDNS autoscaling, see the section called "Scale for high traffic".	May 16, 2024
`1.29.3`	`eks.6`	New platform version with security fixes and enhancements.	April 18, 2024
`1.29.1`	`eks.5`	New platform version with security fixes and enhancements.	March 29, 2024
`1.29.1`	`eks.4`	New platform version with security fixes and enhancements.	March 20, 2024
`1.29.1`	`eks.3`	New platform version with security fixes and enhancements.	March 12, 2024

Kubernetes version	EKS platform version	Release notes	Release date
1.29.0	eks.1	Initial release of Kubernetes version 1.29 for EKS. For more information, see the section called "Kubernetes 1.29".	January 23, 2024

Kubernetes version 1.28

The following admission controllers are enabled for all 1.28 platform versions: NodeRestriction, ExtendedResourceToleration, NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, PodSecurity, Priority, DefaultTolerationSeconds, DefaultStorageClass, StorageObjectInUseProtection, PersistentVolumeClaimResize, RuntimeClass, CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota.

Kubernetes version	EKS platform version	Release notes	Release date
1.28.15	eks.34	New platform version with security fixes and enhancements.	January 17, 2025
1.28.15	eks.29	New platform version with Amazon EKS Hybrid Nodes support and enhancements to control plane observability. See the section called "Hybrid nodes" and	November 15, 2024

Kubernetes version	EKS platform version	Release notes	Release date
		see Amazon EKS enhances performance observability, respectively.	
`1.28.14`	`eks.23`	New platform version with security fixes and enhancements.	October 21, 2024
`1.28.13`	`eks.19`	New platform version with security fixes and enhancements.	September 3, 2024
`1.28.12`	`eks.18`	New platform version with security fixes and enhancements.	August 28, 2024
`1.28.11`	`eks.17`	New platform version with security fixes and enhancements.	August 9, 2024
`1.28.11`	`eks.16`	New platform version with security fixes and enhancements.	July 2, 2024
`1.28.9`	`eks.13`	New platform version with CoreDNS autoscaling, security fixes and enhancements. For more information about CoreDNS autoscaling, see the section called "Scale for high traffic".	May 16, 2024

Kubernetes version	EKS platform version	Release notes	Release date
1.28.8	eks.12	New platform version with security fixes and enhancements.	April 18, 2024
1.28.7	eks.11	New platform version with security fixes and enhancements.	March 29, 2024
1.28.7	eks.10	New platform version with security fixes and enhancements.	March 20, 2024
1.28.6	eks.9	New platform version with security fixes and enhancements.	March 12, 2024
1.28.5	eks.7	New platform version with security fixes and enhancements.	January 17, 2024
1.28.4	eks.6	New platform version with access entries, security fixes and enhancements.	December 14, 2023
1.28.4	eks.5	New platform version with security fixes and enhancements.	December 12, 2023
1.28.3	eks.4	New platform version with Learn how EKS Pod Identity grants pods access to AWS services, security fixes and enhancements.	November 10, 2023

Kubernetes version	EKS platform version	Release notes	Release date
`1.28.3`	`eks.3`	New platform version with security fixes and enhancements.	November 3, 2023
`1.28.2`	`eks.2`	New platform version with security fixes and enhancements.	October 16, 2023
`1.28.1`	`eks.1`	Initial release of Kubernetes version `1.28` for EKS. For more information, see the section called "Kubernetes 1.28".	September 26, 2023

Kubernetes version `1.27`

The following admission controllers are enabled for all `1.27` platform versions: `NodeRestriction`, `ExtendedResourceToleration`, `NamespaceLifecycle`, `LimitRanger`, `ServiceAccount`, `TaintNodesByCondition`, `PodSecurity`, `Priority`, `DefaultTolerationSeconds`, `DefaultStorageClass`, `StorageObjectInUseProtection`, `PersistentVolumeClaimResize`, `RuntimeClass`, `CertificateApproval`, `CertificateSigning`, `CertificateSubjectRestriction`, `DefaultIngressClass`, `MutatingAdmissionWebhook`, `ValidatingAdmissionWebhook`, `ResourceQuota`.

Kubernetes version	EKS platform version	Release notes	Release date
`1.27.16`	`eks.38`	New platform version with security fixes and enhancements.	January 17, 2025
`1.27.16`	`eks.33`	New platform version with Amazon	November 15, 2024

Kubernetes version	EKS platform version	Release notes	Release date
		EKS Hybrid Nodes support, security fixes and enhancements. For more information about Amazon EKS Hybrid Nodes, see the section called “Hybrid nodes”.	
`1.27.16`	`eks.27`	New platform version with security fixes and enhancements.	October 21, 2024
`1.27.16`	`eks.23`	New platform version with security fixes and enhancements.	September 3, 2024
`1.27.16`	`eks.22`	New platform version with security fixes and enhancements.	August 28, 2024
`1.27.16`	`eks.21`	New platform version with security fixes and enhancements.	August 9, 2024
`1.27.15`	`eks.20`	New platform version with security fixes and enhancements.	July 2, 2024

Kubernetes version	EKS platform version	Release notes	Release date
`1.27.13`	`eks.17`	New platform version with CoreDNS autoscaling, security fixes and enhancements. For more information about CoreDNS autoscaling, see the section called "Scale for high traffic".	May 16, 2024
`1.27.12`	`eks.16`	New platform version with security fixes and enhancements.	April 18, 2024
`1.27.11`	`eks.15`	New platform version with security fixes and enhancements.	March 29, 2024
`1.27.11`	`eks.14`	New platform version with security fixes and enhancements.	March 20, 2024
`1.27.10`	`eks.13`	New platform version with security fixes and enhancements.	March 12, 2024
`1.27.9`	`eks.11`	New platform version with security fixes and enhancements.	January 17, 2024
`1.27.8`	`eks.10`	New platform version with access entries, security fixes and enhancements.	December 14, 2023

Kubernetes version	EKS platform version	Release notes	Release date
1.27.8	eks.9	New platform version with security fixes and enhancements.	December 12, 2023
1.27.7	eks.8	New platform version with Learn how EKS Pod Identity grants pods access to AWS services, security fixes and enhancements.	November 10, 2023
1.27.7	eks.7	New platform version with security fixes and enhancements.	November 3, 2023
1.27.6	eks.6	New platform version with security fixes and enhancements.	October 16, 2023
1.27.4	eks.5	New platform version with security fixes and enhancements.	August 30, 2023
1.27.4	eks.4	New platform version with security fixes and enhancements.	July 30, 2023
1.27.3	eks.3	New platform version with security fixes and enhancements.	June 30, 2023
1.27.2	eks.2	New platform version with security fixes and enhancements.	June 9, 2023

Kubernetes version	EKS platform version	Release notes	Release date
1.27.1	eks.1	Initial release of Kubernetes version 1.27 for EKS. For more information, see the section called "Kubernetes 1.27".	May 24, 2023

Kubernetes version 1.26

The following admission controllers are enabled for all 1.26 platform versions: NodeRestriction, ExtendedResourceToleration, NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, PodSecurity, Priority, DefaultTolerationSeconds, DefaultStorageClass, StorageObjectInUseProtection, PersistentVolumeClaimResize, RuntimeClass, CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota.

Kubernetes version	EKS platform version	Release notes	Release date
1.26.15	eks.40	New platform version with security fixes and enhancements.	January 17, 2025
1.26.15	eks.35	New platform version with Amazon EKS Hybrid Nodes support, security fixes and enhancements. For more information about Amazon EKS Hybrid Nodes, see the	November 15, 2024

Kubernetes version	EKS platform version	Release notes	Release date
		section called "Hybrid nodes".	
1.26.15	eks.28	New platform version with security fixes and enhancements.	October 21, 2024
1.26.15	eks.24	New platform version with security fixes and enhancements.	September 3, 2024
1.26.15	eks.23	New platform version with security fixes and enhancements.	August 28, 2024
1.26.15	eks.22	New platform version with security fixes and enhancements.	August 9, 2024
1.26.15	eks.21	New platform version with security fixes and enhancements.	July 2, 2024
1.26.15	eks.18	New platform version with CoreDNS autoscaling, security fixes and enhanceme nts. For more information about CoreDNS autoscali ng, see the section called "Scale for high traffic".	May 16, 2024

Kubernetes version	EKS platform version	Release notes	Release date
`1.26.15`	`eks.17`	New platform version with security fixes and enhancements.	April 18, 2024
`1.26.14`	`eks.16`	New platform version with security fixes and enhancements.	March 29, 2024
`1.26.14`	`eks.15`	New platform version with security fixes and enhancements.	March 20, 2024
`1.26.13`	`eks.14`	New platform version with security fixes and enhancements.	March 12, 2024
`1.26.12`	`eks.12`	New platform version with security fixes and enhancements.	January 17, 2024
`1.26.11`	`eks.11`	New platform version with access entries, security fixes and enhancements.	December 14, 2023
`1.26.11`	`eks.10`	New platform version with security fixes and enhancements.	December 12, 2023
`1.26.10`	`eks.9`	New platform version with Learn how EKS Pod Identity grants pods access to AWS services, security fixes and enhancements.	November 10, 2023

Kubernetes version	EKS platform version	Release notes	Release date
`1.26.10`	`eks.8`	New platform version with security fixes and enhancements.	November 3, 2023
`1.26.9`	`eks.7`	New platform version with security fixes and enhancements.	October 16, 2023
`1.26.7`	`eks.6`	New platform version with security fixes and enhancements.	August 30, 2023
`1.26.7`	`eks.5`	New platform version with security fixes and enhancements.	July 30, 2023
`1.26.6`	`eks.4`	New platform version with security fixes and enhancements.	June 30, 2023
`1.26.5`	`eks.3`	New platform version with security fixes and enhancements.	June 9, 2023
`1.26.4`	`eks.2`	New platform version with security fixes and enhancements.	May 5, 2023
`1.26.2`	`eks.1`	Initial release of Kubernetes version `1.26` for EKS. For more information, see the section called "Kubernetes 1.26".	April 11, 2023

Kubernetes version 1.25

The following admission controllers are enabled for all 1.25 platform versions: NodeRestriction, ExtendedResourceToleration, NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, PodSecurity, Priority, DefaultTolerationSeconds, DefaultStorageClass, StorageObjectInUseProtection, PersistentVolumeClaimResize, RuntimeClass, CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota.

Kubernetes version	EKS platform version	Release notes	Release date
1.25.16	eks.40	New platform version with security fixes and enhancements.	January 17, 2025
1.25.16	eks.35	New platform version with Amazon EKS Hybrid Nodes support, security fixes and enhancements. For more information about Amazon EKS Hybrid Nodes, see the section called “Hybrid nodes”.	November 15, 2024
1.25.16	eks.29	New platform version with security fixes and enhancements.	October 21, 2024
1.25.16	eks.25	New platform version with security fixes and enhancements.	September 3, 2024

Kubernetes version	EKS platform version	Release notes	Release date
1.25.16	eks.24	New platform version with security fixes and enhancements.	August 28, 2024
1.25.16	eks.23	New platform version with security fixes and enhancements.	August 9, 2024
1.25.16	eks.22	New platform version with security fixes and enhancements.	July 2, 2024
1.25.16	eks.19	New platform version with CoreDNS autoscaling, security fixes and enhanceme nts. For more information about CoreDNS autoscali ng, see the section called “Scale for high traffic”.	May 16, 2024
1.25.16	eks.18	New platform version with security fixes and enhancements.	April 18, 2024
1.25.16	eks.17	New platform version with security fixes and enhancements.	March 29, 2024
1.25.16	eks.16	New platform version with security fixes and enhancements.	March 20, 2024

Kubernetes version	EKS platform version	Release notes	Release date
`1.25.16`	`eks.15`	New platform version with security fixes and enhancements.	March 12, 2024
`1.25.16`	`eks.13`	New platform version with security fixes and enhancements.	January 17, 2024
`1.25.16`	`eks.12`	New platform version with access entries, security fixes and enhancements.	December 14, 2023
`1.25.16`	`eks.11`	New platform version with security fixes and enhancements.	December 12, 2023
`1.25.15`	`eks.10`	New platform version with Learn how EKS Pod Identity grants pods access to AWS services, security fixes and enhancements.	November 10, 2023
`1.25.15`	`eks.9`	New platform version with security fixes and enhancements.	November 3, 2023
`1.25.14`	`eks.8`	New platform version with security fixes and enhancements.	October 16, 2023
`1.25.12`	`eks.7`	New platform version with security fixes and enhancements.	August 30, 2023

Kubernetes version	EKS platform version	Release notes	Release date
1.25.12	eks.6	New platform version with security fixes and enhancements.	July 30, 2023
1.25.11	eks.5	New platform version with security fixes and enhancements.	June 30, 2023
1.25.10	eks.4	New platform version with security fixes and enhancements.	June 9, 2023
1.25.9	eks.3	New platform version with security fixes and enhancements.	May 5, 2023
1.25.8	eks.2	New platform version with security fixes and enhancements.	March 24, 2023
1.25.6	eks.1	Initial release of Kubernetes version 1.25 for EKS. For more information, see the section called "Kubernetes 1.25".	February 21, 2023

Kubernetes version 1.24

The following admission controllers are enabled for all 1.24 platform versions: CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, DefaultStorageClass, DefaultTolerationSeconds, ExtendedResourceToleration, LimitRanger, MutatingAdmissionWebhook, NamespaceLifecycle, NodeRestriction, PersistentVolumeClaimResize, `Priority`, `PodSecurityPolicy`, `ResourceQuota`, `RuntimeClass`, `ServiceAccount`, `StorageObjectInUseProtection`, `TaintNodesByCondition`, and `ValidatingAdmissionWebhook`.

Kubernetes version	EKS platform version	Release notes	Release date
`1.24.17`	`eks.44`	New platform version with security fixes and enhancements.	January 17, 2025
`1.24.17`	`eks.39`	New platform version with security fixes and enhancements.	November 15, 2024
`1.24.17`	`eks.32`	New platform version with security fixes and enhancements.	October 21, 2024
`1.24.17`	`eks.28`	New platform version with security fixes and enhancements.	September 3, 2024
`1.24.17`	`eks.27`	New platform version with security fixes and enhancements.	August 28, 2024
`1.24.17`	`eks.26`	New platform version with security fixes and enhancements.	August 9, 2024
`1.24.17`	`eks.25`	New platform version with security fixes and enhancements.	July 2, 2024
`1.24.17`	`eks.22`	New platform version with security fixes and enhancements.	May 16, 2024

Kubernetes version	EKS platform version	Release notes	Release date
1.24.17	eks.21	New platform version with security fixes and enhancements.	April 18, 2024
1.24.17	eks.20	New platform version with security fixes and enhancements.	March 29, 2024
1.24.17	eks.19	New platform version with security fixes and enhancements.	March 20, 2024
1.24.17	eks.18	New platform version with security fixes and enhancements.	March 12, 2024
1.24.17	eks.16	New platform version with security fixes and enhancements.	January 17, 2024
1.24.17	eks.15	New platform version with access entries, security fixes and enhancements.	December 14, 2023
1.24.17	eks.14	New platform version with security fixes and enhancements.	December 12, 2023
1.24.17	eks.13	New platform version with Learn how EKS Pod Identity grants pods access to AWS services, security fixes and enhancements.	November 10, 2023

Kubernetes version	EKS platform version	Release notes	Release date
1.24.17	eks.12	New platform version with security fixes and enhancements.	November 3, 2023
1.24.17	eks.11	New platform version with security fixes and enhancements.	October 16, 2023
1.24.16	eks.10	New platform version with security fixes and enhancements.	August 30, 2023
1.24.16	eks.9	New platform version with security fixes and enhancements.	July 30, 2023
1.24.15	eks.8	New platform version with security fixes and enhancements.	June 30, 2023
1.24.14	eks.7	New platform version with security fixes and enhancements.	June 9, 2023
1.24.13	eks.6	New platform version with security fixes and enhancements.	May 5, 2023
1.24.12	eks.5	New platform version with security fixes and enhancements.	March 24, 2023
1.24.8	eks.4	New platform version with security fixes and enhancements.	January 27, 2023

Kubernetes version	EKS platform version	Release notes	Release date
`1.24.7`	`eks.3`	New platform version with security fixes and enhancements.	December 5, 2022
`1.24.7`	`eks.2`	New platform version with security fixes and enhancements.	November 18, 2022
`1.24.7`	`eks.1`	Initial release of Kubernetes version `1.24` for EKS. For more information, see the section called "Kubernetes 1.24".	November 15, 2022

Get current platform version

1. Open the Amazon EKS console.
2. In the navigation pane, choose **Clusters**.
3. In the list of clusters, choose the **Cluster Name** to check the platform version of.
4. Choose the **Overview** tab.
5. The **Platform Version** is available under in the **Details** section.
6. Determine the **Name** of the cluster you want to check the platform version of.
7. Run the following command:

```
aws eks describe-cluster --name my-cluster --query cluster.platformVersion
```

   An example output is as follows.

```
"eks.10"
```

Change platform version

You cannot change the platform version of an EKS cluster. When new Amazon EKS platform versions become available for a Kubernetes version, EKS automatically upgrades all existing clusters to the latest Amazon EKS platform version for their corresponding Kubernetes version. Automatic upgrades of existing Amazon EKS platform versions are rolled out incrementally. You cannot use the AWS Console or CLI to change the platform version.

If you upgrade your Kubernetes version, your cluster will move onto the most recent platform version for the Kubernetes version.

Scale cluster compute with Karpenter and Cluster Autoscaler

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

EKS Auto Mode

Amazon EKS Auto Mode automatically scales cluster compute resources. If a pod can't fit onto existing nodes, EKS Auto Mode creates a new one. EKS Auto Mode also consolidates workloads and deletes nodes. EKS Auto Mode builds upon Karpenter.

For more information, see:

- *EKS Auto Mode*
- the section called "Create node pool"
- the section called "Deploy inflate workload"

Additional Solutions

Amazon EKS supports two additional autoscaling products:

Karpenter

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

> **Important**
>
> Karpenter is open-source software which AWS customers are responsible for installing, configuring, and managing in their Kubernetes clusters. AWS provides technical support when Karpenter is run unmodified using a compatible version in Amazon EKS clusters. It is essential that customers maintain the availability and security of the Karpenter controller as well as appropriate testing procedures when upgrading it or the Kubernetes cluster in which it's running, just like any other customer-managed software. There is no AWS Service Level Agreement (SLA) for Karpenter and customers are responsible for ensuring that the EC2 instances launched by Karpenter meet their business requirements.

Cluster Autoscaler

The Kubernetes Cluster Autoscaler automatically adjusts the number of nodes in your cluster when pods fail or are rescheduled onto other nodes. The Cluster Autoscaler uses Auto Scaling groups. For more information, see Cluster Autoscaler on AWS.

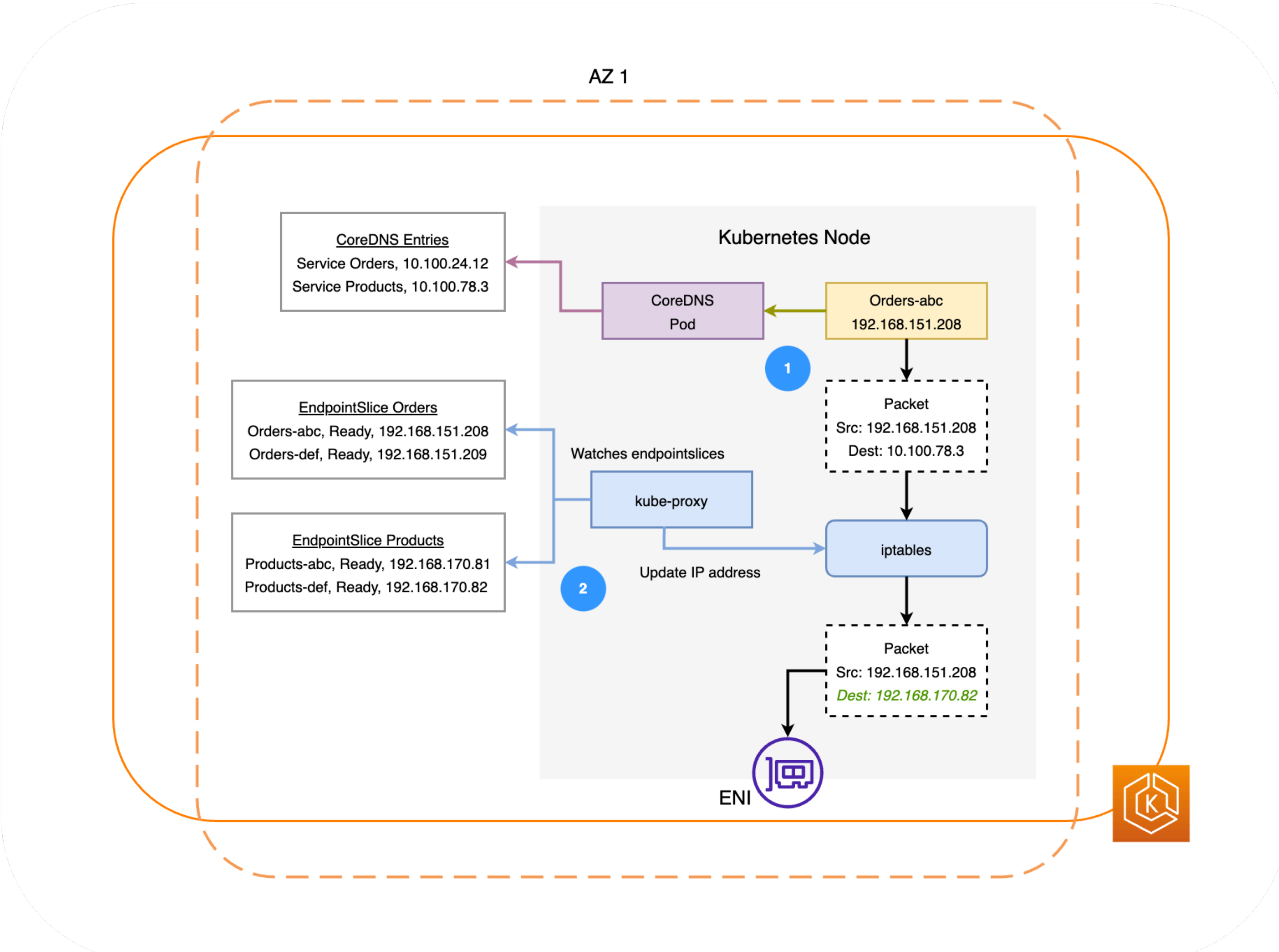
Learn about Amazon Application Recovery Controller's (ARC) Zonal Shift in Amazon EKS

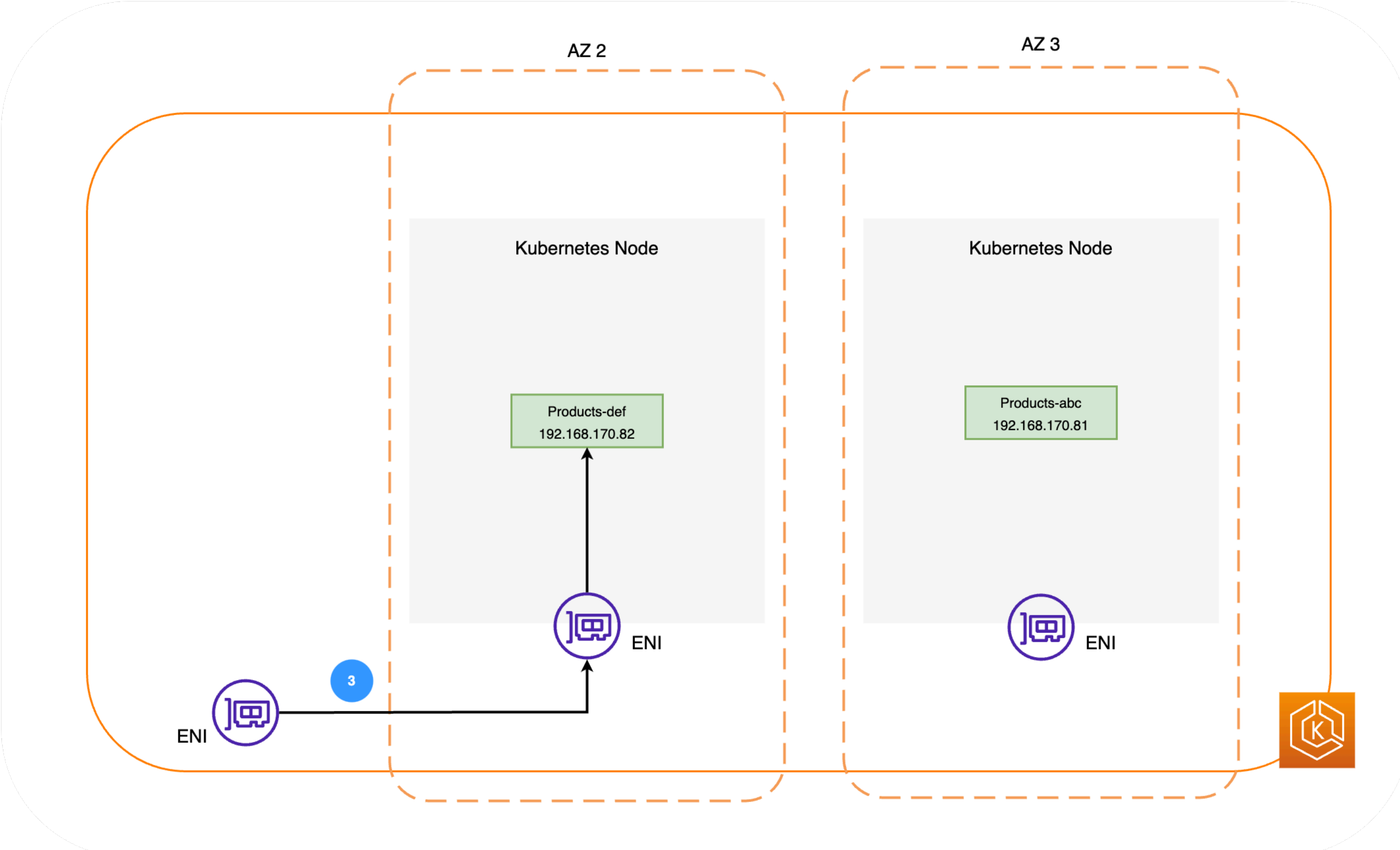
Kubernetes has native features that enable you to make your applications more resilient to events such as the degraded health or impairment of an Availability Zone (AZ). When running your workloads in an Amazon EKS cluster, you can further improve your application environment's fault tolerance and application recovery using Amazon Application Recovery Controller's (ARC) zonal shift or zonal autoshift. ARC zonal shift is designed to be a temporary measure that allows you to move traffic for a resource away from an impaired AZ until the zonal shift expires or you cancel it. You can extend the zonal shift if necessary.

You can start a zonal shift for an EKS cluster, or you can allow AWS to do it for you by enabling zonal autoshift. This shift updates the flow of east-to-west network traffic in your cluster to only consider network endpoints for Pods running on worker nodes in healthy AZs. Additionally, any ALB or NLB handling ingress traffic for applications in your EKS cluster will automatically route traffic to targets in the healthy AZs. For those customers seeking the highest availability goals, in the case that an AZ becomes impaired, it can be important to be able to steer all traffic away from the impaired AZ until it recovers. For this, you can also *enable an ALB or NLB with ARC zonal shift*.

Understanding East-West Network Traffic Flow Between Pods

The following diagram illustrates two example workloads, Orders, and Products. The purpose of this example is to show how workloads and Pods in different AZs communicate.





1. For Orders to communicate with Products, it must first resolve the DNS name of the destination service. Orders will communicate with CoreDNS to fetch the virtual IP address (Cluster IP) for that Service. Once Orders resolves the Products service name, it sends traffic to that target IP.
2. The kube-proxy runs on every node in the cluster and continuously watches the EndpointSlices for Services. When a Service is created, an EndpointSlice is created and managed in the background by the EndpointSlice controller. Each EndpointSlice has a list or table of endpoints containing a subset of Pod addresses along with the nodes that they're running on. The kube-proxy sets up routing rules for each of these Pod endpoints using `iptables` on the nodes. The kube-proxy is also responsible for a basic form of load balancing by redirecting traffic destined to a service's Cluster IP to instead be sent to a Pod's IP address directly. The kube-proxy does this by rewriting the destination IP on the outgoing connection.
3. The network packets are then sent to the Products Pod in AZ 2 via the ENIs on the respective nodes (as depicted in the diagram above).

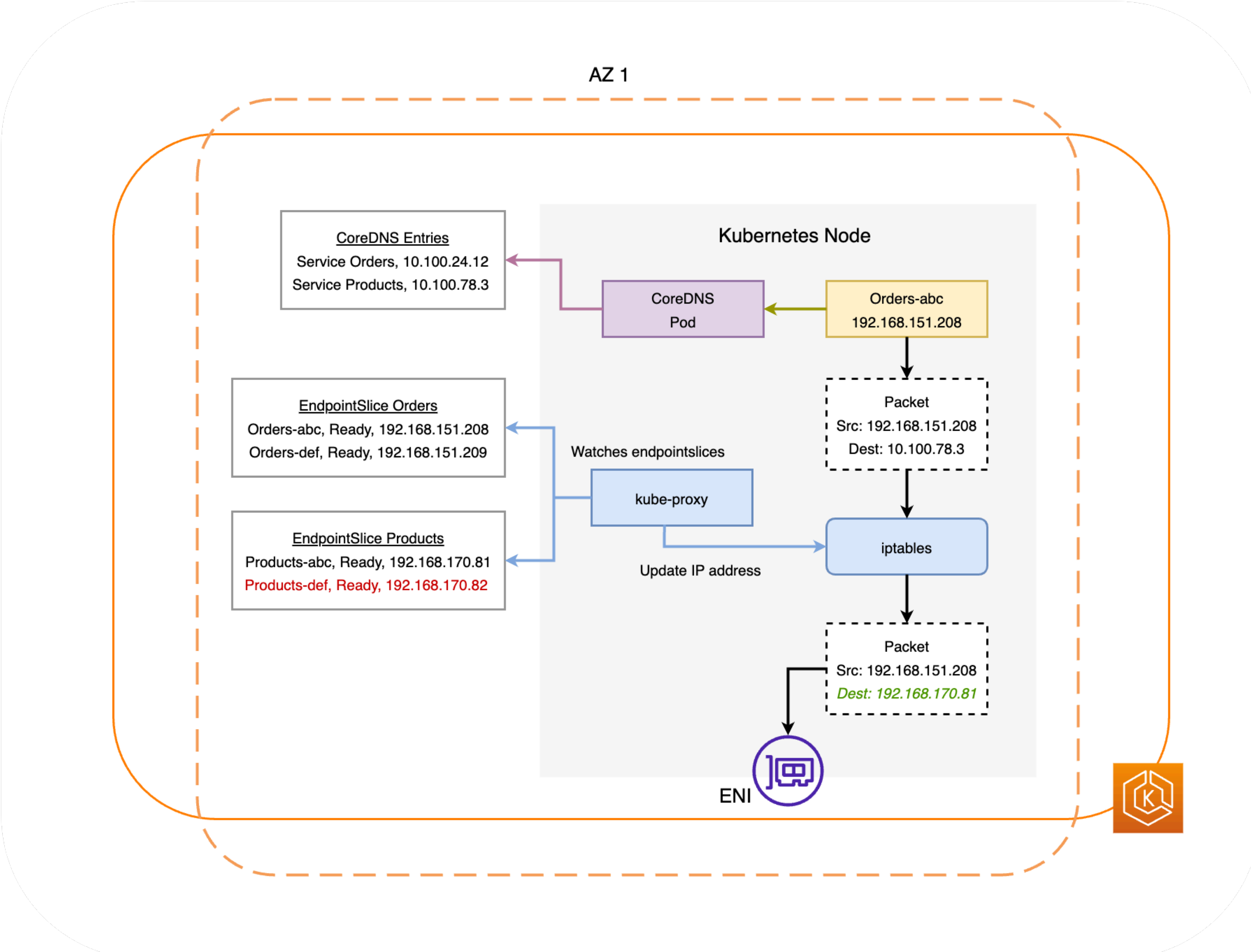
Understanding ARC Zonal Shift in Amazon EKS

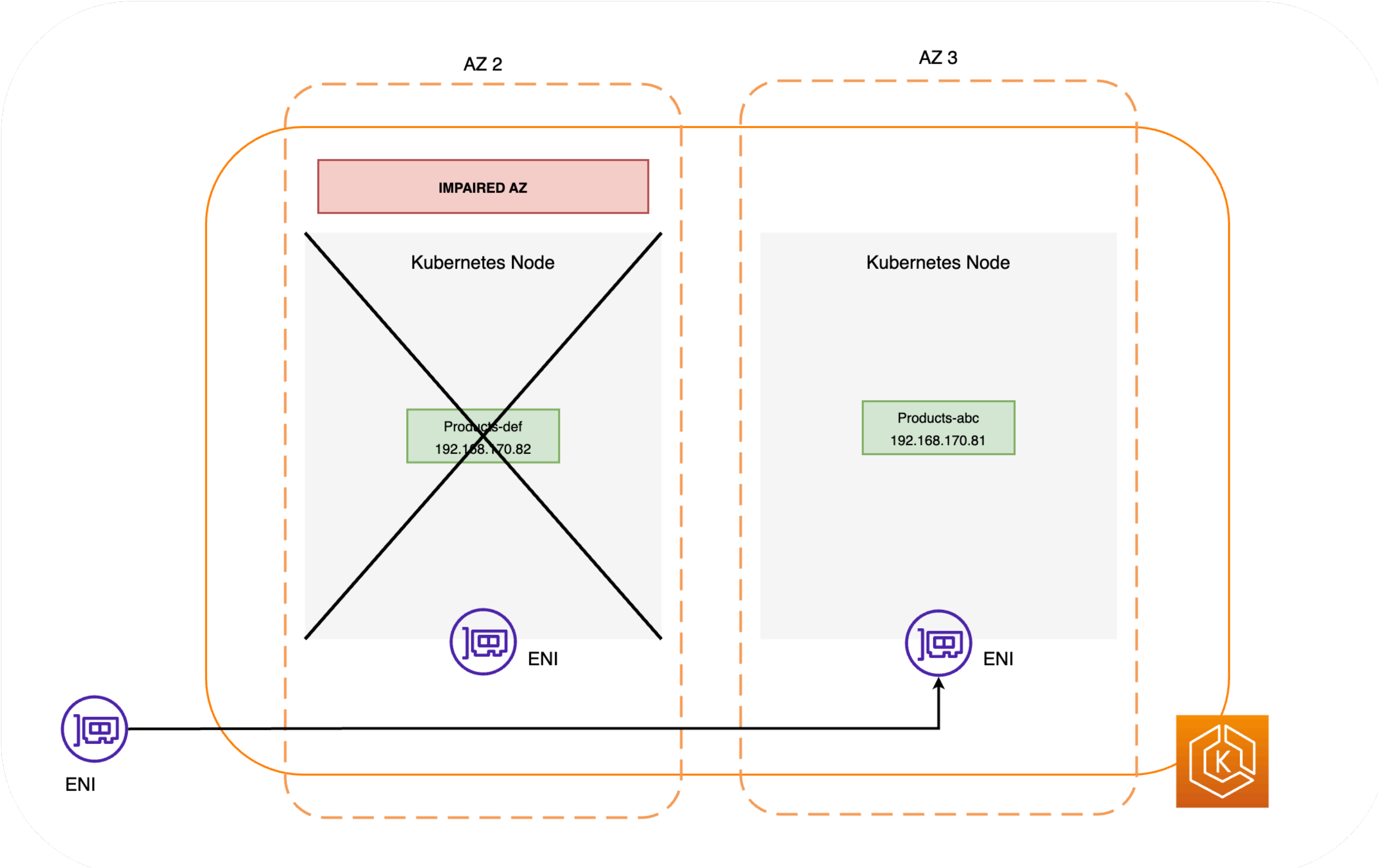
In the case that there is an AZ impairment in your environment, you can initiate a zonal shift for your EKS cluster environment. Alternatively, you can allow AWS to manage this for you with zonal autoshift. With zonal autoshift, AWS will monitor the overall AZ health and respond to a potential AZ impairment by automatically shifting traffic away from the impaired AZ in your cluster environment.

Once your Amazon EKS cluster has zonal shift enabled with ARC, you can trigger a zonal shift or enable zonal autoshift using the ARC Console, the AWS CLI, or the zonal shift and zonal autoshift APIs. During an EKS zonal shift, the following will automatically take place:

- All the nodes in the impacted AZ will be cordoned. This will prevent the Kubernetes Scheduler from scheduling new Pods onto the nodes in the unhealthy AZ.
- If you're using Managed Node Groups, *Availability Zone rebalancing* will be suspended, and your Auto Scaling Group (ASG) will be updated to ensure that new EKS Data Plane nodes are only launched in the healthy AZs.
- The nodes in the unhealthy AZ will not be terminated and the Pods will not be evicted from these nodes. This is to ensure that when a zonal shift expires or gets cancelled, your traffic can be safely returned to the AZ which still has full capacity
- The EndpointSlice controller will find all the Pod endpoints in the impaired AZ and remove them from the relevant EndpointSlices. This will ensure that only Pod endpoints in healthy AZs are targeted to receive network traffic. When a zonal shift is cancelled or expires, the EndpointSlice controller will update the EndpointSlices to include the endpoints in the restored AZ.

The diagrams below depicts a high level flow of how EKS zonal shift ensures that only healthy Pod endpoints are targeted in your cluster environment.


AZ 1
Kubernetes Node
CoreDNS Entries
Service Orders, 10.100.24.12
Service Products, 10.100.78.3
CoreDNS
Pod
Orders-abc
192.168.151.208
Packet
Src: 192.168.151.208
Dest: 10.100.78.3
EndpointSlice Orders
Orders-abc, Ready, 192.168.151.208
Orders-def, Ready, 192.168.151.209
Watches endpointslices
kube-proxy
iptables
EndpointSlice Products
Products-abc, Ready, 192.168.170.81
Products-def, Ready, 192.168.170.82
Update IP address
Packet
Src: 192.168.151.208
Dest: 192.168.170.81
ENI



EKS Zonal Shift Requirements

For zonal shift to work successfully in EKS, you need to setup your cluster environment to be resilient to an AZ impairment beforehand. Below is a list of the steps that you have to follow.

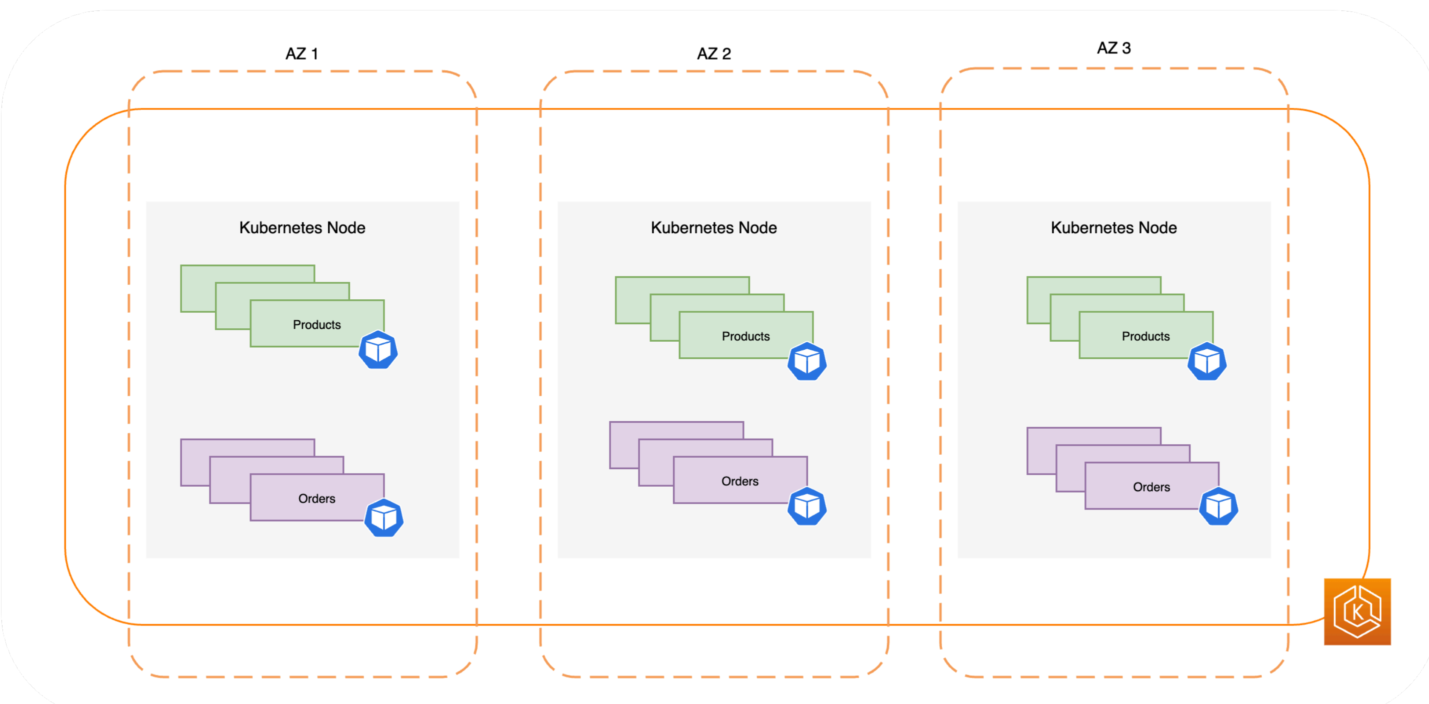
- Provision your cluster's worker nodes across multiple AZs
- Provision enough compute capacity to withstand removal of a single AZ
- Pre-scale your Pods (including CoreDNS) in every AZ
- Spread multiple Pod replicas across all AZs to ensure that shifting away from a single AZ will leave you with sufficient capacity
- Co-locate interdependent or related Pods in the same AZ
- Test that your cluster environment would work as expected with one less AZ by manually starting a zonal shift. Alternatively, you can enable zonal autoshift and reply on the autoshift practice runs. This is not required for zonal shift to work in EKS but it's strongly recommended.

Provision Your EKS Worker Nodes Across Multiple AZs

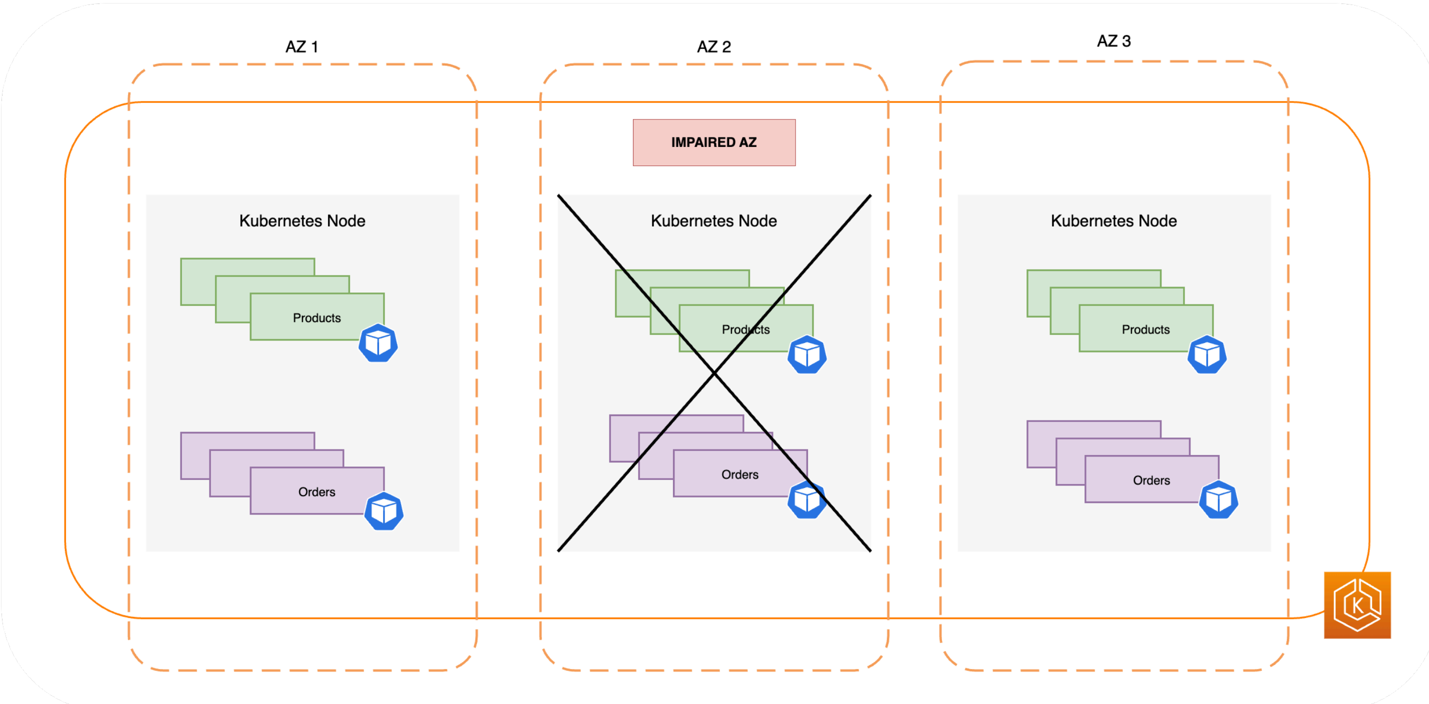
AWS Regions have multiple, separate locations with physical data centers known as Availability Zones (AZs). AZs are designed to be physically isolated from one another to avoid simultaneous impact that could affect an entire Region. When provisioning an EKS cluster, you should deploy your worker nodes across multiple AZs in a Region. This will make your cluster environment more resilient to the impairment of a single AZ, and allow you to maintain high availability (HA) of your applications running in the other AZs. When you start a zonal shift away from the impacted AZ, your EKS environment's in-cluster network will automatically update to only use healthy AZs, while maintaining a highly available posture for your cluster.

Ensuring that you have such a multi-AZ setup for your EKS environment will enhance the overall reliability of your system. However, multi-AZ environments can play a significant role in how application data is transferred and processed, which will in turn have an impact on your environment's network charges. In particular, frequent egress cross-zone traffic (traffic distributed between AZs) can have a major impact on your network-related costs. You can apply different strategies to control the amount of cross-zone traffic between Pods in your EKS cluster and drive down the associated costs. Please refer to *this best practice guide* for more details on how to optimize network costs when running highly available EKS environments.

The diagram below depicts a highly available EKS environment with 3 healthy AZs.



The diagram below depicts how an EKS environment with 3 AZs is resilient to an AZ impairment and remains highly available because of the 2 other healthy AZs.



Provision Enough Compute Capacity to Withstand Removal of a Single AZ

To optimize resource utilization and costs for your compute infrastructure in the EKS Data Plane, it's a best practice to align compute capacity with your workload requirements. However, **if all your worker nodes are at full capacity**, then this makes you reliant on having new worker nodes added to the EKS Data Plane before new Pods can be scheduled. When running critical workloads, it is generally always a good practice to run with redundant capacity online to handle eventualities such as sudden increases in load, node health issues, etc. If you plan to use zonal shift, you are planning to remove an entire AZ of capacity so you need to adjust your redundant compute capacity so that it's sufficient to handle the load even with an AZ offline.

When scaling your compute, the process of adding new nodes to the EKS Data Plane will take some time which can have implications on the real-time performance and availability of your applications, especially in the event of a zonal impairment. Your EKS environment should be resilient to absorb the load of losing an AZ to avoid a degraded experience for your end users or clients. This means minimizing or eliminating any lag between the time at which a new Pod is needed and when it's actually scheduled on a worker node.

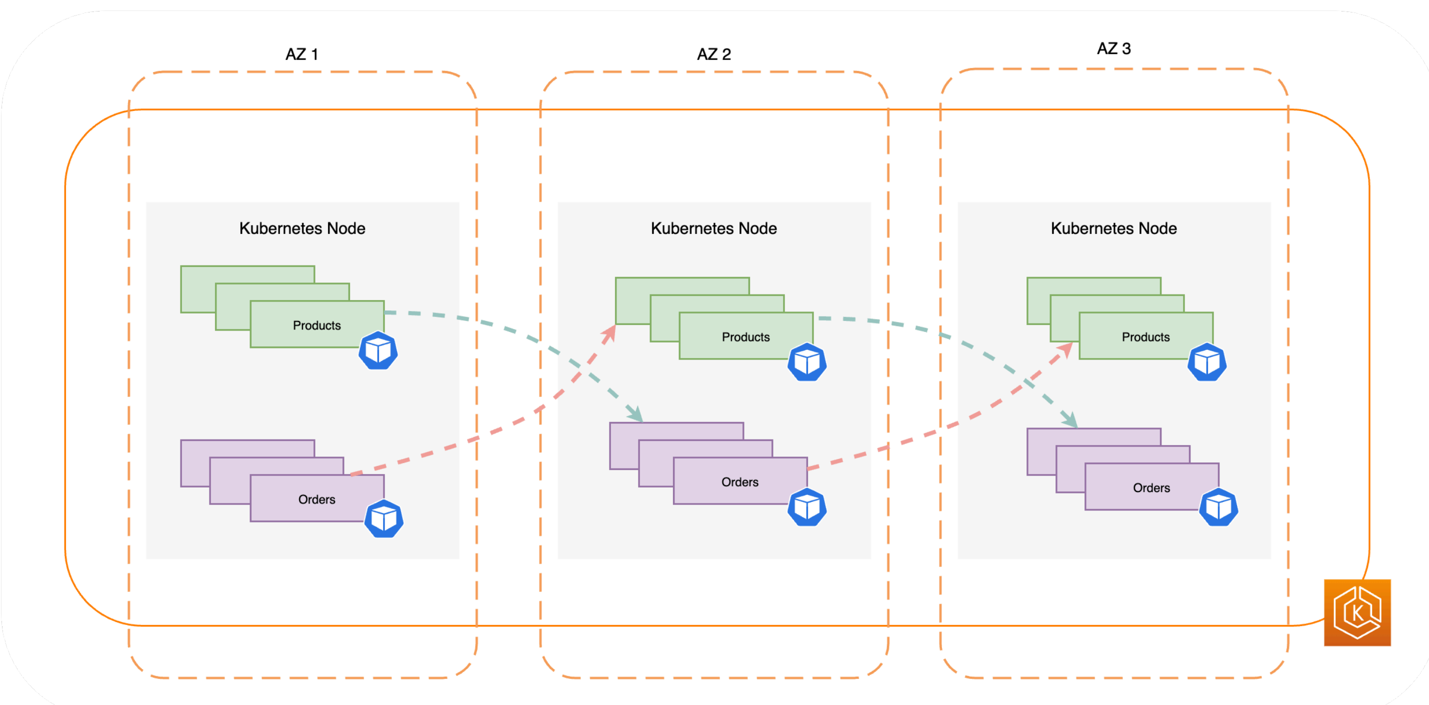
Additionally, in the event of a zonal impairment, you should mitigate the risk of a potential compute capacity constraint which would prevent newly required nodes from being added to your EKS Data Plane in the healthy AZs.

To accomplish this, you should over-provision compute capacity in some of the worker nodes in each of the AZs so that the Kubernetes Scheduler has pre-existing capacity available for new Pod placements, especially when you have one less AZ in your environment.

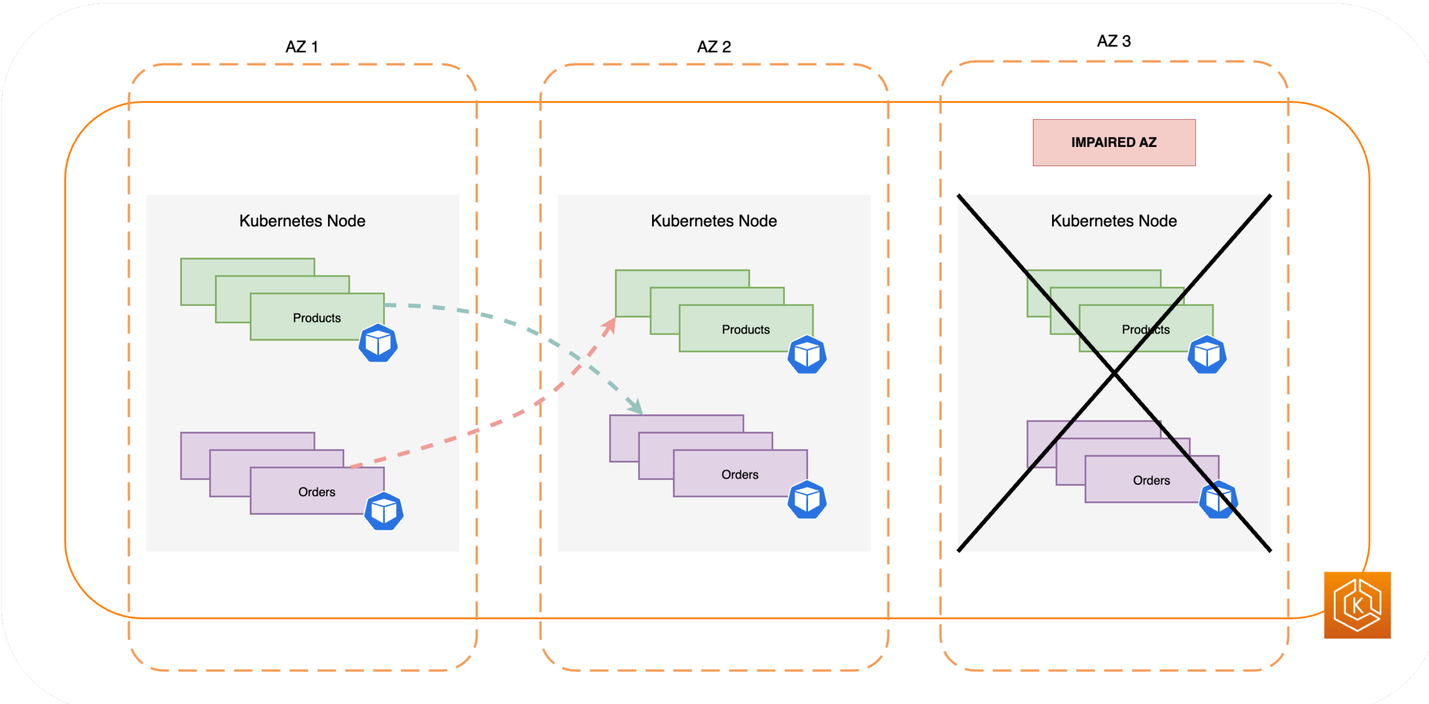
Run & Spread Multiple Pod Replicas Across AZs

Kubernetes allows you to pre-scale your workloads by running multiple instances (Pod replicas) of a single application. Running multiple Pod replicas for an application eliminates a single point of failure and increases its overall performance by reducing the resource strain on a single replica. However, to have both high availability and better fault tolerance for your applications, you should run and spread multiple replicas of an application across different failure domains (also referred to as topology domains) in this case AZs. With topology spread constraints, you can setup your applications to have pre-existing, static stability so that, in the case of an AZ impairment, you'll have enough replicas in the healthy AZs to immediately handle any additional spike or surge in traffic that they may experience.

The diagram below depicts an EKS environment with east-to-west traffic flow when all AZs are healthy.



The diagram below depicts an EKS environment with east-to-west traffic flow when a single AZ fails, and you initiate a zonal shift.



The code snippet below is an example of how to setup your workload with this Kubernetes feature.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orders
spec:
  replicas: 9
  selector:
    matchLabels:
      app:orders
  template:
    metadata:
      labels:
        app: orders
        tier: backend
    spec:
      topologySpreadConstraints:
      - maxSkew: 1
        topologyKey: "topology.kubernetes.io/zone"
        whenUnsatisfiable: ScheduleAnyway
        labelSelector:
```

```
          matchLabels:
            app: orders
```

Most important, you should run multiple replicas of your DNS server software (CoreDNS/kube-dns) and apply similar topology spread constraints if they are not already configured by default. This will help ensure that you have enough DNS Pods in healthy AZs to continue handling service discovery requests for other communicating Pods in the cluster if there's a single AZ impairment. The CoreDNS EKS add-on has default settings for the CoreDNS Pods to be spread across your cluster's Availability Zones if there are nodes in multiple AZs available. You can also replace these default settings with your own custom configurations.

When installing CoreDNS with Helm, you can update the `replicaCount` in the values.yaml file to ensure that you have a sufficient number of replicas in each AZ. In addition, to ensure that these replicas are spread across the different AZs in your cluster environment, you should update the `topologySpreadConstraints` property in the same values.yaml file. The code snippet below demonstrates how to configure CoreDNS for this.

CoreDNS Helm values.yaml

```
replicaCount: 6
topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: ScheduleAnyway
    labelSelector:
      matchLabels:
        k8s-app: kube-dns
```

In the event of an AZ impairment, you can absorb the increased load on the CoreDNS Pods by using an autoscaling system for CoreDNS. The number of DNS instances you require will depend on the number of workloads running in your cluster. CoreDNS is CPU bound which allows it to scale based on CPU using the Horizontal Pod Autoscaler (HPA). Below is an example that you can modify to suit your needs.

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: coredns
  namespace: default
spec:
```

```
  maxReplicas: 20
  minReplicas: 2
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: coredns
  targetCPUUtilizationPercentage: 50
```

Alternatively, EKS can manage the autoscaling of the CoreDNS Deployment in the EKS add-on version of CoreDNS. This CoreDNS autoscaler continuously monitors the cluster state, including the number of nodes and CPU cores. Based on that information, the controller will dynamically adapt the number of replicas of the CoreDNS deployment in an EKS cluster.

To enable the autoscaling configuration in the CoreDNS EKS add-on, you should add the following optional configuration settings:

```
{
  "autoScaling": {
    "enabled": true
  }
}
```

You can also use NodeLocal DNS or the cluster proportional autoscaler to scale CoreDNS. You can read further about scaling CoreDNS horizontally here.

Colocate Interdependent Pods in the Same AZ

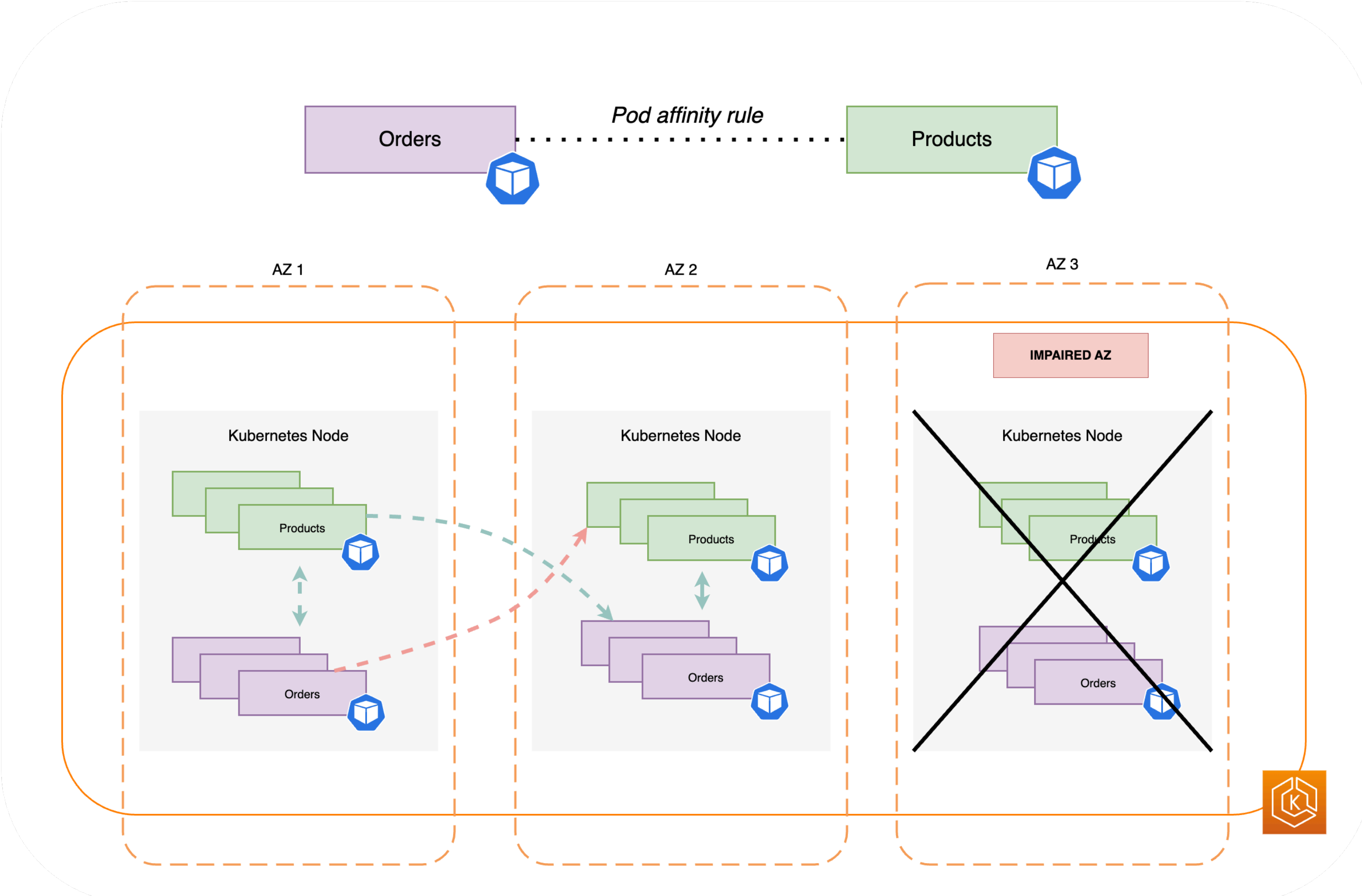
In most cases, you may be running distinct workloads that have to communicate with each other for successful execution of an end-to-end process. If the distinct applications are spread across different AZs but are not colocated in the same AZ, then a single AZ impairment may impact the underlying end-to-end process. For example, if **Application A** has multiple replicas in AZ 1 and AZ 2, but **Application B** has all its replicas in AZ 3, then the loss of AZ 3 will affect any end-to-end processes between these two workloads (**Application A and B**). Combining topology spread constraints with pod affinity can enhance your application's resiliency by spreading Pods across all AZs, as well as configuring a relationship between certain Pods to ensure that they're colocated together.

With pod affinity rules, you can define relationships between workloads to influence the behavior of the Kubernetes Scheduler so that it colocates Pods on the same worker node or in the same AZ. You can also configure how strict these scheduling constraints should be.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: products
  namespace: ecommerce
  labels:
    app.kubernetes.io/version: "0.1.6"

    spec:
      serviceAccountName: graphql-service-account
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - orders
            topologyKey: "kubernetes.io/hostname"
```

The diagram below depicts pods that have been co-located on the same node using pod affinity rules.



Test That Your Cluster Environment Can Handle The Loss of an AZ

After completing the above requirements, the next important step is to test that you have sufficient compute and workload capacity to handle the loss of an AZ. You can do this by manually triggering a zonal shift in EKS. Alternatively, you can enable zonal autoshift and configure practice runs to test that your applications function as expected with one less AZ in your cluster environment.

Frequently Asked Questions

Why should I use this feature?

By using ARC zonal shift or zonal autoshift in your EKS cluster, you can better maintain Kubernetes application availability by automating the quick recovery process of shifting in-cluster network traffic away from an impaired AZ. With ARC, you can avoid long and complicated steps which often lead to an extended recovery period during impaired AZ events.

How does this feature work with other AWS services?

EKS integrates with ARC which provides the primary interface for you to accomplish recovery operations in AWS. To ensure that in-cluster traffic is appropriately routed away from an impaired AZ, modifications are made to the list of network endpoints for Pods running in the Kubernetes data plane. If you’re using AWS Load Balancers for routing external traffic into the cluster, you can already register your load balancers with ARC and trigger a zonal shift on them to prevent traffic flowing into the degraded zone. This feature also interacts with Amazon EC2 Auto Scaling Groups (ASG) that are created by EKS Managed Node Groups (MNG). To prevent an impaired AZ from being used for new Kubernetes Pods or node launches, EKS removes the impaired AZ from the ASG.

How is this feature different from default Kubernetes protections?

This feature works in tandem with several Kubernetes native built-in protections that help customers stay resilient. You can configure Pod readiness and liveness probes that decide when a Pod should take traffic. When these probes fail, Kubernetes removes these Pods as targets for a Service and traffic is no longer sent to the Pod. While this is useful, it’s non-trivial for customers to configure these health checks so that they are guaranteed to fail when a zone is degraded. The ARC zonal shift feature provides you with an additional safety net that helps them isolate a degraded AZ entirely when Kubernetes' native protections have not sufficed. It also provides you with an easy way to test the operational readiness and resilience of your architecture.

Can AWS trigger a zonal shift on my behalf?

Yes, if you want a fully automated way of using ARC zonal shift, you can enable ARC zonal autoshift. With zonal autoshift, you can rely on AWS to monitor the health of the AZs for your EKS cluster, and to automatically trigger a shift when an AZ impairment is detected.

What happens if I use this feature and my worker nodes and workloads are not pre-scaled?

If you are not pre-scaled and rely on provisioning additional nodes or Pods during a zonal shift, then you risk experiencing a delayed recovery. The process of adding new nodes to the Kubernetes data plane will take some time which can have implications on the real-time performance and availability of your applications, especially in the event of a zonal impairment. Additionally, in the event of a zonal impairment, you may encounter a potential compute capacity constraint which would prevent newly required nodes from being added to the healthy AZs.

If your workloads are not pre-scaled and spread across all AZs in your cluster, a zonal impairment may impact the availability of an application that is only running on worker nodes in an impacted AZ. To mitigate the risk of a complete availability outage for your application, EKS has a fail safe for traffic to be sent to Pod endpoints in an impaired zone if that workload has all of its endpoints in the unhealthy AZ. However, it's strongly recommended that you rather pre-scale and spread your applications across all AZs to maintain availability in the event of a zonal issue.

What happens if I'm running a stateful application?

If you are running a stateful application, you will need to assess its fault tolerance depending on the use case and the architecture. If you have an active/standby architecture or pattern, there may be instances where the active is in an impaired AZ. At the application level, if the standby is not activated, you may run into issues with your application. You may also run into issues when new Kubernetes Pods are launched in healthy AZs since they will not be able to attach to the persistent volumes bounded to the impaired AZ.

Does this feature work with Karpenter?

Karpenter support is currently not available with ARC zonal shift and zonal autoshift in EKS. If an AZ is impaired, you can adjust the relevant Karpenter NodePool configuration by removing the unhealthy AZ so that new worker nodes are only launched in the healthy AZs.

Does this feature work with EKS Fargate?

This feature does not work with EKS Fargate. By default, when EKS Fargate recognizes a zonal health event, Pods will prefer to run in the other AZs.

Will the EKS managed Kubernetes control plane be impacted?

No, by default Amazon EKS runs and scales the Kubernetes control plane across multiple AZs to ensure high availability. ARC zonal shift and zonal autoshift will only act on the Kubernetes data plane.

Are there any costs associated with this new feature?

You can use ARC zonal shift and zonal autoshift in your EKS cluster at no additional charge. However, you will continue to pay for provisioned instances and it is strongly recommended that you pre-scale your Kubernetes data plane before using this feature. You should consider the right balance between cost and application availability.

Additional Resources

- How a zonal shift works
- Best practices for zonal shifts in ARC

- Resources and scenarios supported for zonal shift and zonal autoshift
- Operating resilient workloads on Amazon EKS
- Eliminate Kubernetes node scaling lag with pod priority and over-provisioning
- Scale CoreDNS Pods for high DNS traffic

Enable EKS Zonal Shift to avoid impaired Availability Zones

Amazon Application Recovery Controller (ARC) helps you manage and coordinate recovery for your applications across Availability Zones (AZs) and works with many services, including Amazon EKS. With EKS support for ARC zonal shift, you can shift in-cluster network traffic away from an impaired AZ. You can also authorize AWS to monitor the health of your AZs and temporarily shift network traffic away from an unhealthy AZ on your behalf.

How to use EKS Zonal Shift:

1. Enable your EKS cluster with Amazon Application Recovery Controller (ARC). This is done at the cluster level using the Amazon EKS Console, the AWS CLI, CloudFormation, or eksctl.
2. Once enabled, you can manage zonal shifts or zonal autoshifts using the ARC Console, the AWS CLI, or the Zonal Shift and Zonal Autoshift APIs.

Note that after you register an EKS cluster with ARC, you still need to configure ARC. For example, you can use the ARC console to configure Zonal Autoshift.

For more detailed information about how EKS Zonal Shift works, and how to design your workloads to handle impaired availability zones, see the section called “Learn about Zonal Shift”.

Considerations:

- EKS Auto Mode does not support Amazon Application Recovery Controller, Zonal Shift, and Zonal Autoshift

What is Amazon Application Recovery Controller?

Amazon Application Recovery Controller (ARC) helps you prepare for and accomplish faster recovery for applications running on AWS. Zonal shift enables you to quickly recover from Availability Zone (AZ) impairments, by temporarily moving traffic for a supported resource away from an AZ, to healthy AZs in the AWS Region.

Learn more about Amazon Application Recovery Controller (ARC)

What is zonal shift?

Zonal shift is a capability in ARC that allows you to move traffic for a resource like an EKS cluster or an Elastic Load Balancer away from an Availability Zone in an AWS Region to quickly mitigate an issue and quickly recover your application. You might choose to shift traffic, for example, because a bad deployment is causing latency issues, or because the Availability Zone is impaired. A zonal shift requires no advance configuration steps.

Learn more about ARC Zonal Shift

What is zonal autoshift?

Zonal autoshift is a capability in ARC that you can enable to authorize AWS to shift traffic away from an AZ for supported resources, on your behalf, to healthy AZs in the AWS Region. AWS starts an autoshift when internal telemetry indicates that there is an impairment in one AZ in a Region that could potentially impact customers. The internal telemetry incorporates metrics from multiple sources, including the AWS network, and the Amazon EC2 and Elastic Load Balancing services.

AWS ends autoshifts when indicators show that there is no longer an issue or potential issue.

Learn more about ARC Zonal Autoshift

What does EKS do during an autoshift?

EKS updates networking configurations to avoid directing traffic to impaired AZs. Additionally, if you are using Managed Node Groups, EKS will only launch new nodes in the healthy AZs during a zonal shift. When the shift expires or gets cancelled, the networking configurations will be restored to include the AZ that was previously detected as unhealthy.

Learn more about EKS Zonal Shift.

Register EKS cluster with Amazon Application Recovery Controller (ARC) (AWS console)

1. Find the name and region of the EKS cluster you want to register with ARC.
2. Navigate to the EKS console in that region, and select your cluster.
3. On the **Cluster info** page, select the **Overview** tab.

4. Under the **Zonal shift** heading, select the **Manage** button.
5. Select **enable** or **disable** for *EKS Zonal Shift*.

Now your EKS cluster is registered with ARC.

If you want AWS to detect and avoid impaired availability zones, you need to configure ARC Zonal Autoshift. For example, you can do this in the ARC console.

Next Steps

- Learn how to enable zonal autoshift
- Learn how to manually start a zonal shift

Learn how access control works in Amazon EKS

Learn how to manage access to your Amazon EKS cluster. Using Amazon EKS requires knowledge of how both Kubernetes and AWS Identity and Access Management (AWS IAM) handle access control.

This section includes:

the section called "Kubernetes API access" — Learn how to enable applications or users to authenticate to the Kubernetes API. You can use access entries, the aws-auth ConfigMap, or an external OIDC provider.

the section called "Access cluster resources" — Learn how to configure the AWS Management Console to communicate with your Amazon EKS cluster. Use the console to view Kubernetes resources in the cluster, such as namespaces, nodes, and Pods.

the section called "Access cluster with kubectl" — Learn how to configure kubectl to communicate with your Amazon EKS cluster. Use the AWS CLI to create a kubeconfig file.

the section called "Workload access to AWS " — Learn how to associate a Kubernetes service account with AWS IAM Roles. You can use Pod Identity or IAM Roles for Service Accounts (IRSA).

Common Tasks

- Grant developers access to the Kubernetes API. View Kubernetes resources in the AWS Management Console.
  - Solution: Use access entries to associate Kubernetes RBAC permissions with AWS IAM Users or Roles.
- Configure kubectl to talk to an Amazon EKS cluster using AWS Credentials.
  - Solution: Use the AWS CLI to create a kubeconfig file.
- Use an external identity provider, such as Ping Identity, to authenticate users to the Kubernetes API.
  - Solution: Link an external OIDC provider.
- Grant workloads on your Kubernetes cluster the ability to call AWS APIs.
  - Solution: Use Pod Identity to associate an AWS IAM Role to a Kubernetes Service Account.

Background

- Learn how Kubernetes Service Accounts work.
- Review the Kubernetes Role Based Access Control (RBAC) Model
- For more information about managing access to AWS resources, see the AWS IAM User Guide. Alternatively, take a free introductory training on using AWS IAM.

Considerations for EKS Auto Mode

EKS Auto Mode integrates with EKS Pod Identity and EKS EKS access entries.

- EKS Auto Mode uses access entries to grant the EKS control plane Kubernetes permissions. For example, the access policies enable EKS Auto Mode to read information about network endpoints and services.
  - You cannot disable access entries on an EKS Auto Mode cluster.
  - You can optionally enable the `aws-auth` `ConfigMap`.
  - The access entries for EKS Auto Mode are automatically configured. You can view these access entries, but you cannot modify them.
  - If you use a NodeClass to create a custom Node IAM Role, you need to create an access entry for the role using the AmazonEKSAutoNodePolicy access policy.
- If you want to grant workloads permissions for AWS services, use EKS Pod Identity.
  - You do not need to install the Pod Identity agent on EKS Auto Mode clusters.

Grant IAM users and roles access to Kubernetes APIs

Your cluster has an Kubernetes API endpoint. Kubectl uses this API. You can authenticate to this API using two types of identities:

- **An AWS Identity and Access Management (IAM) *principal* (role or user)** – This type requires authentication to IAM. Users can sign in to AWS as an IAM user or with a federated identity by using credentials provided through an identity source. Users can only sign in with a federated identity if your administrator previously set up identity federation using IAM roles. When users access AWS by using federation, they're indirectly assuming a role. When users use this type of identity, you:

  - Can assign them Kubernetes permissions so that they can work with Kubernetes objects on your cluster. For more information about how to assign permissions to your IAM principals so that they're able to access Kubernetes objects on your cluster, see the section called "Grant permissions".
  - Can assign them IAM permissions so that they can work with your Amazon EKS cluster and its resources using the Amazon EKS API, AWS CLI, AWS CloudFormation, AWS Management Console, or `eksctl`. For more information, see Actions defined by Amazon Elastic Kubernetes Service in the Service Authorization Reference.
  - Nodes join your cluster by assuming an IAM role. The ability to access your cluster using IAM principals is provided by the AWS IAM Authenticator for Kubernetes, which runs on the Amazon EKS control plane.
- **A user in your own OpenID Connect (OIDC) provider** – This type requires authentication to your OIDC provider. For more information about setting up your own OIDC provider with your Amazon EKS cluster, see the section called "Link OIDC provider". When users use this type of identity, you:
  - Can assign them Kubernetes permissions so that they can work with Kubernetes objects on your cluster.
  - Can't assign them IAM permissions so that they can work with your Amazon EKS cluster and its resources using the Amazon EKS API, AWS CLI, AWS CloudFormation, AWS Management Console, or `eksctl`.

You can use both types of identities with your cluster. The IAM authentication method cannot be disabled. The OIDC authentication method is optional.

Associate IAM Identities with Kubernetes Permissions

The AWS IAM Authenticator for Kubernetes is installed on your cluster's control plane. It enables AWS Identity and Access Management (IAM) principals (roles and users) that you allow to access Kubernetes resources on your cluster. You can allow IAM principals to access Kubernetes objects on your cluster using one of the following methods:

- **Creating access entries** – If your cluster is at or later than the platform version listed in the Prerequisites section for your cluster's Kubernetes version, we recommend that you use this option.

Use *access entries* to manage the Kubernetes permissions of IAM principals from outside the cluster. You can add and manage access to the cluster by using the EKS API, AWS Command Line Interface, AWS SDKs, AWS CloudFormation, and AWS Management Console. This means you can manage users with the same tools that you created the cluster with.

To get started, follow Change authentication mode to use access entries, then Migrating existing aws-auth ConfigMap entries to access entries.

- **Adding entries to the `aws-auth` `ConfigMap`** – If your cluster's platform version is earlier than the version listed in the Prerequisites section, then you must use this option. If your cluster's platform version is at or later than the platform version listed in the Prerequisites section for your cluster's Kubernetes version, and you've added entries to the `ConfigMap`, then we recommend that you migrate those entries to access entries. You can't migrate entries that Amazon EKS added to the `ConfigMap` however, such as entries for IAM roles used with managed node groups or Fargate profiles. For more information, see the section called "Kubernetes API access".
  - If you have to use the `aws-auth` `ConfigMap` option, you can add entries to the `ConfigMap` using the `eksctl create iamidentitymapping` command. For more information, see Manage IAM users and roles in the `eksctl` documentation.

Set Cluster Authentication Mode

Each cluster has an *authentication mode*. The authentication mode determines which methods you can use to allow IAM principals to access Kubernetes objects on your cluster. There are three authentication modes.

> **Important**
>
> Once the access entry method is enabled, it cannot be disabled.
> If the `ConfigMap` method is not enabled during cluster creation, it cannot be enabled later.
> All clusters created before the introduction of access entries have the `ConfigMap` method enabled.
> If you are using hybrid nodes with your cluster, you must use the `API` or `API_AND_CONFIG_MAP` cluster authentication modes.

The `aws-auth` `ConfigMap` inside the cluster

This is the original authentication mode for Amazon EKS clusters. The IAM principal that created the cluster is the initial user that can access the cluster by using `kubectl`. The initial user must add other users to the list in the `aws-auth` `ConfigMap` and assign permissions that affect the other users within the cluster. These other users can't manage or remove the initial user, as there isn't an entry in the `ConfigMap` to manage.

Both the `ConfigMap` and access entries

With this authentication mode, you can use both methods to add IAM principals to the cluster. Note that each method stores separate entries; for example, if you add an access entry from the AWS CLI, the `aws-auth` `ConfigMap` is not updated.

Access entries only

With this authentication mode, you can use the EKS API, AWS Command Line Interface, AWS SDKs, AWS CloudFormation, and AWS Management Console to manage access to the cluster for IAM principals.

Each access entry has a *type* and you can use the combination of an *access scope* to limit the principal to a specific namespace and an *access policy* to set preconfigured reusable permissions policies. Alternatively, you can use the STANDARD type and Kubernetes RBAC groups to assign custom permissions.

Authentication mode	Methods
`ConfigMap` only (`CONFIG_MAP`)	`aws-auth` `ConfigMap`
EKS API and `ConfigMap` (`API_AND_CONFIG_MAP`)	access entries in the EKS API, AWS Command Line Interface, AWS SDKs, AWS CloudFormation, and AWS Management Console and `aws-auth` `ConfigMap`
EKS API only (`API`)	access entries in the EKS API, AWS Command Line Interface, AWS SDKs, AWS CloudFormation, and AWS Management Console

Note

Amazon EKS Auto Mode requires Access entries.

Grant IAM users access to Kubernetes with EKS access entries

What is EKS access entries?

EKS access entries is the best way to grant users access to the Kubernetes API. For example, you can use access entries to grant developers access to use kubectl.

Fundamentally, an EKS access entry associates a set of Kubernetes permissions with an IAM identity, such as an IAM role. For example, a developer may assume an IAM role and use that to authenticate to an EKS Cluster.

You can attach Kubernetes permissions to access entries in two ways:

- Use an access policy. Access policies are pre-defined Kubernetes permissions templates maintained by AWS. For more information, see the section called “Review access policies”.
- Reference a Kubernetes group. If you associate an IAM Identity with a Kubernetes group, you can create Kubernetes resources that grant the group permissions. For more information, see Using RBAC Authorization in the Kubernetes documentation.

Advantages

Amazon EKS cluster access management enables you to control authentication and authorization for your Kubernetes clusters directly through Amazon EKS APIs. This feature simplifies access management by eliminating the need to switch between AWS and Kubernetes APIs when managing user permissions. Using access entries and access policies, you can define granular permissions for AWS IAM principals, including the ability to modify or revoke cluster-admin permissions from the cluster creator.

The feature integrates with infrastructure as code (IaC) tools like AWS CloudFormation, Terraform, and AWS CDK, allowing you to define access configurations during cluster creation. If misconfigurations occur, you can restore cluster access through the Amazon EKS API without requiring direct Kubernetes API access. This centralized approach reduces operational overhead and improves security by leveraging existing AWS IAM capabilities such as CloudTrail audit logging and multi-factor authentication.

Get Started

1. Determine the IAM Identity and Access policy you want to use.
   - the section called "Review access policies"
2. Enable EKS Access Entries on your cluster. Confirm you have a supported platform version.
   - the section called "Authentication mode"
3. Create an access entry that associates an IAM Identity with Kubernetes permission.
   - the section called "Create access entries"
4. Authenticate to the cluster using the IAM identity.
   - the section called "Set up AWS CLI"
   - the section called "Set up kubectl and eksctl"

Legacy cluster access configuration

When you enable EKS access entries on clusters created before this feature was introduced (clusters with initial platform versions earlier than those specified in Platform Version Requirements), EKS automatically creates an access entry that reflects pre-existing permissions. This access entry shows:

- The IAM identity that originally created the cluster
- The administrative permissions granted to that identity during cluster creation

> **Note**
>
> Previously, this administrative access was granted automatically and couldn't be modified. With EKS access entries enabled, you can now view and delete this legacy access configuration.

Associate access policies with access entries

You can assign one or more access policies to *access entries* of *type* `STANDARD`. Amazon EKS automatically grants the other types of access entries the permissions required to function properly in your cluster. Amazon EKS access policies include Kubernetes permissions, not IAM permissions. Before associating an access policy to an access entry, make sure that you're familiar with the

Kubernetes permissions included in each access policy. For more information, see the section called "Review access policies". If none of the access policies meet your requirements, then don't associate an access policy to an access entry. Instead, specify one or more *group names* for the access entry and create and manage Kubernetes role-based access control objects. For more information, see the section called "Create access entries".

- An existing access entry. To create one, see the section called "Create access entries".
- An AWS Identity and Access Management role or user with the following permissions: `ListAccessEntries`, `DescribeAccessEntry`, `UpdateAccessEntry`, `ListAccessPolicies`, `AssociateAccessPolicy`, and `DisassociateAccessPolicy`. For more information, see Actions defined by Amazon Elastic Kubernetes Service in the *Service Authorization Reference*.

Before associating access policies with access entries, consider the following requirements:

- You can associate multiple access policies to each access entry, but you can only associate each policy to an access entry once. If you associate multiple access policies, the access entry's IAM principal has all permissions included in all associated access policies.
- You can scope an access policy to all resources on a cluster or by specifying the name of one or more Kubernetes namespaces. You can use wildcard characters for a namespace name. For example, if you want to scope an access policy to all namespaces that start with `dev-`, you can specify `dev-*` as a namespace name. Make sure that the namespaces exist on your cluster and that your spelling matches the actual namespace name on the cluster. Amazon EKS doesn't confirm the spelling or existence of the namespaces on your cluster.
- You can change the *access scope* for an access policy after you associate it to an access entry. If you've scoped the access policy to Kubernetes namespaces, you can add and remove namespaces for the association, as necessary.
- If you associate an access policy to an access entry that also has *group names* specified, then the IAM principal has all the permissions in all associated access policies. It also has all the permissions in any Kubernetes `Role` or `ClusterRole` object that is specified in any Kubernetes `Role` and `RoleBinding` objects that specify the group names.
- If you run the `kubectl auth can-i --list` command, you won't see any Kubernetes permissions assigned by access policies associated with an access entry for the IAM principal you're using when you run the command. The command only shows Kubernetes permissions if you've granted them in Kubernetes `Role` or `ClusterRole` objects that you've bound to the group names or username that you specified for an access entry.

- If you impersonate a Kubernetes user or group when interacting with Kubernetes objects on your cluster, such as using the `kubectl` command with `--as username` or `--as-group group-name` , you're forcing the use of Kubernetes RBAC authorization. As a result, the IAM principal has no permissions assigned by any access policies associated to the access entry. The only Kubernetes permissions that the user or group that the IAM principal is impersonating has are the Kubernetes permissions that you've granted them in Kubernetes `Role` or `ClusterRole` objects that you've bound to the group names or user name. For your IAM principal to have the permissions in associated access policies, don't impersonate a Kubernetes user or group. The IAM principal will still also have any permissions that you've granted them in the Kubernetes `Role` or `ClusterRole` objects that you've bound to the group names or user name that you specified for the access entry. For more information, see User impersonation in the Kubernetes documentation.

You can associate an access policy to an access entry using the AWS Management Console or the AWS CLI.

AWS Management Console

1. Open the Amazon EKS console.
2. Choose the name of the cluster that has an access entry that you want to associate an access policy to.
3. Choose the **Access** tab.
4. If the type of the access entry is **Standard**, you can associate or disassociate Amazon EKS **access policies**. If the type of your access entry is anything other than **Standard**, then this option isn't available.
5. Choose **Associate access policy**.
6. For **Policy name**, select the policy with the permissions you want the IAM principal to have. To view the permissions included in each policy, see the section called "Review access policies".
7. For **Access scope**, choose an access scope. If you choose **Cluster**, the permissions in the access policy are granted to the IAM principal for resources in all Kubernetes namespaces. If you choose **Kubernetes namespace**, you can then choose **Add new namespace**. In the **Namespace** field that appears, you can enter the name of a Kubernetes namespace on your cluster. If you want the IAM principal to have the permissions across multiple namespaces, then you can enter multiple namespaces.
8. Choose **Add access policy**.

AWS CLI

1. Version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version, use `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*. The AWS CLI version that is installed in AWS CloudShell might also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the *AWS CloudShell User Guide*.
2. View the available access policies.

```
aws eks list-access-policies --output table
```

An example output is as follows.

```
---------------------------------------------------------------------------------------------------------
|                                          ListAccessPolicies                                          
                   |
+-------------------------------------------------------------------------------------------------------
+
||                                          accessPolicies                                          
                  ||
|+---------------------------------------------------------------------
+--------------------------------+|
||                                   arn                                   |
 name                  ||
|+---------------------------------------------------------------------
+--------------------------------+|
||   {arn-aws}eks::aws:cluster-access-policy/AmazonEKSAdminPolicy         |
 AmazonEKSAdminPolicy           ||
||   {arn-aws}eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy  |
 AmazonEKSClusterAdminPolicy    ||
||   {arn-aws}eks::aws:cluster-access-policy/AmazonEKSEditPolicy          |
 AmazonEKSEditPolicy            ||
||   {arn-aws}eks::aws:cluster-access-policy/AmazonEKSViewPolicy          |
 AmazonEKSViewPolicy            ||
|+---------------------------------------------------------------------
+--------------------------------+|
```

To view the permissions included in each policy, see the section called "Review access policies".

3. View your existing access entries. Replace *my-cluster* with the name of your cluster.

```
aws eks list-access-entries --cluster-name my-cluster
```

An example output is as follows.

```
{
    "accessEntries": [
        "arn:aws:iam::111122223333:role/my-role",
        "arn:aws:iam::111122223333:user/my-user"
    ]
}
```

4. Associate an access policy to an access entry. The following example associates the `AmazonEKSViewPolicy` access policy to an access entry. Whenever the *my-role* IAM role attempts to access Kubernetes objects on the cluster, Amazon EKS will authorize the role to use the permissions in the policy to access Kubernetes objects in the *my-namespace1* and *my-namespace2* Kubernetes namespaces only. Replace *my-cluster* with the name of your cluster, *111122223333* with your AWS account ID, and *my-role* with the name of the IAM role that you want Amazon EKS to authorize access to Kubernetes cluster objects for.

```
aws eks associate-access-policy --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/my-role \
    --access-scope type=namespace,namespaces=my-namespace1,my-namespace2 --policy-arn
 arn:aws:eks::aws:cluster-access-policy/AmazonEKSViewPolicy
```

If you want the IAM principal to have the permissions cluster-wide, replace `type=namespace,namespaces=`*`my-namespace1`*`,`*`my-namespace2`* with `type=cluster`. If you want to associate multiple access policies to the access entry, run the command multiple times, each with a unique access policy. Each associated access policy has its own scope.

> **Note**
>
> If you later want to change the scope of an associated access policy, run the previous command again with the new scope. For example, if you wanted to remove *my-namespace2*, you'd run the command again using `type=namespace,namespaces=`*`my-namespace1`* only. If you wanted to change

the scope from `namespace` to `cluster`, you'd run the command again using `type=cluster`, removing `type=namespace,namespaces=my-namespace1,my-namespace2` .

5. Determine which access policies are associated to an access entry.

```
aws eks list-associated-access-policies --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/my-role
```

An example output is as follows.

```
{
    "clusterName": "my-cluster",
    "principalArn": "arn:aws:iam::111122223333",
    "associatedAccessPolicies": [
        {
            "policyArn": "arn:aws:eks::aws:cluster-access-policy/
AmazonEKSViewPolicy",
            "accessScope": {
                "type": "cluster",
                "namespaces": []
            },
            "associatedAt": "2023-04-17T15:25:21.675000-04:00",
            "modifiedAt": "2023-04-17T15:25:21.675000-04:00"
        },
        {
            "policyArn": "arn:aws:eks::aws:cluster-access-policy/
AmazonEKSAdminPolicy",
            "accessScope": {
                "type": "namespace",
                "namespaces": [
                    "my-namespace1",
                    "my-namespace2"
                ]
            },
            "associatedAt": "2023-04-17T15:02:06.511000-04:00",
            "modifiedAt": "2023-04-17T15:02:06.511000-04:00"
        }
    ]
}
```

In the previous example, the IAM principal for this access entry has view permissions across all namespaces on the cluster, and administrator permissions to two Kubernetes namespaces.

6. Disassociate an access policy from an access entry. In this example, the `AmazonEKSAdminPolicy` policy is disassociated from an access entry. The IAM principal retains the permissions in the `AmazonEKSViewPolicy` access policy for objects in the *my-namespace1* and *my-namespace2* namespaces however, because that access policy is not disassociated from the access entry.

```
aws eks disassociate-access-policy --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/my-role \
    --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy
```

To list available access policies, see the section called "Review access policies".

Migrating existing `aws-auth ConfigMap` entries to access entries

If you've added entries to the `aws-auth` `ConfigMap` on your cluster, we recommend that you create access entries for the existing entries in your `aws-auth` `ConfigMap`. After creating the access entries, you can remove the entries from your `ConfigMap`. You can't associate access policies to entries in the `aws-auth` `ConfigMap`. If you want to associate access polices to your IAM principals, create access entries.

> **Important**
>
> Don't remove existing `aws-auth` `ConfigMap` entries that were created by Amazon EKS when you added a managed node group or a Fargate profile to your cluster. If you remove entries that Amazon EKS created in the `ConfigMap`, your cluster won't function properly. You can however, remove any entries for self-managed node groups after you've created access entries for them.

Prerequisites

- Familiarity with access entries and access policies. For more information, see the section called "Grant permissions" and the section called "Associate access policies".
- An existing cluster with a platform version that is at or later than the versions listed in the Prerequisites of the the section called "Grant permissions" topic.

- Version `0.204.0` or later of the `eksctl` command line tool installed on your device or AWS CloudShell. To install or update `eksctl`, see [Installation](#) in the `eksctl` documentation.
- Kubernetes permissions to modify the `aws-auth` `ConfigMap` in the `kube-system` namespace.
- An AWS Identity and Access Management role or user with the following permissions: `CreateAccessEntry` and `ListAccessEntries`. For more information, see [Actions defined by Amazon Elastic Kubernetes Service](#) in the Service Authorization Reference.

eksctl

1. View the existing entries in your `aws-auth ConfigMap`. Replace *`my-cluster`* with the name of your cluster.

   ```
   eksctl get iamidentitymapping --cluster my-cluster
   ```

   An example output is as follows.

   ```
   ARN
               USERNAME                                GROUPS
                           ACCOUNT
   arn:aws:iam::111122223333:role/EKS-my-cluster-Admins
               Admins                                  system:masters
   arn:aws:iam::111122223333:role/EKS-my-cluster-my-namespace-Viewers
               my-namespace-Viewers                    Viewers
   arn:aws:iam::111122223333:role/EKS-my-cluster-self-managed-ng-1
               system:node:{{EC2PrivateDNSName}}       system:bootstrappers,system:nodes
   arn:aws:iam::111122223333:user/my-user
               my-user
   arn:aws:iam::111122223333:role/EKS-my-cluster-fargateprofile1
                                    system:node:{{SessionName}}
    system:bootstrappers,system:nodes,system:node-proxier
   arn:aws:iam::111122223333:role/EKS-my-cluster-managed-ng
               system:node:{{EC2PrivateDNSName}}       system:bootstrappers,system:nodes
   ```

2. [the section called “Create access entries”](#) for any of the `ConfigMap` entries that you created returned in the previous output. When creating the access entries, make sure to specify the same values for `ARN`, `USERNAME`, `GROUPS`, and `ACCOUNT` returned in your output. In the example output, you would create access entries for all entries except the last two entries, since those entries were created by Amazon EKS for a Fargate profile and a managed node group.

3. Delete the entries from the ConfigMap for any access entries that you created. If you don't delete the entry from the ConfigMap, the settings for the access entry for the IAM principal ARN override the ConfigMap entry. Replace *111122223333* with your AWS account ID and *EKS-my-cluster-my-namespace-Viewers* with the name of the role in the entry in your ConfigMap. If the entry you're removing is for an IAM user, rather than an IAM role, replace role with user and *EKS-my-cluster-my-namespace-Viewers* with the user name.

```
eksctl delete iamidentitymapping --arn arn:aws:iam::111122223333:role/EKS-my-cluster-my-namespace-Viewers --cluster my-cluster
```

Review access policy permissions

Access policies include rules that contain Kubernetes verbs (permissions) and resources. Access policies don't include IAM permissions or resources. Similar to Kubernetes Role and ClusterRole objects, access policies only include allow rules. You can't modify the contents of an access policy. You can't create your own access policies. If the permissions in the access policies don't meet your needs, then create Kubernetes RBAC objects and specify *group names* for your access entries. For more information, see the section called "Create access entries". The permissions contained in access policies are similar to the permissions in the Kubernetes user-facing cluster roles. For more information, see User-facing roles in the Kubernetes documentation.

Choose any access policy to see its contents. Each row of each table in each access policy is a separate rule.

AmazonEKSAdminPolicy

This access policy includes permissions that grant an IAM principal most permissions to resources. When associated to an access entry, its access scope is typically one or more Kubernetes namespaces. If you want an IAM principal to have administrator access to all resources on your cluster, associate the the section called "AmazonEKSClusterAdminPolicy" access policy to your access entry instead.

ARN – arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy

Kubernetes API groups	Kubernetes resources	Kubernetes verbs (permissions)
apps	daemonsets , deployments , deployments/rollback , deployments/scale , replicasets , replicasets/scale, statefulsets , statefulsets/scale	create, delete, deletecollection , patch, update
apps	controllerrevisions , daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , replicasets , replicasets/scale , replicasets/status , statefulsets , statefulsets/scale , statefulsets/status	get, list, watch
authorization.k8s.io	localsubjectaccessreviews	create
autoscaling	horizontalpodautoscalers	create, delete, deletecollection , patch, update
autoscaling	horizontalpodautoscalers , horizontalpodautoscalers/status	get, list, watch
batch	cronjobs, jobs	create, delete, deletecollection , patch, update

Kubernetes API groups	Kubernetes resources	Kubernetes verbs (permissions)
batch	cronjobs, cronjobs/status , jobs, jobs/status	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch
extensions	daemonsets , deployments , deployments/rollback , deployments/scale , ingresses , networkpolicies , replicasets , replicasets/scale , replicationcontrollers/scale	create, delete, deletecollection , patch, update
extensions	daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , ingresses , ingresses/status , networkpolicies , replicasets , replicasets/scale , replicasets/status , replicationcontrollers/scale	get, list, watch
networking.k8s.io	ingresses , ingresses/status , networkpolicies	get, list, watch
networking.k8s.io	ingresses , networkpolicies	create, delete, deletecollection , patch, update

Kubernetes API groups	Kubernetes resources	Kubernetes verbs (permissions)
policy	poddisruptionbudgets	create, delete, deletecollection , patch, update
policy	poddisruptionbudgets , poddisruptionbudgets/status	get, list, watch
rbac.authorization.k8s.io	rolebindings , roles	create, delete, deletecollection , get, list, patch, update, watch
	configmaps , endpoints , persistentvolumeclaims , persistentvolumeclaims/status , pods, replicationcontrollers , replicationcontrollers/scale , serviceaccounts , services, services/status	get,list, watch
	pods/attach , pods/exec , pods/portforward , pods/proxy , secrets, services/proxy	get, list, watch

Kubernetes API groups	Kubernetes resources	Kubernetes verbs (permissions)
	configmaps , events, persistentvolumeclaims , replicationcontrollers , replicationcontrollers/scale , secrets, serviceaccounts , services, services/proxy	create, delete, deletecollection , patch, update
	pods, pods/attach , pods/exec , pods/portforward , pods/proxy	create, delete, deletecollection , patch, update
	serviceaccounts	impersonate
	bindings, events, limitranges , namespaces/status , pods/log, pods/status , replicationcontrollers/status , resourcequotas , resourcequotas/status	get, list, watch
	namespaces	get,list, watch

AmazonEKSClusterAdminPolicy

This access policy includes permissions that grant an IAM principal administrator access to a cluster. When associated to an access entry, its access scope is typically the cluster, rather than a Kubernetes namespace. If you want an IAM principal to have a more limited administrative scope, consider associating the the section called “AmazonEKSAdminPolicy” access policy to your access entry instead.

ARN – `arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy`

Kubernetes API groups	Kubernetes nonResourceURLs	Kubernetes resources	Kubernetes verbs (permissions)
*		*	*
	*		*

AmazonEKSAdminViewPolicy

This access policy includes permissions that grant an IAM principal access to list/view all resources in a cluster. Note this includes Kubernetes Secrets.

ARN – `arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminViewPolicy`

Kubernetes API groups	Kubernetes resources	Kubernetes verbs (permissi ons)
*	*	get, list, watch

AmazonEKSEditPolicy

This access policy includes permissions that allow an IAM principal to edit most Kubernetes resources.

ARN – `arn:aws:eks::aws:cluster-access-policy/AmazonEKSEditPolicy`

Kubernetes API groups	Kubernetes resources	Kubernetes verbs (permissi ons)
apps	daemonsets , deploymen ts , deployments/ rollback , deploymen ts/scale , replicase ts , replicasets/	create, delete, deletecol lection , patch, update

Kubernetes API groups	Kubernetes resources	Kubernetes verbs (permissions)
	scale, statefulsets , statefulsets/scale	
apps	controllerrevisions , daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , replicasets , replicasets/scale , replicasets/status , statefulsets , statefulsets/scale , statefulsets/status	get, list, watch
autoscaling	horizontalpodautoscalers , horizontalpodautoscalers/status	get, list, watch
autoscaling	horizontalpodautoscalers	create, delete, deletecollection , patch, update
batch	cronjobs, jobs	create, delete, deletecollection , patch, update
batch	cronjobs, cronjobs/status , jobs, jobs/status	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch

Kubernetes API groups	Kubernetes resources	Kubernetes verbs (permissions)
extensions	daemonsets , deployments , deployments/rollback , deployments/scale , ingresses , networkpolicies , replicasets , replicasets/scale , replicationcontrollers/scale	create, delete, deletecollection , patch, update
extensions	daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , ingresses , ingresses/status , networkpolicies , replicasets , replicasets/scale , replicasets/status , replicationcontrollers/scale	get, list, watch
networking.k8s.io	ingresses , networkpolicies	create, delete, deletecollection , patch, update
networking.k8s.io	ingresses , ingresses/status , networkpolicies	get, list, watch
policy	poddisruptionbudgets	create, delete, deletecollection , patch, update
policy	poddisruptionbudgets , poddisruptionbudgets/status	get, list, watch

Kubernetes API groups	Kubernetes resources	Kubernetes verbs (permissions)
	namespaces	get, list, watch
	pods/attach , pods/exec , pods/portforward , pods/proxy , secrets, services/proxy	get, list, watch
	serviceaccounts	impersonate
	pods, pods/attach , pods/exec , pods/portforward , pods/proxy	create, delete, deletecollection , patch, update
	configmaps , events, persistentvolumeclaims , replicationcontrollers , replicationcontrollers/scale , secrets, serviceaccounts , services, services/proxy	create, delete, deletecollection , patch, update
	configmaps , endpoints , persistentvolumeclaims , persistentvolumeclaims/status , pods, replicationcontrollers , replicationcontrollers/scale , serviceaccounts , services, services/status	get, list, watch

Kubernetes API groups	Kubernetes resources	Kubernetes verbs (permissions)
	bindings, events, limitranges , namespaces/status , pods/log, pods/status , replicationcontrollers/status , resourcequotas , resourcequotas/status	get, list, watch

AmazonEKSViewPolicy

This access policy includes permissions that allow an IAM principal to view most Kubernetes resources.

ARN – `arn:aws:eks::aws:cluster-access-policy/AmazonEKSViewPolicy`

Kubernetes API groups	Kubernetes resources	Kubernetes verbs (permissions)
apps	controllerrevisions , daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , replicasets , replicasets/scale , replicasets/status , statefulsets , statefulsets/scale , statefulsets/status	get, list, watch
autoscaling	horizontalpodautoscalers , horizonta	get, list, watch

Kubernetes API groups	Kubernetes resources	Kubernetes verbs (permissions)
	lpodautoscalers/status	
batch	cronjobs, cronjobs/status , jobs, jobs/status	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch
extensions	daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , ingresses , ingresses/status , networkpolicies , replicasets , replicasets/scale , replicasets/status , replicationcontrollers/scale	get, list, watch
networking.k8s.io	ingresses , ingresses/status , networkpolicies	get, list, watch
policy	poddisruptionbudgets , poddisruptionbudgets/status	get, list, watch

Kubernetes API groups	Kubernetes resources	Kubernetes verbs (permissions)
	configmaps , endpoints , persistentvolumeclaims , persistentvolumeclaims/status , pods, replicationcontrollers , replicationcontrollers/scale , serviceaccounts , services, services/status	get, list, watch
	bindings, events, limitranges , namespaces/status , pods/log, pods/status , replicationcontrollers/status , resourcequotas , resourcequotas/status	get, list, watch
	namespaces	get, list, watch

AmazonEKSAutoNodePolicy

ARN – arn:aws:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy

This policy includes the following permissions that allow Amazon EKS components to complete the following tasks:

- kube-proxy – Monitor network endpoints and services, and manage related events. This enables cluster-wide network proxy functionality.
- ipamd – Manage AWS VPC networking resources and container network interfaces (CNI). This allows the IP address management daemon to handle pod networking.
- coredns – Access service discovery resources like endpoints and services. This enables DNS resolution within the cluster.

- `ebs-csi-driver` – Work with storage-related resources for Amazon EBS volumes. This allows dynamic provisioning and attachment of persistent volumes.
- `neuron` – Monitor nodes and pods for AWS Neuron devices. This enables management of AWS Inferentia and Trainium accelerators.
- `node-monitoring-agent` – Access node diagnostics and events. This enables cluster health monitoring and diagnostics collection.

Each component uses a dedicated service account and is restricted to only the permissions required for its specific function.

If you manually specifiy a Node IAM role in a NodeClass, you need to create an Access Entry that associates the new Node IAM role with this Access Policy.

AmazonEKSBlockStoragePolicy

ARN – `arn:aws:eks::aws:cluster-access-policy/AmazonEKSBlockStoragePolicy`

This policy includes permissions that allow Amazon EKS to manage leader election and coordination resources for storage operations:

- `coordination.k8s.io` – Create and manage lease objects for leader election. This enables EKS storage components to coordinate their activities across the cluster through a leader election mechanism.

The policy is scoped to specific lease resources used by the EKS storage components to prevent conflicting access to other coordination resources in the cluster.

Amazon EKS automatically creates an access entry with this access policy for the cluster IAM role when Auto Mode is enabled, ensuring that the necessary permissions are in place for the block storage capability to function properly.

AmazonEKSLoadBalancingPolicy

ARN – `arn:aws:eks::aws:cluster-access-policy/AmazonEKSLoadBalancingPolicy`

This policy includes permissions that allow Amazon EKS to manage leader election resources for load balancing:

- `coordination.k8s.io` – Create and manage lease objects for leader election. This enables EKS load balancing components to coordinate activities across multiple replicas by electing a leader.

The policy is scoped specifically to load balancing lease resources to ensure proper coordination while preventing access to other lease resources in the cluster.

Amazon EKS automatically creates an access entry with this access policy for the cluster IAM role when Auto Mode is enabled, ensuring that the necessary permissions are in place for the networking capability to function properly.

AmazonEKSNetworkingPolicy

ARN – `arn:aws:eks::aws:cluster-access-policy/AmazonEKSNetworkingPolicy`

This policy includes permissions that allow Amazon EKS to manage leader election resources for networking:

- `coordination.k8s.io` – Create and manage lease objects for leader election. This enables EKS networking components to coordinate IP address allocation activities by electing a leader.

The policy is scoped specifically to networking lease resources to ensure proper coordination while preventing access to other lease resources in the cluster.

Amazon EKS automatically creates an access entry with this access policy for the cluster IAM role when Auto Mode is enabled, ensuring that the necessary permissions are in place for the networking capability to function properly.

AmazonEKSComputePolicy

ARN – `arn:aws:eks::aws:cluster-access-policy/AmazonEKSComputePolicy`

This policy includes permissions that allow Amazon EKS to manage leader election resources for compute operations:

- `coordination.k8s.io` – Create and manage lease objects for leader election. This enables EKS compute components to coordinate node scaling activities by electing a leader.

The policy is scoped specifically to compute management lease resources while allowing basic read access (`get`, `watch`) to all lease resources in the cluster.

Amazon EKS automatically creates an access entry with this access policy for the cluster IAM role when Auto Mode is enabled, ensuring that the necessary permissions are in place for the networking capability to function properly.

AmazonEKSBlockStorageClusterPolicy

ARN – `arn:aws:eks::aws:cluster-access-policy/AmazonEKSBlockStorageClusterPolicy`

This policy grants permissions necessary for the block storage capability of Amazon EKS Auto Mode. It enables efficient management of block storage resources within Amazon EKS clusters. The policy includes the following permissions:

CSI Driver Management:

- Create, read, update, and delete CSI drivers, specifically for block storage.

Volume Management:

- List, watch, create, update, patch, and delete persistent volumes.
- List, watch, and update persistent volume claims.
- Patch persistent volume claim statuses.

Node and Pod Interaction:

- Read node and pod information.
- Manage events related to storage operations.

Storage Classes and Attributes:

- Read storage classes and CSI nodes.
- Read volume attribute classes.

Volume Attachments:

- List, watch, and modify volume attachments and their statuses.

Snapshot Operations:

- Manage volume snapshots, snapshot contents, and snapshot classes.
- Handle operations for volume group snapshots and related resources.

This policy is designed to support comprehensive block storage management within Amazon EKS clusters running in Auto Mode. It combines permissions for various operations including provisioning, attaching, resizing, and snapshotting of block storage volumes.

Amazon EKS automatically creates an access entry with this access policy for the cluster IAM role when Auto Mode is enabled, ensuring that the necessary permissions are in place for the block storage capability to function properly.

AmazonEKSComputeClusterPolicy

ARN – `arn:aws:eks::aws:cluster-access-policy/AmazonEKSComputeClusterPolicy`

This policy grants permissions necessary for the compute management capability of Amazon EKS Auto Mode. It enables efficient orchestration and scaling of compute resources within Amazon EKS clusters. The policy includes the following permissions:

Node Management:

- Create, read, update, delete, and manage status of NodePools and NodeClaims.
- Manage NodeClasses, including creation, modification, and deletion.

Scheduling and Resource Management:

- Read access to pods, nodes, persistent volumes, persistent volume claims, replication controllers, and namespaces.
- Read access to storage classes, CSI nodes, and volume attachments.
- List and watch deployments, daemon sets, replica sets, and stateful sets.
- Read pod disruption budgets.

Event Handling:

- Create, read, and manage cluster events.

Node Deprovisioning and Pod Eviction:

- Update, patch, and delete nodes.
- Create pod evictions and delete pods when necessary.

Custom Resource Definition (CRD) Management:

- Create new CRDs.
- Manage specific CRDs related to node management (NodeClasses, NodePools, NodeClaims, and NodeDiagnostics).

This policy is designed to support comprehensive compute management within Amazon EKS clusters running in Auto Mode. It combines permissions for various operations including node provisioning, scheduling, scaling, and resource optimization.

Amazon EKS automatically creates an access entry with this access policy for the cluster IAM role when Auto Mode is enabled, ensuring that the necessary permissions are in place for the compute management capability to function properly.

AmazonEKSLoadBalancingClusterPolicy

ARN – `arn:aws:eks::aws:cluster-access-policy/AmazonEKSLoadBalancingClusterPolicy`

This policy grants permissions necessary for the load balancing capability of Amazon EKS Auto Mode. It enables efficient management and configuration of load balancing resources within Amazon EKS clusters. The policy includes the following permissions:

Event and Resource Management:

- Create and patch events.
- Read access to pods, nodes, endpoints, and namespaces.
- Update pod statuses.

Service and Ingress Management:

- Full management of services and their statuses.
- Comprehensive control over ingresses and their statuses.
- Read access to endpoint slices and ingress classes.

Target Group Bindings:

- Create and modify target group bindings and their statuses.

- Read access to ingress class parameters.

Custom Resource Definition (CRD) Management:

- Create and read all CRDs.
- Specific management of targetgroupbindings.eks.amazonaws.com and ingressclassparams.eks.amazonaws.com CRDs.

Webhook Configuration:

- Create and read mutating and validating webhook configurations.
- Manage the eks-load-balancing-webhook configuration.

This policy is designed to support comprehensive load balancing management within Amazon EKS clusters running in Auto Mode. It combines permissions for various operations including service exposure, ingress routing, and integration with AWS load balancing services.

Amazon EKS automatically creates an access entry with this access policy for the cluster IAM role when Auto Mode is enabled, ensuring that the necessary permissions are in place for the load balancing capability to function properly.

AmazonEKSNetworkingClusterPolicy

ARN – `arn:aws:eks::aws:cluster-access-policy/AmazonEKSNetworkingClusterPolicy`

AmazonEKSNetworkingClusterPolicy

This policy grants permissions necessary for the networking capability of Amazon EKS Auto Mode. It enables efficient management and configuration of networking resources within Amazon EKS clusters. The policy includes the following permissions:

Node and Pod Management:

- Read access to NodeClasses and their statuses.
- Read access to NodeClaims and their statuses.
- Read access to pods.

CNI Node Management:

- Permissions for CNINodes and their statuses, including create, read, update, delete, and patch.

Custom Resource Definition (CRD) Management:

- Create and read all CRDs.
- Specific management (update, patch, delete) of the cninodes.eks.amazonaws.com CRD.

Event Management:

- Create and patch events.

This policy is designed to support comprehensive networking management within Amazon EKS clusters running in Auto Mode. It combines permissions for various operations including node networking configuration, CNI (Container Network Interface) management, and related custom resource handling.

The policy allows the networking components to interact with node-related resources, manage CNI-specific node configurations, and handle custom resources critical for networking operations in the cluster.

Amazon EKS automatically creates an access entry with this access policy for the cluster IAM role when Auto Mode is enabled, ensuring that the necessary permissions are in place for the networking capability to function properly.

AmazonEKSHybridPolicy

This access policy includes permissions that grant EKS access to the nodes of a cluster. When associated to an access entry, its access scope is typically the cluster, rather than a Kubernetes namespace. This policy is used by Amazon EKS hybrid nodes.

ARN – `arn:aws:eks::aws:cluster-access-policy/AmazonEKSHybridPolicy`

Kubernetes API groups	Kubernetes nonResourceURLs	Kubernetes resources	Kubernetes verbs (permissions)
`*`		`nodes`	`list`

AmazonEKSClusterInsightsPolicy

ARN – `arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterInsightsPolicy`

This policy grants read-only permissions for Amazon EKS Cluster Insights functionality. The policy includes the following permissions:

Node Access: - List and view cluster nodes - Read node status information

DaemonSet Access: - Read access to kube-proxy configuration

This policy is automatically managed by the EKS service for Cluster Insights. For more information, see the section called “Cluster insights”.

Access policy updates

View details about updates to access policies, since they were introduced. For automatic alerts about changes to this page, subscribe to the RSS feed in *Document history*.

Change	Description	Date
Add policy for EKS Cluster Insights	Publish `AmazonEKSClusterInsightsPolicy`	December 2, 2024
Add policies for Amazon EKS Hybrid	Publish `AmazonEKSHybridPolicy`	December 2, 2024
Add policies for Amazon EKS Auto Mode	These access policies give the Cluster IAM Role and Node IAM Role permission to call Kubernetes APIs. AWS uses these to automate routine tasks for storage, compute, and networking resources.	December 2, 2024
Add `AmazonEKSAdminViewPolicy`	Add a new policy for expanded view access,	April 23, 2024

Change	Description	Date
	including resources like Secrets.	
Access policies introduced.	Amazon EKS introduced access policies.	May 29, 2023

Change authentication mode to use access entries

To begin using access entries, you must change the authentication mode of the cluster to either the `API_AND_CONFIG_MAP` or `API` modes. This adds the API for access entries.

AWS Console

1. Open the [Amazon EKS console](https://console.aws.amazon.com/eks/home#/clusters).
2. Choose the name of the cluster that you want to create an access entry in.
3. Choose the **Access** tab.
4. The **Authentication mode** shows the current authentication mode of the cluster. If the mode says EKS API, you can already add access entries and you can skip the remaining steps.
5. Choose **Manage access**.
6. For **Cluster authentication mode**, select a mode with the EKS API. Note that you can't change the authentication mode back to a mode that removes the EKS API and access entries.
7. Choose **Save changes**. Amazon EKS begins to update the cluster, the status of the cluster changes to Updating, and the change is recorded in the **Update history** tab.
8. Wait for the status of the cluster to return to Active. When the cluster is Active, you can follow the steps in the section called "Create access entries" to add access to the cluster for IAM principals.

AWS CLI

1. Install the AWS CLI, as described in Installing in the *AWS Command Line Interface User Guide*.
2. Run the following command. Replace *`my-cluster`* with the name of your cluster. If you want to disable the `ConfigMap` method permanently, replace `API_AND_CONFIG_MAP` with `API`.

Amazon EKS begins to update the cluster, the status of the cluster changes to UPDATING, and the change is recorded in the **aws eks list-updates** .

```
aws eks update-cluster-config --name my-cluster --access-config
 authenticationMode=API_AND_CONFIG_MAP
```

3. Wait for the status of the cluster to return to Active. When the cluster is Active, you can follow the steps in the section called "Create access entries" to add access to the cluster for IAM principals.

Required platform version

To use *access entries*, the cluster must have a platform version that is the same or later than the version listed in the following table, or a Kubernetes version that is later than the versions listed in the table. If your Kubernetes version is not listed, all platform versions support access entries.

Kubernetes version	Platform version
1.30	eks.2
1.29	eks.1
1.28	eks.6
1.27	eks.10
1.26	eks.11
1.25	eks.12
1.24	eks.15
1.23	eks.17

For more information, see the section called "Platform versions".

Create access entries

Before creating access entries, consider the following:

- A properly set authentication mode. See the section called “Authentication mode”.
- An *access entry* includes the Amazon Resource Name (ARN) of one, and only one, existing IAM principal. An IAM principal can't be included in more than one access entry. Additional considerations for the ARN that you specify:
  - IAM best practices recommend accessing your cluster using IAM *roles* that have short-term credentials, rather than IAM *users* that have long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.
  - If the ARN is for an IAM role, it *can* include a path. ARNs in `aws-auth ConfigMap` entries, *can't* include a path. For example, your ARN can be `arn:aws:iam::<111122223333>:role/<development/apps/my-role>` or `arn:aws:iam::<111122223333>:role/<my-role>`.
  - If the type of the access entry is anything other than `STANDARD` (see next consideration about types), the ARN must be in the same AWS account that your cluster is in. If the type is `STANDARD`, the ARN can be in the same, or different, AWS account than the account that your cluster is in.
  - You can't change the IAM principal after the access entry is created.
  - If you ever delete the IAM principal with this ARN, the access entry isn't automatically deleted. We recommend that you delete the access entry with an ARN for an IAM principal that you delete. If you don't delete the access entry and ever recreate the IAM principal, even if it has the same ARN, the access entry won't work. This is because even though the ARN is the same for the recreated IAM principal, the `roleID` or `userID` (you can see this with the `aws sts get-caller-identity` AWS CLI command) is different for the recreated IAM principal than it was for the original IAM principal. Even though you don't see the IAM principal's `roleID` or `userID` for an access entry, Amazon EKS stores it with the access entry.
- Each access entry has a *type*. If you don't specify a type, Amazon EKS automatically sets the type to `STANDARD`
  - `EC2_LINUX` - For an IAM role used with Linux or Bottlerocket self-managed nodes
  - `EC2_WINDOWS` - For an IAM role used with Windows self-managed nodes
  - `FARGATE_LINUX` - For an IAM role used with AWS Fargate (Fargate)
  - `HYBRID_LINUX` - For an IAM role used with hybrid nodes
  - `STANDARD` - Default type if none specified
  - `EC2` - For EKS Auto Mode custom node classes. For more information, see the section called “Create node class access entry”.

  - You can't change the type after the access entry is created.
- It's unnecessary to create an access entry for an IAM role that's used for a managed node group or a Fargate profile. EKS will create access entries (if enabled), or update the auth config map (if access entries are unavailable)
- If the type of the access entry is `STANDARD`, you can specify a *username* for the access entry. If you don't specify a value for username, Amazon EKS sets one of the following values for you, depending on the type of the access entry and whether the IAM principal that you specified is an IAM role or IAM user. Unless you have a specific reason for specifying your own username, we recommend that don't specify one and let Amazon EKS auto-generate it for you. If you specify your own username:
  - It can't start with `system:`, `eks:`, `aws:`, `amazon:`, or `iam:`.
  - If the username is for an IAM role, we recommend that you add `{{SessionName}}` to the end of your username. If you add `{{SessionName}}` to your username, the username must include a colon *before* {{SessionName}}. When this role is assumed, the name of the session specified when assuming the role is automatically passed to the cluster and will appear in CloudTrail logs. For example, you can't have a username of `john{{SessionName}}`. The username would have to be `:john{{SessionName}}` or `jo:hn{{SessionName}}`. The colon only has to be before `{{SessionName}}`. The username generated by Amazon EKS in the following table includes an ARN. Since an ARN includes colons, it meets this requirement. The colon isn't required if you don't include `{{SessionName}}` in your username. Note that the special character "@" is replaced with "-" in the session name.

IAM principal type	Type	Username value that Amazon EKS automatically sets
User	`STANDARD`	The ARN of the user. Example: `arn:aws:iam::<111122223333>:user/<my-user>`
Role	`STANDARD`	The STS ARN of the role when it's assumed. Amazon EKS appends `{{SessionName}}` to the role.

IAM principal type	Type	Username value that Amazon EKS automatically sets
		Example: `arn:aws:sts::<111122223333>:assumed-role/<my-role>/{{SessionName}}` If the ARN of the role that you specified contained a path, Amazon EKS removes it in the generated username.
Role	`EC2_LINUX` or `EC2_Windows`	`system:node:{{EC2PrivateDNSName}}`
Role	`FARGATE_LINUX`	`system:node:{{SessionName}}`
Role	`HYBRID_LINUX`	`system:node:{{SessionName}}`

You can change the username after the access entry is created.

- If an access entry's type is `STANDARD`, and you want to use Kubernetes RBAC authorization, you can add one or more *group names* to the access entry. After you create an access entry you can add and remove group names. For the IAM principal to have access to Kubernetes objects on your cluster, you must create and manage Kubernetes role-based authorization (RBAC) objects. Create Kubernetes `RoleBinding` or `ClusterRoleBinding` objects on your cluster that specify the group name as a `subject` for `kind: Group`. Kubernetes authorizes the IAM principal access to any cluster objects that you've specified in a Kubernetes `Role` or `ClusterRole` object that you've also specified in your binding's `roleRef`. If you specify group names, we recommend that you're familiar with the Kubernetes role-based authorization (RBAC) objects. For more information, see [Using RBAC Authorization] in the Kubernetes documentation.

> **Important**
>
> Amazon EKS doesn't confirm that any Kubernetes RBAC objects that exist on your cluster include any of the group names that you specify. For example, if you create an access entry for group that currently doesn't exist, EKS will create the group instead of returning an error.

Instead of, or in addition to, Kubernetes authorizing the IAM principal access to Kubernetes objects on your cluster, you can associate Amazon EKS *access policies* to an access entry. Amazon EKS authorizes IAM principals to access Kubernetes objects on your cluster with the permissions in the access policy. You can scope an access policy's permissions to Kubernetes namespaces that you specify. Use of access policies don't require you to manage Kubernetes RBAC objects. For more information, see the section called "Associate access policies".

- If you create an access entry with type `EC2_LINUX` or `EC2_Windows`, the IAM principal creating the access entry must have the `iam:PassRole` permission. For more information, see Granting a user permissions to pass a role to an AWS service in the *IAM User Guide*.
- Similar to standard IAM behavior, access entry creation and updates are eventually consistent, and may take several seconds to be effective after the initial API call returns successfully. You must design your applications to account for these potential delays. We recommend that you don't include access entry creates or updates in the critical, high- availability code paths of your application. Instead, make changes in a separate initialization or setup routine that you run less frequently. Also, be sure to verify that the changes have been propagated before production workflows depend on them.
- Access entries do not support service linked roles. You cannot create access entries where the principal ARN is a service linked role. You can identify service linked roles by their ARN, which is in the format `arn:aws:iam::*:role/aws-service-role/*`.

You can create an access entry using the AWS Management Console or the AWS CLI.

AWS Management Console

1. Open the Amazon EKS console.
2. Choose the name of the cluster that you want to create an access entry in.
3. Choose the **Access** tab.

4. Choose **Create access entry**.
5. For **IAM principal**, select an existing IAM role or user. IAM best practices recommend accessing your cluster using IAM *roles* that have short-term credentials, rather than IAM *users* that have long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.
6. For **Type**, if the access entry is for the node role used for self-managed Amazon EC2 nodes, select **EC2 Linux** or **EC2 Windows**. Otherwise, accept the default (**Standard**).
7. If the **Type** you chose is **Standard** and you want to specify a **Username**, enter the username.
8. If the **Type** you chose is **Standard** and you want to use Kubernetes RBAC authorization for the IAM principal, specify one or more names for **Groups**. If you don't specify any group names and want to use Amazon EKS authorization, you can associate an access policy in a later step, or after the access entry is created.
9. (Optional) For **Tags**, assign labels to the access entry. For example, to make it easier to find all resources with the same tag.
10. Choose **Next**.
11. On the **Add access policy** page, if the type you chose was **Standard** and you want Amazon EKS to authorize the IAM principal to have permissions to the Kubernetes objects on your cluster, complete the following steps. Otherwise, choose **Next**.
    a. For **Policy name**, choose an access policy. You can't view the permissions of the access policies, but they include similar permissions to those in the Kubernetes user-facing `ClusterRole` objects. For more information, see User-facing roles in the Kubernetes documentation.
    b. Choose one of the following options:
       - **Cluster** – Choose this option if you want Amazon EKS to authorize the IAM principal to have the permissions in the access policy for all Kubernetes objects on your cluster.
       - **Kubernetes namespace** – Choose this option if you want Amazon EKS to authorize the IAM principal to have the permissions in the access policy for all Kubernetes objects in a specific Kubernetes namespace on your cluster. For **Namespace**, enter the name of the Kubernetes namespace on your cluster. If you want to add additional namespaces, choose **Add new namespace** and enter the namespace name.
    c. If you want to add additional policies, choose **Add policy**. You can scope each policy differently, but you can add each policy only once.
    d. Choose **Next**.

12Review the configuration for your access entry. If anything looks incorrect, choose **Previous** to go back through the steps and correct the error. If the configuration is correct, choose **Create**.

AWS CLI

1. Install the AWS CLI, as described in Installing in the AWS Command Line Interface User Guide.
2. To create an access entry You can use any of the following examples to create access entries:
   - Create an access entry for a self-managed Amazon EC2 Linux node group. Replace *my-cluster* with the name of your cluster, *111122223333* with your AWS account ID, and *EKS-my-cluster-self-managed-ng-1* with the name of your node IAM role. If your node group is a Windows node group, then replace *EC2_LINUX* with EC2_Windows.

     ```
     aws eks create-access-entry --cluster-name my-cluster --principal-arn
      arn:aws:iam::111122223333:role/EKS-my-cluster-self-managed-ng-1 --type EC2_LINUX
     ```

     You can't use the --kubernetes-groups option when you specify a type other than STANDARD. You can't associate an access policy to this access entry, because its type is a value other than STANDARD.
   - Create an access entry that allows an IAM role that's not used for an Amazon EC2 self-managed node group, that you want Kubernetes to authorize access to your cluster with. Replace *my-cluster* with the name of your cluster, *111122223333* with your AWS account ID, and *my-role* with the name of your IAM role. Replace *Viewers* with the name of a group that you've specified in a Kubernetes RoleBinding or ClusterRoleBinding object on your cluster.

     ```
     aws eks create-access-entry --cluster-name my-cluster --principal-arn
      arn:aws:iam::111122223333:role/my-role --type STANDARD --user Viewers --
     kubernetes-groups Viewers
     ```
   - Create an access entry that allows an IAM user to authenticate to your cluster. This example is provided because this is possible, though IAM best practices recommend accessing your cluster using IAM *roles* that have short-term credentials, rather than IAM *users* that have long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

     ```
     aws eks create-access-entry --cluster-name my-cluster --principal-arn
      arn:aws:iam::111122223333:user/my-user --type STANDARD --username my-user
     ```

If you want this user to have more access to your cluster than the permissions in the Kubernetes API discovery roles, then you need to associate an access policy to the access entry, since the `--kubernetes-groups` option isn't used. For more information, see the section called "Associate access policies" and API discovery roles in the Kubernetes documentation.

Update access entries

You can update an access entry using the AWS Management Console or the AWS CLI.

AWS Management Console

1. Open the Amazon EKS console.
2. Choose the name of the cluster that you want to create an access entry in.
3. Choose the **Access** tab.
4. Choose the access entry that you want to update.
5. Choose **Edit**.
6. For **Username**, you can change the existing value.
7. For **Groups**, you can remove existing group names or add new group names. If the following groups names exist, don't remove them: **system:nodes** or **system:bootstrappers**. Removing these groups can cause your cluster to function improperly. If you don't specify any group names and want to use Amazon EKS authorization, associate an access policy in a later step.
8. For **Tags**, you can assign labels to the access entry. For example, to make it easier to find all resources with the same tag. You can also remove existing tags.
9. Choose **Save changes**.
10. If you want to associate an access policy to the entry, see the section called "Associate access policies".

AWS CLI

1. Install the AWS CLI, as described in Installing in the AWS Command Line Interface User Guide.
2. To update an access entry Replace *`my-cluster`* with the name of your cluster, *`111122223333`* with your AWS account ID, and *`EKS-my-cluster-my-namespace-Viewers`* with the name of an IAM role.

```
aws eks update-access-entry --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/EKS-my-cluster-my-namespace-Viewers --kubernetes-
groups Viewers
```

You can't use the `--kubernetes-groups` option if the type of the access entry is a value other than `STANDARD`. You also can't associate an access policy to an access entry with a type other than `STANDARD`.

Delete access entries

If you discover that you deleted an access entry in error, you can always recreate it. If the access entry that you're deleting is associated to any access policies, the associations are automatically deleted. You don't have to disassociate access policies from an access entry before deleting the access entry.

You can delete an access entry using the AWS Management Console or the AWS CLI.

AWS Management Console

1. Open the [Amazon EKS console](#).
2. Choose the name of the cluster that you want to delete an access entry from.
3. Choose the **Access** tab.
4. In the **Access entries** list, choose the access entry that you want to delete.
5. Choose Delete.
6. In the confirmation dialog box, choose **Delete**.

AWS CLI

1. Install the AWS CLI, as described in [Installing](#) in the AWS Command Line Interface User Guide.
2. To delete an access entry Replace *`my-cluster`* with the name of your cluster, *`111122223333`* with your AWS account ID, and *`my-role`* with the name of the IAM role that you no longer want to have access to your cluster.

```
aws eks delete-access-entry --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/my-role
```

Grant IAM users access to Kubernetes with a ConfigMap

> **Important**
>
> The `aws-auth ConfigMap` is deprecated. For the recommended method to manage access to Kubernetes APIs, see the section called "Grant permissions".

Access to your cluster using IAM principals is enabled by the AWS IAM Authenticator for Kubernetes, which runs on the Amazon EKS control plane. The authenticator gets its configuration information from the `aws-auth` `ConfigMap`. For all `aws-auth` `ConfigMap` settings, see Full Configuration Format on GitHub.

Add IAM principals to your Amazon EKS cluster

When you create an Amazon EKS cluster, the IAM principal that creates the cluster is automatically granted `system:masters` permissions in the cluster's role-based access control (RBAC) configuration in the Amazon EKS control plane. This principal doesn't appear in any visible configuration, so make sure to keep track of which principal originally created the cluster. To grant additional IAM principals the ability to interact with your cluster, edit the `aws-auth ConfigMap` within Kubernetes and create a Kubernetes `rolebinding` or `clusterrolebinding` with the name of a `group` that you specify in the `aws-auth ConfigMap`.

> **Note**
>
> For more information about Kubernetes role-based access control (RBAC) configuration, see Using RBAC Authorization in the Kubernetes documentation.

1. Determine which credentials `kubectl` is using to access your cluster. On your computer, you can see which credentials `kubectl` uses with the following command. Replace *`~/.kube/config`* with the path to your `kubeconfig` file if you don't use the default path.

```
cat ~/.kube/config
```

   An example output is as follows.

```
[...]
```

```
contexts:
- context:
    cluster: my-cluster.region-code.eksctl.io
    user: admin@my-cluster.region-code.eksctl.io
  name: admin@my-cluster.region-code.eksctl.io
current-context: admin@my-cluster.region-code.eksctl.io
[...]
```

In the previous example output, the credentials for a user named *admin* are configured for a cluster named *my-cluster*. If this is the user that created the cluster, then it already has access to your cluster. If it's not the user that created the cluster, then you need to complete the remaining steps to enable cluster access for other IAM principals. IAM best practices recommend that you grant permissions to roles instead of users. You can see which other principals currently have access to your cluster with the following command:

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

An example output is as follows.

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

Data
====
mapRoles:
----
- groups:
  - system:bootstrappers
  - system:nodes
  rolearn: arn:aws:iam::111122223333:role/my-node-role
  username: system:node:{{EC2PrivateDNSName}}


BinaryData
====

Events:  <none>
```

The previous example is a default `aws-auth` `ConfigMap`. Only the node instance role has access to the cluster.

2. Make sure that you have existing Kubernetes `roles` and `rolebindings` or `clusterroles` and `clusterrolebindings` that you can map IAM principals to. For more information about these resources, see [Using RBAC Authorization](#) in the Kubernetes documentation.

   a. View your existing Kubernetes `roles` or `clusterroles`. `Roles` are scoped to a `namespace`, but `clusterroles` are scoped to the cluster.

   ```
   kubectl get roles -A
   ```

   ```
   kubectl get clusterroles
   ```

   b. View the details of any `role` or `clusterrole` returned in the previous output and confirm that it has the permissions (`rules`) that you want your IAM principals to have in your cluster.

   Replace *`role-name`* with a `role` name returned in the output from the previous command. Replace *`kube-system`* with the namespace of the `role`.

   ```
   kubectl describe role role-name -n kube-system
   ```

   Replace *`cluster-role-name`* with a `clusterrole` name returned in the output from the previous command.

   ```
   kubectl describe clusterrole cluster-role-name
   ```

   c. View your existing Kubernetes `rolebindings` or `clusterrolebindings`. `Rolebindings` are scoped to a `namespace`, but `clusterrolebindings` are scoped to the cluster.

   ```
   kubectl get rolebindings -A
   ```

   ```
   kubectl get clusterrolebindings
   ```

   d. View the details of any `rolebinding` or `clusterrolebinding` and confirm that it has a `role` or `clusterrole` from the previous step listed as a `roleRef` and a group name listed for `subjects`.

Replace *role-binding-name* with a `rolebinding` name returned in the output from the previous command. Replace *kube-system* with the `namespace` of the `rolebinding`.

```
kubectl describe rolebinding role-binding-name -n kube-system
```

An example output is as follows.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: eks-console-dashboard-restricted-access-role-binding
  namespace: default
subjects:
- kind: Group
  name: eks-console-dashboard-restricted-access-group
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: eks-console-dashboard-restricted-access-role
  apiGroup: rbac.authorization.k8s.io
```

Replace *cluster-role-binding-name* with a `clusterrolebinding` name returned in the output from the previous command.

```
kubectl describe clusterrolebinding cluster-role-binding-name
```

An example output is as follows.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks-console-dashboard-full-access-binding
subjects:
- kind: Group
  name: eks-console-dashboard-full-access-group
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: eks-console-dashboard-full-access-clusterrole
```

```
  apiGroup: rbac.authorization.k8s.io
```

3. Edit the `aws-auth` `ConfigMap`. You can use a tool such as `eksctl` to update the `ConfigMap` or you can update it manually by editing it.

> **Important**
>
> We recommend using `eksctl`, or another tool, to edit the `ConfigMap`. For information about other tools you can use, see Use tools to make changes to the aws-authConfigMap in the Amazon EKS best practices guides. An improperly formatted `aws-auth` `ConfigMap` can cause you to lose access to your cluster.

- View steps to edit configmap with eksctl.
- View steps to edit configmap manually.

Edit Configmap with Eksctl

1. You need version `0.204.0` or later of the `eksctl` command line tool installed on your device or AWS CloudShell. To install or update `eksctl`, see Installation in the `eksctl` documentation.
2. View the current mappings in the `ConfigMap`. Replace *`my-cluster`* with the name of your cluster. Replace *`region-code`* with the AWS Region that your cluster is in.

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

An example output is as follows.

```
ARN
            USERNAME                                GROUPS
 ACCOUNT
arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-
NodeInstanceRole-1XLS7754U3ZPA    system:node:{{EC2PrivateDNSName}}
 system:bootstrappers,system:nodes
```

3. Add a mapping for a role. Replace *`my-role`* with your role name. Replace *`eks-console-dashboard-full-access-group`* with the name of the group specified in your Kubernetes `RoleBinding` or `ClusterRoleBinding` object. Replace *`111122223333`* with your account ID. You can replace *`admin`* with any name you choose.

```
eksctl create iamidentitymapping --cluster my-cluster --region=region-code \
    --arn arn:aws:iam::111122223333:role/my-role --username admin --group eks-
console-dashboard-full-access-group \
    --no-duplicate-arns
```

> **Important**
>
> The role ARN can't include a path such as `role/my-team/developers/my-role`. The format of the ARN must be `arn:aws:iam::111122223333:role/my-role` . In this example, `my-team/developers/` needs to be removed.

An example output is as follows.

```
[...]
2022-05-09 14:51:20 [#]  adding identity "{arn-aws}iam::111122223333:role/my-role" to
 auth ConfigMap
```

4. Add a mapping for a user. IAM best practices recommend that you grant permissions to roles instead of users. Replace *`my-user`* with your user name. Replace *`eks-console-dashboard-restricted-access-group`* with the name of the group specified in your Kubernetes `RoleBinding` or `ClusterRoleBinding` object. Replace *`111122223333`* with your account ID. You can replace *`my-user`* with any name you choose.

```
eksctl create iamidentitymapping --cluster my-cluster --region=region-code \
    --arn arn:aws:iam::111122223333:user/my-user --username my-user --group eks-
console-dashboard-restricted-access-group \
    --no-duplicate-arns
```

An example output is as follows.

```
[...]
2022-05-09 14:53:48 [#]  adding identity "arn:aws:iam::111122223333:user/my-user" to
 auth ConfigMap
```

5. View the mappings in the `ConfigMap` again.

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

An example output is as follows.

```
ARN
           USERNAME                                GROUPS
      ACCOUNT
arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-
NodeInstanceRole-1XLS7754U3ZPA    system:node:{{EC2PrivateDNSName}}
 system:bootstrappers,system:nodes
arn:aws:iam::111122223333:role/admin
           my-role                                 eks-console-dashboard-full-access-
group
arn:aws:iam::111122223333:user/my-user
            my-user                                 eks-console-dashboard-restricted-
access-group
```

Edit Configmap manually

1. Open the `ConfigMap` for editing.

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

> **Note**
>
> If you receive an error stating " `Error from server (NotFound): configmaps "aws-auth" not found` ", then use the procedure in Apply the aws-auth ConfigMap to your cluster to apply the stock `ConfigMap`.

2. Add your IAM principals to the `ConfigMap`. An IAM group isn't an IAM principal, so it can't be added to the `ConfigMap`.

   - **To add an IAM role (for example, for federated users):** Add the role details to the `mapRoles` section of the `ConfigMap`, under `data`. Add this section if it does not already exist in the file. Each entry supports the following parameters:
     - **rolearn**: The ARN of the IAM role to add. This value can't include a path. For example, you can't specify an ARN such as `arn:aws:iam::111122223333:role/my-team/developers/role-name`. The ARN needs to be `arn:aws:iam::111122223333:role/role-name` instead.
     - **username**: The user name within Kubernetes to map to the IAM role.

  - **groups**: The group or list of Kubernetes groups to map the role to. The group can be a default group, or a group specified in a `clusterrolebinding` or `rolebinding`. For more information, see [Default roles and role bindings](#) in the Kubernetes documentation.

- **To add an IAM user:** [IAM best practices](#) recommend that you grant permissions to roles instead of users. Add the user details to the `mapUsers` section of the `ConfigMap`, under `data`. Add this section if it does not already exist in the file. Each entry supports the following parameters:
  - **userarn**: The ARN of the IAM user to add.
  - **username**: The user name within Kubernetes to map to the IAM user.
  - **groups**: The group, or list of Kubernetes groups to map the user to. The group can be a default group, or a group specified in a `clusterrolebinding` or `rolebinding`. For more information, see [Default roles and role bindings](#) in the Kubernetes documentation.

3. For example, the following YAML block contains:
   - A `mapRoles` section that maps the IAM node instance to Kubernetes groups so that nodes can register themselves with the cluster and the `my-console-viewer-role` IAM role that is mapped to a Kubernetes group that can view all Kubernetes resources for all clusters. For a list of the IAM and Kubernetes group permissions required for the `my-console-viewer-role` IAM role, see [the section called "Required permissions"](#).
   - A `mapUsers` section that maps the `admin` IAM user from the default AWS account to the `system:masters` Kubernetes group and the `my-user` user from a different AWS account that is mapped to a Kubernetes group that can view Kubernetes resources for a specific namespace. For a list of the IAM and Kubernetes group permissions required for the `my-user` IAM user, see [the section called "Required permissions"](#).

   Add or remove lines as necessary and replace all *`example values`* with your own values.

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file
 will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
```

```
      rolearn: arn:aws:iam::111122223333:role/my-role
      username: system:node:{{EC2PrivateDNSName}}
    - groups:
      - eks-console-dashboard-full-access-group
      rolearn: arn:aws:iam::111122223333:role/my-console-viewer-role
      username: my-console-viewer-role
  mapUsers: |
    - groups:
      - system:masters
      userarn: arn:aws:iam::111122223333:user/admin
      username: admin
    - groups:
      - eks-console-dashboard-restricted-access-group
      userarn: arn:aws:iam::444455556666:user/my-user
      username: my-user
```

4. Save the file and exit your text editor.

Apply the `aws-auth` `ConfigMap` to your cluster

The `aws-auth` `ConfigMap` is automatically created and applied to your cluster when you create a managed node group or when you create a node group using `eksctl`. It is initially created to allow nodes to join your cluster, but you also use this `ConfigMap` to add role-based access control (RBAC) access to IAM principals. If you've launched self-managed nodes and haven't applied the `aws-auth` `ConfigMap` to your cluster, you can do so with the following procedure.

1. Check to see if you've 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://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/aws-auth-cm.yaml
   ```

b. In the `aws-auth-cm.yaml` file, set the `rolearn` to the Amazon Resource Name (ARN) of the IAM role associated with your nodes. You can do this with a text editor, or by replacing *`my-node-instance-role`* and running the following command:

```
sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-
role|' aws-auth-cm.yaml
```

Don't modify any other lines in this file.

> **Important**
>
> The role ARN can't include a path such as `role/my-team/developers/my-role`. The format of the ARN must be `arn:aws:iam::`*`111122223333`*`:role/`*`my-role`* . In this example, `my-team/developers/` needs to be removed.

You can inspect the AWS CloudFormation stack outputs for your node groups and look for the following values:

- **InstanceRoleARN** – For node groups that were created with `eksctl`
- **NodeInstanceRole** – For node groups that were created with Amazon EKS vended AWS CloudFormation templates in the AWS Management Console

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

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

> **Note**
>
> If you receive any authorization or resource type errors, see the section called "Unauthorized or access denied (`kubectl`)" in the troubleshooting topic.

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

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

Enter `Ctrl+C` to return to a shell prompt.

Grant users access to Kubernetes with an external OIDC provider

Amazon EKS supports using OpenID Connect (OIDC) identity providers as a method to authenticate users to your cluster. OIDC identity providers can be used with, or as an alternative to AWS Identity and Access Management (IAM). For more information about using IAM, see the section called "Kubernetes API access". After configuring authentication to your cluster, you can create Kubernetes `roles` and `clusterroles` to assign permissions to the roles, and then bind the roles to the identities using Kubernetes `rolebindings` and `clusterrolebindings`. For more information, see Using RBAC Authorization in the Kubernetes documentation.

- You can associate one OIDC identity provider to your cluster.
- Kubernetes doesn't provide an OIDC identity provider. You can use an existing public OIDC identity provider, or you can run your own identity provider. For a list of certified providers, see OpenID Certification on the OpenID site.
- The issuer URL of the OIDC identity provider must be publicly accessible, so that Amazon EKS can discover the signing keys. Amazon EKS doesn't support OIDC identity providers with self-signed certificates.
- You can't disable IAM authentication to your cluster, because it's still required for joining nodes to a cluster.
- An Amazon EKS cluster must still be created by an AWS IAM principal, rather than an OIDC identity provider user. This is because the cluster creator interacts with the Amazon EKS APIs, rather than the Kubernetes APIs.
- OIDC identity provider-authenticated users are listed in the cluster's audit log if CloudWatch logs are turned on for the control plane. For more information, see the section called "Enable or disable control plane logs".
- You can't sign in to the AWS Management Console with an account from an OIDC provider. You can only the section called "Access cluster resources" by signing into the AWS Management Console with an AWS Identity and Access Management account.

Associate an OIDC identity provider

Before you can associate an OIDC identity provider with your cluster, you need the following information from your provider:

Issuer URL

The URL of the OIDC identity provider that allows the API server to discover public signing keys for verifying tokens. The URL must begin with `https://` and should correspond to the `iss` claim in the provider's OIDC ID tokens. In accordance with the OIDC standard, path components are allowed but query parameters are not. Typically the URL consists of only a host name, like `https://server.example.org` or `https://example.com`. This URL should point to the level below `.well-known/openid-configuration` and must be publicly accessible over the internet.

Client ID (also known as *audience*)

The ID for the client application that makes authentication requests to the OIDC identity provider.

You can associate an identity provider using `eksctl` or the AWS Management Console.

Associate an identity provider using eksctl

1. Create a file named *`associate-identity-provider.yaml`* with the following contents. Replace the *`example values`* with your own. The values in the `identityProviders` section are obtained from your OIDC identity provider. Values are only required for the `name`, `type`, `issuerUrl`, and `clientId` settings under `identityProviders`.

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

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

identityProviders:
  - name: my-provider
    type: oidc
    issuerUrl: https://example.com
    clientId: kubernetes
    usernameClaim: email
    usernamePrefix: my-username-prefix
    groupsClaim: my-claim
    groupsPrefix: my-groups-prefix
```

```
    requiredClaims:
      string: string
    tags:
      env: dev
```

> **Important**
>
> Don't specify `system:`, or any portion of that string, for `groupsPrefix` or `usernamePrefix`.

2. Create the provider.

```
eksctl associate identityprovider -f associate-identity-provider.yaml
```

3. To use `kubectl` to work with your cluster and OIDC identity provider, see Using kubectl in the Kubernetes documentation.

Associate an identity provider using the AWS Console

1. Open the Amazon EKS console.
2. Select your cluster, and then select the **Access** tab.
3. In the **OIDC Identity Providers** section, select* Associate Identity Provider*.
4. On the **Associate OIDC Identity Provider** page, enter or select the following options, and then select **Associate**.
   - For **Name**, enter a unique name for the provider.
   - For **Issuer URL**, enter the URL for your provider. This URL must be accessible over the internet.
   - For **Client ID**, enter the OIDC identity provider's client ID (also known as **audience**).
   - For **Username claim**, enter the claim to use as the username.
   - For **Groups claim**, enter the claim to use as the user's group.
   - (Optional) Select **Advanced options**, enter or select the following information.
     - **Username prefix** – Enter a prefix to prepend to username claims. The prefix is prepended to username claims to prevent clashes with existing names. If you do not provide a value, and the username is a value other than `email`, the prefix defaults to the value for **Issuer URL**. You can use the value `-` to disable all prefixing. Don't specify `system:` or any portion of that string.

- **Groups prefix** – Enter a prefix to prepend to groups claims. The prefix is prepended to group claims to prevent clashes with existing names (such as `system: groups`). For example, the value `oidc:` creates group names like `oidc:engineering` and `oidc:infra`. Don't specify `system:` or any portion of that string..
- **Required claims** – Select **Add claim** and enter one or more key value pairs that describe required claims in the client ID token. The pairs describe required claims in the ID Token. If set, each claim is verified to be present in the ID token with a matching value.

   a. To use `kubectl` to work with your cluster and OIDC identity provider, see Using kubectl in the Kubernetes documentation.

Example IAM policy

If you want to prevent an OIDC identity provider from being associated with a cluster, create and associate the following IAM policy to the IAM accounts of your Amazon EKS administrators. For more information, see Creating IAM policies and Adding IAM identity permissions in the *IAM User Guide* and Actions in the Service Authorization Reference.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "denyOIDC",
            "Effect": "Deny",
            "Action": [
                "eks:AssociateIdentityProviderConfig"
            ],
            "Resource": "arn:aws:eks:us-west-2.amazonaws.com:111122223333:cluster/*"

        },
        {
            "Sid": "eksAdmin",
            "Effect": "Allow",
            "Action": [
                "eks:*"
            ],
            "Resource": "*"
        }
    ]
}
```

The following example policy allows OIDC identity provider association if the `clientID` is `kubernetes` and the `issuerUrl` is [`https://cognito-idp.us-west-2amazonaws.com/*`](https://cognito-idp.us-west-2amazonaws.com/*)
.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCognitoOnly",
            "Effect": "Deny",
            "Action": "eks:AssociateIdentityProviderConfig",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-instance",
            "Condition": {
                "StringNotLikeIfExists": {
                    "eks:issuerUrl": "https://cognito-idp.us-west-2.amazonaws.com/*"
                }
            }
        },
        {
            "Sid": "DenyOtherClients",
            "Effect": "Deny",
            "Action": "eks:AssociateIdentityProviderConfig",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-instance",
            "Condition": {
                "StringNotEquals": {
                    "eks:clientId": "kubernetes"
                }
            }
        },
        {
            "Sid": "AllowOthers",
            "Effect": "Allow",
            "Action": "eks:*",
            "Resource": "*"
        }
    ]
}
```

Disassociate an OIDC identity provider from your cluster

If you disassociate an OIDC identity provider from your cluster, users included in the provider can no longer access the cluster. However, you can still access the cluster with IAM principals.

1. Open the Amazon EKS console.
2. In the **OIDC Identity Providers** section, select **Disassociate**, enter the identity provider name, and then select `Disassociate`.

View Kubernetes resources in the AWS Management Console

You can view the Kubernetes resources deployed to your cluster with the AWS Management Console. You can't view Kubernetes resources with the AWS CLI or eksctl. To view Kubernetes resources using a command-line tool, use kubectl.

> **Note**
>
> To view the **Resources** tab and **Nodes** section on the **Compute** tab in the AWS Management Console, the IAM principal that you're using must have specific IAM and Kubernetes permissions. For more information, see the section called "Required permissions".

1. Open the Amazon EKS console.
2. In the **Clusters** list, select the cluster that contains the Kubernetes resources that you want to view.
3. Select the **Resources** tab.
4. Select a **Resource type** group that you want to view resources for, such as **Workloads**. You see a list of resource types in that group.
5. Select a resource type, such as **Deployments**, in the **Workloads** group. You see a description of the resource type, a link to the Kubernetes documentation for more information about the resource type, and a list of resources of that type that are deployed on your cluster. If the list is empty, then there are no resources of that type deployed to your cluster.
6. Select a resource to view more information about it. Try the following examples:
   - Select the **Workloads** group, select the **Deployments** resource type, and then select the **coredns** resource. When you select a resource, you are in **Structured view**, by default. For some resource types, you see a **Pods** section in **Structured view**. This section lists the Pods managed by the workload. You can select any Pod listed to view information about the Pod. Not all resource types display information in **Structured View**. If you select **Raw view** in the top right corner of the page for the resource, you see the complete JSON response from the Kubernetes API for the resource.

- Select the **Cluster** group and then select the **Nodes** resource type. You see a list of all nodes in your cluster. The nodes can be any Amazon EKS node type. This is the same list that you see in the **Nodes** section when you select the **Compute** tab for your cluster. Select a node resource from the list. In **Structured view**, you also see a **Pods** section. This section shows you all Pods running on the node.

Required permissions

To view the **Resources** tab and **Nodes** section on the **Compute** tab in the AWS Management Console, the IAM principal that you're using must have specific minimum IAM and Kubernetes permissions. Complete the following steps to assign the required permissions to your IAM principals.

1. Make sure that the `eks:AccessKubernetesApi`, and other necessary IAM permissions to view Kubernetes resources, are assigned to the IAM principal that you're using. For more information about how to edit permissions for an IAM principal, see Controlling access for principals in the IAM User Guide. For more information about how to edit permissions for a role, see Modifying a role permissions policy (console) in the IAM User Guide.

   The following example policy includes the necessary permissions for a principal to view Kubernetes resources for all clusters in your account. Replace *111122223333* with your AWS account ID.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:ListFargateProfiles",
                "eks:DescribeNodegroup",
                "eks:ListNodegroups",
                "eks:ListUpdates",
                "eks:AccessKubernetesApi",
                "eks:ListAddons",
                "eks:DescribeCluster",
                "eks:DescribeAddonVersions",
                "eks:ListClusters",
                "eks:ListIdentityProviderConfigs",
                "iam:ListRoles"
```

```
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ssm:GetParameter",
            "Resource": "arn:aws:ssm:*:111122223333:parameter/*"
        }
    ]
}
```

To view nodes in connected clusters, the Amazon EKS connector IAM role should be able to impersonate the principal in the cluster. This allows the Amazon EKS Connector to map the principal to a Kubernetes user.

2. Create a Kubernetes `rolebinding` or `clusterrolebinding` that is bound to a Kubernetes `role` or `clusterrole` that has the necessary permissions to view the Kubernetes resources. To learn more about Kubernetes roles and role bindings, see Using RBAC Authorization in the Kubernetes documentation. You can apply one of the following manifests to your cluster that create a `role` and `rolebinding` or a `clusterrole` and `clusterrolebinding` with the necessary Kubernetes permissions:

   View Kubernetes resources in all namespaces

   - The group name in the file is `eks-console-dashboard-full-access-group`. Apply the manifest to your cluster with the following command:

     ```
     kubectl apply -f https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-full-access.yaml
     ```

   View Kubernetes resources in a specific namespace

   - The namespace in this file is `default`. The group name in the file is `eks-console-dashboard-restricted-access-group`. Apply the manifest to your cluster with the following command:

     ```
     kubectl apply -f https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-restricted-access.yaml
     ```

     If you need to change the Kubernetes group name, namespace, permissions, or any other configuration in the file, then download the file and edit it before applying it to your cluster:

a. Download the file with one of the following commands:

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-full-access.yaml
```

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-restricted-access.yaml
```

b. Edit the file as necessary.

c. Apply the manifest to your cluster with one of the following commands:

```
kubectl apply -f eks-console-full-access.yaml
```

```
kubectl apply -f eks-console-restricted-access.yaml
```

3. Map the IAM principal to the Kubernetes user or group in the `aws-auth` `ConfigMap`. You can use a tool such as `eksctl` to update the `ConfigMap` or you can update it manually by editing it.

> **Important**
>
> We recommend using `eksctl`, or another tool, to edit the `ConfigMap`. For information about other tools you can use, see Use tools to make changes to the aws-authConfigMap in the Amazon EKS best practices guides. An improperly formatted `aws-auth` `ConfigMap` can cause you to lose access to your cluster.

Edit with eksctl

1. You need version `0.204.0` or later of the `eksctl` command line tool installed on your device or AWS CloudShell. To install or update `eksctl`, see Installation in the `eksctl` documentation.

2. View the current mappings in the `ConfigMap`. Replace *`my-cluster`* with the name of your cluster. Replace *`region-code`* with the AWS Region that your cluster is in.

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

An example output is as follows.

```
ARN
            USERNAME                                GROUPS
 ACCOUNT
arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-
NodeInstanceRole-1XLS7754U3ZPA    system:node:{{EC2PrivateDNSName}}
 system:bootstrappers,system:nodes
```

3. Add a mapping for a role. This example assume that you attached the IAM permissions in the first step to a role named *my-console-viewer-role*. Replace *111122223333* with your account ID.

```
eksctl create iamidentitymapping \
    --cluster my-cluster \
    --region=region-code \
    --arn arn:aws:iam::111122223333:role/my-console-viewer-role \
    --group eks-console-dashboard-full-access-group \
    --no-duplicate-arns
```

Important

The role ARN can't include a path such as `role/my-team/developers/my-role`. The format of the ARN must be `arn:aws:iam::`*111122223333*`:role/`*my-role* . In this example, `my-team/developers/` needs to be removed.

An example output is as follows.

```
[...]
2022-05-09 14:51:20 [#]  adding identity "arn:aws:iam::111122223333:role/my-console-
viewer-role" to auth ConfigMap
```

4. Add a mapping for a user. IAM best practices recommend that you grant permissions to roles instead of users. This example assume that you attached the IAM permissions in the first step to a user named *my-user*. Replace *111122223333* with your account ID.

```
eksctl create iamidentitymapping \
    --cluster my-cluster \
    --region=region-code \
    --arn arn:aws:iam::111122223333:user/my-user \
```

```
    --group eks-console-dashboard-restricted-access-group \
    --no-duplicate-arns
```

An example output is as follows.

```
[...]
2022-05-09 14:53:48 [#]  adding identity "arn:aws:iam::111122223333:user/my-user" to
 auth ConfigMap
```

5. View the mappings in the `ConfigMap` again.

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

An example output is as follows.

```
ARN
           USERNAME                                        GROUPS
      ACCOUNT
arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-
NodeInstanceRole-1XLS7754U3ZPA    system:node:{{EC2PrivateDNSName}}
 system:bootstrappers,system:nodes
arn:aws:iam::111122223333:role/my-console-viewer-role
                                                   eks-console-dashboard-full-access-
group
arn:aws:iam::111122223333:user/my-user
                                                    eks-console-dashboard-restricted-
access-group
```

Edit ConfigMap manually

For more information about adding users or roles to the `aws-auth` `ConfigMap`, see the section called "Add IAM principals to your Amazon EKS cluster".

1. Open the `aws-auth` `ConfigMap` for editing.

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

2. Add the mappings to the `aws-auth` `ConfigMap`, but don't replace any of the existing mappings. The following example adds mappings between IAM principals with permissions added in the first step and the Kubernetes groups created in the previous step:

- The *my-console-viewer-role* role and the `eks-console-dashboard-full-access-group`.
- The *my-user* user and the `eks-console-dashboard-restricted-access-group`.

  These examples assume that you attached the IAM permissions in the first step to a role named *my-console-viewer-role* and a user named *my-user*. Replace *111122223333* with your AWS account ID.

```
apiVersion: v1
data:
mapRoles: |
  - groups:
    - eks-console-dashboard-full-access-group
    rolearn: arn:aws:iam::111122223333:role/my-console-viewer-role
    username: my-console-viewer-role
mapUsers: |
  - groups:
    - eks-console-dashboard-restricted-access-group
    userarn: arn:aws:iam::111122223333:user/my-user
    username: my-user
```

> **Important**
>
> The role ARN can't include a path such as `role/my-team/developers/my-console-viewer-role`. The format of the ARN must be `arn:aws:iam::`*111122223333*`:role/`*my-console-viewer-role*. In this example, `my-team/developers/` needs to be removed.

3. Save the file and exit your text editor.

Connect kubectl to an EKS cluster by creating a kubeconfig file

In this topic, you create a `kubeconfig` file for your cluster (or update an existing one).

The `kubectl` command-line tool uses configuration information in `kubeconfig` files to communicate with the API server of a cluster. For more information, see Organizing Cluster Access Using kubeconfig Files in the Kubernetes documentation.

Amazon EKS uses the `aws eks get-token` command with `kubectl` for cluster authentication. By default, the AWS CLI uses the same credentials that are returned with the following command:

```
aws sts get-caller-identity
```

- An existing Amazon EKS cluster. To deploy one, see *Get started*.
- The `kubectl` command line tool is installed on your device or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. For example, if your cluster version is `1.29`, you can use `kubectl` version `1.28`, `1.29`, or `1.30` with it. To install or upgrade `kubectl`, see the section called "Set up kubectl and eksctl".
- Version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version, use `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*. The AWS CLI version that is installed in AWS CloudShell might also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the *AWS CloudShell User Guide*.
- An IAM user or role with permission to use the `eks:DescribeCluster` API action for the cluster that you specify. For more information, see the section called "Identity-based policies". If you use an identity from your own OpenID Connect provider to access your cluster, then see Using kubectl in the Kubernetes documentation to create or update your `kube config` file.

Create `kubeconfig` file automatically

- Version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version, use `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*. The AWS CLI version that is installed in AWS CloudShell might also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the *AWS CloudShell User Guide*.
- Permission to use the `eks:DescribeCluster` API action for the cluster that you specify. For more information, see the section called "Identity-based policies".

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

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

   By default, the resulting configuration file is created at the default `kubeconfig` path (`.kube`) in your home directory or merged with an existing `config` file at that location. You can specify another path with the `--kubeconfig` option.

   You can specify an IAM role ARN with the `--role-arn` option to use for authentication when you issue `kubectl` commands. Otherwise, the IAM principal 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 all available options, run the `aws eks update-kubeconfig help` command or see update-kubeconfig in the *AWS CLI Command Reference*.

2. Test your configuration.

```
kubectl get svc
```

   An example output is as follows.

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

   If you receive any authorization or resource type errors, see the section called "Unauthorized or access denied (`kubectl`)" in the troubleshooting topic.

Grant Kubernetes workloads access to AWS using Kubernetes Service Accounts

A Kubernetes service account provides an identity for processes that run in a Pod. For more information see Managing Service Accounts in the Kubernetes documentation. If your Pod needs access to AWS services, you can map the service account to an AWS Identity and Access Management identity to grant that access. For more information, see the section called "Credentials with IRSA" or the section called "Pod Identity".

Service account tokens

The BoundServiceAccountTokenVolume feature is enabled by default in Kubernetes versions. This feature improves the security of service account tokens by allowing workloads running on Kubernetes to request JSON web tokens that are audience, time, and key bound. Service account tokens have an expiration of one hour. In earlier Kubernetes versions, the tokens didn't have an expiration. This means that clients that rely on these tokens must refresh the tokens within an hour. The following Kubernetes client SDKs refresh tokens automatically within the required time frame:

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

If your workload is using an earlier client version, then you must update it. To enable a smooth migration of clients to the newer time-bound service account tokens, Kubernetes adds an extended expiry period to the service account token over the default one hour. For Amazon EKS clusters, the extended expiry period is 90 days. Your Amazon EKS cluster's Kubernetes API server rejects requests with tokens that are greater than 90 days old. We recommend that you check your applications and their dependencies to make sure that the Kubernetes client SDKs are the same or later than the versions listed previously.

When the API server receives requests with tokens that are greater than one hour old, it annotates the API audit log event with `annotations.authentication.k8s.io/stale-token`. The value of the annotation looks like the following example:

```
subject: system:serviceaccount:common:fluent-bit, seconds after warning threshold:
 4185802.
```

If your cluster has control plane logging enabled, then the annotations are in the audit logs. You can use the following CloudWatch Logs Insights query to identify all the Pods in your Amazon EKS cluster that are using stale tokens:

```
fields @timestamp
|filter @logStream like /kube-apiserver-audit/
|filter @message like /seconds after warning threshold/
|parse @message "subject: *, seconds after warning threshold:*\"" as subject,
 elapsedtime
```

The `subject` refers to the service account that the Pod used. The `elapsedtime` indicates the elapsed time (in seconds) after reading the latest token. The requests to the API server are denied when the `elapsedtime` exceeds 90 days (7,776,000 seconds). You should proactively update your applications' Kubernetes client SDK to use one of the version listed previously that automatically refresh the token. If the service account token used is close to 90 days and you don't have sufficient time to update your client SDK versions before token expiration, then you can terminate existing Pods and create new ones. This results in refetching of the service account token, giving you an additional 90 days to update your client version SDKs.

If the Pod is part of a deployment, the suggested way to terminate Pods while keeping high availability is to perform a roll out with the following command. Replace *my-deployment* with the name of your deployment.

```
kubectl rollout restart deployment/my-deployment
```

Cluster add-ons

The following cluster add-ons have been updated to use the Kubernetes client SDKs that automatically refetch service account tokens. We recommend making sure that the listed versions, or later versions, are installed on your cluster.

- Amazon VPC CNI plugin for Kubernetes and metrics helper plugins version `1.8.0` and later. To check your current version or update it, see the section called "Amazon VPC CNI" and cni-metrics-helper.
- CoreDNS version `1.8.4` and later. To check your current version or update it, see the section called "CoreDNS".
- AWS Load Balancer Controller version `2.0.0` and later. To check your current version or update it, see the section called " AWS Load Balancer Controller".
- A current `kube-proxy` version. To check your current version or update it, see the section called "kube-proxy".

- AWS for Fluent Bit version `2.25.0` or later. To update your current version, see Releases on GitHub.
- Fluentd image version 1.14.6-1.2 or later and Fluentd filter plugin for Kubernetes metadata version 2.11.1 or later.

Granting AWS Identity and Access Management permissions to workloads on Amazon Elastic Kubernetes Service clusters

Amazon EKS provides two ways to grant AWS Identity and Access Management permissions to workloads that run in Amazon EKS clusters: *IAM roles for service accounts*, and *EKS Pod Identities*.

IAM roles for service accounts

IAM roles for service accounts (IRSA) configures Kubernetes applications running on AWS with fine-grained IAM permissions to access various other AWS resources such as Amazon S3 buckets, Amazon DynamoDB tables, and more. You can run multiple applications together in the same Amazon EKS cluster, and ensure each application has only the minimum set of permissions that it needs. IRSA was build to support various Kubernetes deployment options supported by AWS such as Amazon EKS, Amazon EKS Anywhere, Red Hat OpenShift Service on AWS, and self managed Kubernetes clusters on Amazon EC2 instances. Thus, IRSA was build using foundational AWS service like IAM, and did not take any direct dependency on the Amazon EKS service and the EKS API. For more information, see the section called "Credentials with IRSA".

EKS Pod Identities

EKS Pod Identity offers cluster administrators a simplified workflow for authenticating applications to access various other AWS resources such as Amazon S3 buckets, Amazon DynamoDB tables, and more. EKS Pod Identity is for EKS only, and as a result, it simplifies how cluster administrators can configure Kubernetes applications to obtain IAM permissions. These permissions can now be easily configured with fewer steps directly through AWS Management Console, EKS API, and AWS CLI, and there isn't any action to take inside the cluster in any Kubernetes objects. Cluster administrators don't need to switch between the EKS and IAM services, or use privileged IAM operations to configure permissions required by your applications. IAM roles can now be used across multiple clusters without the need to update the role trust policy when creating new clusters. IAM credentials supplied by EKS Pod Identity include role session tags, with attributes such as cluster name, namespace, service account name. Role session tags enable administrators to author a single role that can work across service accounts by allowing access to AWS resources based on matching tags. For more information, see the section called “Pod Identity”.

Comparing EKS Pod Identity and IRSA

At a high level, both EKS Pod Identity and IRSA enables you to grant IAM permissions to applications running on Kubernetes clusters. But they are fundamentally different in how you configure them, the limits supported, and features enabled. Below, we compare some of the key facets of both solutions.

Attribute	EKS Pod Identity	IRSA
Role extensibility	You have to setup each role once to establish trust with the newly-introduced Amazon EKS service principal `pods.eks.amazonaws.com`. After this one-time step, you don’t need to update the role’s trust policy each time that it is used in a new cluster.	You have to update the IAM role’s trust policy with the new EKS cluster OIDC provider endpoint each time you want to use the role in a new cluster.
Cluster scalability	EKS Pod Identity doesn’t require users to setup IAM OIDC provider, so this limit doesn’t apply.	Each EKS cluster has an OpenID Connect (OIDC) issuer URL associated with it. To use IRSA, a unique OpenID Connect provider needs to be created for each EKS cluster in IAM. IAM has a default global limit of 100 OIDC providers for each AWS account. If you plan to have more than 100 EKS clusters for each AWS account with IRSA, then you will reach the IAM OIDC provider limit.

Attribute	EKS Pod Identity	IRSA
Role scalability	EKS Pod Identity doesn't require users to define trust relationship between IAM role and service account in the trust policy, so this limit doesn't apply.	In IRSA, you define the trust relationship between an IAM role and service account in the role's trust policy. By default, the length of trust policy size is `2048`. This means that you can typically define 4 trust relationships in a single trust policy. While you can get the trust policy length limit increased, you are typically limited to a max of 8 trust relationships within a single trust policy.
Role reusability	AWS STS temporary credentials supplied by EKS Pod Identity include role session tags, such as cluster name, namespace, service account name. Role session tags enable administrators to author a single IAM role that can be used with multiple service accounts, with different effective permission, by allowing access to AWS resources based on tags attached to them. This is also called attribute-based access control (ABAC). For more information, see the section called "Grant Pods access".	AWS STS session tags are not supported. You can reuse a role between clusters but every pod receives all of the permissions of the role.

Attribute	EKS Pod Identity	IRSA
Environments supported	EKS Pod Identity is only available on Amazon EKS.	IRSA can be used such as Amazon EKS, Amazon EKS Anywhere, Red Hat OpenShift Service on AWS, and self managed Kubernetes clusters on Amazon EC2 instances.
EKS versions supported	EKS Kubernetes versions `1.24` or later. For the specific platform versions, see the section called "EKS Pod Identity cluster versions".	All of the supported EKS cluster versions.

IAM roles for service accounts

Applications in a Pod's containers can use an AWS SDK or the AWS CLI to make API requests to AWS services using AWS Identity and Access Management (IAM) permissions. Applications must sign their AWS API requests with AWS credentials. IAM roles for service accounts (IRSA) provide the ability to manage credentials for your applications, similar to the way that Amazon EC2 instance profiles provide credentials to Amazon EC2 instances. Instead of creating and distributing your AWS credentials to the containers or using the Amazon EC2 instance's role, you associate an IAM role with a Kubernetes service account and configure your Pods to use the service account. You can't use IAM roles for service accounts with local clusters for Amazon EKS on AWS Outposts.

IAM roles for service accounts provide the following benefits:

- **Least privilege** – You can scope IAM permissions to a service account, and only Pods that use that service account have access to those permissions. This feature also eliminates the need for third-party solutions such as `kiam` or `kube2iam`.
- **Credential isolation** – A Pod's containers can only retrieve credentials for the IAM role that's associated with the service account that the container uses. A container never has access to credentials that are used by other containers in other Pods. When using IAM roles for service accounts, the Pod's containers also have the permissions assigned to the Amazon EKS node IAM role, unless you block Pod access to the Amazon EC2 Instance Metadata Service (IMDS). For more information, see Restrict access to the instance profile assigned to the worker node.

- **Auditability** – Access and event logging is available through AWS CloudTrail to help ensure retrospective auditing.

Enable IAM roles for service accounts by completing the following procedures:

1. Create an IAM OIDC provider for your cluster – You only complete this procedure once for each cluster.

> **Note**
>
> If you enabled the EKS VPC endpoint, the EKS OIDC service endpoint couldn't be accessed from inside that VPC. Consequently, your operations such as creating an OIDC provider with `eksctl` in the VPC will not work and will result in a timeout when attempting to request `https://oidc.eks.*region*.amazonaws.com`. An example error message follows:
>
> ```
> server cant find oidc.eks.region.amazonaws.com: NXDOMAIN
> ```
>
> To complete this step, you can run the command outside the VPC, for example in AWS CloudShell or on a computer connected to the internet. Alternatively, you can create a split-horizon conditional resolver in the VPC, such as Route 53 Resolver to use a different resolver for the OIDC Issuer URL and not use the VPC DNS for it. For an example of conditional forwarding in CoreDNS, see the Amazon EKS feature request on GitHub.

2. Assign IAM roles to Kubernetes service accounts – Complete this procedure for each unique set of permissions that you want an application to have.
3. Configure Pods to use a Kubernetes service account – Complete this procedure for each Pod that needs access to AWS services.
4. Use IRSA with the AWS SDK – Confirm that the workload uses an AWS SDK of a supported version and that the workload uses the default credential chain.

IAM, Kubernetes, and OpenID Connect (OIDC) background information

In 2014, AWS Identity and Access Management added support for federated identities using OpenID Connect (OIDC). This feature allows you to authenticate AWS API calls with supported identity providers and receive a valid OIDC JSON web token (JWT). You can pass this token to the AWS STS `AssumeRoleWithWebIdentity` API operation and receive IAM temporary role credentials. You can use these credentials to interact with any AWS service, including Amazon S3 and DynamoDB.

Each JWT token is signed by a signing key pair. The keys are served on the OIDC provider managed by Amazon EKS and the private key rotates every 7 days. Amazon EKS keeps the public keys until they expire. If you connect external OIDC clients, be aware that you need to refresh the signing keys before the public key expires. Learn how to Fetch signing keys to validate OIDC tokens.

Kubernetes has long used service accounts as its own internal identity system. Pods can authenticate with the Kubernetes API server using an auto-mounted token (which was a non-OIDC JWT) that only the Kubernetes API server could validate. These legacy service account tokens don't expire, and rotating the signing key is a difficult process. In Kubernetes version `1.12`, support was added for a new `ProjectedServiceAccountToken` feature. This feature is an OIDC JSON web token that also contains the service account identity and supports a configurable audience.

Amazon EKS hosts a public OIDC discovery endpoint for each cluster that contains the signing keys for the `ProjectedServiceAccountToken` JSON web tokens so external systems, such as IAM, can validate and accept the OIDC tokens that are issued by Kubernetes.

Create an IAM OIDC provider for your cluster

Your cluster has an OpenID Connect (OIDC) issuer URL associated with it. To use AWS Identity and Access Management (IAM) roles for service accounts, an IAM OIDC provider must exist for your cluster's OIDC issuer URL.

- An existing Amazon EKS cluster. To deploy one, see *Get started*.
- Version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version, use `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*. The AWS CLI version that is installed in AWS CloudShell might also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the *AWS CloudShell User Guide*.
- The `kubectl` command line tool is installed on your device or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. For example, if your cluster version is `1.29`, you can use `kubectl` version `1.28`, `1.29`, or `1.30` with it. To install or upgrade `kubectl`, see the section called "Set up kubectl and eksctl".

- An existing `kubectl config` file that contains your cluster configuration. To create a `kubectl config` file, see the section called "Access cluster with kubectl".

You can create an IAM OIDC provider for your cluster using `eksctl` or the AWS Management Console.

Create OIDC provider (eksctl)

1. Version `0.204.0` or later of the `eksctl` command line tool installed on your device or AWS CloudShell. To install or update `eksctl`, see Installation in the `eksctl` documentation.
2. Determine the OIDC issuer ID for your cluster.

   Retrieve your cluster's OIDC issuer ID and store it in a variable. Replace *`my-cluster`* with your own value.

   ```
   cluster_name=my-cluster
   ```

```
oidc_id=$(aws eks describe-cluster --name $cluster_name --query
 "cluster.identity.oidc.issuer" --output text | cut -d '/' -f 5)
```

```
echo $oidc_id
```

1. Determine whether an IAM OIDC provider with your cluster's issuer ID is already in your account.

   ```
   aws iam list-open-id-connect-providers | grep $oidc_id | cut -d "/" -f4
   ```

   If output is returned, then you already have an IAM OIDC provider for your cluster and you can skip the next step. If no output is returned, then you must create an IAM OIDC provider for your cluster.

2. Create an IAM OIDC identity provider for your cluster with the following command.

   ```
   eksctl utils associate-iam-oidc-provider --cluster $cluster_name --approve
   ```

> **Note**
>
> If you enabled the EKS VPC endpoint, the EKS OIDC service endpoint couldn't be accessed from inside that VPC. Consequently, your operations such as creating an OIDC provider with `eksctl` in the VPC will not work and will result in a timeout when attempting to request `https://oidc.eks.region.amazonaws.com`. An example error message follows:

```
** server cant find oidc.eks.region.amazonaws.com: NXDOMAIN
```

To complete this step, you can run the command outside the VPC, for example in AWS CloudShell or on a computer connected to the internet. Alternatively, you can create a split-horizon conditional resolver in the VPC, such as Route 53 Resolver to use a different resolver for the OIDC Issuer URL and not use the VPC DNS for it. For an example of conditional forwarding in CoreDNS, see the Amazon EKS feature request on GitHub.

Create OIDC provider (AWS Console)

1. Open the Amazon EKS console.
2. In the left pane, select **Clusters**, and then select the name of your cluster on the **Clusters** page.
3. In the **Details** section on the **Overview** tab, note the value of the **OpenID Connect provider URL**.
4. Open the IAM console at https://console.aws.amazon.com/iam/.
5. In the left navigation pane, choose **Identity Providers** under **Access management**. If a **Provider** is listed that matches the URL for your cluster, then you already have a provider for your cluster. If a provider isn't listed that matches the URL for your cluster, then you must create one.
6. To create a provider, choose **Add provider**.
7. For **Provider type**, select **OpenID Connect**.
8. For **Provider URL**, enter the OIDC provider URL for your cluster.
9. For **Audience**, enter `sts.amazonaws.com`.
10. (Optional) Add any tags, for example a tag to identify which cluster is for this provider.
11. Choose **Add provider**.

Next step: the section called "Assign IAM role"

Assign IAM roles to Kubernetes service accounts

This topic covers how to configure a Kubernetes service account to assume an AWS Identity and Access Management (IAM) role. Any Pods that are configured to use the service account can then access any AWS service that the role has permissions to access.

Prerequisites

- An existing cluster. If you don't have one, you can create one by following one of the guides in *Get started*.
- An existing IAM OpenID Connect (OIDC) provider for your cluster. To learn if you already have one or how to create one, see the section called "IAM OIDC provider".
- Version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version, use `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*. The AWS CLI version that is installed in AWS CloudShell might also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the *AWS CloudShell User Guide*.
- The `kubectl` command line tool is installed on your device or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. For example, if your cluster version is `1.29`, you can use `kubectl` version `1.28`, `1.29`, or `1.30` with it. To install or upgrade `kubectl`, see the section called "Set up kubectl and eksctl".
- An existing `kubectl config` file that contains your cluster configuration. To create a `kubectl config` file, see the section called "Access cluster with kubectl".

Step 1: Create IAM Policy

If you want to associate an existing IAM policy to your IAM role, skip to the next step.

1. Create an IAM policy. You can create your own policy, or copy an AWS managed policy that already grants some of the permissions that you need and customize it to your specific requirements. For more information, see Creating IAM policies in the *IAM User Guide*.

2. Create a file that includes the permissions for the AWS services that you want your Pods to access. For a list of all actions for all AWS services, see the Service Authorization Reference.

   You can run the following command to create an example policy file that allows read-only access to an Amazon S3 bucket. You can optionally store configuration information or a bootstrap script in this bucket, and the containers in your Pod can read the file from the bucket and load it into your application. If you want to create this example policy, copy the following contents to your device. Replace *my-pod-secrets-bucket* with your bucket name and run the command.

```
cat >my-policy.json <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
        }
    ]
}
EOF
```

3. Create the IAM policy.

```
aws iam create-policy --policy-name my-policy --policy-document file://my-policy.json
```

Step 2: Create and associate IAM Role

Create an IAM role and associate it with a Kubernetes service account. You can use either `eksctl` or the AWS CLI.

Create and associate role (eksctl)

Version `0.204.0` or later of the `eksctl` command line tool installed on your device or AWS CloudShell. To install or update `eksctl`, see Installation in the `eksctl` documentation.

Replace *my-service-account* with the name of the Kubernetes service account that you want `eksctl` to create and associate with an IAM role. Replace *default* with the namespace that you want `eksctl` to create the service account in. Replace *my-cluster* with the name of your cluster.

Replace *my-role* with the name of the role that you want to associate the service account to. If it doesn't already exist, `eksctl` creates it for you. Replace *111122223333* with your account ID and *my-policy* with the name of an existing policy.

```
eksctl create iamserviceaccount --name my-service-account --namespace default --cluster
 my-cluster --role-name my-role \
    --attach-policy-arn arn:aws:iam::111122223333:policy/my-policy --approve
```

> **Important**
>
> If the role or service account already exist, the previous command might fail. `eksctl` has different options that you can provide in those situations. For more information run `eksctl create iamserviceaccount --help`.

Create and associate role (AWS CLI)

If you have an existing Kubernetes service account that you want to assume an IAM role, then you can skip this step.

1. Create a Kubernetes service account. Copy the following contents to your device. Replace *my-service-account* with your desired name and *default* with a different namespace, if necessary. If you change *default*, the namespace must already exist.

```
cat >my-service-account.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-service-account
  namespace: default
EOF
kubectl apply -f my-service-account.yaml
```

2. Set your AWS account ID to an environment variable with the following command.

```
account_id=$(aws sts get-caller-identity --query "Account" --output text)
```

3. Set your cluster's OIDC identity provider to an environment variable with the following command. Replace *my-cluster* with the name of your cluster.

```
oidc_provider=$(aws eks describe-cluster --name my-cluster --region $AWS_REGION --
query "cluster.identity.oidc.issuer" --output text | sed -e "s/^https:\/\///")
```

4. Set variables for the namespace and name of the service account. Replace *my-service-account* with the Kubernetes service account that you want to assume the role. Replace *default* with the namespace of the service account.

```
export namespace=default
export service_account=my-service-account
```

5. Run the following command to create a trust policy file for the IAM role. If you want to allow all service accounts within a namespace to use the role, then copy the following contents to your device. Replace *StringEquals* with `StringLike` and replace *$service_account* with `*`. You can add multiple entries in the `StringEquals` or `StringLike` conditions to allow multiple service accounts or namespaces to assume the role. To allow roles from a different AWS account than the account that your cluster is in to assume the role, see the section called “Cross-account IAM” for more information.

```
cat >trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::$account_id:oidc-provider/$oidc_provider"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "$oidc_provider:aud": "sts.amazonaws.com",
          "$oidc_provider:sub": "system:serviceaccount:$namespace:$service_account"
        }
      }
    }
  ]
}
EOF
```

6. Create the role. Replace *my-role* with a name for your IAM role, and *my-role-description* with a description for your role.

```
aws iam create-role --role-name my-role --assume-role-policy-document file://trust-relationship.json --description "my-role-description"
```

7. Attach an IAM policy to your role. Replace *my-role* with the name of your IAM role and *my-policy* with the name of an existing policy that you created.

```
aws iam attach-role-policy --role-name my-role --policy-arn=arn:aws:iam::$account_id:policy/my-policy
```

8. Annotate your service account with the Amazon Resource Name (ARN) of the IAM role that you want the service account to assume. Replace *my-role* with the name of your existing IAM role. Suppose that you allowed a role from a different AWS account than the account that your cluster is in to assume the role in a previous step. Then, make sure to specify the AWS account and role from the other account. For more information, see the section called "Cross-account IAM".

```
kubectl annotate serviceaccount -n $namespace $service_account eks.amazonaws.com/role-arn=arn:aws:iam::$account_id:role/my-role
```

9. (Optional) Configure the AWS Security Token Service endpoint for a service account. AWS recommends using a regional AWS STS endpoint instead of the global endpoint. This reduces latency, provides built-in redundancy, and increases session token validity.

Step 3: Confirm configuration

1. Confirm that the IAM role's trust policy is configured correctly.

```
aws iam get-role --role-name my-role --query Role.AssumeRolePolicyDocument
```

An example output is as follows.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:default:my-service-
account",
                    "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}
```

2. Confirm that the policy that you attached to your role in a previous step is attached to the role.

```
aws iam list-attached-role-policies --role-name my-role --query
 AttachedPolicies[].PolicyArn --output text
```

An example output is as follows.

```

               arn:aws:iam::111122223333:policy/my-policy
```

3. Set a variable to store the Amazon Resource Name (ARN) of the policy that you want to use. Replace *my-policy* with the name of the policy that you want to confirm permissions for.

```
export policy_arn=arn:aws:iam::111122223333:policy/my-policy
```

4. View the default version of the policy.

```
aws iam get-policy --policy-arn $policy_arn
```

An example output is as follows.

```
{
    "Policy": {
```

```
        "PolicyName": "my-policy",
        "PolicyId": "EXAMPLEBIOWGLDEXAMPLE",
        "Arn": "arn:aws:iam::111122223333:policy/my-policy",
        "Path": "/",
        "DefaultVersionId": "v1",
        [...]
    }
}
```

5. View the policy contents to make sure that the policy includes all the permissions that your Pod needs. If necessary, replace *1* in the following command with the version that's returned in the previous output.

```
aws iam get-policy-version --policy-arn $policy_arn --version-id v1
```

An example output is as follows.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
        }
    ]
}
```

If you created the example policy in a previous step, then your output is the same. If you created a different policy, then the *example* content is different.

6. Confirm that the Kubernetes service account is annotated with the role.

```
kubectl describe serviceaccount my-service-account -n default
```

An example output is as follows.

```
Name:                    my-service-account
Namespace:               default
Annotations:             eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/my-
role
```

```
Image pull secrets:   <none>
Mountable secrets:    my-service-account-token-qqjfl
Tokens:               my-service-account-token-qqjfl
[...]
```

Next steps

- the section called “Assign to Pod”

Configure Pods to use a Kubernetes service account

If a Pod needs to access AWS services, then you must configure it to use a Kubernetes service account. The service account must be associated to an AWS Identity and Access Management (IAM) role that has permissions to access the AWS services.

- An existing cluster. If you don’t have one, you can create one using one of the guides in *Get started*.
- An existing IAM OpenID Connect (OIDC) provider for your cluster. To learn if you already have one or how to create one, see the section called “IAM OIDC provider”.
- An existing Kubernetes service account that’s associated with an IAM role. The service account must be annotated with the Amazon Resource Name (ARN) of the IAM role. The role must have an associated IAM policy that contains the permissions that you want your Pods to have to use AWS services. For more information about how to create the service account and role, and configure them, see the section called “Assign IAM role”.
- Version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version, use `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*. The AWS CLI version that is installed in AWS CloudShell might also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the *AWS CloudShell User Guide*.
- The `kubectl` command line tool is installed on your device or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. For example, if your cluster version is `1.29`, you can use `kubectl` version `1.28`, `1.29`, or `1.30` with it. To install or upgrade `kubectl`, see the section called “Set up kubectl and eksctl”.

- An existing `kubectl config` file that contains your cluster configuration. To create a `kubectl config` file, see the section called "Access cluster with kubectl".

1. Use the following command to create a deployment manifest that you can deploy a Pod to confirm configuration with. Replace the *`example values`* with your own values.

```
cat >my-deployment.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      serviceAccountName: my-service-account
      containers:
      - name: my-app
        image: public.ecr.aws/nginx/nginx:X.XX
EOF
```

2. Deploy the manifest to your cluster.

```
kubectl apply -f my-deployment.yaml
```

3. Confirm that the required environment variables exist for your Pod.

   a. View the Pods that were deployed with the deployment in the previous step.

   ```
   kubectl get pods | grep my-app
   ```

   An example output is as follows.

   ```
   my-app-6f4dfff6cb-76cv9   1/1     Running   0          3m28s
   ```

   b. View the ARN of the IAM role that the Pod is using.

```
kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep AWS_ROLE_ARN:
```

An example output is as follows.

```
AWS_ROLE_ARN:                 arn:aws:iam::111122223333:role/my-role
```

The role ARN must match the role ARN that you annotated the existing service account with. For more about annotating the service account, see the section called “Assign IAM role”.

c. Confirm that the Pod has a web identity token file mount.

```
kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep AWS_WEB_IDENTITY_TOKEN_FILE:
```

An example output is as follows.

```
AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/serviceaccount/
token
```

The `kubelet` requests and stores the token on behalf of the Pod. By default, the `kubelet` refreshes the token if the token is older than 80 percent of its total time to live or older than 24 hours. You can modify the expiration duration for any account other than the default service account by using the settings in your Pod spec. For more information, see Service Account Token Volume Projection in the Kubernetes documentation.

The Amazon EKS Pod Identity Webhook on the cluster watches for Pods that use a service account with the following annotation:

```
eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/my-role
```

The webhook applies the previous environment variables to those Pods. Your cluster doesn’t need to use the webhook to configure the environment variables and token file mounts. You can manually configure Pods to have these environment variables. The supported versions of the AWS SDK look for these environment variables first in the credential chain provider. The role credentials are used for Pods that meet this criteria.

4. Confirm that your Pods can interact with the AWS services using the permissions that you assigned in the IAM policy attached to your role.

> **Note**
>
> When a Pod uses AWS credentials from an IAM role that's associated with a service account, the AWS CLI or other SDKs in the containers for that Pod use the credentials that are provided by that role. If you don't restrict access to the credentials that are provided to the Amazon EKS node IAM role, the Pod still has access to these credentials. For more information, see Restrict access to the instance profile assigned to the worker node.

If your Pods can't interact with the services as you expected, complete the following steps to confirm that everything is properly configured.

a. Confirm that your Pods use an AWS SDK version that supports assuming an IAM role through an OpenID Connect web identity token file. For more information, see the section called "Supported SDKs".

b. Confirm that the deployment is using the service account.

```
kubectl describe deployment my-app | grep "Service Account"
```

An example output is as follows.

```
Service Account:  my-service-account
```

c. If your Pods still can't access services, review the steps that are described in Assign IAM roles to Kubernetes service accounts to confirm that your role and service account are configured properly.

Configure the AWS Security Token Service endpoint for a service account

If you're using a Kubernetes service account with IAM roles for service accounts, then you can configure the type of AWS Security Token Service endpoint that's used by the service account if your cluster and platform version are the same or later than those listed in the following table. If your Kubernetes or platform version are earlier than those listed in the table, then your service accounts can only use the global endpoint.

Kubernetes version	Platform version	Default endpoint type
1.31	eks.4	Regional
1.30	eks.2	Regional
1.29	eks.1	Regional
1.28	eks.1	Regional
1.27	eks.1	Regional
1.26	eks.1	Regional
1.25	eks.1	Regional
1.24	eks.2	Regional
1.23	eks.1	Regional

AWS recommends using the regional AWS STS endpoints instead of the global endpoint. This reduces latency, provides built-in redundancy, and increases session token validity. The AWS Security Token Service must be active in the AWS Region where the Pod is running. Moreover, your application must have built-in redundancy for a different AWS Region in the event of a failure of the service in the AWS Region. For more information, see Managing AWS STS in an AWS Region in the IAM User Guide.

- An existing cluster. If you don't have one, you can create one using one of the guides in *Get started*.
- An existing IAM OIDC provider for your cluster. For more information, see the section called "IAM OIDC provider".
- An existing Kubernetes service account configured for use with the Amazon EKS IAM for service accounts feature.

The following examples all use the aws-node Kubernetes service account used by the Amazon VPC CNI plugin. You can replace the `example values` with your own service accounts, Pods, namespaces, and other resources.

1. Select a Pod that uses a service account that you want to change the endpoint for. Determine which AWS Region that the Pod runs in. Replace *aws-node-6mfgv* with your Pod name and *kube-system* with your Pod's namespace.

```
kubectl describe pod aws-node-6mfgv -n kube-system |grep Node:
```

An example output is as follows.

```
ip-192-168-79-166.us-west-2/192.168.79.166
```

In the previous output, the Pod is running on a node in the us-west-2 AWS Region.

2. Determine the endpoint type that the Pod's service account is using.

```
kubectl describe pod aws-node-6mfgv -n kube-system |grep AWS_STS_REGIONAL_ENDPOINTS
```

An example output is as follows.

```
AWS_STS_REGIONAL_ENDPOINTS: regional
```

If the current endpoint is global, then `global` is returned in the output. If no output is returned, then the default endpoint type is in use and has not been overridden.

3. If your cluster or platform version are the same or later than those listed in the table, then you can change the endpoint type used by your service account from the default type to a different type with one of the following commands. Replace *aws-node* with the name of your service account and *kube-system* with the namespace for your service account.

   - If your default or current endpoint type is global and you want to change it to regional:

   ```
   kubectl annotate serviceaccount -n kube-system aws-node eks.amazonaws.com/sts-regional-endpoints=true
   ```

   If you're using IAM roles for service accounts to generate pre-signed S3 URLs in your application running in Pods' containers, the format of the URL for regional endpoints is similar to the following example:

   ```
   https://bucket.s3.us-west-2.amazonaws.com/path?...&X-Amz-Credential=your-access-key-id/date/us-west-2/s3/aws4_request&...
   ```

- If your default or current endpoint type is regional and you want to change it to global:

```
kubectl annotate serviceaccount -n kube-system aws-node eks.amazonaws.com/sts-regional-endpoints=false
```

  If your application is explicitly making requests to AWS STS global endpoints and you don't override the default behavior of using regional endpoints in Amazon EKS clusters, then requests will fail with an error. For more information, see the section called "Pod containers receive the following error: `An error occurred (SignatureDoesNotMatch) when calling the GetCallerIdentity operation: Credential should be scoped to a valid region`".

  If you're using IAM roles for service accounts to generate pre-signed S3 URLs in your application running in Pods' containers, the format of the URL for global endpoints is similar to the following example:

```
https://bucket.s3.amazonaws.com/path?...&X-Amz-Credential=your-access-key-id/date/us-west-2/s3/aws4_request&...
```

If you have automation that expects the pre-signed URL in a certain format or if your application or downstream dependencies that use pre-signed URLs have expectations for the AWS Region targeted, then make the necessary changes to use the appropriate AWS STS endpoint.

4. Delete and re-create any existing Pods that are associated with the service account to apply the credential environment variables. The mutating web hook doesn't apply them to Pods that are already running. You can replace *`Pods`*, *`kube-system`*, and *`-l k8s-app=aws-node`* with the information for the Pods that you set your annotation for.

```
kubectl delete Pods -n kube-system -l k8s-app=aws-node
```

5. Confirm that the all Pods restarted.

```
kubectl get Pods -n kube-system -l k8s-app=aws-node
```

6. View the environment variables for one of the Pods. Verify that the `AWS_STS_REGIONAL_ENDPOINTS` value is what you set it to in a previous step.

```
kubectl describe pod aws-node-kzbtr -n kube-system |grep AWS_STS_REGIONAL_ENDPOINTS
```

An example output is as follows.

```
AWS_STS_REGIONAL_ENDPOINTS=regional
```

Authenticate to another account with IRSA

You can configure cross-account IAM permissions either by creating an identity provider from another account's cluster or by using chained `AssumeRole` operations. In the following examples, *Account A* owns an Amazon EKS cluster that supports IAM roles for service accounts. Pods that are running on that cluster must assume IAM permissions from *Account B*.

Example Create an identity provider from another account's cluster

Example

In this example, Account A provides Account B with the OpenID Connect (OIDC) issuer URL from their cluster. Account B follows the instructions in Create an IAM OIDC provider for your cluster and the section called "Assign IAM role" using the OIDC issuer URL from Account A's cluster. Then, a cluster administrator annotates the service account in Account A's cluster to use the role from Account B (*444455556666*).

```
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::444455556666:role/account-b-role
```

Example Use chained `AssumeRole` operations

Example

In this example, Account B creates an IAM policy with the permissions to give to Pods in Account A's cluster. Account B (*444455556666*) attaches that policy to an IAM role with a trust relationship that allows `AssumeRole` permissions to Account A (*111122223333*).

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {}
    }
  ]
}
```

Account A creates a role with a trust policy that gets credentials from the identity provider created with the cluster's OIDC issuer address.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.region-
code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity"
    }
  ]
}
```

Account A attaches a policy to that role with the following permissions to assume the role that Account B created.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::444455556666:role/account-b-role"
        }
    ]
}
```

The application code for Pods to assume Account B's role uses two profiles: `account_b_role` and `account_a_role`. The `account_b_role` profile uses the `account_a_role` profile as its source. For the AWS CLI, the `~/.aws/config` file is similar to the following.

```
[profile account_b_role]
source_profile = account_a_role
role_arn=arn:aws:iam::444455556666:role/account-b-role

[profile account_a_role]
web_identity_token_file = /var/run/secrets/eks.amazonaws.com/serviceaccount/token
role_arn=arn:aws:iam::111122223333:role/account-a-role
```

To specify chained profiles for other AWS SDKs, consult the documentation for the SDK that you're using. For more information, see Tools to Build on AWS.

Use IRSA with the AWS SDK

Using the credentials

To use the credentials from IAM roles for service accounts (IRSA), your code can use any AWS SDK to create a client for an AWS service with an SDK, and by default the SDK searches in a chain of locations for AWS Identity and Access Management credentials to use. The IAM roles for service accounts credentials will be used if you don't specify a credential provider when you create the client or otherwise initialized the SDK.

This works because IAM roles for service accounts have been added as a step in the default credential chain. If your workloads currently use credentials that are earlier in the chain of credentials, those credentials will continue to be used even if you configure an IAM roles for service accounts for the same workload.

The SDK automatically exchanges the service account OIDC token for temporary credentials from AWS Security Token Service by using the `AssumeRoleWithWebIdentity` action. Amazon EKS and this SDK action continue to rotate the temporary credentials by renewing them before they expire.

When using IAM roles for service accounts, the containers in your Pods must use an AWS SDK version that supports assuming an IAM role through an OpenID Connect web identity token file. Make sure that you're using the following versions, or later, for your AWS SDK:

- Java (Version 2) – 2.10.11
- Java – 1.11.704

- Go – 1.23.13
- Python (Boto3) – 1.9.220
- Python (botocore) – 1.12.200
- AWS CLI – 1.16.232
- Node – 2.525.0 and 3.27.0
- Ruby – 3.58.0
- C++ – 1.7.174
- .NET – 3.3.659.1 – You must also include `AWSSDK.SecurityToken`.
- PHP – 3.110.7

Many popular Kubernetes add-ons, such as the Cluster Autoscaler, the Route internet traffic with AWS Load Balancer Controller, and the Amazon VPC CNI plugin for Kubernetes support IAM roles for service accounts.

To ensure that you're using a supported SDK, follow the installation instructions for your preferred SDK at Tools to Build on AWS when you build your containers.

Fetch signing keys to validate OIDC tokens

Kubernetes issues a `ProjectedServiceAccountToken` to each Kubernetes Service Account. This token is an OIDC token, which is further a type of JSON web token (JWT). Amazon EKS hosts a public OIDC endpoint for each cluster that contains the signing keys for the token so external systems can validate it.

To validate a `ProjectedServiceAccountToken`, you need to fetch the OIDC public signing keys, also called the JSON Web Key Set (JWKS). Use these keys in your application to validate the token. For example, you can use the PyJWT Python library to validate tokens using these keys. For more information on the `ProjectedServiceAccountToken`, see the section called "IAM, Kubernetes, and OpenID Connect (OIDC) background information".

Prerequisites

- An existing AWS Identity and Access Management (IAM) OpenID Connect (OIDC) provider for your cluster. To determine whether you already have one, or to create one, see the section called "IAM OIDC provider".
- **AWS CLI** — A command line tool for working with AWS services, including Amazon EKS. For more information, see Installing in the AWS Command Line Interface User Guide. After installing

the AWS CLI, we recommend that you also configure it. For more information, see Quick configuration with aws configure in the AWS Command Line Interface User Guide.

Procedure

1. Retrieve the OIDC URL for your Amazon EKS cluster using the AWS CLI.

```
$ aws eks describe-cluster --name my-cluster --query 'cluster.identity.oidc.issuer'
"https://oidc.eks.us-west-2.amazonaws.com/id/8EBDXXXX00BAE"
```

2. Retrieve the public signing key using curl, or a similar tool. The result is a JSON Web Key Set (JWKS).

> **Important**
>
> Amazon EKS throttles calls to the OIDC endpoint. You should cache the public signing key. Respect the `cache-control` header included in the response.

> **Important**
>
> Amazon EKS rotates the OIDC signing key every seven days.

```
$ curl https://oidc.eks.us-west-2.amazonaws.com/id/8EBDXXXX00BAE/keys
{"keys":
[{"kty":"RSA","kid":"2284XXXX4a40","use":"sig","alg":"RS256","n":"wklbXXXXMVfQ","e":"AQAB"}]
```

Learn how EKS Pod Identity grants pods access to AWS services

Applications in a Pod's containers can use an AWS SDK or the AWS CLI to make API requests to AWS services using AWS Identity and Access Management (IAM) permissions. Applications must sign their AWS API requests with AWS credentials.

EKS Pod Identities provide the ability to manage credentials for your applications, similar to the way that Amazon EC2 instance profiles provide credentials to Amazon EC2 instances. Instead of creating and distributing your AWS credentials to the containers or using the Amazon EC2 instance's role, you associate an IAM role with a Kubernetes service account and configure your Pods to use the service account.

Each EKS Pod Identity association maps a role to a service account in a namespace in the specified cluster. If you have the same application in multiple clusters, you can make identical associations in each cluster without modifying the trust policy of the role.

If a pod uses a service account that has an association, Amazon EKS sets environment variables in the containers of the pod. The environment variables configure the AWS SDKs, including the AWS CLI, to use the EKS Pod Identity credentials.

Benefits of EKS Pod Identities

EKS Pod Identities provide the following benefits:

- **Least privilege** – You can scope IAM permissions to a service account, and only Pods that use that service account have access to those permissions. This feature also eliminates the need for third-party solutions such as `kiam` or `kube2iam`.
- **Credential isolation** – A Pod's containers can only retrieve credentials for the IAM role that's associated with the service account that the container uses. A container never has access to credentials that are used by other containers in other Pods. When using Pod Identities, the Pod's containers also have the permissions assigned to the the section called "Node IAM role", unless you block Pod access to the Amazon EC2 Instance Metadata Service (IMDS). For more information, see Restrict access to the instance profile assigned to the worker node.
- **Auditability** – Access and event logging is available through AWS CloudTrail to help facilitate retrospective auditing.

EKS Pod Identity is a simpler method than the section called "Credentials with IRSA", as this method doesn't use OIDC identity providers. EKS Pod Identity has the following enhancements:

- **Independent operations** – In many organizations, creating OIDC identity providers is a responsibility of different teams than administering the Kubernetes clusters. EKS Pod Identity has clean separation of duties, where all configuration of EKS Pod Identity associations is done in Amazon EKS and all configuration of the IAM permissions is done in IAM.
- **Reusability** – EKS Pod Identity uses a single IAM principal instead of the separate principals for each cluster that IAM roles for service accounts use. Your IAM administrator adds the following principal to the trust policy of any role to make it usable by EKS Pod Identities.

```
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            }
```

- **Scalability** — Each set of temporary credentials are assumed by the EKS Auth service in EKS Pod Identity, instead of each AWS SDK that you run in each pod. Then, the Amazon EKS Pod Identity Agent that runs on each node issues the credentials to the SDKs. Thus the load is reduced to once for each node and isn't duplicated in each pod. For more details of the process, see the section called "How it works".

For more information to compare the two alternatives, see the section called "Workload access to AWS ".

Overview of setting up EKS Pod Identities

Turn on EKS Pod Identities by completing the following procedures:

1. the section called "Set up the Agent" — You only complete this procedure once for each cluster. You do not need to complete this step if EKS Auto Mode is enabled on your cluster.
2. the section called "Assign IAM role" — Complete this procedure for each unique set of permissions that you want an application to have.
3. the section called "Pod service account" — Complete this procedure for each Pod that needs access to AWS services.
4. the section called "Supported SDKs" — Confirm that the workload uses an AWS SDK of a supported version and that the workload uses the default credential chain.

EKS Pod Identity considerations

- You can associate one IAM role to each Kubernetes service account in each cluster. You can change which role is mapped to the service account by editing the EKS Pod Identity association.
- You can only associate roles that are in the same AWS account as the cluster. You can delegate access from another account to the role in this account that you configure for EKS Pod Identities to use. For a tutorial about delegating access and `AssumeRole`, see Delegate access across AWS accounts using IAM roles in the *IAM User Guide*.
- The EKS Pod Identity Agent is required. It runs as a Kubernetes `DaemonSet` on your nodes and only provides credentials to pods on the node that it runs on. For more information about EKS

Pod Identity Agent compatibility, see the following section the section called "EKS Pod Identity restrictions".

- If you are using Security Group for Pods along with Pod Identity Agent, you may need to set the `POD_SECURITY_GROUP_ENFORCING_MODE` Flag for the AWS VPC CNI. For more information on security group for pods considerations, see the section called "Security groups for Pods".
- The EKS Pod Identity Agent uses the `hostNetwork` of the node and it uses port `80` and port `2703` on a link-local address on the node. This address is `169.254.170.23` for IPv4 and `[fd00:ec2::23]` for IPv6 clusters.

  If you disable `IPv6` addresses, or otherwise prevent localhost `IPv6` IP addresses, the agent can't start. To start the agent on nodes that can't use `IPv6`, follow the steps in the section called "Disable IPv6" to disable the `IPv6` configuration.
- If your Pods use a proxy, you must ensure you add `169.254.170.23` for IPv4 and `[fd00:ec2::23]` for IPv6 in the `no_proxy/NO_PROXY` environment variables injected into the pods. Otherwise requests from the application pods to the `eks-pod-identity-agent` DaemonSets would fail as the requests would be sent to the proxy and the proxy wouldn't be able to route the IP.

EKS Pod Identity cluster versions

To use EKS Pod Identities, the cluster must have a platform version that is the same or later than the version listed in the following table, or a Kubernetes version that is later than the versions listed in the table.

Kubernetes version	Platform version
Kubernetes versions not listed	All platform versions support
`1.28`	`eks.4`
`1.27`	`eks.8`
`1.26`	`eks.9`
`1.25`	`eks.10`
`1.24`	`eks.13`

EKS Pod Identity restrictions

EKS Pod Identities are available on the following:

- Amazon EKS cluster versions listed in the previous topic the section called "EKS Pod Identity cluster versions".
- Worker nodes in the cluster that are Linux Amazon EC2 instances.

EKS Pod Identities aren't available on the following:

- AWS Outposts.
- Amazon EKS Anywhere.
- Kubernetes clusters that you create and run on Amazon EC2. The EKS Pod Identity components are only available on Amazon EKS.

You can't use EKS Pod Identities with:

- Pods that run anywhere except Linux Amazon EC2 instances. Linux and Windows pods that run on AWS Fargate (Fargate) aren't supported. Pods that run on Windows Amazon EC2 instances aren't supported.

Understand how EKS Pod Identity works

Amazon EKS Pod Identity associations provide the ability to manage credentials for your applications, similar to the way that Amazon EC2 instance profiles provide credentials to Amazon EC2 instances.

Amazon EKS Pod Identity provides credentials to your workloads with an additional *EKS Auth* API and an agent pod that runs on each node.

In your add-ons, such as *Amazon EKS add-ons* and self-managed controller, operators, and other add-ons, the author needs to update their software to use the latest AWS SDKs. For the list of compatibility between EKS Pod Identity and the add-ons produced by Amazon EKS, see the previous section the section called "EKS Pod Identity restrictions".

Using EKS Pod Identities in your code

In your code, you can use the AWS SDKs to access AWS services. You write code to create a client for an AWS service with an SDK, and by default the SDK searches in a chain of locations for AWS

Identity and Access Management credentials to use. After valid credentials are found, the search is stopped. For more information about the default locations used, see the Credential provider chain in the AWS SDKs and Tools Reference Guide.

EKS Pod Identities have been added to the *Container credential provider* which is searched in a step in the default credential chain. If your workloads currently use credentials that are earlier in the chain of credentials, those credentials will continue to be used even if you configure an EKS Pod Identity association for the same workload. This way you can safely migrate from other types of credentials by creating the association first, before removing the old credentials.

The container credentials provider provides temporary credentials from an agent that runs on each node. In Amazon EKS, the agent is the Amazon EKS Pod Identity Agent and on Amazon Elastic Container Service the agent is the `amazon-ecs-agent`. The SDKs use environment variables to locate the agent to connect to.

In contrast, *IAM roles for service accounts* provides a *web identity* token that the AWS SDK must exchange with AWS Security Token Service by using `AssumeRoleWithWebIdentity`.

How EKS Pod Identity Agent works with a Pod

1. When Amazon EKS starts a new pod that uses a service account with an EKS Pod Identity association, the cluster adds the following content to the Pod manifest:

```
    env:
    - name: AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE
      value: "/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/eks-pod-
identity-token"
    - name: AWS_CONTAINER_CREDENTIALS_FULL_URI
      value: "http://169.254.170.23/v1/credentials"
    volumeMounts:
    - mountPath: "/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/"
      name: eks-pod-identity-token
  volumes:
  - name: eks-pod-identity-token
    projected:
      defaultMode: 420
      sources:
      - serviceAccountToken:
          audience: pods.eks.amazonaws.com
          expirationSeconds: 86400 # 24 hours
          path: eks-pod-identity-token
```

2. Kubernetes selects which node to run the pod on. Then, the Amazon EKS Pod Identity Agent on the node uses the AssumeRoleForPodIdentity action to retrieve temporary credentials from the EKS Auth API.
3. The EKS Pod Identity Agent makes these credentials available for the AWS SDKs that you run inside your containers.
4. You use the SDK in your application without specifying a credential provider to use the default credential chain. Or, you specify the container credential provider. For more information about the default locations used, see the Credential provider chain in the AWS SDKs and Tools Reference Guide.
5. The SDK uses the environment variables to connect to the EKS Pod Identity Agent and retrieve the credentials.

> **Note**
>
> If your workloads currently use credentials that are earlier in the chain of credentials, those credentials will continue to be used even if you configure an EKS Pod Identity association for the same workload.

Set up the Amazon EKS Pod Identity Agent

Amazon EKS Pod Identity associations provide the ability to manage credentials for your applications, similar to the way that Amazon EC2 instance profiles provide credentials to Amazon EC2 instances.

Amazon EKS Pod Identity provides credentials to your workloads with an additional *EKS Auth* API and an agent pod that runs on each node.

> **Tip**
>
> You do not need to install the EKS Pod Identity Agent on EKS Auto Mode Clusters. This capability is built into EKS Auto Mode.

Considerations

- By default, the EKS Pod Identity Agent listens on an `IPv4` and `IPv6` address for pods to request credentials. The agent uses the loopback (localhost) IP address `169.254.170.23` for `IPv4` and the localhost IP address `[fd00:ec2::23]` for `IPv6`.
- If you disable `IPv6` addresses, or otherwise prevent localhost `IPv6` IP addresses, the agent can't start. To start the agent on nodes that can't use `IPv6`, follow the steps in the section called "Disable IPv6" to disable the `IPv6` configuration.

Creating the Amazon EKS Pod Identity Agent

Agent prerequisites

- An existing Amazon EKS cluster. To deploy one, see *Get started*. The cluster version and platform version must be the same or later than the versions listed in EKS Pod Identity cluster versions.
- The node role has permissions for the agent to do the `AssumeRoleForPodIdentity` action in the EKS Auth API. You can use the AWS managed policy: AmazonEKSWorkerNodePolicy or add a custom policy similar to the following:

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

  This action can be limited by tags to restrict which roles can be assumed by pods that use the agent.
- The nodes can reach and download images from Amazon ECR. The container image for the add-on is in the registries listed in View Amazon container image registries for Amazon EKS add-ons.

Note that you can change the image location and provide `imagePullSecrets` for EKS add-ons in the **Optional configuration settings** in the AWS Management Console, and in the `--configuration-values` in the AWS CLI.

- The nodes can reach the Amazon EKS Auth API. For private clusters, the `eks-auth` endpoint in AWS PrivateLink is required.

Setup agent with AWS console

1. Open the [Amazon EKS console](https://console.aws.amazon.com/eks/home#/clusters).
2. In the left navigation pane, select **Clusters**, and then select the name of the cluster that you want to configure the EKS Pod Identity Agent add-on for.
3. Choose the **Add-ons** tab.
4. Choose **Get more add-ons**.
5. Select the box in the top right of the add-on box for EKS Pod Identity Agent and then choose **Next**.
6. On the **Configure selected add-ons settings** page, select any version in the **Version** dropdown list.
7. (Optional) Expand **Optional configuration settings** to enter additional configuration. For example, you can provide an alternative container image location and `ImagePullSecrets`. The JSON Schema with accepted keys is shown in **Add-on configuration schema**.

   Enter the configuration keys and values in **Configuration values**.
8. Choose **Next**.
9. Confirm that the EKS Pod Identity Agent pods are running on your cluster.

   ```
   kubectl get pods -n kube-system | grep 'eks-pod-identity-agent'
   ```

   An example output is as follows.

   ```
   eks-pod-identity-agent-gmqp7                                          1/1     Running
      1 (24h ago)   24h
   eks-pod-identity-agent-prnsh                                          1/1     Running
      1 (24h ago)   24h
   ```

You can now use EKS Pod Identity associations in your cluster. For more information, see the section called “Assign IAM role”.

Setup agent with AWS CLI

1. Run the following AWS CLI command. Replace `my-cluster` with the name of your cluster.

```
aws eks create-addon --cluster-name my-cluster --addon-name eks-pod-identity-agent --
addon-version v1.0.0-eksbuild.1
```

> **Note**
>
> The EKS Pod Identity Agent doesn’t use the `service-account-role-arn` for *IAM roles for service accounts*. You must provide the EKS Pod Identity Agent with permissions in the node role.

2. Confirm that the EKS Pod Identity Agent pods are running on your cluster.

```
kubectl get pods -n kube-system | grep 'eks-pod-identity-agent'
```

An example output is as follows.

```
eks-pod-identity-agent-gmqp7                                          1/1     Running
   1 (24h ago)   24h
eks-pod-identity-agent-prnsh                                          1/1     Running
   1 (24h ago)   24h
```

You can now use EKS Pod Identity associations in your cluster. For more information, see the section called “Assign IAM role”.

Assign an IAM role to a Kubernetes service account

This topic covers how to configure a Kubernetes service account to assume an AWS Identity and Access Management (IAM) role with EKS Pod Identity. Any Pods that are configured to use the service account can then access any AWS service that the role has permissions to access.

To create an EKS Pod Identity association, there is only a single step; you create the association in EKS through the AWS Management Console, AWS CLI, AWS SDKs, AWS CloudFormation and other tools. There isn't any data or metadata about the associations inside the cluster in any Kubernetes objects and you don't add any annotations to the service accounts.

- An existing cluster. If you don't have one, you can create one by following one of the guides in *Get started*.
- The IAM principal that is creating the association must have `iam:PassRole`.
- The latest version of the AWS CLI installed and configured on your device or AWS CloudShell. You can check your current version with `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the AWS Command Line Interface User Guide. The AWS CLI version installed in the AWS CloudShell may also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the AWS CloudShell User Guide.
- The `kubectl` command line tool is installed on your device or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. For example, if your cluster version is `1.29`, you can use `kubectl` version `1.28`, `1.29`, or `1.30` with it. To install or upgrade `kubectl`, see the section called "Set up kubectl and eksctl".
- An existing `kubectl config` file that contains your cluster configuration. To create a `kubectl config` file, see the section called "Access cluster with kubectl".

Create a Pod Identity association (AWS Console)

1. Open the Amazon EKS console.
2. In the left navigation pane, select **Clusters**, and then select the name of the cluster that you want to configure the EKS Pod Identity Agent add-on for.
3. Choose the **Access** tab.
4. In the **Pod Identity associations**, choose **Create**.
5. For the **IAM role**, select the IAM role with the permissions that you want the workload to have.

> **Note**
>
> The list only contains roles that have the following trust policy which allows EKS Pod Identity to use them.

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

`sts:AssumeRole` — EKS Pod Identity uses `AssumeRole` to assume the IAM role before passing the temporary credentials to your pods.

`sts:TagSession` — EKS Pod Identity uses `TagSession` to include *session tags* in the requests to AWS STS.

You can use these tags in the *condition keys* in the trust policy to restrict which service accounts, namespaces, and clusters can use this role.

For a list of Amazon EKS condition keys, see Conditions defined by Amazon Elastic Kubernetes Service in the *Service Authorization Reference*. To learn which actions and resources you can use a condition key with, see Actions defined by Amazon Elastic Kubernetes Service.

6. For the **Kubernetes namespace**, select the Kubernetes namespace that contains the service account and workload. Optionally, you can specify a namespace by name that doesn't exist in the cluster.
7. For the **Kubernetes service account**, select the Kubernetes service account to use. The manifest for your Kubernetes workload must specify this service account. Optionally, you can specify a service account by name that doesn't exist in the cluster.
8. (Optional) For the **Tags**, choose **Add tag** to add metadata in a key and value pair. These tags are applied to the association and can be used in IAM policies.

You can repeat this step to add multiple tags.

9. Choose **Create**.

Create a Pod Identity association (AWS CLI)

1. If you want to associate an existing IAM policy to your IAM role, skip to the next step.

   Create an IAM policy. You can create your own policy, or copy an AWS managed policy that already grants some of the permissions that you need and customize it to your specific requirements. For more information, see Creating IAM policies in the *IAM User Guide*.

   a. Create a file that includes the permissions for the AWS services that you want your Pods to access. For a list of all actions for all AWS services, see the Service Authorization Reference.

      You can run the following command to create an example policy file that allows read-only access to an Amazon S3 bucket. You can optionally store configuration information or a bootstrap script in this bucket, and the containers in your Pod can read the file from the bucket and load it into your application. If you want to create this example policy, copy the following contents to your device. Replace *my-pod-secrets-bucket* with your bucket name and run the command.

```
cat >my-policy.json <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
        }
    ]
}
EOF
```

   b. Create the IAM policy.

```
aws iam create-policy --policy-name my-policy --policy-document file://my-
policy.json
```

2. Create an IAM role and associate it with a Kubernetes service account.

a. If you have an existing Kubernetes service account that you want to assume an IAM role, then you can skip this step.

   Create a Kubernetes service account. Copy the following contents to your device. Replace *my-service-account* with your desired name and *default* with a different namespace, if necessary. If you change *default*, the namespace must already exist.

```
cat >my-service-account.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-service-account
  namespace: default
EOF
kubectl apply -f my-service-account.yaml
```

   Run the following command.

```
kubectl apply -f my-service-account.yaml
```

b. Run the following command to create a trust policy file for the IAM role.

```
cat >trust-relationship.json <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
EOF
```

c. Create the role. Replace *my-role* with a name for your IAM role, and *my-role-description* with a description for your role.

```
aws iam create-role --role-name my-role --assume-role-policy-document file://
trust-relationship.json --description "my-role-description"
```

d. Attach an IAM policy to your role. Replace *my-role* with the name of your IAM role and *my-policy* with the name of an existing policy that you created.

```
aws iam attach-role-policy --role-name my-role --policy-
arn=arn:aws:iam::111122223333:policy/my-policy
```

> **Note**
>
> Unlike IAM roles for service accounts, EKS Pod Identity doesn't use an annotation on the service account.

e. Run the following command to create the association. Replace `my-cluster` with the name of the cluster, replace *my-service-account* with your desired name and *default* with a different namespace, if necessary.

```
aws eks create-pod-identity-association --cluster-name my-cluster --role-arn
 arn:aws:iam::111122223333:role/my-role --namespace default --service-account my-
service-account
```

An example output is as follows.

```
{
    "association": {
        "clusterName": "my-cluster",
        "namespace": "default",
        "serviceAccount": "my-service-account",
        "roleArn": "arn:aws:iam::111122223333:role/my-role",
        "associationArn": "arn:aws::111122223333:podidentityassociation/my-
cluster/a-abcdefghijklmnop1",
        "associationId": "a-abcdefghijklmnop1",
        "tags": {},
        "createdAt": 1700862734.922,
        "modifiedAt": 1700862734.922
    }
```

```
}
```

> **Note**
>
> You can specify a namespace and service account by name that doesn't exist in the cluster. You must create the namespace, service account, and the workload that uses the service account for the EKS Pod Identity association to function.

Confirm configuration

1. Confirm that the IAM role's trust policy is configured correctly.

```
aws iam get-role --role-name my-role --query Role.AssumeRolePolicyDocument
```

An example output is as follows.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow EKS Auth service to assume this role for Pod Identities",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

2. Confirm that the policy that you attached to your role in a previous step is attached to the role.

```
aws iam list-attached-role-policies --role-name my-role --query
 AttachedPolicies[].PolicyArn --output text
```

An example output is as follows.

```
arn:aws:iam::111122223333:policy/my-policy
```

3. Set a variable to store the Amazon Resource Name (ARN) of the policy that you want to use. Replace *my-policy* with the name of the policy that you want to confirm permissions for.

```
export policy_arn=arn:aws:iam::111122223333:policy/my-policy
```

4. View the default version of the policy.

```
aws iam get-policy --policy-arn $policy_arn
```

An example output is as follows.

```
{
    "Policy": {
        "PolicyName": "my-policy",
        "PolicyId": "EXAMPLEBIOWGLDEXAMPLE",
        "Arn": "arn:aws:iam::111122223333:policy/my-policy",
        "Path": "/",
        "DefaultVersionId": "v1",
        [...]
    }
}
```

5. View the policy contents to make sure that the policy includes all the permissions that your Pod needs. If necessary, replace *1* in the following command with the version that's returned in the previous output.

```
aws iam get-policy-version --policy-arn $policy_arn --version-id v1
```

An example output is as follows.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
```

```
        }
    ]
}
```

If you created the example policy in a previous step, then your output is the same. If you created a different policy, then the *example* content is different.

Next Steps

the section called "Pod service account"

Configure Pods to access AWS services with service accounts

If a Pod needs to access AWS services, then you must configure it to use a Kubernetes service account. The service account must be associated to an AWS Identity and Access Management (IAM) role that has permissions to access the AWS services.

- An existing cluster. If you don't have one, you can create one using one of the guides in *Get started*.
- An existing Kubernetes service account and an EKS Pod Identity association that associates the service account with an IAM role. The role must have an associated IAM policy that contains the permissions that you want your Pods to have to use AWS services. For more information about how to create the service account and role, and configure them, see the section called "Assign IAM role".
- The latest version of the AWS CLI installed and configured on your device or AWS CloudShell. You can check your current version with `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the AWS Command Line Interface User Guide. The AWS CLI version installed in the AWS CloudShell may also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the AWS CloudShell User Guide.
- The `kubectl` command line tool is installed on your device or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. For example, if your cluster version is `1.29`, you can use `kubectl` version `1.28`, `1.29`, or `1.30` with it. To install or upgrade `kubectl`, see the section called "Set up kubectl and eksctl".
- An existing `kubectl config` file that contains your cluster configuration. To create a `kubectl config` file, see the section called "Access cluster with kubectl".

1. Use the following command to create a deployment manifest that you can deploy a Pod to confirm configuration with. Replace the *example values* with your own values.

```
cat >my-deployment.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      serviceAccountName: my-service-account
      containers:
      - name: my-app
        image: public.ecr.aws/nginx/nginx:X.XX
EOF
```

2. Deploy the manifest to your cluster.

```
kubectl apply -f my-deployment.yaml
```

3. Confirm that the required environment variables exist for your Pod.

   a. View the Pods that were deployed with the deployment in the previous step.

   ```
   kubectl get pods | grep my-app
   ```

   An example output is as follows.

   ```
   my-app-6f4dfff6cb-76cv9   1/1     Running   0          3m28s
   ```

   b. Confirm that the Pod has a service account token file mount.

   ```
   kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep
    AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE:
   ```

An example output is as follows.

```
AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE:  /var/run/secrets/
pods.eks.amazonaws.com/serviceaccount/eks-pod-identity-token
```

4. Confirm that your Pods can interact with the AWS services using the permissions that you assigned in the IAM policy attached to your role.

> **Note**
>
> When a Pod uses AWS credentials from an IAM role that's associated with a service account, the AWS CLI or other SDKs in the containers for that Pod use the credentials that are provided by that role. If you don't restrict access to the credentials that are provided to the Amazon EKS node IAM role, the Pod still has access to these credentials. For more information, see Restrict access to the instance profile assigned to the worker node.

If your Pods can't interact with the services as you expected, complete the following steps to confirm that everything is properly configured.

a. Confirm that your Pods use an AWS SDK version that supports assuming an IAM role through an EKS Pod Identity association. For more information, see the section called "Supported SDKs".

b. Confirm that the deployment is using the service account.

```
kubectl describe deployment my-app | grep "Service Account"
```

An example output is as follows.

```
Service Account:  my-service-account
```

Grant Pods access to {aws} resources based on tags

Attribute-based access control (ABAC) grants rights to users through policies which combine attributes together. EKS Pod Identity attaches tags to the temporary credentials to each Pod with attributes such as cluster name, namespace, and service account name. These role session tags enable administrators to author a single role that can work across service accounts by allowing access to AWS resources based on matching tags. By adding support for role session tags, customers can enforce tighter security boundaries between clusters, and workloads within clusters, while reusing the same IAM roles and IAM policies.

For example, the following policy allows the `s3:GetObject` action if the object is tagged with the name of the EKS cluster.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectTagging"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/eks-cluster-name": "${aws:PrincipalTag/eks-
cluster-name}"
                }
            }
        }
    ]
}
```

List of session tags added by EKS Pod Identity

The following list contains all of the keys for tags that are added to the `AssumeRole` request made by Amazon EKS. To use these tags in policies, use `${aws:PrincipalTag/` followed by the key, for example `${aws:PrincipalTag/kubernetes-namespace}`.

- `eks-cluster-arn`

- `eks-cluster-name`
- `kubernetes-namespace`
- `kubernetes-service-account`
- `kubernetes-pod-name`
- `kubernetes-pod-uid`

Cross-account tags

All of the session tags that are added by EKS Pod Identity are *transitive*; the tag keys and values are passed to any `AssumeRole` actions that your workloads use to switch roles into another account. You can use these tags in policies in other accounts to limit access in cross-account scenarios. For more infromation, see [Chaining roles with session tags](#) in the *IAM User Guide*.

Custom tags

EKS Pod Identity can't add additional custom tags to the `AssumeRole` action that it performs. However, tags that you apply to the IAM role are always available though the same format: `${aws:PrincipalTag/` followed by the key, for example `${aws:PrincipalTag/MyCustomTag}`.

> **Note**
>
> Tags added to the session through the `sts:AssumeRole` request take precedence in the case of conflict. For example, say that:
>
> - Amazon EKS adds a key `eks-cluster-name` and value `my-cluster` to the session when EKS assumes the customer role and
> - You add an `eks-cluster-name` tag to the IAM role with the value `my-own-cluster`.
>
> In this case, the former takes precedence and the value for the `eks-cluster-name` tag will be `my-cluster`.

Use pod identity with the AWS SDK

Using EKS Pod Identity credentials

To use the credentials from a EKS Pod Identity association, your code can use any AWS SDK to create a client for an AWS service with an SDK, and by default the SDK searches in a chain of locations for AWS Identity and Access Management credentials to use. The EKS Pod Identity credentials will be used if you don't specify a credential provider when you create the client or otherwise initialized the SDK.

This works because EKS Pod Identities have been added to the *Container credential provider* which is searched in a step in the default credential chain. If your workloads currently use credentials that are earlier in the chain of credentials, those credentials will continue to be used even if you configure an EKS Pod Identity association for the same workload.

For more information about how EKS Pod Identities work, see the section called "How it works".

When using Learn how EKS Pod Identity grants pods access to AWS services, the containers in your Pods must use an AWS SDK version that supports assuming an IAM role from the EKS Pod Identity Agent. Make sure that you're using the following versions, or later, for your AWS SDK:

- Java (Version 2) – 2.21.30
- Java – 1.12.746
- Go v1 – v1.47.11
- Go v2 – release-2023-11-14
- Python (Boto3) – 1.34.41
- Python (botocore) – 1.34.41
- AWS CLI – 1.30.0

  AWS CLI – 2.15.0
- JavaScript v2 – 2.1550.0
- JavaScript v3 – v3.458.0
- Kotlin – v1.0.1
- Ruby – 3.188.0
- Rust – release-2024-03-13

- C++ – 1.11.263
- .NET – 3.7.734.0
- PowerShell – 4.1.502
- PHP – 3.287.1

To ensure that you're using a supported SDK, follow the installation instructions for your preferred SDK at Tools to Build on AWS when you build your containers.

For a list of add-ons that support EKS Pod Identity, see the section called "Pod Identity Support Reference".

Disable IPv6 in the EKS Pod Identity Agent

AWS Management Console

1. To disable `IPv6` in the EKS Pod Identity Agent, add the following configuration to the **Optional configuration settings** of the EKS Add-on.
   a. Open the Amazon EKS console.
   b. In the left navigation pane, select **Clusters**, and then select the name of the cluster that you want to configure the add-on for.
   c. Choose the **Add-ons** tab.
   d. Select the box in the top right of the EKS Pod Identity Agent add-on box and then choose **Edit**.
   e. On the **Configure EKS Pod Identity Agent** page:
      i. Select the **Version** that you'd like to use. We recommend that you keep the same version as the previous step, and update the version and configuration in separate actions.
      ii. Expand the **Optional configuration settings**.
      iii. Enter the JSON key `"agent":` and value of a nested JSON object with a key `"additionalArgs":` in **Configuration values**. The resulting text must be a valid JSON object. If this key and value are the only data in the text box, surround the key and value with curly braces `{ }`. The following example shows network policy is enabled:

```
{
    "agent": {
        "additionalArgs": {
            "-b": "169.254.170.23"
```

```
        }
    }
}
```

This configuration sets the `IPv4` address to be the only address used by the agent.

f. To apply the new configuration by replacing the EKS Pod Identity Agent pods, choose **Save changes**.

   Amazon EKS applies changes to the EKS Add-ons by using a *rollout* of the Kubernetes `DaemonSet` for EKS Pod Identity Agent. You can track the status of the rollout in the **Update history** of the add-on in the AWS Management Console and with `kubectl rollout status daemonset/eks-pod-identity-agent --namespace kube-system`.

   `kubectl rollout` has the following commands:

```
$ kubectl rollout

history  -- View rollout history
pause    -- Mark the provided resource as paused
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
undo     -- Undo a previous rollout
```

   If the rollout takes too long, Amazon EKS will undo the rollout, and a message with the type of **Addon Update** and a status of **Failed** will be added to the **Update history** of the add-on. To investigate any issues, start from the history of the rollout, and run `kubectl logs` on a EKS Pod Identity Agent pod to see the logs of EKS Pod Identity Agent.

2. If the new entry in the **Update history** has a status of **Successful**, then the rollout has completed and the add-on is using the new configuration in all of the EKS Pod Identity Agent pods.

AWS CLI

1. To disable `IPv6` in the EKS Pod Identity Agent, add the following configuration to the **configuration values** of the EKS Add-on.

   Run the following AWS CLI command. Replace `my-cluster` with the name of your cluster and the IAM role ARN with the role that you are using.

```
aws eks update-addon --cluster-name my-cluster --addon-name eks-pod-identity-agent \
    --resolve-conflicts PRESERVE --configuration-values '{"agent":{"additionalArgs":
 { "-b": "169.254.170.23"}}}'
```

This configuration sets the `IPv4` address to be the only address used by the agent.

Amazon EKS applies changes to the EKS Add-ons by using a *rollout* of the Kubernetes DaemonSet for EKS Pod Identity Agent. You can track the status of the rollout in the **Update history** of the add-on in the AWS Management Console and with `kubectl rollout status daemonset/eks-pod-identity-agent --namespace kube-system`.

`kubectl rollout` has the following commands:

```
kubectl rollout

history  -- View rollout history
pause    -- Mark the provided resource as paused
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
undo     -- Undo a previous rollout
```

If the rollout takes too long, Amazon EKS will undo the rollout, and a message with the type of **Addon Update** and a status of **Failed** will be added to the **Update history** of the add-on. To investigate any issues, start from the history of the rollout, and run `kubectl logs` on a EKS Pod Identity Agent pod to see the logs of EKS Pod Identity Agent.

Create IAM role with trust policy required by EKS Pod Identity

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
```

```
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

`sts:AssumeRole`

EKS Pod Identity uses `AssumeRole` to assume the IAM role before passing the temporary credentials to your pods.

`sts:TagSession`

EKS Pod Identity uses `TagSession` to include *session tags* in the requests to AWS STS.

You can use these tags in the *condition keys* in the trust policy to restrict which service accounts, namespaces, and clusters can use this role.

For a list of Amazon EKS condition keys, see Conditions defined by Amazon Elastic Kubernetes Service in the *Service Authorization Reference*. To learn which actions and resources you can use a condition key with, see Actions defined by Amazon Elastic Kubernetes Service.

Manage compute resources by using nodes

A Kubernetes node is a machine that runs containerized applications. Each node has the following components:

- **Container runtime** – Software that's responsible for running the containers.
- **kubelet** – Makes sure that containers are healthy and running within their associated Pod.
- **kube-proxy** – Maintains network rules that allow communication to your Pods.

For more information, see Nodes in the Kubernetes documentation.

Your Amazon EKS cluster can schedule Pods on any combination of EKS Auto Mode managed nodes, self-managed nodes, Amazon EKS managed node groups, AWS Fargate, and Amazon EKS Hybrid Nodes. To learn more about nodes deployed in your cluster, see the section called "Access cluster resources".

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

Compare compute options

The following table provides several criteria to evaluate when deciding which options best meet your requirements. Self-managed nodes are another option which support all of the criteria listed, but they require a lot more manual maintenance. For more information, see the section called "Self-managed nodes".

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

Criteria	EKS managed node groups	EKS Auto Mode	Amazon EKS Hybrid Nodes
Can be deployed to AWS Outposts	No	No	No
Can be deployed to an AWS Local Zone	Yes	No	No
Can run containers that require Windows	Yes	No	No
Can run containers that require Linux	Yes	Yes	Yes
Can run workloads that require the Inferentia chip	Yes – Amazon Linux nodes only	Yes	No
Can run workloads that require a GPU	Yes – Amazon Linux nodes only	Yes	Yes
Can run workloads that require Arm processors	Yes	Yes	Yes
Can run AWS Bottlerocket	Yes	Yes	No
Pods share CPU, memory, storage, and network resources with other Pods.	Yes	Yes	Yes
Must deploy and manage Amazon EC2 instances	Yes	No - Learn about EC2 managed instances	Yes – the on-premis es physical or virtual machines are managed by you

Criteria	EKS managed node groups	EKS Auto Mode	Amazon EKS Hybrid Nodes
			with your choice of tooling.
Must secure, maintain, and patch the operating system of Amazon EC2 instances	Yes	No	Yes – the operating system running on your physical or virtual machines are managed by you with your choice of tooling.
Can provide bootstrap arguments at deployment of a node, such as extra kubelet arguments.	Yes – Using `eksctl` or a launch template with a custom AMI.	No - Use a `NodeClass` to configure nodes	Yes - you can customize bootstrap arguments with nodeadm. See the section called “Hybrid nodes nodeadm”.
Can assign IP addresses to Pods from a different CIDR block than the IP address assigned to the node.	Yes – Using a launch template with a custom AMI. For more information, see the section called “Launch templates”.	No	Yes - see the section called “Configure CNI”.
Can SSH into node	Yes	No - Learn how to troubleshoot nodes	Yes
Can deploy your own custom AMI to nodes	Yes – Using a launch template	No	Yes
Can deploy your own custom CNI to nodes	Yes – Using a launch template with a custom AMI	No	Yes

Criteria	EKS managed node groups	EKS Auto Mode	Amazon EKS Hybrid Nodes
Must update node AMI on your own	Yes – If you deployed an Amazon EKS optimized AMI, you're notified in the Amazon EKS console when updates are available. You can perform the update with one-click in the console. If you deployed a custom AMI, you're not notified in the Amazon EKS console when updates are available. You must perform the update on your own.	No	Yes - the operating system running on your physical or virtual machines is managed by you with your choice of tooling. See the section called "Prepare operating system".

Criteria	EKS managed node groups	EKS Auto Mode	Amazon EKS Hybrid Nodes
Must update node Kubernetes version on your own	Yes – If you deployed an Amazon EKS optimized AMI, you're notified in the Amazon EKS console when updates are available. You can perform the update with one-click in the console. If you deployed a custom AMI, you're not notified in the Amazon EKS console when updates are available. You must perform the update on your own.	No	Yes - you manage hybrid nodes upgrades with your own choice of tooling or with `nodeadm`. See the section called "Upgrade hybrid nodes".
Can use Amazon EBS storage with Pods	Yes	Yes, as an integrated capability. Learn how to create a storage class.	No
Can use Amazon EFS storage with Pods	Yes	Yes	No
Can use Amazon FSx for Lustre storage with Pods	Yes	Yes	No
Can use Network Load Balancer for services	Yes	Yes	Yes - must use target type `ip`.

Criteria	EKS managed node groups	EKS Auto Mode	Amazon EKS Hybrid Nodes
Pods can run in a public subnet	Yes	Yes	No - pods run in on-premises environment.
Can assign different VPC security groups to individual Pods	Yes – Linux nodes only	No	No
Can run Kubernetes DaemonSets	Yes	Yes	Yes
Support `HostPort` and `HostNetwork` in the Pod manifest	Yes	Yes	Yes
AWS Region availability	All Amazon EKS supported regions	All Amazon EKS supported regions	All Amazon EKS supported regions except the AWS GovCloud (US) Regions and the China Regions.
Can run containers on Amazon EC2 dedicated hosts	Yes	No	No

Criteria	EKS managed node groups	EKS Auto Mode	Amazon EKS Hybrid Nodes
Pricing	Cost of Amazon EC2 instance that runs multiple Pods. For more information, see Amazon EC2 pricing.	When EKS Auto Mode is enabled in your cluster, you pay a separate fee, in addition to the standard EC2 instance charges, for the instances launched using Auto Mode's compute capability. The amount varies with the instance type launched and the AWS region where your cluster is located. For more information, see Amazon EKS pricing.	Cost of hybrid nodes vCPU per hour. For more information, see Amazon EKS pricing.

Simplify node lifecycle with managed node groups

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

With Amazon EKS managed node groups, you don't need to separately provision or register the Amazon EC2 instances that provide compute capacity to run your Kubernetes applications. You can create, automatically update, or terminate nodes for your cluster with a single operation. Node updates and terminations automatically drain nodes to ensure that your applications stay available.

Every managed node is provisioned as part of an Amazon EC2 Auto Scaling group that's managed for you by Amazon EKS. Every resource including the instances and Auto Scaling groups runs within your AWS account. Each node group runs across multiple Availability Zones that you define.

Managed node groups can also optionally leverage node auto repair, which continuously monitors the health of nodes. It automatically reacts to detected problems and replaces nodes when possible. This helps overall availability of the cluster with minimal manual intervention. For more information, see the section called "Node health".

You can add a managed node group to new or existing clusters using the Amazon EKS console, `eksctl`, AWS CLI, AWS API, or infrastructure as code tools including AWS CloudFormation. Nodes launched as part of a managed node group are automatically tagged for auto-discovery by the Kubernetes Cluster Autoscaler. You can use the node group to apply Kubernetes labels to nodes and update them at any time.

There are no additional costs to use Amazon EKS managed node groups, you only pay for the AWS resources you provision. These include Amazon EC2 instances, Amazon EBS volumes, Amazon EKS cluster hours, and any other AWS infrastructure. There are no minimum fees and no upfront commitments.

To get started with a new Amazon EKS cluster and managed node group, see the section called "Create cluster (Console and CLI)".

To add a managed node group to an existing cluster, see the section called "Create".

Managed node groups concepts

- Amazon EKS managed node groups create and manage Amazon EC2 instances for you.
- Every managed node is provisioned as part of an Amazon EC2 Auto Scaling group that's managed for you by Amazon EKS. Moreover, every resource including Amazon EC2 instances and Auto Scaling groups run within your AWS account.
- The Auto Scaling group of a managed node group spans every subnet that you specify when you create the group.
- Amazon EKS tags managed node group resources so that they are configured to use the Kubernetes Cluster Autoscaler.

> **Important**
>
> If you are running a stateful application across multiple Availability Zones that is backed by Amazon EBS volumes and using the Kubernetes Cluster Autoscaler, you should configure multiple node groups, each scoped to a single Availability Zone. In addition, you should enable the `--balance-similar-node-groups` feature.

- You can use a custom launch template for a greater level of flexibility and customization when deploying managed nodes. For example, you can specify extra `kubelet` arguments and use a custom AMI. For more information, see the section called "Launch templates". If you don't use a custom launch template when first creating a managed node group, there is an auto-generated launch template. Don't manually modify this auto-generated template or errors occur.
- Amazon EKS follows the shared responsibility model for CVEs and security patches on managed node groups. When managed nodes run an Amazon EKS optimized AMI, Amazon EKS is responsible for building patched versions of the AMI when bugs or issues are reported. We can publish a fix. However, you're responsible for deploying these patched AMI versions to your managed node groups. When managed nodes run a custom AMI, you're responsible for building patched versions of the AMI when bugs or issues are reported and then deploying the AMI. For more information, see the section called "Update".
- Amazon EKS managed node groups can be launched in both public and private subnets. If you launch a managed node group in a public subnet on or after April 22, 2020, the subnet must have `MapPublicIpOnLaunch` set to true for the instances to successfully join a cluster. If the public subnet was created using `eksctl` or the Amazon EKS vended AWS CloudFormation templates on or after March 26, 2020, then this setting is already set to true. If the public subnets were created before March 26, 2020, you must change the setting manually. For more information, see Modifying the public IPv4 addressing attribute for your subnet.
- When deploying a managed node group in private subnets, you must ensure that it can access Amazon ECR for pulling container images. You can do this by connecting a NAT gateway to the route table of the subnet or by adding the following AWS PrivateLink VPC endpoints:
  - Amazon ECR API endpoint interface – `com.amazonaws.`*`region-code`*`.ecr.api`
  - Amazon ECR Docker registry API endpoint interface – `com.amazonaws.`*`region-code`*`.ecr.dkr`
  - Amazon S3 gateway endpoint – `com.amazonaws.`*`region-code`*`.s3`

  For other commonly-used services and endpoints, see the section called "Private clusters".
- Managed node groups can't be deployed on AWS Outposts or in AWS Wavelength. Managed node groups can be created on AWS Local Zones. For more information, see the section called " AWS Local Zones".
- You can create multiple managed node groups within a single cluster. For example, you can create one node group with the standard Amazon EKS optimized Amazon Linux AMI for some workloads and another with the GPU variant for workloads that require GPU support.

- If your managed node group encounters an Amazon EC2 instance status check failure, Amazon EKS returns an error code to help you to diagnose the issue. For more information, see the section called "Managed node group error codes".
- Amazon EKS adds Kubernetes labels to managed node group instances. These Amazon EKS provided labels are prefixed with `eks.amazonaws.com`.
- Amazon EKS automatically drains nodes using the Kubernetes API during terminations or updates.
- Pod disruption budgets aren't respected when terminating a node with `AZRebalance` or reducing the desired node count. These actions try to evict Pods on the node. But if it takes more than 15 minutes, the node is terminated regardless of whether all Pods on the node are terminated. To extend the period until the node is terminated, add a lifecycle hook to the Auto Scaling group. For more information, see Add lifecycle hooks in the *Amazon EC2 Auto Scaling User Guide*.
- In order to run the drain process correctly after receiving a Spot interruption notification or a capacity rebalance notification, `CapacityRebalance` must be set to `true`.
- Updating managed node groups respects the Pod disruption budgets that you set for your Pods. For more information, see the section called "Update behavior details".
- There are no additional costs to use Amazon EKS managed node groups. You only pay for the AWS resources that you provision.
- If you want to encrypt Amazon EBS volumes for your nodes, you can deploy the nodes using a launch template. To deploy managed nodes with encrypted Amazon EBS volumes without using a launch template, encrypt all new Amazon EBS volumes created in your account. For more information, see Encryption by default in the *Amazon EC2 User Guide*.

Managed node group capacity types

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

On-Demand

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

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

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

Spot

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

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

- To ensure that your Spot nodes are provisioned in the optimal Spot capacity pools, the allocation strategy is set to one of the following:
  - `price-capacity-optimized` (PCO) – When creating new node groups in a cluster with Kubernetes version `1.28` or higher, the allocation strategy is set to `price-capacity-optimized`. However, the allocation strategy won't be changed for node groups already

created with `capacity-optimized` before Amazon EKS managed node groups started to support PCO.
    - `capacity-optimized` (CO) – When creating new node groups in a cluster with Kubernetes version `1.27` or lower, the allocation strategy is set to `capacity-optimized`.

  To increase the number of Spot capacity pools available for allocating capacity from, configure a managed node group to use multiple instance types.
- Amazon EC2 Spot Capacity Rebalancing is enabled so that Amazon EKS can gracefully drain and rebalance your Spot nodes to minimize application disruption when a Spot node is at elevated risk of interruption. For more information, see [Amazon EC2 Auto Scaling Capacity Rebalancing] in the *Amazon EC2 Auto Scaling User Guide*.
    - When a Spot node receives a rebalance recommendation, Amazon EKS automatically attempts to launch a new replacement Spot node.
    - If a Spot two-minute interruption notice arrives before the replacement Spot node is in a `Ready` state, Amazon EKS starts draining the Spot node that received the rebalance recommendation. Amazon EKS drains the node on a best-effort basis. As a result, there's no guarantee that Amazon EKS will wait for the replacement node to join the cluster before draining the existing node.
    - When a replacement Spot node is bootstrapped and in the `Ready` state on Kubernetes, Amazon EKS cordons and drains the Spot node that received the rebalance recommendation. Cordoning the Spot node ensures that the service controller doesn't send any new requests to this Spot node. It also removes it from its list of healthy, active Spot nodes. Draining the Spot node ensures that running Pods are evicted gracefully.
- Amazon EKS adds the following Kubernetes label to all nodes in your managed node group that specifies the capacity type: `eks.amazonaws.com/capacityType: SPOT`. You can use this label to schedule fault tolerant applications on Spot nodes.

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

- Spot Instances are a good fit for stateless, fault-tolerant, flexible applications. These include batch and machine learning training workloads, big data ETLs such as Apache Spark, queue processing applications, and stateless API endpoints. Because Spot is spare Amazon EC2 capacity, which can change over time, we recommend that you use Spot capacity for interruption-tolerant workloads. More specifically, Spot capacity is suitable for workloads that can tolerate periods where the required capacity isn't available.

- We recommend that you use On-Demand for applications that are fault intolerant. This includes cluster management tools such as monitoring and operational tools, deployments that require `StatefulSets`, and stateful applications, such as databases.
- To maximize the availability of your applications while using Spot Instances, we recommend that you configure a Spot managed node group to use multiple instance types. We recommend applying the following rules when using multiple instance types:
  - Within a managed node group, if you're using the Cluster Autoscaler, we recommend using a flexible set of instance types with the same amount of vCPU and memory resources. This is to ensure that the nodes in your cluster scale as expected. For example, if you need four vCPUs and eight GiB memory, use `c3.xlarge`, `c4.xlarge`, `c5.xlarge`, `c5d.xlarge`, `c5a.xlarge`, `c5n.xlarge`, or other similar instance types.
  - To enhance application availability, we recommend deploying multiple Spot managed node groups. For this, each group should use a flexible set of instance types that have the same vCPU and memory resources. For example, if you need 4 vCPUs and 8 GiB memory, we recommend that you create one managed node group with `c3.xlarge`, `c4.xlarge`, `c5.xlarge`, `c5d.xlarge`, `c5a.xlarge`, `c5n.xlarge`, or other similar instance types, and a second managed node group with `m3.xlarge`, `m4.xlarge`, `m5.xlarge`, `m5d.xlarge`, `m5a.xlarge`, `m5n.xlarge` or other similar instance types.
  - When deploying your node group with the Spot capacity type that's using a custom launch template, use the API to pass multiple instance types. Don't pass a single instance type through the launch template. For more information about deploying a node group using a launch template, see the section called "Launch templates".

Create a managed node group for your cluster

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

If this is your first time launching an Amazon EKS managed node group, we recommend that you instead follow one of our guides in *Get started*. These guides provide walkthroughs for creating an Amazon EKS cluster with nodes.

> **Important**
>
> - Amazon EKS nodes are standard Amazon EC2 instances. You're billed based on the normal Amazon EC2 prices. For more information, see Amazon EC2 Pricing.
> - You can't create managed nodes in an AWS Region where you have AWS Outposts or AWS Wavelength enabled. You can create self-managed nodes instead. For more information, see the section called "Amazon Linux", the section called "Windows", and the section called "Bottlerocket". You can also create a self-managed Amazon Linux node group on an Outpost. For more information, see the section called "Nodes".
> - If you don't specify an AMI ID for the `bootstrap.sh` file included with Amazon EKS optimized Linux or Bottlerocket, managed node groups enforce a maximum number on the value of `maxPods`. For instances with less than 30 vCPUs, the maximum number is `110`. For instances with greater than 30 vCPUs, the maximum number jumps to `250`. These numbers are based on Kubernetes scalability thresholds and recommended settings by internal Amazon EKS scalability team testing. For more information, see the Amazon VPC CNI plugin increases pods per node limits blog post.

- An existing Amazon EKS cluster. To deploy one, see the section called "Create a cluster".
- An existing IAM role for the nodes to use. To create one, see the section called "Node IAM role". If this role doesn't have either of the policies for the VPC CNI, the separate role that follows is required for the VPC CNI pods.
- (Optional, but recommended) The Amazon VPC CNI plugin for Kubernetes add-on configured with its own IAM role that has the necessary IAM policy attached to it. For more information, see the section called "Configure for IRSA".
- Familiarity with the considerations listed in Choose an optimal Amazon EC2 node instance type. Depending on the instance type you choose, there may be additional prerequisites for your cluster and VPC.
- To add a Windows managed node group, you must first enable Windows support for your cluster. For more information, see the section called "Enable Windows support".

You can create a managed node group with either of the following:

- the section called "`eksctl`"
- the section called "AWS Management Console"

eksctl

Create a managed node group with eksctl

This procedure requires `eksctl` version `0.204.0` or later. You can check your version with the following command:

```
eksctl version
```

For instructions on how to install or upgrade `eksctl`, see Installation in the `eksctl` documentation.

1. (Optional) If the **AmazonEKS_CNI_Policy** managed IAM policy is attached to your Amazon EKS node IAM role, we recommend assigning it to an IAM role that you associate to the Kubernetes `aws-node` service account instead. For more information, see the section called “Configure for IRSA”.
2. Create a managed node group with or without using a custom launch template. Manually specifying a launch template allows for greater customization of a node group. For example, it can allow deploying a custom AMI or providing arguments to the `boostrap.sh` script in an Amazon EKS optimized AMI. For a complete list of every available option and default, enter the following command.

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

   In the following command, replace *my-cluster* with the name of your cluster and replace *my-mng* with the name of your node group. The node group name can't be longer than 63 characters. It must start with letter or digit, but can also include hyphens and underscores for the remaining characters.

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

Without a launch template

eksctl creates a default Amazon EC2 launch template in your account and deploys the node group using a launch template that it creates based on options that you specify. Before specifying a value for --node-type, see the section called “Amazon EC2 instance types”.

Replace *ami-family* with an allowed keyword. For more information, see Setting the node AMI Family in the eksctl documentation. Replace *my-key* with the name of your Amazon EC2 key pair or public key. This key is used to SSH into your nodes after they launch.

> **Note**
>
> For Windows, this command doesn’t enable SSH. Instead, it associates your Amazon EC2 key pair with the instance and allows you to RDP into the instance.

If you don’t already have an Amazon EC2 key pair, you can create one in the AWS Management Console. For Linux information, see Amazon EC2 key pairs and Linux instances in the *Amazon EC2 User Guide*. For Windows information, see Amazon EC2 key pairs and Windows instances in the *Amazon EC2 User Guide*.

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

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

For more information, see Restrict access to the instance profile assigned to the worker node.

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

```
eksctl create nodegroup \
  --cluster my-cluster \
  --region region-code \
  --name my-mng \
  --node-ami-family ami-family \
  --node-type m5.large \
```

```
  --nodes 3 \
  --nodes-min 2 \
  --nodes-max 4 \
  --ssh-access \
  --ssh-public-key my-key
```

Your instances can optionally assign a significantly higher number of IP addresses to Pods, assign IP addresses to Pods from a different CIDR block than the instance's, and be deployed to a cluster without internet access. For more information, see the section called "Increase IP addresses", the section called "Custom networking", and the section called "Private clusters" for additional options to add to the previous command.

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

With a launch template

The launch template must already exist and must meet the requirements specified in Launch template configuration basics. We recommend blocking Pod access to IMDS if the following conditions are true:

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

For more information, see Restrict access to the instance profile assigned to the worker node.

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

a. Copy the following contents to your device. Replace the *example values* and then run the modified command to create the `eks-nodegroup.yaml` file. Several settings that you specify when deploying without a launch template are moved into the launch template. If you don't specify a `version`, the template's default version is used.

```
cat >eks-nodegroup.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-cluster
  region: region-code
managedNodeGroups:
- name: my-mng
  launchTemplate:
    id: lt-id
    version: "1"
EOF
```

For a complete list of `eksctl` config file settings, see Config file schema in the `eksctl` documentation. Your instances can optionally assign a significantly higher number of IP addresses to Pods, assign IP addresses to Pods from a different CIDR block than the instance's, use the `containerd` runtime, and be deployed to a cluster without outbound internet access. For more information, see the section called "Increase IP addresses", the section called "Custom networking", the section called "Test Amazon Linux 2 migration from Docker to `containerd`", and the section called "Private clusters" for additional options to add to the config file.

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

If you specified an AMI ID in your launch template, specify the maximum number of Pods that can run on each node of your node group if you're using custom networking or want to increase the number of IP addresses assigned to your instance. For more information, see the section called "Amazon EKS recommended maximum Pods for each Amazon EC2 instance type".

b. Deploy the nodegroup with the following command.

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

AWS Management Console

Create a managed node group using the AWS Management Console

1. Wait for your cluster status to show as `ACTIVE`. You can't create a managed node group for a cluster that isn't already `ACTIVE`.
2. Open the Amazon EKS console.
3. Choose the name of the cluster that you want to create a managed node group in.
4. Select the **Compute** tab.
5. Choose **Add node group**.
6. On the **Configure node group** page, fill out the parameters accordingly, and then choose **Next**.
   - **Name** – Enter a unique name for your managed node group. The node group name can't be longer than 63 characters. It must start with letter or digit, but can also include hyphens and underscores for the remaining characters.
   - **Node IAM role** – Choose the node instance role to use with your node group. For more information, see the section called "Node IAM role".

   Important

   - You can't use the same role that is used to create any clusters.
   - We recommend using a role that's not currently in use by any self-managed node group. Otherwise, you plan to use with a new self-managed node group. For more information, see the section called "Delete".

   - **Use launch template** – (Optional) Choose if you want to use an existing launch template. Select a **Launch Template Name**. Then, select a **Launch template version**. If you don't select a version, then Amazon EKS uses the template's default version. Launch templates allow for more customization of your node group, such as allowing you to deploy a custom AMI, assign a significantly higher number of IP addresses to Pods, assign IP addresses to Pods from a different CIDR block than the instance's, enable the `containerd` runtime for your instances, and deploying nodes to a cluster without outbound internet access. For more information, see the section called "Increase IP addresses", the section called "Custom networking", the section called "Test Amazon Linux 2 migration from Docker to `containerd`", and the section called "Private clusters".

The launch template must meet the requirements in Customize managed nodes with launch templates. If you don't use your own launch template, the Amazon EKS API creates a default Amazon EC2 launch template in your account and deploys the node group using the default launch template.

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

- **Kubernetes labels** – (Optional) You can choose to apply Kubernetes labels to the nodes in your managed node group.
- **Kubernetes taints** – (Optional) You can choose to apply Kubernetes taints to the nodes in your managed node group. The available options in the **Effect** menu are `NoSchedule`, `NoExecute`, and `PreferNoSchedule`. For more information, see the section called "Taint GPU nodes".
- **Tags** – (Optional) You can choose to tag your Amazon EKS managed node group. These tags don't propagate to other resources in the node group, such as Auto Scaling groups or instances. For more information, see the section called "Tagging your resources".

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

- **AMI type** – Select an AMI type.If you are deploying Arm instances, be sure to review the considerations in Amazon EKS optimized Arm Amazon Linux AMIs before deploying.

  If you specified a launch template on the previous page, and specified an AMI in the launch template, then you can't select a value. The value from the template is displayed. The AMI specified in the template must meet the requirements in Specifying an AMI.
- **Capacity type** – Select a capacity type. For more information about choosing a capacity type, see the section called "Managed node group capacity types". You can't mix different capacity types within the same node group. If you want to use both capacity types, create separate node groups, each with their own capacity and instance types. See the section called "Reserve GPUs for MNG" for information on provisioning and scaling GPU-accelerated worker nodes.
- **Instance types** – By default, one or more instance type is specified. To remove a default instance type, select the X on the right side of the instance type. Choose the instance types to

use in your managed node group. For more information, see the section called “Amazon EC2 instance types”.

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

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

  If you specified a launch template on the previous page, then you can't select a value because it must be specified in the launch template.
- **Desired size** – Specify the current number of nodes that the managed node group should maintain at launch.

> **Note**
>
> Amazon EKS doesn't automatically scale your node group in or out. However, you can configure the Kubernetes Cluster Autoscaler to do this for you. For more information, see Cluster Autoscaler on AWS.

- **Minimum size** – Specify the minimum number of nodes that the managed node group can scale in to.
- **Maximum size** – Specify the maximum number of nodes that the managed node group can scale out to.
- **Node group update configuration** – (Optional) You can select the number or percentage of nodes to be updated in parallel. These nodes will be unavailable during the update. For **Maximum unavailable**, select one of the following options and specify a **Value**:
  - **Number** – Select and specify the number of nodes in your node group that can be updated in parallel.
  - **Percentage** – Select and specify the percentage of nodes in your node group that can be updated in parallel. This is useful if you have a large number of nodes in your node group.
- **Node auto repair configuration** – (Optional) If you activate the **Enable node auto repair** checkbox, Amazon EKS will automatically replace nodes when detected issues occur. For more information, see the section called “Node health”.

8. On the **Specify networking** page, fill out the parameters accordingly, and then choose **Next**.

   - **Subnets** – Choose the subnets to launch your managed nodes into.

     > **Important**
     >
     > If you are running a stateful application across multiple Availability Zones that is backed by Amazon EBS volumes and using the Kubernetes Cluster Autoscaler, you should configure multiple node groups, each scoped to a single Availability Zone. In addition, you should enable the `--balance-similar-node-groups` feature.

     > **Important**
     >
     > - If you choose a public subnet, and your cluster has only the public API server endpoint enabled, then the subnet must have `MapPublicIPOnLaunch` set to `true` for the instances to successfully join a cluster. If the subnet was created using `eksctl` or the Amazon EKS vended AWS CloudFormation templates on or after March 26, 2020, then this setting is already set to `true`. If the subnets were created with `eksctl` or the AWS CloudFormation templates before March 26, 2020, then you need to change the setting manually. For more information, see Modifying the public IPv4 addressing attribute for your subnet.
     > - If you use a launch template and specify multiple network interfaces, Amazon EC2 won't auto-assign a public `IPv4` address, even if `MapPublicIpOnLaunch` is set to `true`. For nodes to join the cluster in this scenario, you must either enable the cluster's private API server endpoint, or launch nodes in a private subnet with outbound internet access provided through an alternative method, such as a NAT Gateway. For more information, see Amazon EC2 instance IP addressing in the *Amazon EC2 User Guide*.

   - **Configure SSH access to nodes** (Optional). Enabling SSH allows you to connect to your instances and gather diagnostic information if there are issues. We highly recommend enabling remote access when you create a node group. You can't enable remote access after the node group is created.

     If you chose to use a launch template, then this option isn't shown. To enable remote access to your nodes, specify a key pair in the launch template and ensure that the proper port is open to the nodes in the security groups that you specify in the launch template. For more information, see the section called “Using custom security groups”.

> **Note**
>
> For Windows, this command doesn’t enable SSH. Instead, it associates your Amazon EC2 key pair with the instance and allows you to RDP into the instance.

- For **SSH key pair** (Optional), choose an Amazon EC2 SSH key to use. For Linux information, see Amazon EC2 key pairs and Linux instances in the *Amazon EC2 User Guide*. For Windows information, see Amazon EC2 key pairs and Windows instances in the *Amazon EC2 User Guide*. If you chose to use a launch template, then you can’t select one. When an Amazon EC2 SSH key is provided for node groups using Bottlerocket AMIs, the administrative container is also enabled. For more information, see Admin container on GitHub.
- For **Allow SSH remote access from**, if you want to limit access to specific instances, then select the security groups that are associated to those instances. If you don’t select specific security groups, then SSH access is allowed from anywhere on the internet (`0.0.0.0/0`).

9. On the **Review and create** page, review your managed node group configuration and choose **Create**.

   If nodes fail to join the cluster, then see the section called “Nodes fail to join cluster” in the Troubleshooting chapter.

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

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

11. (GPU nodes only) If you chose a GPU instance type and an Amazon EKS optimized accelerated AMI, then you must apply the NVIDIA device plugin for Kubernetes as a DaemonSet on your cluster. Replace *vX.X.X* with your desired NVIDIA/k8s-device-plugin version before running the following command.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
deployments/static/nvidia-device-plugin.yml
```

Install Kubernetes add-ons

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

- The IAM principal that created the cluster is the only principal that can make calls to the Kubernetes API server with `kubectl` or the AWS Management Console. If you want other IAM principals to have access to your cluster, then you need to add them. For more information, see the section called "Kubernetes API access" and the section called "Required permissions".
- We recommend blocking Pod access to IMDS if the following conditions are true:
  - You plan to assign IAM roles to all of your Kubernetes service accounts so that Pods only have the minimum permissions that they need.
  - No Pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current AWS Region.

  For more information, see Restrict access to the instance profile assigned to the worker node.
- Configure the Kubernetes Cluster Autoscaler to automatically adjust the number of nodes in your node groups.
- Deploy a sample application to your cluster.
- Organize and monitor cluster resources with important tools for managing your cluster.

Update a managed node group for your cluster

When you initiate a managed node group update, Amazon EKS automatically updates your nodes for you, completing the steps listed in Understand each phase of node updates. If you're using an Amazon EKS optimized AMI, Amazon EKS automatically applies the latest security patches and operating system updates to your nodes as part of the latest AMI release version.

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

- You have updated the Kubernetes version for your Amazon EKS cluster and want to update your nodes to use the same Kubernetes version.
- A new AMI release version is available for your managed node group. For more information about AMI versions, see these sections:

  - the section called "Get version information"
  - the section called "Bottlerocket"
  - the section called "Get version information"
- You want to adjust the minimum, maximum, or desired count of the instances in your managed node group.
- You want to add or remove Kubernetes labels from the instances in your managed node group.
- You want to add or remove AWS tags from your managed node group.
- You need to deploy a new version of a launch template with configuration changes, such as an updated custom AMI.
- You have deployed version `1.9.0` or later of the Amazon VPC CNI add-on, enabled the add-on for prefix delegation, and want new AWS Nitro System instances in a node group to support a significantly increased number of Pods. For more information, see the section called "Increase IP addresses".
- You have enabled IP prefix delegation for Windows nodes and want new AWS Nitro System instances in a node group to support a significantly increased number of Pods. For more information, see the section called "Increase IP addresses".

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

When a node in a managed node group is terminated due to a scaling operation or update, the Pods in that node are drained first. For more information, see the section called "Update behavior details".

Update a node group version

You can update a node group version with either of the following:

- the section called "`eksctl`"
- the section called "AWS Management Console"

The version that you update to can't be greater than the control plane's version.

eksctl

Update a managed node group using eksctl

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

```
eksctl upgrade nodegroup \
  --name=node-group-name \
  --cluster=my-cluster \
  --region=region-code
```

Note

If you're upgrading a node group that's deployed with a launch template to a new launch template version, add `--launch-template-version` *version-number* to the preceding command. The launch template must meet the requirements described in Customize managed nodes with launch templates. If the launch template includes a custom AMI, the AMI must meet the requirements in Specifying an AMI. When you upgrade your node group to a newer version of your launch template, every node is recycled to match the new configuration of the launch template version that's specified.

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

You can upgrade a node group to the same version as the control plane's Kubernetes version. For example, if you have a cluster running Kubernetes `1.32`, you can upgrade nodes currently running Kubernetes `1.31` to version `1.32` with the following command.

```
eksctl upgrade nodegroup \
  --name=node-group-name \
  --cluster=my-cluster \
  --region=region-code \
  --kubernetes-version=1.32
```

AWS Management Console

Update a managed node group using the AWS Management Console

1. Open the Amazon EKS console.
2. Choose the cluster that contains the node group to update.
3. If at least one node group has an available update, a box appears at the top of the page notifying you of the available update. If you select the **Compute** tab, you'll see **Update now** in the **AMI release version** column in the **Node groups** table for the node group that has an available update. To update the node group, choose **Update now**.

   You won't see a notification for node groups that were deployed with a custom AMI. If your nodes are deployed with a custom AMI, complete the following steps to deploy a new updated custom AMI.

   a. Create a new version of your AMI.
   b. Create a new launch template version with the new AMI ID.
   c. Upgrade the nodes to the new version of the launch template.
4. On the **Update node group version** dialog box, activate or deactivate the following options:
   - **Update node group version** – This option is unavailable if you deployed a custom AMI or your Amazon EKS optimized AMI is currently on the latest version for your cluster.
   - **Change launch template version** – This option is unavailable if the node group is deployed without a custom launch template. You can only update the launch template version for a node group that has been deployed with a custom launch template. Select the **Launch template version** that you want to update the node group to. If your node group is configured with a custom AMI, then the version that you select must also specify an AMI. When you upgrade to a newer version of your launch template, every node is recycled to match the new configuration of the launch template version specified.
5. For **Update strategy**, select one of the following options:
   - **Rolling update** – This option respects the Pod disruption budgets for your cluster. Updates fail if there's a Pod disruption budget issue that causes Amazon EKS to be unable to gracefully drain the Pods that are running on this node group.
   - **Force update** – This option doesn't respect Pod disruption budgets. Updates occur regardless of Pod disruption budget issues by forcing node restarts to occur.
6. Choose **Update**.

Edit a node group configuration

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

1. Open the Amazon EKS console.
2. Choose the cluster that contains the node group to edit.
3. Select the **Compute** tab.
4. Select the node group to edit, and then choose **Edit**.
5. (Optional) On the **Edit node group** page, do the following:
   a. Edit the **Node group scaling configuration**.
      - **Desired size** – Specify the current number of nodes that the managed node group should maintain.
      - **Minimum size** – Specify the minimum number of nodes that the managed node group can scale in to.
      - **Maximum size** – Specify the maximum number of nodes that the managed node group can scale out to. For the maximum number of nodes supported in a node group, see the section called “Service quotas”.

   b. (Optional) Add or remove **Kubernetes labels** to the nodes in your node group. The labels shown here are only the labels that you have applied with Amazon EKS. Other labels may exist on your nodes that aren’t shown here.

   c. (Optional) Add or remove **Kubernetes taints** to the nodes in your node group. Added taints can have the effect of either `NoSchedule`, `NoExecute`, or `PreferNoSchedule`. For more information, see the section called “Taint GPU nodes”.

   d. (Optional) Add or remove **Tags** from your node group resource. These tags are only applied to the Amazon EKS node group. They don’t propagate to other resources, such as subnets or Amazon EC2 instances in the node group.

   e. (Optional) Edit the **Node Group update configuration**. Select either **Number** or **Percentage**.
      - **Number** – Select and specify the number of nodes in your node group that can be updated in parallel. These nodes will be unavailable during update.
      - **Percentage** – Select and specify the percentage of nodes in your node group that can be updated in parallel. These nodes will be unavailable during update. This is useful if you have many nodes in your node group.

   f. When you’re finished editing, choose **Save changes**.

> **Important**
>
> When updating the node group configuration, modifying the `NodegroupScalingConfig` does not respect Pod disruption budgets (PDBs). Unlike the update node group process (which drains nodes and respects PDBs during the upgrade phase), updating the scaling configuration causes nodes to be terminated immediately through an Auto Scaling Group (ASG) scale-down call. This happens without considering PDBs, regardless of the target size you're scaling down to. That means when you reduce the `desiredSize` of an Amazon EKS managed node group, Pods are evicted as soon as the nodes are terminated, without honoring any PDBs.

Understand each phase of node updates

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

Setup phase

The setup phase has these steps:

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

Scale up phase

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

The scale up phase has these steps:

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

     For example, if your node group has five Availability Zones and `maxUnavailable` as one, the upgrade process can launch a maximum of 10 nodes. However when `maxUnavailable` is 20 (or anything higher than 10), the process would launch 20 new nodes.
2. After scaling the Auto Scaling Group, it checks if the nodes using the latest configuration are present in the node group. This step succeeds only when it meets these criteria:
   - At least one new node is launched in every Availability Zone where the node exists.
   - Every new node should be in `Ready` state.
   - New nodes should have Amazon EKS applied labels.

     These are the Amazon EKS applied labels on the worker nodes in a regular node group:
     - `eks.amazonaws.com/nodegroup-image=$amiName`
     - `eks.amazonaws.com/nodegroup=$nodeGroupName`

     These are the Amazon EKS applied labels on the worker nodes in a custom launch template or AMI node group:
     - `eks.amazonaws.com/nodegroup-image=$amiName`
     - `eks.amazonaws.com/nodegroup=$nodeGroupName`
     - `eks.amazonaws.com/sourceLaunchTemplateId=$launchTemplateId`
     - `eks.amazonaws.com/sourceLaunchTemplateVersion=$launchTemplateVersion`
3. It marks nodes as unschedulable to avoid scheduling new Pods. It also labels nodes with `node.kubernetes.io/exclude-from-external-load-balancers=true` to remove the nodes from load balancers before terminating the nodes.

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

Insufficient capacity in the Availability Zone

There is a possibility that the Availability Zone might not have capacity of requested instance types. It's recommended to configure multiple instance types while creating a managed node group.

EC2 instance limits in your account

You may need to increase the number of Amazon EC2 instances your account can run simultaneously using Service Quotas. For more information, see EC2 Service Quotas in the *Amazon Elastic Compute Cloud User Guide for Linux Instances*.

Custom user data

Custom user data can sometimes break the bootstrap process. This scenario can lead to the `kubelet` not starting on the node or nodes not getting expected Amazon EKS labels on them. For more information, see the section called "Specifying an AMI".

Any changes which make a node unhealthy or not ready

Node disk pressure, memory pressure, and similar conditions can lead to a node not going to `Ready` state.

Each node most bootstrap within 15 minutes

If any node takes more than 15 minutes to bootstrap and join the cluster, it will cause the upgrade to time out. This is the total runtime for bootstrapping a new node measured from when a new node is required to when it joins the cluster. When upgrading a managed node group, the time counter starts as soon as the Auto Scaling Group size increases.

Upgrade phase

The upgrade phase behaves in two different ways, depending on the *update strategy*. There are two update strategies: **default** and **minimal**.

We recommend the default strategy in most scenarios. It creates new nodes before terminating the old ones, so that the available capacity is maintained during the upgrade phase. The minimal strategy is useful in scenarios where you are constrained to resources or costs, for example with hardware accelerators such as GPUs. It terminating the old nodes before creating the new ones, so that total capacity never increases beyond your configured quantity.

The *default* update strategy has these steps:

1. It increases the quantity of nodes (desired count) in the Auto Scaling Group, causing the node group to create additional nodes.
2. It randomly selects a node that needs to be upgraded, up to the maximum unavailable configured for the node group.

3. It drains the Pods from the node. If the Pods don't leave the node within 15 minutes and there's no force flag, the upgrade phase fails with a `PodEvictionFailure` error. For this scenario, you can apply the force flag with the `update-nodegroup-version` request to delete the Pods.
4. It cordons the node after every Pod is evicted and waits for 60 seconds. This is done so that the service controller doesn't send any new requests to this node and removes this node from its list of active nodes.
5. It sends a termination request to the Auto Scaling Group for the cordoned node.
6. It repeats the previous upgrade steps until there are no nodes in the node group that are deployed with the earlier version of the launch template.

The *minimal* update strategy has these steps:

1. It randomly selects a node that needs to be upgraded, up to the maximum unavailable configured for the node group.
2. It drains the Pods from the node. If the Pods don't leave the node within 15 minutes and there's no force flag, the upgrade phase fails with a `PodEvictionFailure` error. For this scenario, you can apply the force flag with the `update-nodegroup-version` request to delete the Pods.
3. It cordons the node after every Pod is evicted and waits for 60 seconds. This is done so that the service controller doesn't send any new requests to this node and removes this node from its list of active nodes.
4. It sends a termination request to the Auto Scaling Group for the cordoned node. The Auto Scaling Group creates a new node to replace the missing capacity.
5. It repeats the previous upgrade steps until there are no nodes in the node group that are deployed with the earlier version of the launch template.

`PodEvictionFailure` errors during the upgrade phase

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

Aggressive PDB

Aggressive PDB is defined on the Pod or there are multiple PDBs pointing to the same Pod.

Deployment tolerating all the taints

Once every Pod is evicted, it's expected for the node to be empty because the node is tainted in the earlier steps. However, if the deployment tolerates every taint, then the node is more likely to be non-empty, leading to Pod eviction failure.

Scale down phase

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

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

Customize managed nodes with launch templates

For the highest level of customization, you can deploy managed nodes using your own launch template. Using a launch template allows capabilities such as the following:

- Provide bootstrap arguments at deployment of a node, such as extra kubelet arguments.
- Assign IP addresses to Pods from a different CIDR block than the IP address assigned to the node.
- Deploy your own custom AMI to nodes.
- Deploy your own custom CNI to nodes.

When you give your own launch template upon first creating a managed node group, you will also have greater flexibility later. As long as you deploy a managed node group with your own launch template, you can iteratively update it with a different version of the same launch template. When you update your node group to a different version of your launch template, all nodes in the group are recycled to match the new configuration of the specified launch template version.

Managed node groups are always deployed with a launch template to be used with the Amazon EC2 Auto Scaling group. When you don't provide a launch template, the Amazon EKS API creates one automatically with default values in your account. However, we don't recommend that you modify auto-generated launch templates. Furthermore, existing node groups that don't use a custom launch template can't be updated directly. Instead, you must create a new node group with a custom launch template to do so.

Launch template configuration basics

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

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

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

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

Amazon EKS node group configuration – Prohibited	Launch template

(Only if you specified a custom AMI in a launch template) **AMI type** under **Node group compute configuration** on **Set compute and scaling configuration** page – Console displays **Specified in launch template** and the AMI ID that was specified.
 	
If **Application and OS Images (Amazon Machine Image)** wasn't specified in the launch template, you can select an AMI in the node group configuration.
	Application and OS Images (Amazon Machine Image) under **Launch template contents** – You must specify an ID if you have either of the following requirements:
	
	• Using a custom AMI. If you specify an AMI that doesn't meet the requirements listed in Specifying an AMI, the node group deployment will fail.
	• Want to provide user data to provide arguments to the `bootstrap.sh` file included with an Amazon EKS optimized AMI. You can enable your instances to assign a significantly higher number of IP addresses to Pods, assign IP addresses to Pods from a different CIDR block than the instance's, or deploy a private cluster without outbound internet access. For more information, see the following topics:
	  ◦ Assign more IP addresses to Amazon EKS nodes with prefixes
	  ◦ Deploy pods in alternate subnets with custom networking
	  ◦ Deploy private clusters with limited internet access
	  ◦ Specifying an AMI

Disk size under **Node group compute configuration** on **Set compute and scaling configuration** page – Console displays **Specified in launch template**.	**Size** under **Storage (Volumes)** (**Add new volume**). You must specify this in the launch template.

Amazon EKS node group configuration – Prohibited	Launch template
SSH key pair under **Node group configuration** on the **Specify Networking** page – The console displays the key that was specified in the launch template or displays **Not specified in launch template**.	**Key pair name** under **Key pair (login)**.
You can't specify source security groups that are allowed remote access when using a launch template.	**Security groups** under **Network settings** for the instance or **Security groups** under **Network interfaces (Add network interface)**, but not both. For more information, see the section called "Using custom security groups".

Note

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

Tagging Amazon EC2 instances

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

Using custom security groups

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

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

Amazon EC2 user data

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

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

- Including users or groups

- Installing packages

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

> **Note**
>
> For more information about advanced `kubelet` customization, including manually starting it or passing in custom configuration parameters, see the section called "Specifying an AMI". If a custom AMI ID is specified in a launch template, Amazon EKS doesn't merge user data.

The following details provide more information about the user data section.

Amazon Linux 2 user data

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

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

- The closing boundary, which signals the end of the MIME multi-part file: `--==MYBOUNDARY==--`

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

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

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

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

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

Amazon Linux 2023 user data

Amazon Linux 2023 (AL2023) introduces a new node initialization process `nodeadm` that uses a YAML configuration schema. If you're using self-managed node groups or an AMI with a launch template, you'll now need to provide additional cluster metadata explicitly when creating a new node group. An example of the minimum required parameters is as follows, where `apiServerEndpoint`, `certificateAuthority`, and service `cidr` are now required:

```
---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    cidr: 10.100.0.0/16
```

You'll typically set this configuration in your user data, either as-is or embedded within a MIME multi-part document:

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

--BOUNDARY
```

```
Content-Type: application/node.eks.aws

---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig spec: [...]

--BOUNDARY--
```

In AL2, the metadata from these parameters was discovered from the Amazon EKS `DescribeCluster` API call. With AL2023, this behavior has changed since the additional API call risks throttling during large node scale ups. This change doesn't affect you if you're using managed node groups without a launch template or if you're using Karpenter. For more information on `certificateAuthority` and service `cidr`, see [`DescribeCluster`] in the *Amazon EKS API Reference*.

Bottlerocket user data

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

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

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

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

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

- Quotes within quoted keys: `'quoted "value"' = "value"`

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

Windows user data

Windows user data uses PowerShell commands. When creating a managed node group, your custom user data combines with Amazon EKS managed user data. Your PowerShell commands come first, followed by the managed user data commands, all within one `<powershell></powershell>` tag.

> **Note**
>
> When no AMI ID is specified in the launch template, don't use the Windows Amazon EKS Bootstrap script in user data to configure Amazon EKS.

Example user data is as follows.

```
<powershell>
Write-Host "Running custom user data script"
</powershell>
```

Specifying an AMI

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

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

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

eksctl without specifying a launch template

Create a file named *my-nodegroup.yaml* with the following contents. Replace every *example value* with your own values. The `--apiserver-endpoint`, `--b64-cluster-ca`, and `--dns-cluster-ip` arguments are optional. However, defining them allows the `bootstrap.sh` script to avoid making a `describeCluster` call. This is useful in private cluster setups or clusters where you're scaling in and out nodes frequently. For more information on the `bootstrap.sh` script, see the bootstrap.sh file on GitHub.

- The only required argument is the cluster name (*my-cluster*).
- To retrieve an optimized AMI ID for `ami-`*1234567890abcdef0* , you can use the tables in the following sections:
  - Retrieve recommended Amazon Linux AMI IDs
  - Retrieve recommended Bottlerocket AMI IDs
  - Retrieve recommended Microsoft Windows AMI IDs
- To retrieve the *certificate-authority* for your cluster, run the following command.

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

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

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

- The value for `--dns-cluster-ip` is your service CIDR with `.10` at the end. To retrieve the *service-cidr* for your cluster, run the following command. For example, if the returned value for is `ipv4 10.100.0.0/16`, then your value is *10.100.0.10*.

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr"
 --output text --name my-cluster --region region-code
```

- This example provides a `kubelet` argument to set a custom `max-pods` value using the `bootstrap.sh` script included with the Amazon EKS optimized AMI. The node group name can't be longer than 63 characters. It must start with letter or digit, but can also include hyphens and underscores for the remaining characters. For help with selecting *my-max-pods-value*, see the section called "Amazon EKS recommended maximum Pods for each Amazon EC2 instance type".

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

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

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

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

Create a node group with the following command.

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

User data in a launch template

Specify the following information in the user data section of your launch template. Replace every *`example value`* with your own values. The `--apiserver-endpoint`, `--b64-cluster-ca`, and `--dns-cluster-ip` arguments are optional. However, defining them allows the `bootstrap.sh` script to avoid making a `describeCluster` call. This is useful in private cluster setups or clusters where you're scaling in and out nodes frequently. For more information on the `bootstrap.sh` script, see the bootstrap.sh file on GitHub.

- The only required argument is the cluster name (*my-cluster*).
- To retrieve the *certificate-authority* for your cluster, run the following command.

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

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

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

- The value for `--dns-cluster-ip` is your service CIDR with `.10` at the end. To retrieve the *service-cidr* for your cluster, run the following command. For example, if the returned value for is `ipv4 10.100.0.0/16`, then your value is *10.100.0.10*.

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr"
 --output text --name my-cluster --region region-code
```

- This example provides a `kubelet` argument to set a custom `max-pods` value using the `bootstrap.sh` script included with the Amazon EKS optimized AMI. For help with selecting *my-max-pods-value*, see the section called "Amazon EKS recommended maximum Pods for each Amazon EC2 instance type".

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

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

#!/bin/bash
set -ex
/etc/eks/bootstrap.sh my-cluster \
  --b64-cluster-ca certificate-authority \
  --apiserver-endpoint api-server-endpoint \
  --dns-cluster-ip service-cidr.10 \
  --kubelet-extra-args '--max-pods=my-max-pods-value' \
  --use-max-pods false

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

Provide user data to pass arguments to the `Start-EKSBootstrap.ps1` file included with an Amazon EKS optimized Windows AMI

Bootstrapping is a term used to describe adding commands that can be run when an instance starts. You can pass arguments to the `Start-EKSBootstrap.ps1` script by using `eksctl` without specifying a launch template. Or you can do so by specifying the information in the user data section of a launch template.

If you want to specify a custom Windows AMI ID, keep in mind the following considerations:

- You must use a launch template and give the required bootstrap commands in the user data section. To retrieve your desired Windows ID, you can use the table in Create nodes with optimized Windows AMIs.
- There are several limits and conditions. For example, you must add `eks:kube-proxy-windows` to your AWS IAM Authenticator configuration map. For more information, see the section called “Limits and conditions when specifying an AMI ID”.

Specify the following information in the user data section of your launch template. Replace every *`example value`* with your own values. The `-APIServerEndpoint`, `-Base64ClusterCA`, and `-DNSClusterIP` arguments are optional. However, defining them allows the `Start-EKSBootstrap.ps1` script to avoid making a `describeCluster` call.

- The only required argument is the cluster name (*`my-cluster`*).
- To retrieve the *`certificate-authority`* for your cluster, run the following command.

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

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

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

- The value for `--dns-cluster-ip` is your service CIDR with `.10` at the end. To retrieve the *`service-cidr`* for your cluster, run the following command. For example, if the returned value for is `ipv4 10.100.0.0/16`, then your value is *`10.100.0.10`*.

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr" --
output text --name my-cluster --region region-code
```

- For additional arguments, see the section called "Bootstrap script configuration parameters".

> **Note**
>
> If you're using custom service CIDR, then you need to specify it using the `-ServiceCIDR` parameter. Otherwise, the DNS resolution for Pods in the cluster will fail.

```
<powershell>
[string]$EKSBootstrapScriptFile = "$env:ProgramFiles\Amazon\EKS\Start-EKSBootstrap.ps1"
& $EKSBootstrapScriptFile -EKSClusterName my-cluster `
  -Base64ClusterCA certificate-authority `
  -APIServerEndpoint api-server-endpoint `
  -DNSClusterIP service-cidr.10
</powershell>
```

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

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

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

Limits and conditions when specifying an AMI ID

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

- You must create a new node group to switch between specifying an AMI ID in a launch template and not specifying an AMI ID.
- You aren't notified in the console when a newer AMI version is available. To update your node group to a newer AMI version, you need to create a new version of your launch template with an updated AMI ID. Then, you need to update the node group with the new launch template version.
- The following fields can't be set in the API if you specify an AMI ID:
  - `amiType`
  - `releaseVersion`
  - `version`
- Any `taints` set in the API are applied asynchronously if you specify an AMI ID. To apply taints prior to a node joining the cluster, you must pass the taints to `kubelet` in your user data using the `--register-with-taints` command line flag. For more information, see kubelet in the Kubernetes documentation.
- When specifying a custom AMI ID for Windows managed node groups, add `eks:kube-proxy-windows` to your AWS IAM Authenticator configuration map. This is required for DNS to function properly.

  a. Open the AWS IAM Authenticator configuration map for editing.

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

  b. Add this entry to the `groups` list under each `rolearn` associated with Windows nodes. Your configuration map should look similar to aws-auth-cm-windows.yaml.

  ```
  - eks:kube-proxy-windows
  ```

  c. Save the file and exit your text editor.

Delete a managed node group from your cluster

This topic describes how you can delete an Amazon EKS managed node group. When you delete a managed node group, Amazon EKS first sets the minimum, maximum, and desired size of your Auto Scaling group to zero. This then causes your node group to scale down.

Before each instance is terminated, Amazon EKS sends a signal to drain the Pods from that node. If the Pods haven't drained after a few minutes, Amazon EKS lets Auto Scaling continue the termination of the instance. After every instance is terminated, the Auto Scaling group is deleted.

> **Important**
>
> If you delete a managed node group that uses a node IAM role that isn't used by any other managed node group in the cluster, the role is removed from the `aws-auth` `ConfigMap`. If any of the self-managed node groups in the cluster are using the same node IAM role, the self-managed nodes move to the `NotReady` status. Additionally, the cluster operation is also disrupted. To add a mapping for the role you're using only for the self-managed node groups, see the section called "Create access entries", if your cluster's platform version is at least minimum version listed in the prerequisites section of Grant IAM users access to Kubernetes with EKS access entries. If your platform version is earlier than the required minimum version for access entries, you can add the entry back to the `aws-auth` `ConfigMap`. For more information, enter `eksctl create iamidentitymapping --help` in your terminal.

You can delete a managed node group with:

- the section called "`eksctl`"
- the section called "AWS Management Console"
- the section called "AWS CLI"

`eksctl`

Delete a managed node group with `eksctl`

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

```
eksctl delete nodegroup \
```

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

For more options, see [Deleting and draining nodegroups] in the `eksctl` documentation.

AWS Management Console

Delete a managed node group with AWS Management Console

1. Open the [Amazon EKS console].
2. On the **Clusters** page, choose the cluster that contains the node group to delete.
3. On the selected cluster page, choose the **Compute** tab.
4. In the **Node groups** section, choose the node group to delete. Then choose **Delete**.
5. In the **Delete node group** confirmation dialog box, enter the name of the node group. Then choose **Delete**.

AWS CLI

Delete a managed node group with AWS CLI

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

```
aws eks delete-nodegroup \
  --cluster-name my-cluster \
  --nodegroup-name my-mng \
  --region region-code
```

2. Use the arrow keys on your keyboard to scroll through the response output. Press the q key when you're finished.

   For more options, see the `delete-nodegroup` command in the *AWS CLI Command Reference*.

Maintain nodes yourself with self-managed nodes

A cluster contains one or more Amazon EC2 nodes that Pods are scheduled on. Amazon EKS nodes run in your AWS account and connect to the control plane of your cluster through the cluster API server endpoint. You're billed for them based on Amazon EC2 prices. For more information, see Amazon EC2 pricing.

A cluster can contain several node groups. Each node group contains one or more nodes that are deployed in an Amazon EC2 Auto Scaling group. The instance type of the nodes within the group can vary, such as when using attribute-based instance type selection with Karpenter. All instances in a node group must use the Amazon EKS node IAM role.

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

If you restrict access to the public endpoint of your cluster using CIDR blocks, we recommend that you also enable private endpoint access. This is so that nodes can communicate with the cluster. Without the private endpoint enabled, the CIDR blocks that you specify for public access must include the egress sources from your VPC. For more information, see the section called "Configure endpoint access".

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

Topics

Create self-managed Amazon Linux nodes

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

Management Console. If you need to launch nodes on AWS Outposts, see the section called "Nodes".

- An existing Amazon EKS cluster. To deploy one, see the section called "Create a cluster". If you have subnets in the AWS Region where you have AWS Outposts, AWS Wavelength, or AWS Local Zones enabled, those subnets must not have been passed in when you created your cluster.
- An existing IAM role for the nodes to use. To create one, see the section called "Node IAM role". If this role doesn't have either of the policies for the VPC CNI, the separate role that follows is required for the VPC CNI pods.
- (Optional, but recommended) The Amazon VPC CNI plugin for Kubernetes add-on configured with its own IAM role that has the necessary IAM policy attached to it. For more information, see the section called "Configure for IRSA".
- Familiarity with the considerations listed in Choose an optimal Amazon EC2 node instance type. Depending on the instance type you choose, there may be additional prerequisites for your cluster and VPC.

You can launch self-managed Linux nodes using either of the following:

- the section called "`eksctl`"
- the section called "AWS Management Console"

`eksctl`

Launch self-managed Linux nodes using `eksctl`

1. Install version `0.204.0` or later of the `eksctl` command line tool installed on your device or AWS CloudShell. To install or update `eksctl`, see Installation in the `eksctl` documentation.
2. (Optional) If the **AmazonEKS_CNI_Policy** managed IAM policy is attached to your Amazon EKS node IAM role, we recommend assigning it to an IAM role that you associate to the Kubernetes `aws-node` service account instead. For more information, see the section called "Configure for IRSA".
3. The following command creates a node group in an existing cluster. Replace *`al-nodes`* with a name for your node group. The node group name can't be longer than 63 characters. It must start with letter or digit, but can also include hyphens and underscores for the remaining characters. Replace *`my-cluster`* with the name of your cluster. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphanumeric

character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in. Replace the remaining *example value* with your own values. The nodes are created with the same Kubernetes version as the control plane, by default.

Before choosing a value for `--node-type`, review [Choose an optimal Amazon EC2 node instance type](#).

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

Create your node group with the following command.

> **Important**
>
> If you want to deploy a node group to AWS Outposts, Wavelength, or Local Zone subnets, there are additional considerations:
>
> - The subnets must not have been passed in when you created the cluster.
> - You must create the node group with a config file that specifies the subnets and `volumeType: gp2`. For more information, see Create a nodegroup from a config file and Config file schema in the `eksctl` documentation.

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

To deploy a node group that:

- can assign a significantly higher number of IP addresses to Pods than the default configuration, see the section called “Increase IP addresses”.
- can assign `IPv4` addresses to Pods from a different CIDR block than that of the instance, see the section called “Custom networking”.
- can assign `IPv6` addresses to Pods and services, see the section called “IPv6”.
- use the `containerd` runtime, you must deploy the node group using a `config` file. For more information, see the section called “Test Amazon Linux 2 migration from Docker to `containerd`”.
- don’t have outbound internet access, see the section called “Private clusters”.

For a complete list of all available options and defaults, enter the following command.

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

If nodes fail to join the cluster, then see the section called “Nodes fail to join cluster” in the Troubleshooting chapter.

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

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

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

   For more information, see Restrict access to the instance profile assigned to the worker node.

AWS Management Console

Step 1: Launch self-managed Linux nodes using AWS Management Console

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

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2022-12-23/
amazon-eks-nodegroup.yaml
```

2. Wait for your cluster status to show as `ACTIVE`. If you launch your nodes before the cluster is active, the nodes fail to register with the cluster and you will have to relaunch them.
3. Open the AWS CloudFormation console.
4. Choose **Create stack** and then select **With new resources (standard)**.
5. For **Specify template**, select **Upload a template file** and then select **Choose file**.
6. Select the `amazon-eks-nodegroup.yaml` file that you downloaded.
7. Select **Next**.
8. On the **Specify stack details** page, enter the following parameters accordingly, and then choose **Next**:
   - **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it *`my-cluster-nodes`*. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.
   - **ClusterName**: Enter the name that you used when you created your Amazon EKS cluster. This name must equal the cluster name or your nodes can't join the cluster.
   - **ClusterControlPlaneSecurityGroup**: Choose the **SecurityGroups** value from the AWS CloudFormation output that you generated when you created your VPC.

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

     a. Open the Amazon EKS console.
     b. Choose the name of the cluster.
     c. Choose the **Networking** tab.
     d. Use the **Additional security groups** value as a reference when selecting from the **ClusterControlPlaneSecurityGroup** dropdown list.
   - **NodeGroupName**: Enter a name for your node group. This name can be used later to identify the Auto Scaling node group that's created for your nodes. The node group name can't be longer than 63 characters. It must start with letter or digit, but can also include hyphens and underscores for the remaining characters.

- **NodeAutoScalingGroupMinSize**: Enter the minimum number of nodes that your node Auto Scaling group can scale in to.
- **NodeAutoScalingGroupDesiredCapacity**: Enter the desired number of nodes to scale to when your stack is created.
- **NodeAutoScalingGroupMaxSize**: Enter the maximum number of nodes that your node Auto Scaling group can scale out to.
- **NodeInstanceType**: Choose an instance type for your nodes. For more information, see the section called “Amazon EC2 instance types”.
- **NodeImageIdSSMParam**: Pre-populated with the Amazon EC2 Systems Manager parameter of a recent Amazon EKS optimized AMI for a variable Kubernetes version. To use a different Kubernetes minor version supported with Amazon EKS, replace *1.XX* with a different supported version. We recommend specifying the same Kubernetes version as your cluster.

  You can also replace *amazon-linux-2* with a different AMI type. For more information, see the section called “Get latest IDs”.

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

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

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

- **BootstrapArguments**: Specify any optional arguments to pass to the node bootstrap script, such as extra `kubelet` arguments. For more information, view the bootstrap script usage information on GitHub.

  To deploy a node group that:

  - can assign a significantly higher number of IP addresses to Pods than the default configuration, see the section called "Increase IP addresses".
  - can assign `IPv4` addresses to Pods from a different CIDR block than that of the instance, see the section called "Custom networking".
  - can assign `IPv6` addresses to Pods and services, see the section called "IPv6".
  - use the `containerd` runtime, you must deploy the node group using a `config` file. For more information, see the section called "Test Amazon Linux 2 migration from Docker to `containerd`".
  - don't have outbound internet access, see the section called "Private clusters".
- **DisableIMDSv1**: By default, each node supports the Instance Metadata Service Version 1 (IMDSv1) and IMDSv2. You can disable IMDSv1. To prevent future nodes and Pods in the node group from using MDSv1, set **DisableIMDSv1** to **true**. For more information about IMDS, see Configuring the instance metadata service. For more information about restricting access to it on your nodes, see Restrict access to the instance profile assigned to the worker node.
- **VpcId**: Enter the ID for the VPC that you created.
- **Subnets**: Choose the subnets that you created for your VPC. If you created your VPC using the steps that are described in Create an Amazon VPC for your Amazon EKS cluster, specify only the private subnets within the VPC for your nodes to launch into. You can see which subnets are private by opening each subnet link from the **Networking** tab of your cluster.

> **Important**
>
> - If any of the subnets are public subnets, then they must have the automatic public IP address assignment setting enabled. If the setting isn't enabled for the public subnet, then any nodes that you deploy to that public subnet won't be assigned a public IP address and won't be able to communicate with the cluster or other

AWS services. If the subnet was deployed before March 26, 2020 using either of the Amazon EKS AWS CloudFormation VPC templates, or by using `eksctl`, then automatic public IP address assignment is disabled for public subnets. For information about how to enable public IP address assignment for a subnet, see Modifying the public IPv4 addressing attribute for your subnet. If the node is deployed to a private subnet, then it's able to communicate with the cluster and other AWS services through a NAT gateway.

- If the subnets don't have internet access, make sure that you're aware of the considerations and extra steps in Deploy private clusters with limited internet access.
- If you select AWS Outposts, Wavelength, or Local Zone subnets, the subnets must not have been passed in when you created the cluster.

9. Select your desired choices on the **Configure stack options** page, and then choose **Next**.
10. Select the check box to the left of **I acknowledge that AWS CloudFormation might create IAM resources.**, and then choose **Create stack**.
11. When your stack has finished creating, select it in the console and choose **Outputs**.
12. Record the **NodeInstanceRole** for the node group that was created. You need this when you configure your Amazon EKS nodes.

Step 2: Enable nodes to join your cluster

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

1. Check to see if you already have an `aws-auth` `ConfigMap`.

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

2. If you are shown an `aws-auth` `ConfigMap`, then update it as needed.

   a. Open the `ConfigMap` for editing.

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

b. Add a new `mapRoles` entry as needed. Set the `rolearn` value to the **NodeInstanceRole** value that you recorded in the previous procedure.

```
[...]
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
[...]
```

c. Save the file and exit your text editor.

3. If you received an error stating "`Error from server (NotFound): configmaps "aws-auth" not found`, then apply the stock `ConfigMap`.

a. Download the configuration map.

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

b. In the `aws-auth-cm.yaml` file, set the `rolearn` value to the **NodeInstanceRole** value that you recorded in the previous procedure. You can do this with a text editor, or by replacing *`my-node-instance-role`* and running the following command:

```
sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-
role|' aws-auth-cm.yaml
```

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

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

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

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

Enter `Ctrl+C` to return to a shell prompt.

> **Note**
>
> If you receive any authorization or resource type errors, see the section called "Unauthorized or access denied (kubectl)" in the troubleshooting topic.

If nodes fail to join the cluster, then see the section called "Nodes fail to join cluster" in the Troubleshooting chapter.

5. (GPU nodes only) If you chose a GPU instance type and the Amazon EKS optimized accelerated AMI, you must apply the NVIDIA device plugin for Kubernetes as a DaemonSet on your cluster. Replace *vX.X.X* with your desired NVIDIA/k8s-device-plugin version before running the following command.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
deployments/static/nvidia-device-plugin.yml
```

Step 3: Additional actions

1. (Optional) Deploy a sample application to test your cluster and Linux nodes.
2. (Optional) If the **AmazonEKS_CNI_Policy** managed IAM policy (if you have an IPv4 cluster) or the *AmazonEKS_CNI_IPv6_Policy* (that you created yourself if you have an IPv6 cluster) is attached to your Amazon EKS node IAM role, we recommend assigning it to an IAM role that you associate to the Kubernetes aws-node service account instead. For more information, see the section called "Configure for IRSA".
3. We recommend blocking Pod access to IMDS if the following conditions are true:
   - You plan to assign IAM roles to all of your Kubernetes service accounts so that Pods only have the minimum permissions that they need.
   - No Pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current AWS Region.

   For more information, see Restrict access to the instance profile assigned to the worker node.

Create self-managed Bottlerocket nodes

Note

Managed node groups might offer some advantages for your use case. For more information, see the section called "Managed node groups".

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

For information about in-place upgrades, see Bottlerocket Update Operator on GitHub.

Important

- Amazon EKS nodes are standard Amazon EC2 instances, and you are billed for them based on normal Amazon EC2 instance prices. For more information, see Amazon EC2 pricing.
- You can launch Bottlerocket nodes in Amazon EKS extended clusters on AWS Outposts, but you can't launch them in local clusters on AWS Outposts. For more information, see *Amazon EKS on AWS Outposts*.
- You can deploy to Amazon EC2 instances with `x86` or Arm processors. However, you can't deploy to instances that have Inferentia chips.
- Bottlerocket is compatible with AWS CloudFormation. However, there is no official CloudFormation template that can be copied to deploy Bottlerocket nodes for Amazon EKS.
- Bottlerocket images don't come with an SSH server or a shell. You can use out-of-band access methods to allow SSH enabling the admin container and to pass some bootstrapping configuration steps with user data. For more information, see these sections in the bottlerocket README.md on GitHub:
  - Exploration
  - Admin container

- Kubernetes settings

This procedure requires `eksctl` version `0.204.0` or later. You can check your version with the following command:

```
eksctl version
```

For instructions on how to install or upgrade `eksctl`, see Installation in the `eksctl` documentation.NOTE: This procedure only works for clusters that were created with `eksctl`.

1. Copy the following contents to your device. Replace *`my-cluster`* with the name of your cluster. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in. Replace *`ng-bottlerocket`* with a name for your node group. The node group name can't be longer than 63 characters. It must start with letter or digit, but can also include hyphens and underscores for the remaining characters. To deploy on Arm instances, replace *`m5.large`* with an Arm instance type. Replace *`my-ec2-keypair-name`* with the name of an Amazon EC2 SSH key pair that you can use to connect using SSH into your nodes with after they launch. If you don't already have an Amazon EC2 key pair, you can create one in the AWS Management Console. For more information, see Amazon EC2 key pairs in the *Amazon EC2 User Guide*. Replace all remaining *`example values`* with your own values. Once you've made the replacements, run the modified command to create the `bottlerocket.yaml` file.

   If specifying an Arm Amazon EC2 instance type, then review the considerations in Amazon EKS optimized Arm Amazon Linux AMIs before deploying. For instructions on how to deploy using a custom AMI, see Building Bottlerocket on GitHub and Custom AMI support in the `eksctl` documentation. To deploy a managed node group, deploy a custom AMI using a launch template. For more information, see the section called "Launch templates".

   Important

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

```
cat >bottlerocket.yaml <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: '1.32'

iam:
  withOIDC: true

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

2. Deploy your nodes with the following command.

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

An example output is as follows.

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

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

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

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

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

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

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

   For more information, see Restrict access to the instance profile assigned to the worker node.

Create self-managed Microsoft Windows nodes

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

> **Important**
>
> - Amazon EKS nodes are standard Amazon EC2 instances, and you are billed for them based on normal Amazon EC2 instance prices. For more information, see Amazon EC2 pricing.
> - You can launch Windows nodes in Amazon EKS extended clusters on AWS Outposts, but you can't launch them in local clusters on AWS Outposts. For more information, see *Amazon EKS on AWS Outposts*.

Enable Windows support for your cluster. We recommend that you review important considerations before you launch a Windows node group. For more information, see the section called "Enable Windows support".

You can launch self-managed Windows nodes with either of the following:

- the section called "`eksctl`"
- the section called "AWS Management Console"

`eksctl`

Launch self-managed Windows nodes using `eksctl`

This procedure requires that you have installed `eksctl`, and that your `eksctl` version is at least `0.204.0`. You can check your version with the following command.

```
eksctl version
```

For instructions on how to install or upgrade `eksctl`, see Installation in the `eksctl` documentation.

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

1. (Optional) If the **AmazonEKS_CNI_Policy** managed IAM policy (if you have an `IPv4` cluster) or the *`AmazonEKS_CNI_IPv6_Policy`* (that you created yourself if you have an `IPv6` cluster) is attached to your Amazon EKS node IAM role, we recommend assigning it to an IAM role that you associate to the Kubernetes `aws-node` service account instead. For more information, see the section called "Configure for IRSA".
2. This procedure assumes that you have an existing cluster. If you don't already have an Amazon EKS cluster and an Amazon Linux node group to add a Windows node group to, we recommend that you follow the section called "Create cluster (eksctl)". This guide provides a complete walkthrough for how to create an Amazon EKS cluster with Amazon Linux nodes.

   Create your node group with the following command. Replace *`region-code`* with the AWS Region that your cluster is in. Replace *`my-cluster`* with your cluster name. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in. Replace *`ng-windows`* with a name for your node group. The node group name can't be longer than 63 characters. It must start with letter or digit, but can also include hyphens and underscores for the remaining characters. For Kubernetes version `1.24` or later, you can replace *`2019`* with `2022` to use Windows Server 2022. Replace the rest of the *`example values`* with your own values.

   > **Important**
   >
   > To deploy a node group to AWS Outposts, AWS Wavelength, or AWS Local Zone subnets, don't pass the AWS Outposts, Wavelength, or Local Zone subnets when you create the cluster. Create the node group with a config file, specifying the AWS Outposts, Wavelength, or Local Zone subnets. For more information, see Create a nodegroup from a config file and Config file schema in the `eksctl` documentation.

```
eksctl create nodegroup \
    --region region-code \
    --cluster my-cluster \
```

```
    --name ng-windows \
    --node-type t2.large \
    --nodes 3 \
    --nodes-min 1 \
    --nodes-max 4 \
    --managed=false \
    --node-ami-family WindowsServer2019FullContainer
```

> **Note**
>
> - If nodes fail to join the cluster, see the section called "Nodes fail to join cluster" in the Troubleshooting guide.
> - To see the available options for `eksctl` commands, enter the following command.
>
> ```
> eksctl command -help
> ```

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

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

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

   For more information, see Restrict access to the instance profile assigned to the worker node.

AWS Management Console

Prerequisites

- An existing Amazon EKS cluster and a Linux node group. If you don't have these resources, we recommend that you create them using one of our guides in *Get started*. These guides describe how to create an Amazon EKS cluster with Linux nodes.

- An existing VPC and security group that meet the requirements for an Amazon EKS cluster. For more information, see the section called "VPC and subnet requirements" and the section called "Security group requirements". The guides in *Get started* create a VPC that meets the requirements. Alternatively, you can also follow Create an Amazon VPC for your Amazon EKS cluster to create one manually.
- An existing Amazon EKS cluster that uses a VPC and security group that meets the requirements of an Amazon EKS cluster. For more information, see the section called "Create a cluster". If you have subnets in the AWS Region where you have AWS Outposts, AWS Wavelength, or AWS Local Zones enabled, those subnets must not have been passed in when you created the cluster.

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

1. Wait for your cluster status to show as `ACTIVE`. If you launch your nodes before the cluster is active, the nodes fail to register with the cluster and you need to relaunch them.
2. Open the AWS CloudFormation console
3. Choose **Create stack**.
4. For **Specify template**, select **Amazon S3 URL**.
5. Copy the following URL and paste it into **Amazon S3 URL**.

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2023-02-09/amazon-eks-windows-nodegroup.yaml
```

6. Select **Next** twice.
7. On the **Quick create stack** page, enter the following parameters accordingly:
   - **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it `my-cluster-nodes`.
   - **ClusterName**: Enter the name that you used when you created your Amazon EKS cluster.

   > **Important**
   >
   > This name must exactly match the name that you used in Step 1: Create your Amazon EKS cluster. Otherwise, your nodes can't join the cluster.

   - **ClusterControlPlaneSecurityGroup**: Choose the security group from the AWS CloudFormation output that you generated when you created your VPC. The following steps show one method to retrieve the applicable group.

a. Open the Amazon EKS console.

b. Choose the name of the cluster.

c. Choose the **Networking** tab.

d. Use the **Additional security groups** value as a reference when selecting from the **ClusterControlPlaneSecurityGroup** dropdown list.

- **NodeGroupName**: Enter a name for your node group. This name can be used later to identify the Auto Scaling node group that's created for your nodes. The node group name can't be longer than 63 characters. It must start with letter or digit, but can also include hyphens and underscores for the remaining characters.
- **NodeAutoScalingGroupMinSize**: Enter the minimum number of nodes that your node Auto Scaling group can scale in to.
- **NodeAutoScalingGroupDesiredCapacity**: Enter the desired number of nodes to scale to when your stack is created.
- **NodeAutoScalingGroupMaxSize**: Enter the maximum number of nodes that your node Auto Scaling group can scale out to.
- **NodeInstanceType**: Choose an instance type for your nodes. For more information, see the section called "Amazon EC2 instance types".

> **Note**
>
> The supported instance types for the latest version of the Amazon VPC CNI plugin for Kubernetes are listed in vpc_ip_resource_limit.go on GitHub. You might need to update your CNI version to use the latest supported instance types. For more information, see the section called "Amazon VPC CNI".

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

you can create one in the AWS Management Console. For more information, see Amazon EC2 key pairs in the *Amazon EC2 User Guide*.

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

- **BootstrapArguments**: Specify any optional arguments to pass to the node bootstrap script, such as extra `kubelet` arguments using `-KubeletExtraArgs`.
- **DisableIMDSv1**: By default, each node supports the Instance Metadata Service Version 1 (IMDSv1) and IMDSv2. You can disable IMDSv1. To prevent future nodes and Pods in the node group from using MDSv1, set **DisableIMDSv1** to **true**. For more information about IMDS, see Configuring the instance metadata service.
- **VpcId**: Select the ID for the VPC that you created.
- **NodeSecurityGroups**: Select the security group that was created for your Linux node group when you created your VPC. If your Linux nodes have more than one security group attached to them, specify all of them. This for, for example, if the Linux node group was created with `eksctl`.
- **Subnets**: Choose the subnets that you created. If you created your VPC using the steps in Create an Amazon VPC for your Amazon EKS cluster, then specify only the private subnets within the VPC for your nodes to launch into.

> **Important**
>
> - If any of the subnets are public subnets, then they must have the automatic public IP address assignment setting enabled. If the setting isn't enabled for the public subnet, then any nodes that you deploy to that public subnet won't be assigned a public IP address and won't be able to communicate with the cluster or other AWS services. If the subnet was deployed before March 26, 2020 using either of the Amazon EKS AWS CloudFormation VPC templates, or by using `eksctl`, then automatic public IP address assignment is disabled for public subnets. For information about how to enable public IP address assignment for a subnet, see Modifying the public IPv4 addressing attribute for your subnet. If the node is deployed to a private subnet, then it's able to communicate with the cluster and other AWS services through a NAT gateway.

- If the subnets don't have internet access, then make sure that you're aware of the considerations and extra steps in [Deploy private clusters with limited internet access](#).
- If you select AWS Outposts, Wavelength, or Local Zone subnets, then the subnets must not have been passed in when you created the cluster.

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

Step 2: Enable nodes to join your cluster

1. Check to see if you already have an `aws-auth` `ConfigMap`.

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

2. If you are shown an `aws-auth` `ConfigMap`, then update it as needed.

   a. Open the `ConfigMap` for editing.

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

   b. Add new `mapRoles` entries as needed. Set the `rolearn` values to the **NodeInstanceRole** values that you recorded in the previous procedures.

   ```
   [...]
   data:
     mapRoles: |
   - rolearn: <ARN of linux instance role (not instance profile)>
         username: system:node:{{EC2PrivateDNSName}}
         groups:
           - system:bootstrappers
           - system:nodes
       - rolearn: <ARN of windows instance role (not instance profile)>
         username: system:node:{{EC2PrivateDNSName}}
         groups:
           - system:bootstrappers
           - system:nodes
           - eks:kube-proxy-windows
   ```

```
[...]
```

c. Save the file and exit your text editor.

3. If you received an error stating "`Error from server (NotFound): configmaps "aws-auth" not found`, then apply the stock `ConfigMap`.

a. Download the configuration map.

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

b. In the `aws-auth-cm-windows.yaml` file, set the `rolearn` values to the applicable **NodeInstanceRole** values that you recorded in the previous procedures. You can do this with a text editor, or by replacing the *`example values`* and running the following command:

```
sed -i.bak -e 's|<ARN of linux instance role (not instance profile)>|my-node-linux-instance-role|' \
    -e 's|<ARN of windows instance role (not instance profile)>|my-node-windows-instance-role|' aws-auth-cm-windows.yaml
```

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

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

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

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

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

Enter `Ctrl+C` to return to a shell prompt.

> **Note**
> If you receive any authorization or resource type errors, see the section called "Unauthorized or access denied (`kubectl`)" in the troubleshooting topic.

If nodes fail to join the cluster, then see the section called “Nodes fail to join cluster” in the Troubleshooting chapter.

Step 3: Additional actions

1. (Optional) Deploy a sample application to test your cluster and Windows nodes.
2. (Optional) If the **AmazonEKS_CNI_Policy** managed IAM policy (if you have an IPv4 cluster) or the *AmazonEKS_CNI_IPv6_Policy* (that you created yourself if you have an IPv6 cluster) is attached to your Amazon EKS node IAM role, we recommend assigning it to an IAM role that you associate to the Kubernetes aws-node service account instead. For more information, see the section called “Configure for IRSA”.
3. We recommend blocking Pod access to IMDS if the following conditions are true:
   - You plan to assign IAM roles to all of your Kubernetes service accounts so that Pods only have the minimum permissions that they need.
   - No Pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current AWS Region.

   For more information, see Restrict access to the instance profile assigned to the worker node.

Create self-managed Ubuntu Linux nodes

> **Note**
>
> Managed node groups might offer some advantages for your use case. For more information, see the section called “Managed node groups”.

This topic describes how to launch Auto Scaling groups of Ubuntu on Amazon Elastic Kubernetes Service (EKS) or Ubuntu Pro on Amazon Elastic Kubernetes Service (EKS) nodes that register with your Amazon EKS cluster. Ubuntu and Ubuntu Pro for EKS are based on the official Ubuntu Minimal LTS, include the custom AWS kernel that is jointly developed with AWS, and have been built specifically for EKS. Ubuntu Pro adds additional security coverage by supporting EKS extended support periods, kernel livepatch, FIPS compliance and the ability to run unlimited Pro containers.

After the nodes join the cluster, you can deploy containerized applications to them. For more information, visit the documentation for Ubuntu on AWS and Custom AMI support in the `eksctl` documentation.

> **Important**
>
> - Amazon EKS nodes are standard Amazon EC2 instances, and you are billed for them based on normal Amazon EC2 instance prices. For more information, see Amazon EC2 pricing.
> - You can launch Ubuntu nodes in Amazon EKS extended clusters on AWS Outposts, but you can't launch them in local clusters on AWS Outposts. For more information, see *Amazon EKS on AWS Outposts*.
> - You can deploy to Amazon EC2 instances with `x86` or Arm processors. However, instances that have Inferentia chips might need to install the Neuron SDK first.

This procedure requires `eksctl` version `0.204.0` or later. You can check your version with the following command:

```
eksctl version
```

For instructions on how to install or upgrade `eksctl`, see Installation in the `eksctl` documentation.NOTE: This procedure only works for clusters that were created with `eksctl`.

1. Copy the following contents to your device. Replace `my-cluster` with the name of your cluster. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphabetic character and can't be longer than 100 characters. Replace `ng-ubuntu` with a name for your node group. The node group name can't be longer than 63 characters. It must start with letter or digit, but can also include hyphens and underscores for the remaining characters. To deploy on Arm instances, replace `m5.large` with an Arm instance type. Replace `my-ec2-keypair-name` with the name of an Amazon EC2 SSH key pair that you can use to connect using SSH into your nodes with after they launch. If you don't already have an Amazon EC2 key pair, you can create one in the AWS Management Console. For more information, see Amazon EC2 key pairs in the Amazon EC2 User Guide. Replace all remaining *`example values`* with your own values. Once you've made the replacements, run the modified command to create the `ubuntu.yaml` file.

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

```
cat >ubuntu.yaml <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: '1.32'

iam:
  withOIDC: true

nodeGroups:
  - name: ng-ubuntu
    instanceType: m5.large
    desiredCapacity: 3
    amiFamily: Ubuntu2204
    iam:
       attachPolicyARNs:
          - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
          - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
          - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
          - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
    ssh:
        allow: true
        publicKeyName: my-ec2-keypair-name
EOF
```

To create an Ubuntu Pro node group, just change the `amiFamily` value to `UbuntuPro2204`.

2. Deploy your nodes with the following command.

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

An example output is as follows.

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

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

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

   For more information, see Restrict access to the instance profile assigned to the worker node.

Update self-managed nodes for your cluster

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

> **Important**
>
> This topic covers node updates for self-managed nodes. If you are using Simplify node lifecycle with managed node groups, see the section called "Update".

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

Migrate applications to a new node group

Create a new node group and migrate your Pods to that group. Migrating to a new node group is more graceful than simply updating the AMI ID in an existing AWS CloudFormation stack.

This is because the migration process taints the old node group as `NoSchedule` and drains the nodes after a new stack is ready to accept the existing Pod workload.

Update an AWS CloudFormation node stack

Update the AWS CloudFormation stack for an existing node group to use the new AMI. This method isn't supported for node groups that were created with `eksctl`.

Migrate applications to a new node group

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

- the section called “`eksctl`”
- the section called “AWS Management Console and AWS CLI”

`eksctl`

Migrate your applications to a new node group with `eksctl`

For more information on using eksctl for migration, see Unmanaged nodegroups in the `eksctl` documentation.

This procedure requires `eksctl` version `0.204.0` or later. You can check your version with the following command:

```
eksctl version
```

For instructions on how to install or upgrade `eksctl`, see Installation in the `eksctl` documentation.

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

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

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

An example output is as follows.

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

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

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

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

     For more information, see [Restrict access to the instance profile assigned to the worker node](https://aws.github.io/aws-eks-best-practices/security/docs/iam/#restrict-access-to-the-instance-profile-assigned-to-the-worker-node).

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

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

```
eksctl create nodegroup \
  --cluster my-cluster \
  --version 1.32 \
  --name standard-nodes-new \
  --node-type t3.medium \
  --nodes 3 \
  --nodes-min 1 \
  --nodes-max 4 \
```

```
--managed=false
```

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

```
kubectl get nodes
```

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

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

AWS Management Console and AWS CLI

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

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

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

4. Update the security groups for both node groups so that they can communicate with each other. For more information, see the section called “Security group requirements”.

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

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

```
oldNodes="old_node_CFN_stack_name"
newNodes="new_node_CFN_stack_name"

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

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

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

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

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

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

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

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: ARN of instance role (not instance profile)
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes>
```

```
    - rolearn: arn:aws:iam::111122223333:role/nodes-1-16-NodeInstanceRole-
U11V27W93CX5
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
```

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

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

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

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

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

8. Use the following command to taint each of the nodes that you want to remove with `NoSchedule`. This is so that new Pods aren't scheduled or rescheduled on the nodes that you're replacing. For more information, see Taints and Tolerations in the Kubernetes documentation.

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

If you're upgrading your nodes to a new Kubernetes version, you can identify and taint all of the nodes of a particular Kubernetes version (in this case, `1.30`) with the following code snippet. The version number can't be later than the Kubernetes version of your control plane. It also can't be more than two minor versions earlier than the Kubernetes version of your control plane. We recommend that you use the same version as your control plane.

```
K8S_VERSION=1.30
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
```

   An example output is as follows. This cluster is using CoreDNS for DNS resolution, but your cluster can return `kube-dns` instead):

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

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

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

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

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

   If you're upgrading your nodes to a new Kubernetes version, identify and drain all of the nodes of a particular Kubernetes version (in this case, *`1.30`*) with the following code snippet.

```
K8S_VERSION=1.30
nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==
\"v$K8S_VERSION\")].metadata.name}")
for node in ${nodes[@]}
do
    echo "Draining $node"
    kubectl drain $node --ignore-daemonsets --delete-local-data
done
```

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

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

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

```
oldNodes="old_node_CFN_stack_name"
newNodes="new_node_CFN_stack_name"

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

b. Open the AWS CloudFormation console.

c. Select your old node stack.

d. Choose **Delete**.

e. In the **Delete stack** confirmation dialog box, choose **Delete stack**.

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

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

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

```
apiVersion: v1
```

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

Save and close the file to apply the updated configmap.

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

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

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

15(Optional) Verify that you're using the latest version of the Amazon VPC CNI plugin for Kubernetes. You might need to update your CNI version to use the latest supported instance types. For more information, see the section called "Amazon VPC CNI".

16If your cluster is using `kube-dns` for DNS resolution (see [migrate-determine-dns-step]), scale in the `kube-dns` deployment to one replica.

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

Update an AWS CloudFormation node stack

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

> **Important**
>
> This topic covers node updates for self-managed nodes. For information about using Simplify node lifecycle with managed node groups, see the section called "Update".

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

> **Note**
>
> This method isn't supported for node groups that were created with `eksctl`. If you created your cluster or node group with `eksctl`, see the section called "Migration".

1. Determine the DNS provider for your cluster.

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

   An example output is as follows. This cluster is using CoreDNS for DNS resolution, but your cluster might return `kube-dns` instead. Your output might look different depending on the version of `kubectl` that you're using.

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

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

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

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

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

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

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

   b. In the left navigation pane, choose **Launch Configurations**, and note the instance type for your existing node launch configuration.

   c. In the left navigation pane, choose **Auto Scaling Groups**, and note the **Desired** instance count for your existing node Auto Scaling group.

5. Open the AWS CloudFormation console.

6. Select your node group stack, and then choose **Update**.

7. Select **Replace current template** and select **Amazon S3 URL**.

8. For **Amazon S3 URL**, paste the following URL into the text area to ensure that you're using the latest version of the node AWS CloudFormation template. Then, choose **Next**:

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2022-12-23/amazon-eks-
nodegroup.yaml
```

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

   - **NodeAutoScalingGroupDesiredCapacity** – Enter the desired instance count that you recorded in a previous step. Or, enter your new desired number of nodes to scale to when your stack is updated.
   - **NodeAutoScalingGroupMaxSize** – Enter the maximum number of nodes to which your node Auto Scaling group can scale out. This value must be at least one node more than your desired capacity. This is so that you can perform a rolling update of your nodes without reducing your node count during the update.
   - **NodeInstanceType** – Choose the instance type your recorded in a previous step. Alternatively, choose a different instance type for your nodes. Before choosing a different instance type, review Choose an optimal Amazon EC2 node instance type. Each Amazon EC2 instance type supports a maximum number of elastic network interfaces (network interface) and each

network interface supports a maximum number of IP addresses. Because each worker node and Pod ,is assigned its own IP address, it's important to choose an instance type that will support the maximum number of Pods that you want to run on each Amazon EC2 node. For a list of the number of network interfaces and IP addresses supported by instance types, see IP addresses per network interface per instance type. For example, the `m5.large` instance type supports a maximum of 30 IP addresses for the worker node and Pods.

> **Note**
>
> The supported instance types for the latest version of the Amazon VPC CNI plugin for Kubernetes are shown in vpc_ip_resource_limit.go on GitHub. You might need to update your Amazon VPC CNI plugin for Kubernetes version to use the latest supported instance types. For more information, see the section called "Amazon VPC CNI".

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

- **NodeImageIdSSMParam** – The Amazon EC2 Systems Manager parameter of the AMI ID that you want to update to. The following value uses the latest Amazon EKS optimized AMI for Kubernetes version `1.32`.

```
/aws/service/eks/optimized-ami/1.32/amazon-linux-2/recommended/image_id
```

You can replace *`1.32`* with a supported Kubernetes version that's the same. Or, it should be up to one version earlier than the Kubernetes version running on your control plane. We recommend that you keep your nodes at the same version as your control plane. You can also replace *`amazon-linux-2`* with a different AMI type. For more information, see the section called "Get latest IDs".

> **Note**
>
> Using the Amazon EC2 Systems Manager parameter enables you to update your nodes in the future without having to look up and specify an AMI ID. If your AWS CloudFormation stack is using this value, any stack update always launches the latest

recommended Amazon EKS optimized AMI for your specified Kubernetes version. This is even the case even if you don’t change any values in the template.

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

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

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

10. (Optional) On the **Options** page, tag your stack resources. Choose **Next**.

11. On the **Review** page, review your information, acknowledge that the stack might create IAM resources, and then choose **Update stack**.

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

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

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

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

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

14(Optional) Verify that you're using the latest version of the Amazon VPC CNI plugin for Kubernetes. You might need to update your Amazon VPC CNI plugin for Kubernetes version to use the latest supported instance types. For more information, see the section called "Amazon VPC CNI".

Simplify compute management with AWS Fargate

This topic discusses using Amazon EKS to run Kubernetes Pods on AWS Fargate. Fargate is a technology that provides on-demand, right-sized compute capacity for containers. With Fargate, you don't have to provision, configure, or scale groups of virtual machines on your own to run containers. You also don't need to choose server types, decide when to scale your node groups, or optimize cluster packing.

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

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

AWS Fargate considerations

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

- Each Pod that runs on Fargate has its own isolation boundary. They don't share the underlying kernel, CPU resources, memory resources, or elastic network interface with another Pod.
- Network Load Balancers and Application Load Balancers (ALBs) can be used with Fargate with IP targets only. For more information, see the section called "Create a network load balancer" and the section called "Application load balancing".
- Fargate exposed services only run on target type IP mode, and not on node IP mode. The recommended way to check the connectivity from a service running on a managed node and a service running on Fargate is to connect via service name.

- Pods must match a Fargate profile at the time that they're scheduled to run on Fargate. Pods that don't match a Fargate profile might be stuck as `Pending`. If a matching Fargate profile exists, you can delete pending Pods that you have created to reschedule them onto Fargate.
- Daemonsets aren't supported on Fargate. If your application requires a daemon, reconfigure that daemon to run as a sidecar container in your Pods.
- Privileged containers aren't supported on Fargate.
- Pods running on Fargate can't specify `HostPort` or `HostNetwork` in the Pod manifest.
- The default `nofile` and `nproc` soft limit is 1024 and the hard limit is 65535 for Fargate Pods.
- GPUs aren't currently available on Fargate.
- Pods that run on Fargate are only supported on private subnets (with NAT gateway access to AWS services, but not a direct route to an Internet Gateway), so your cluster's VPC must have private subnets available. For clusters without outbound internet access, see the section called "Private clusters".
- You can use the Adjust pod resources with Vertical Pod Autoscaler to set the initial correct size of CPU and memory for your Fargate Pods, and then use the Scale pod deployments with Horizontal Pod Autoscaler to scale those Pods. If you want the Vertical Pod Autoscaler to automatically re-deploy Pods to Fargate with larger CPU and memory combinations, set the mode for the Vertical Pod Autoscaler to either `Auto` or `Recreate` to ensure correct functionality. For more information, see the Vertical Pod Autoscaler documentation on GitHub.
- DNS resolution and DNS hostnames must be enabled for your VPC. For more information, see Viewing and updating DNS support for your VPC.
- Amazon EKS Fargate adds defense-in-depth for Kubernetes applications by isolating each Pod within a Virtual Machine (VM). This VM boundary prevents access to host-based resources used by other Pods in the event of a container escape, which is a common method of attacking containerized applications and gain access to resources outside of the container.

  Using Amazon EKS doesn't change your responsibilities under the shared responsibility model. You should carefully consider the configuration of cluster security and governance controls. The safest way to isolate an application is always to run it in a separate cluster.
- Fargate profiles support specifying subnets from VPC secondary CIDR blocks. You might want to specify a secondary CIDR block. This is because there's a limited number of IP addresses available in a subnet. As a result, there's also a limited number of Pods that can be created in the cluster. By using different subnets for Pods, you can increase the number of available IP addresses. For more information, see Adding IPv4 CIDR blocks to a VPC.

- The Amazon EC2 instance metadata service (IMDS) isn't available to Pods that are deployed to Fargate nodes. If you have Pods that are deployed to Fargate that need IAM credentials, assign them to your Pods using IAM roles for service accounts. If your Pods need access to other information available through IMDS, then you must hard code this information into your Pod spec. This includes the AWS Region or Availability Zone that a Pod is deployed to.
- You can't deploy Fargate Pods to AWS Outposts, AWS Wavelength, or AWS Local Zones.
- Amazon EKS must periodically patch Fargate Pods to keep them secure. We attempt the updates in a way that reduces impact, but there are times when Pods must be deleted if they aren't successfully evicted. There are some actions you can take to minimize disruption. For more information, see the section called "OS patching events".
- The Amazon VPC CNI plugin for Amazon EKS is installed on Fargate nodes. You can't use Alternate CNI plugins for Amazon EKS clusters with Fargate nodes.
- A Pod running on Fargate automatically mounts an Amazon EFS file system, without needing manual driver installation steps. You can't use dynamic persistent volume provisioning with Fargate nodes, but you can use static provisioning.
- Amazon EKS doesn't support Fargate Spot.
- You can't mount Amazon EBS volumes to Fargate Pods.
- You can run the Amazon EBS CSI controller on Fargate nodes, but the Amazon EBS CSI node DaemonSet can only run on Amazon EC2 instances.
- After a Kubernetes Job is marked `Completed` or `Failed`, the Pods that the Job creates normally continue to exist. This behavior allows you to view your logs and results, but with Fargate you will incur costs if you don't clean up the Job afterwards.

  To automatically delete the related Pods after a Job completes or fails, you can specify a time period using the time-to-live (TTL) controller. The following example shows specifying `.spec.ttlSecondsAfterFinished` in your Job manifest.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: busybox
spec:
  template:
    spec:
      containers:
      - name: busybox
        image: busybox
```

```
        command: ["/bin/sh", "-c", "sleep 10"]
      restartPolicy: Never
  ttlSecondsAfterFinished: 60 # <-- TTL controller
```

Fargate Comparison Table

Criteria	AWS Fargate
Can be deployed to AWS Outposts	No
Can be deployed to an AWS Local Zone	No
Can run containers that require Windows	No
Can run containers that require Linux	Yes
Can run workloads that require the Inferentia chip	No
Can run workloads that require a GPU	No
Can run workloads that require Arm processor s	No
Can run AWS Bottlerocket	No
Pods share a kernel runtime environment with other Pods	No – Each Pod has a dedicated kernel
Pods share CPU, memory, storage, and network resources with other Pods.	No – Each Pod has dedicated resources and can be sized independently to maximize resource utilization.
Pods can use more hardware and memory than requested in Pod specs	No – The Pod can be re-deployed using a larger vCPU and memory configuration though.
Must deploy and manage Amazon EC2 instances	No

Criteria	AWS Fargate
Must secure, maintain, and patch the operating system of Amazon EC2 instances	No
Can provide bootstrap arguments at deployment of a node, such as extra kubelet arguments.	No
Can assign IP addresses to Pods from a different CIDR block than the IP address assigned to the node.	No
Can SSH into node	No – There's no node host operating system to SSH to.
Can deploy your own custom AMI to nodes	No
Can deploy your own custom CNI to nodes	No
Must update node AMI on your own	No
Must update node Kubernetes version on your own	No – You don't manage nodes.
Can use Amazon EBS storage with Pods	No
Can use Amazon EFS storage with Pods	Yes
Can use Amazon FSx for Lustre storage with Pods	No
Can use Network Load Balancer for services	Yes, when using the Create a network load balancer
Pods can run in a public subnet	No
Can assign different VPC security groups to individual Pods	Yes

Criteria	AWS Fargate
Can run Kubernetes DaemonSets	No
Support `HostPort` and `HostNetwork` in the Pod manifest	No
AWS Region availability	Some Amazon EKS supported regions
Can run containers on Amazon EC2 dedicated hosts	No
Pricing	Cost of an individual Fargate memory and CPU configuration. Each Pod has its own cost. For more information, see AWS Fargate pricing.

Get started with AWS Fargate for your cluster

This topic describes how to get started running Pods on AWS Fargate with your Amazon EKS cluster.

If you restrict access to the public endpoint of your cluster using CIDR blocks, we recommend that you also enable private endpoint access. This way, Fargate Pods can communicate with the cluster. Without the private endpoint enabled, the CIDR blocks that you specify for public access must include the outbound sources from your VPC. For more information, see the section called "Configure endpoint access".

Prerequisite

An existing cluster. If you don't already have an Amazon EKS cluster, see *Get started*.

Step 1: Ensure that existing nodes can communicate with Fargate Pods

If you're working with a new cluster with no nodes, or a cluster with only managed node groups (see the section called "Managed node groups"), you can skip to the section called "Step 2: Create a Fargate Pod execution role".

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

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

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

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

Step 2: Create a Fargate Pod execution role

When your cluster creates Pods on AWS Fargate, the components that run on the Fargate infrastructure must make calls to AWS APIs on your behalf. The Amazon EKS Pod execution role provides the IAM permissions to do this. To create an AWS Fargate Pod execution role, see the section called "Pod execution IAM role".

> **Note**
>
> If you created your cluster with eksctl using the --fargate option, your cluster already has a Pod execution role that you can find in the IAM console with the pattern eksctl-my-cluster-FargatePodExecutionRole-ABCDEFGHIJKL. Similarly, if you use eksctl to create your Fargate profiles, eksctl creates your Pod execution role if one isn't already created.

Step 3: Create a Fargate profile for your cluster

Before you can schedule Pods that are running on Fargate in your cluster, you must define a Fargate profile that specifies which Pods use Fargate when they're launched. For more information, see the section called "Define profiles".

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

You can create a Fargate profile using either of these tools:

- the section called "`eksctl`"
- the section called "AWS Management Console"

`eksctl`

This procedure requires `eksctl` version `0.204.0` or later. You can check your version with the following command:

```
eksctl version
```

For instructions on how to install or upgrade `eksctl`, see Installation in the `eksctl` documentation.

To create a Fargate profile with `eksctl`

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

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

You can use certain wildcards for *`my-kubernetes-namespace`* and *`key=value`* labels. For more information, see the section called "Fargate profile wildcards".

AWS Management Console

To create a Fargate profile with AWS Management Console

1. Open the [Amazon EKS console](#).
2. Choose the cluster to create a Fargate profile for.
3. Choose the **Compute** tab.
4. Under **Fargate profiles**, choose **Add Fargate profile**.
5. On the **Configure Fargate profile** page, do the following:
   a. For **Name**, enter a name for your Fargate profile. The name must be unique.
   b. For **Pod execution role**, choose the Pod execution role to use with your Fargate profile. Only the IAM roles with the `eks-fargate-pods.amazonaws.com` service principal are shown. If you don't see any roles listed, you must create one. For more information, see the section called "Pod execution IAM role".
   c. Modify the selected **Subnets** as needed.

   > **Note**
   >
   > Only private subnets are supported for Pods that are running on Fargate.

   d. For **Tags**, you can optionally tag your Fargate profile. These tags don't propagate to other resources that are associated with the profile such as Pods.
   e. Choose **Next**.
6. On the **Configure Pod selection** page, do the following:
   a. For **Namespace**, enter a namespace to match for Pods.
      - You can use specific namespaces to match, such as `kube-system` or `default`.
      - You can use certain wildcards (for example, `prod-*`) to match multiple namespaces (for example, `prod-deployment` and `prod-test`). For more information, see the section called "Fargate profile wildcards".
   b. (Optional) Add Kubernetes labels to the selector. Specifically add them to the one that the Pods in the specified namespace need to match.
      - You can add the label `infrastructure: fargate` to the selector so that only Pods in the specified namespace that also have the `infrastructure: fargate` Kubernetes label match the selector.

- You can use certain wildcards (for example, `key?: value?`) to match multiple namespaces (for example, `keya: valuea` and `keyb: valueb`). For more information, see the section called "Fargate profile wildcards".

c. Choose **Next**.

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

Step 4: Update CoreDNS

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

> **Note**
>
> If you created your cluster with `eksctl` using the `--fargate` option, then you can skip to the section called "Next steps".

1. Create a Fargate profile for CoreDNS with the following command. Replace *`my-cluster`* with your cluster name, *`111122223333`* with your account ID, *`AmazonEKSFargatePodExecutionRole`* with the name of your Pod execution role, and *`0000000000000001`*, *`0000000000000002`*, and *`0000000000000003`* with the IDs of your private subnets. If you don't have a Pod execution role, you must create one first (see the section called "Step 2: Create a Fargate Pod execution role").

> **Important**
>
> The role ARN can't include a path other than `/`. For example, if the name of your role is `development/apps/my-role`, you need to change it to `my-role` when specifying the ARN for the role. The format of the role ARN must be `arn:aws:iam::111122223333:role/role-name`.

```
aws eks create-fargate-profile \
    --fargate-profile-name coredns \
    --cluster-name my-cluster \
```

```
    --pod-execution-role-arn arn:aws:iam::111122223333:role/
AmazonEKSFargatePodExecutionRole \
    --selectors namespace=kube-system,labels={k8s-app=kube-dns} \
    --subnets subnet-0000000000000001 subnet-0000000000000002 subnet-0000000000000003
```

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

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

Next steps

- You can start migrating your existing applications to run on Fargate with the following workflow.
  a. the section called "Create a Fargate profile" that matches your application's Kubernetes namespace and Kubernetes labels.
  b. Delete and re-create any existing Pods so that they're scheduled on Fargate. For example, the following command triggers a rollout of the `coredns` deployment. You can modify the namespace and deployment type to update your specific Pods.

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

- Deploy the the section called "Application load balancing" to allow Ingress objects for your Pods running on Fargate.
- You can use the the section called "Vertical Pod Autoscaler" to set the initial correct size of CPU and memory for your Fargate Pods, and then use the the section called "Horizontal Pod Autoscaler" to scale those Pods. If you want the Vertical Pod Autoscaler to automatically re-deploy Pods to Fargate with higher CPU and memory combinations, set the Vertical Pod Autoscaler's mode to either `Auto` or `Recreate`. This is to ensure correct functionality. For more information, see the Vertical Pod Autoscaler documentation on GitHub.
- You can set up the AWS Distro for OpenTelemetry (ADOT) collector for application monitoring by following these instructions.

Define which Pods use AWS Fargate when launched

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

As an administrator, you can use a Fargate profile to declare which Pods run on Fargate. You can do this through the profile's selectors. You can add up to five selectors to each profile. Each selector must contain a namespace. The selector can also include labels. The label field consists of multiple optional key-value pairs. Pods that match a selector are scheduled on Fargate. Pods are matched using a namespace and the labels that are specified in the selector. If a namespace selector is defined without labels, Amazon EKS attempts to schedule all the Pods that run in that namespace onto Fargate using the profile. If a to-be-scheduled Pod matches any of the selectors in the Fargate profile, then that Pod is scheduled on Fargate.

If a Pod matches multiple Fargate profiles, you can specify which profile a Pod uses by adding the following Kubernetes label to the Pod specification: `eks.amazonaws.com/fargate-profile: my-fargate-profile`. The Pod must match a selector in that profile to be scheduled onto Fargate. Kubernetes affinity/anti-affinity rules do not apply and aren't necessary with Amazon EKS Fargate Pods.

When you create a Fargate profile, you must specify a Pod execution role. This execution role is for the Amazon EKS components that run on the Fargate infrastructure using the profile. It's added to the cluster's Kubernetes Role Based Access Control (RBAC) for authorization. That way, the `kubelet` that runs on the Fargate infrastructure can register with your Amazon EKS cluster and appear in your cluster as a node. The Pod execution role also provides IAM permissions to the Fargate infrastructure to allow read access to Amazon ECR image repositories. For more information, see the section called "Pod execution IAM role".

Fargate profiles can't be changed. However, you can create a new updated profile to replace an existing profile, and then delete the original.

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

If any Fargate profiles in a cluster are in the `DELETING` status, you must wait until after the Fargate profile is deleted before you create other profiles in that cluster.

> **Note**
>
> Fargate does not currently support Kubernetes topologySpreadConstraints.

Amazon EKS and Fargate spread Pods across each of the subnets that's defined in the Fargate profile. However, you might end up with an uneven spread. If you must have an even spread, use two Fargate profiles. Even spread is important in scenarios where you want to deploy two replicas and don't want any downtime. We recommend that each profile has only one subnet.

Fargate profile components

The following components are contained in a Fargate profile.

Pod execution role

When your cluster creates Pods on AWS Fargate, the `kubelet` that's running on the Fargate infrastructure must make calls to AWS APIs on your behalf. For example, it needs to make calls to pull container images from Amazon ECR. The Amazon EKS Pod execution role provides the IAM permissions to do this.

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

Subnets

The IDs of subnets to launch Pods into that use this profile. At this time, Pods that are running on Fargate aren't assigned public IP addresses. Therefore, only private subnets with no direct route to an Internet Gateway are accepted for this parameter.

Selectors

The selectors to match for Pods to use this Fargate profile. You might specify up to five selectors in a Fargate profile. The selectors have the following components:

- **Namespace** – You must specify a namespace for a selector. The selector only matches Pods that are created in this namespace. However, you can create multiple selectors to target multiple namespaces.

- **Labels –** You can optionally specify Kubernetes labels to match for the selector. The selector only matches Pods that have all of the labels that are specified in the selector.

Fargate profile wildcards

In addition to characters allowed by Kubernetes, you're allowed to use `*` and `?` in the selector criteria for namespaces, label keys, and label values:

- `*` represents none, one, or multiple characters. For example, `prod*` can represent `prod` and `prod-metrics`.
- `?` represents a single character (for example, `value?` can represent `valuea`). However, it can't represent `value` and `value-a`, because `?` can only represent exactly one character.

These wildcard characters can be used in any position and in combination (for example, `prod*`, `*dev`, and `frontend*?`). Other wildcards and forms of pattern matching, such as regular expressions, aren't supported.

If there are multiple matching profiles for the namespace and labels in the Pod spec, Fargate picks up the profile based on alphanumeric sorting by profile name. For example, if both profile A (with the name `beta-workload`) and profile B (with the name `prod-workload`) have matching selectors for the Pods to be launched, Fargate picks profile A (`beta-workload`) for the Pods. The Pods have labels with profile A on the Pods (for example, `eks.amazonaws.com/fargate-profile=beta-workload`).

If you want to migrate existing Fargate Pods to new profiles that use wildcards, there are two ways to do so:

- Create a new profile with matching selectors, then delete the old profiles. Pods labeled with old profiles are rescheduled to new matching profiles.
- If you want to migrate workloads but aren't sure what Fargate labels are on each Fargate Pod, you can use the following method. Create a new profile with a name that sorts alphanumerically first among the profiles on the same cluster. Then, recycle the Fargate Pods that need to be migrated to new profiles.

Create a Fargate profile

This section describes how to create a Fargate profile. You also must have created a Pod execution role to use for your Fargate profile. For more information, see the section called "Pod execution IAM role". Pods that are running on Fargate are only supported on private subnets with NAT gateway access to AWS services, but not a direct route to an Internet Gateway. This is so that your cluster's VPC must have private subnets available.

You can create a profile with the following:

- the section called "`eksctl`"
- the section called "AWS Management Console"

`eksctl`

To create a Fargate profile with `eksctl`

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

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

You can use certain wildcards for `my-kubernetes-namespace` and `key=value` labels. For more information, see the section called "Fargate profile wildcards".

AWS Management Console

To create a Fargate profile with AWS Management Console

1. Open the Amazon EKS console.
2. Choose the cluster to create a Fargate profile for.
3. Choose the **Compute** tab.
4. Under **Fargate profiles**, choose **Add Fargate profile**.

5. On the **Configure Fargate profile** page, do the following:

    a. For **Name**, enter a unique name for your Fargate profile, such as `my-profile`.

    b. For **Pod execution role**, choose the Pod execution role to use with your Fargate profile. Only the IAM roles with the `eks-fargate-pods.amazonaws.com` service principal are shown. If you don't see any roles listed, you must create one. For more information, see the section called "Pod execution IAM role".

    c. Modify the selected **Subnets** as needed.

    > **Note**
    >
    > Only private subnets are supported for Pods that are running on Fargate.

    d. For **Tags**, you can optionally tag your Fargate profile. These tags don't propagate to other resources that are associated with the profile, such as Pods.

    e. Choose **Next**.

6. On the **Configure Pod selection** page, do the following:

    a. For **Namespace**, enter a namespace to match for Pods.

    - You can use specific namespaces to match, such as `kube-system` or `default`.
    - You can use certain wildcards (for example, `prod-*`) to match multiple namespaces (for example, `prod-deployment` and `prod-test`). For more information, see the section called "Fargate profile wildcards".

    b. (Optional) Add Kubernetes labels to the selector. Specifically, add them to the one that the Pods in the specified namespace need to match.

    - You can add the label `infrastructure: fargate` to the selector so that only Pods in the specified namespace that also have the `infrastructure: fargate` Kubernetes label match the selector.
    - You can use certain wildcards (for example, `key?: value?`) to match multiple namespaces (for example, `keya: valuea` and `keyb: valueb`). For more information, see the section called "Fargate profile wildcards".

    c. Choose **Next**.

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

Delete a Fargate profile

This topic describes how to delete a Fargate profile. When you delete a Fargate profile, any Pods that were scheduled onto Fargate with the profile are deleted. If those Pods match another Fargate profile, then they're scheduled on Fargate with that profile. If they no longer match any Fargate profiles, then they aren't scheduled onto Fargate and might remain as pending.

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

You can delete a profile with any of the following tools:

- the section called "`eksctl`"
- the section called "AWS Management Console"
- the section called "AWS CLI"

`eksctl`

Delete a Fargate profile with `eksctl`

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

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

AWS Management Console

Delete a Fargate profile with AWS Management Console

1. Open the Amazon EKS console.
2. In the left navigation pane, choose **Clusters**. In the list of clusters, choose the cluster that you want to delete the Fargate profile from.
3. Choose the **Compute** tab.
4. Choose the Fargate profile to delete, and then choose **Delete**.
5. On the **Delete Fargate profile** page, enter the name of the profile, and then choose **Delete**.

AWS CLI

Delete a Fargate profile with AWS CLI

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

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

Understand Fargate Pod configuration details

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

Pod CPU and memory

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

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

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

- The maximum request out of any Init containers is used to determine the Init request vCPU and memory requirements.
- Requests for all long-running containers are added up to determine the long-running request vCPU and memory requirements.

- The larger of the previous two values is chosen for the vCPU and memory request to use for your Pod.
- Fargate adds 256 MB to each Pod's memory reservation for the required Kubernetes components (`kubelet`, `kube-proxy`, and `containerd`).

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

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

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

vCPU value	Memory value
.25 vCPU	0.5 GB, 1 GB, 2 GB
.5 vCPU	1 GB, 2 GB, 3 GB, 4 GB
1 vCPU	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB
2 vCPU	Between 4 GB and 16 GB in 1-GB increments
4 vCPU	Between 8 GB and 30 GB in 1-GB increments
8 vCPU	Between 16 GB and 60 GB in 4-GB increments
16 vCPU	Between 32 GB and 120 GB in 8-GB increments

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

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

```
kubectl describe pod --namespace default pod-name
```

An example output is as follows.

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

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

Fargate storage

A Pod running on Fargate automatically mounts an Amazon EFS file system, without needing manual driver installation steps. You can't use dynamic persistent volume provisioning with Fargate nodes, but you can use static provisioning. For more information, see Amazon EFS CSI Driver on GitHub.

When provisioned, each Pod running on Fargate receives a default 20 GiB of ephemeral storage. This type of storage is deleted after a Pod stops. New Pods launched onto Fargate have encryption of the ephemeral storage volume enabled by default. The ephemeral Pod storage is encrypted with an AES-256 encryption algorithm using AWS Fargate managed keys.

> **Note**
>
> The default usable storage for Amazon EKS Pods that run on Fargate is less than 20 GiB. This is because some space is used by the `kubelet` and other Kubernetes modules that are loaded inside the Pod.

You can increase the total amount of ephemeral storage up to a maximum of 175 GiB. To configure the size with Kubernetes, specify the requests of `ephemeral-storage` resource to each container in a Pod. When Kubernetes schedules Pods, it ensures that the sum of the resource requests for each Pod is less than the capacity of the Fargate task. For more information, see Resource Management for Pods and Containers in the Kubernetes documentation.

Amazon EKS Fargate provisions more ephemeral storage than requested for the purposes of system use. For example, a request of 100 GiB will provision a Fargate task with 115 GiB ephemeral storage.

Set actions for AWS Fargate OS patching events

Amazon EKS periodically patches the OS for AWS Fargate nodes to keep them secure. As part of the patching process, we recycle the nodes to install OS patches. Updates are attempted in a way that creates the least impact on your services. However, if Pods aren't successfully evicted, there are times when they must be deleted. The following are actions that you can take to minimize potential disruptions:

- Set appropriate Pod disruption budgets (PDBs) to control the number of Pods that are down simultaneously.
- Create Amazon EventBridge rules to handle failed evictions before the Pods are deleted.
- Manually restart your affected pods before the eviction date posted in the notification you receive.
- Create a notification configuration in AWS User Notifications.

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

When the AWS Fargate OS is updated, Amazon EKS will send you a notification that includes your affected resources and the date of upcoming pod evictions. If the provided eviction date is inconvenient, you have the option to manually restart your affected pods before the eviction date posted in the notification. Any pods created before the time at which you receive the notification are subject to eviction. Refer to the Kubernetes Documentation for further instructions on how to manually restart your pods.

To limit the number of Pods that are down at one time when Pods are recycled, you can set Pod disruption budgets (PDBs). You can use PDBs to define minimum availability based on the requirements of each of your applications while still allowing updates to occur. Your PDB's minimum availability must be less than 100%. For more information, see Specifying a Disruption Budget for your Application in the Kubernetes Documentation.

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

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

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

```
{
    "version": "0",
    "id": "12345678-90ab-cdef-0123-4567890abcde",
    "detail-type": "EKS Fargate Pod Scheduled Termination",
    "source": "aws.eks",
    "account": "111122223333",
    "time": "2021-06-27T12:52:44Z",
    "region": "region-code",
    "resources": [
        "default/my-database-deployment"
    ],
    "detail": {
        "clusterName": "my-cluster",
        "fargateProfileName": "my-fargate-profile",
        "podName": "my-pod-name",
        "podNamespace": "default",
        "evictErrorMessage": "Cannot evict pod as it would violate the pod's disruption
 budget",
        "scheduledTerminationTime": "2021-06-30T12:52:44.832Z[UTC]"
    }
}
```

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

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

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

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

A suitable target can be set for the event to capture it. For a complete list of available targets, see Amazon EventBridge targets in the *Amazon EventBridge User Guide*. You can also create a notification configuration in AWS User Notifications. When using the AWS Management Console to create the notification, under **Event Rules**, choose **Elastic Kubernetes Service (EKS)** for **AWS service name** and **EKS Fargate Pod Scheduled Termination** for **Event type**. For more information, see Getting started with AWS User Notifications in the AWS User Notifications User Guide.

See FAQs: Fargate Pod eviction notice in *AWS re:Post* for frequently asked questions regarding EKS Pod Evictions.

Collect AWS Fargate app and usage metrics

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

Application metrics

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

Usage metrics

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

AWS Fargate usage metrics correspond to AWS service quotas. You can configure alarms that alert you when your usage approaches a service quota. For more information about Fargate service quotas, see the section called "Service quotas".

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

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

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

Dimension	Description
`Service`	The name of the AWS service containing the resource. For AWS Fargate usage metrics, the value for this dimension is `Fargate`.
`Type`	The type of entity that's being reported. Currently, the only valid value for AWS Fargate usage metrics is `Resource`.
`Resource`	The type of resource that's running. Currently, AWS Fargate returns informati on on your Fargate On-Demand usage. The resource value for Fargate On-Demand usage is `OnDemand`.

Dimension	Description
	[NOTE] ==== Fargate On-Demand usage combines Amazon EKS Pods using Fargate, Amazon ECS tasks using the Fargate launch type and Amazon ECS tasks using the `FARGATE` capacity provider. ====
`Class`	The class of resource being tracked. Currently, AWS Fargate doesn't use the class dimension.

Creating a CloudWatch alarm to monitor Fargate resource usage metrics

AWS Fargate provides CloudWatch usage metrics that correspond to the AWS service quotas for Fargate On-Demand resource usage. In the Service Quotas console, you can visualize your usage on a graph. You can also configure alarms that alert you when your usage approaches a service quota. For more information, see the section called "Collect metrics".

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

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

Start AWS Fargate logging for your cluster

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

- Named `aws-logging`
- Created in a dedicated namespace called `aws-observability`
- Can't exceed 5300 characters.

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

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

- An existing Fargate profile that specifies an existing Kubernetes namespace that you deploy Fargate Pods to. For more information, see [the section called "Step 3: Create a Fargate profile for your cluster"](#).
- An existing Fargate Pod execution role. For more information, see [the section called "Step 2: Create a Fargate Pod execution role"](#).

Log router configuration

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

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

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

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

```
    aws-observability: enabled
```

b. Create the namespace.

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

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

> **Important**
>
> The main sections included in a typical `Fluent Conf` are `Service`, `Input`, `Filter`, and `Output`. The Fargate log router however, only accepts:
>
> - The `Filter` and `Output` sections.
> - A `Parser` section.
>
> If you provide any other sections, they will be rejected.

The Fargate log router manages the `Service` and `Input` sections. It has the following `Input` section, which can't be modified and isn't needed in your `ConfigMap`. However, you can get insights from it, such as the memory buffer limit and the tag applied for logs.

```
[INPUT]
    Name tail
    Buffer_Max_Size 66KB
    DB /var/log/flb_kube.db
    Mem_Buf_Limit 45MB
    Path /var/log/containers/*.log
    Read_From_Head On
    Refresh_Interval 10
    Rotate_Wait 30
    Skip_Long_Lines On
    Tag kube.*
```

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

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

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

Important

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

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

Note

You can also use Amazon Kinesis Data Streams for your log destination. If you use Kinesis Data Streams, make sure that the pod execution role has been granted the kinesis:PutRecords permission. For more information, see Amazon Kinesis Data Streams [Permissions](#) in the *Fluent Bit: Official Manual*.

Example

CloudWatch

To create a ConfigMap for CloudWatch

You have two output options when using CloudWatch:

- An output plugin written in C
- An output plugin written in Golang

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

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

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
data:
  flb_log_cw: "false"  # Set to true to ship Fluent Bit process logs to
 CloudWatch.
  filters.conf: |
    [FILTER]
        Name parser
        Match *
        Key_name log
        Parser crio
    [FILTER]
        Name kubernetes
        Match kube.*
        Merge_Log On
        Keep_Log Off
        Buffer_Size 0
        Kube_Meta_Cache_TTL 300s
  output.conf: |
```

```
    [OUTPUT]
        Name cloudwatch_logs
        Match   kube.*
        region region-code
        log_group_name my-logs
        log_stream_prefix from-fluent-bit-
        log_retention_days 60
        auto_create_group true
  parsers.conf: |
    [PARSER]
        Name crio
        Format Regex
        Regex ^(?<time>[^ ]+) (?<stream>stdout|stderr) (?<logtag>P|F) (?
<log>.*)$
        Time_Key    time
        Time_Format %Y-%m-%dT%H:%M:%S.%L%z
```

b. Apply the manifest to your cluster.

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

c. Download the CloudWatch IAM policy to your computer. You can also view the policy on GitHub.

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

Amazon OpenSearch Service

To create a `ConfigMap` for Amazon OpenSearch Service

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

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

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
```

```
data:
  output.conf: |
    [OUTPUT]
      Name  es
      Match *
      Host  search-example-gjxdcilagiprbglqn42jsty66y.region-
code.es.amazonaws.com
      Port  443
      Index example
      Type  example_type
      AWS_Auth On
      AWS_Region region-code
      tls   On
```

b. Apply the manifest to your cluster.

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

c. Download the OpenSearch IAM policy to your computer. You can also view the policy on GitHub.

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

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

Firehose

To create a `ConfigMap` for Firehose

You have two output options when sending logs to Firehose:

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

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

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

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

b. Apply the manifest to your cluster.

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

c. Download the Firehose IAM policy to your computer. You can also view the policy on GitHub.

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

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

```
aws iam create-policy --policy-name eks-fargate-logging-policy --policy-document
 file://permissions.json
```

4. Attach the IAM policy to the pod execution role specified for your Fargate profile with the following command. Replace *111122223333* with your account ID. Replace *AmazonEKSFargatePodExecutionRole* with your Pod execution role (for more information, see the section called "Step 2: Create a Fargate Pod execution role").

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::111122223333:policy/eks-fargate-logging-policy \
```

```
    --role-name AmazonEKSFargatePodExecutionRole
```

Kubernetes filter support

This feature requires the following minimum Kubernetes version and platform level, or later.

Kubernetes version	Platform level
1.23 and later	eks.1

The Fluent Bit Kubernetes filter allows you to add Kubernetes metadata to your log files. For more information about the filter, see Kubernetes in the Fluent Bit documentation. You can apply a filter using the API server endpoint.

```
filters.conf: |
    [FILTER]
        Name             kubernetes
        Match            kube.*
        Merge_Log           On
        Buffer_Size         0
        Kube_Meta_Cache_TTL 300s
```

Important

- `Kube_URL`, `Kube_CA_File`, `Kube_Token_Command`, and `Kube_Token_File` are service owned configuration parameters and must not be specified. Amazon EKS Fargate populates these values.
- `Kube_Meta_Cache_TTL` is the time Fluent Bit waits until it communicates with the API server for the latest metadata. If `Kube_Meta_Cache_TTL` isn't specified, Amazon EKS Fargate appends a default value of 30 minutes to lessen the load on the API server.

To ship Fluent Bit process logs to your account

You can optionally ship Fluent Bit process logs to Amazon CloudWatch using the following `ConfigMap`. Shipping Fluent Bit process logs to CloudWatch requires additional log ingestion and storage costs. Replace *region-code* with the AWS Region that your cluster is in.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
  labels:
data:
  # Configuration files: server, input, filters and output
  # ======================================================
  flb_log_cw: "true"  # Ships Fluent Bit process logs to CloudWatch.

  output.conf: |
    [OUTPUT]
        Name cloudwatch
        Match kube.*
        region region-code
        log_group_name fluent-bit-cloudwatch
        log_stream_prefix from-fluent-bit-
        auto_create_group true
```

The logs are in the AWS Region that the cluster resides in under CloudWatch. The log group name is *my-cluster*`-fluent-bit-logs` and the Fluent Bit logstream name is `fluent-bit-`*podname*`-`*pod-namespace* .

> **Note**
>
> - The process logs are shipped only when the Fluent Bit process successfully starts. If there is a failure while starting Fluent Bit, the process logs are missed. You can only ship process logs to CloudWatch.
> - To debug shipping process logs to your account, you can apply the previous `ConfigMap` to get the process logs. Fluent Bit failing to start is usually due to your `ConfigMap` not being parsed or accepted by Fluent Bit while starting.

To stop shipping Fluent Bit process logs

Shipping Fluent Bit process logs to CloudWatch requires additional log ingestion and storage costs. To exclude process logs in an existing `ConfigMap` setup, do the following steps.

1. Locate the CloudWatch log group automatically created for your Amazon EKS cluster's Fluent Bit process logs after enabling Fargate logging. It follows the format `{cluster_name}-fluent-bit-logs`.
2. Delete the existing CloudWatch log streams created for each Pod's process logs in the CloudWatch log group.
3. Edit the `ConfigMap` and set `flb_log_cw: "false"`.
4. Restart any existing Pods in the cluster.

Test application

1. Deploy a sample Pod.
   a. Save the following contents to a file named `sample-app.yaml` on your computer.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
  namespace: same-namespace-as-your-fargate-profile
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - name: http
              containerPort: 80
```

b. Apply the manifest to the cluster.

```
kubectl apply -f sample-app.yaml
```

2. View the NGINX logs using the destination(s) that you configured in the `ConfigMap`.

Size considerations

We suggest that you plan for up to 50 MB of memory for the log router. If you expect your application to generate logs at very high throughput then you should plan for up to 100 MB.

Troubleshooting

To confirm whether the logging feature is enabled or disabled for some reason, such as an invalid `ConfigMap`, and why it's invalid, check your Pod events with `kubectl describe pod` *pod-name* . The output might include Pod events that clarify whether logging is enabled or not, such as the following example output.

```
[...]
Annotations:           CapacityProvisioned: 0.25vCPU 0.5GB
                       Logging: LoggingDisabled: LOGGING_CONFIGMAP_NOT_FOUND
                       kubernetes.io/psp: eks.privileged
[...]
Events:
  Type     Reason           Age        From
            Message
  ----     ------           ----       ----
            -------
  Warning  LoggingDisabled  <unknown>  fargate-scheduler
             Disabled logging because aws-logging configmap was not found. configmap
 "aws-logging" not found
```

The Pod events are ephemeral with a time period depending on the settings. You can also view a Pod's annotations using `kubectl describe pod` *pod-name* . In the Pod annotation, there is information about whether the logging feature is enabled or disabled and the reason.

Choose an optimal Amazon EC2 node instance type

Amazon EC2 provides a wide selection of instance types for worker nodes. Each instance type offers different compute, memory, storage, and network capabilities. Each instance is also grouped in an instance family based on these capabilities. For a list, see Available instance types in the *Amazon EC2 User Guide*. Amazon EKS releases several variations of Amazon EC2 AMIs to enable support. To make sure that the instance type you select is compatible with Amazon EKS, consider the following criteria.

- All Amazon EKS AMIs don't currently support the `g5g` and `mac` families.
- Arm and non-accelerated Amazon EKS AMIs don't support the `g3`, `g4`, `inf`, and `p` families.
- Accelerated Amazon EKS AMIs don't support the `a`, `c`, `hpc`, `m`, and `t` families.
- For Arm-based instances, Amazon Linux 2023 (AL2023) only supports instance types that use Graviton2 or later processors. AL2023 doesn't support `A1` instances.

When choosing between instance types that are supported by Amazon EKS, consider the following capabilities of each type.

Number of instances in a node group

In general, fewer, larger instances are better, especially if you have a lot of Daemonsets. Each instance requires API calls to the API server, so the more instances you have, the more load on the API server.

Operating system

Review the supported instance types for Linux, Windows, and Bottlerocket. Before creating Windows instances, review Deploy Windows nodes on EKS clusters.

Hardware architecture

Do you need x86 or Arm? Before deploying Arm instances, review Amazon EKS optimized Arm Amazon Linux AMIs. Do you need instances built on the Nitro System (Linux or Windows) or that have Accelerated capabilities? If you need accelerated capabilities, you can only use Linux with Amazon EKS.

Maximum number of Pods

Since each Pod is assigned its own IP address, the number of IP addresses supported by an instance type is a factor in determining the number of Pods that can run on the instance. To manually determine how many Pods an instance type supports, see the section called “Amazon EKS recommended maximum Pods for each Amazon EC2 instance type”.

> **Note**
>
> If you're using an Amazon EKS optimized Amazon Linux 2 AMI that's `v20220406` or newer, you can use a new instance type without upgrading to the latest AMI. For these AMIs, the AMI auto-calculates the necessary `max-pods` value if it isn't listed in the eni-max-pods.txt file. Instance types that are currently in preview may not be supported by Amazon EKS by default. Values for `max-pods` for such types still need to be added to `eni-max-pods.txt` in our AMI.

AWS Nitro System instance types optionally support significantly more IP addresses than non-Nitro System instance types. However, not all IP addresses assigned for an instance are available to Pods. To assign a significantly larger number of IP addresses to your instances, you must have version `1.9.0` or later of the Amazon VPC CNI add-on installed in your cluster and configured appropriately. For more information, see the section called "Increase IP addresses". To assign the largest number of IP addresses to your instances, you must have version `1.10.1` or later of the Amazon VPC CNI add-on installed in your cluster and deploy the cluster with the `IPv6` family.

IP family

You can use any supported instance type when using the `IPv4` family for a cluster, which allows your cluster to assign private `IPv4` addresses to your Pods and Services. But if you want to use the `IPv6` family for your cluster, then you must use AWS Nitro System instance types or bare metal instance types. Only `IPv4` is supported for Windows instances. Your cluster must be running version `1.10.1` or later of the Amazon VPC CNI add-on. For more information about using `IPv6`, see the section called "IPv6".

Version of the Amazon VPC CNI add-on that you're running

The latest version of the Amazon VPC CNI plugin for Kubernetes supports these instance types. You may need to update your Amazon VPC CNI add-on version to take advantage of the latest supported instance types. For more information, see the section called "Amazon VPC CNI". The latest version supports the latest features for use with Amazon EKS. Earlier versions don't support all features. You can view features supported by different versions in the Changelog on GitHub.

AWS Region that you're creating your nodes in

Not all instance types are available in all AWS Regions.

Whether you're using security groups for Pods

If you're using security groups for Pods, only specific instance types are supported. For more information, see the section called "Security groups for Pods".

Amazon EKS recommended maximum Pods for each Amazon EC2 instance type

Since each Pod is assigned its own IP address, the number of IP addresses supported by an instance type is a factor in determining the number of Pods that can run on the instance. Amazon EKS provides a script that you can download and run to determine the Amazon EKS recommended maximum number of Pods to run on each instance type. The script uses hardware attributes of each instance, and configuration options, to determine the maximum Pods number. You can use the number returned in these steps to enable capabilities such as assigning IP addresses to Pods from a different subnet than the instance's and significantly increasing the number of IP addresses for your instance. If you're using a managed node group with multiple instance types, use a value that would work for all instance types.

1. Download a script that you can use to calculate the maximum number of Pods for each instance type.

```
curl -O https://raw.githubusercontent.com/awslabs/amazon-eks-ami/master/templates/
al2/runtime/max-pods-calculator.sh
```

2. Mark the script as executable on your computer.

```
chmod +x max-pods-calculator.sh
```

3. Run the script, replacing *m5.large* with the instance type that you plan to deploy and *1.9.0-eksbuild.1* with your Amazon VPC CNI add-on version. To determine your add-on version, see the update procedures in Assign IPs to Pods with the Amazon VPC CNI.

```
./max-pods-calculator.sh --instance-type m5.large --cni-version 1.9.0-eksbuild.1
```

An example output is as follows.

```
29
```

You can add the following options to the script to see the maximum Pods supported when using optional capabilities.

- `--cni-custom-networking-enabled` – Use this option when you want to assign IP addresses from a different subnet than your instance's. For more information, see the section called "Custom networking". Adding this option to the previous script with the same example values yields `20`.
- `--cni-prefix-delegation-enabled` – Use this option when you want to assign significantly more IP addresses to each elastic network interface. This capability requires an Amazon Linux instance that run on the Nitro System and version `1.9.0` or later of the Amazon VPC CNI add-on. For more information, see the section called "Increase IP addresses". Adding this option to the previous script with the same example values yields `110`.

You can also run the script with the `--help` option to see all available options.

> **Note**
>
> The max Pods calculator script limits the return value to `110` based on Kubernetes scalability thresholds and recommended settings. If your instance type has greater than 30 vCPUs, this limit jumps to `250`, a number based on internal Amazon EKS scalability team testing. For more information, see the Amazon VPC CNI plugin increases pods per node limits blog post.

Considerations for EKS Auto Mode

EKS Auto Mode limits the number of pods on nodes to the lower of:

- 110 pods hard cap
- The result of the max pods calculation described above.

Create nodes with pre-built optimized images

You can deploy nodes with pre-built Amazon EKS optimized Amazon Machine Images (AMIs) or your own custom AMIs when you use managed node groups or self-managed nodes. If you are running hybrid nodes, see the section called "Prepare operating system". For information about each type of Amazon EKS optimized AMI, see one of the following topics. For instructions on how to create your own custom AMI, see the section called "Custom builds".

With Amazon EKS Auto Mode, EKS manages the EC2 instance including selecting and updating the AMI.

Topics

Migrate from `dockershim` to `containerd`

Kubernetes no longer supports `dockershim`. The Kubernetes team removed the runtime in Kubernetes version `1.24`. For more information, see Kubernetes is Moving on From Dockershim: Commitments and Next Steps on the Kubernetes Blog.

Amazon EKS also ended support for `dockershim` starting with the Kubernetes version `1.24` release. Amazon EKS AMIs that are officially published have `containerd` as the only runtime starting with version `1.24`. This topic covers some details, but more information is available in All you need to know about moving to containerd on Amazon EKS.

There's a `kubectl` plugin that you can use to see which of your Kubernetes workloads mount the Docker socket volume. For more information, see Detector for Docker Socket (DDS) on GitHub. Amazon EKS AMIs that run Kubernetes versions that are earlier than `1.24` use Docker as the default runtime. However, these Amazon EKS AMIs have a bootstrap flag option that you can use to test out your workloads on any supported cluster using `containerd`. For more information, see the section called "Test Amazon Linux 2 migration from Docker to `containerd`".

We will continue to publish AMIs for existing Kubernetes versions until the end of their support date. For more information, see the section called "Amazon EKS Kubernetes release calendar". If you require more time to test your workloads on `containerd`, use a supported version before `1.24`. But, when you want to upgrade official Amazon EKS AMIs to version `1.24` or later, make sure to validate that your workloads run on `containerd`.

The `containerd` runtime provides more reliable performance and security. `containerd` is the runtime that's being standardized on across Amazon EKS. Fargate and Bottlerocket already use `containerd` only. `containerd` helps to minimize the number of Amazon EKS AMI releases that are required to address `dockershim` Common Vulnerabilities and Exposures (CVEs). Because `dockershim` already uses `containerd` internally, you might not need to make any changes. However, there are some situations where changes might or must be required:

- You must make changes to applications that mount the Docker socket. For example, container images that are built with a container are impacted. Many monitoring tools also mount the Docker socket. You might need to wait for updates or re-deploy workloads for runtime monitoring.
- You might need to make changes for applications that are reliant on specific Docker settings. For example, the `HTTPS_PROXY` protocol is no longer supported. You must update applications that use this protocol. For more information, see dockerd in the Docker Documentation.
- If you use the Amazon ECR credential helper to pull images, you must switch to the `kubelet` image credential provider. For more information, see Configure a kubelet image credential provider in the Kubernetes documentation.
- Because Amazon EKS `1.24` no longer supports Docker, some flags that the Amazon EKS bootstrap script previously supported are no longer supported. Before moving to Amazon EKS `1.24` or later, you must remove any reference to flags that are now unsupported:
  - `--container-runtime dockerd` (`containerd` is the only supported value)
  - `--enable-docker-bridge`
  - `--docker-config-json`
- If you already have Fluentd configured for Container Insights, then you must migrate Fluentd to Fluent Bit before changing to `containerd`. The Fluentd parsers are configured to only parse log messages in JSON format. Unlike `dockerd`, the `containerd` container runtime has log messages that aren't in JSON format. If you don't migrate to Fluent Bit, some of the configured Fluentd's parsers will generate a massive amount of errors inside the Fluentd container. For more information on migrating, see Set up Fluent Bit as a DaemonSet to send logs to CloudWatch Logs.
- If you use a custom AMI and you are upgrading to Amazon EKS `1.24`, then you must make sure that IP forwarding is enabled for your worker nodes. This setting wasn't needed with Docker but is required for `containerd`. It is needed to troubleshoot Pod-to-Pod, Pod-to-external, or Pod-to-apiserver network connectivity.

To verify this setting on a worker node, run either of the following commands:

- `sysctl net.ipv4.ip_forward`
- `cat /proc/sys/net/ipv4/ip_forward`

If the output is `0`, then run either of the following commands to activate the `net.ipv4.ip_forward` kernel variable:

- `sysctl -w net.ipv4.ip_forward=1`
- `echo 1 > /proc/sys/net/ipv4/ip_forward`

For the setting's activation on Amazon EKS AMIs for Amazon Linux 2 in the `containerd` runtime, see `install-worker.sh` on GitHub.

Test Amazon Linux 2 migration from Docker to `containerd`

For Kubernetes version `1.23`, you can use an optional bootstrap flag to enable the `containerd` runtime for Amazon EKS optimized AL2 AMIs. This feature gives you a clear path to migrate to `containerd` when updating to version `1.24` or later. Amazon EKS ended support for Docker starting with the Kubernetes version `1.24` launch. The `containerd` runtime is widely adopted in the Kubernetes community and is a graduated project with the CNCF. You can test it by adding a node group to a new or existing cluster.

You can enable the boostrap flag by creating one of the following types of node groups.

Self-managed

Create the node group using the instructions in Create self-managed Amazon Linux nodes. Specify an Amazon EKS optimized AMI and the following text for the `BootstrapArguments` parameter.

```
--container-runtime containerd
```

Managed

If you use `eksctl`, create a file named `my-nodegroup.yaml` with the following contents. Replace every *`example value`* with your own values. The node group name can't be longer than 63 characters. It must start with letter or digit, but can also include hyphens and underscores for the remaining characters. To retrieve an optimized AMI ID for `ami-`*`1234567890abcdef0`*, see the section called "Get latest IDs".

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-cluster
  region: region-code
  version: 1.23
managedNodeGroups:
  - name: my-nodegroup
    ami: ami-1234567890abcdef0
    overrideBootstrapCommand: |
      #!/bin/bash
      /etc/eks/bootstrap.sh my-cluster --container-runtime containerd
```

> **Note**
>
> If you launch many nodes simultaneously, you may also want to specify values for the `--apiserver-endpoint`, `--b64-cluster-ca`, and `--dns-cluster-ip` bootstrap arguments to avoid errors. For more information, see the section called "Specifying an AMI".

Run the following command to create the node group.

```
eksctl create nodegroup -f my-nodegroup.yaml
```

If you prefer to use a different tool to create your managed node group, you must deploy the node group using a launch template. In your launch template, specify an Amazon EKS optimized AMI ID, then deploy the node group using a launch template and provide the following user data. This user data passes arguments into the `bootstrap.sh` file. For more information about the bootstrap file, see bootstrap.sh on GitHub.

```
/etc/eks/bootstrap.sh my-cluster --container-runtime containerd
```

Create nodes with optimized Amazon Linux AMIs

The Amazon EKS optimized Amazon Linux AMIs are built on top of Amazon Linux 2 (AL2) and Amazon Linux 2023 (AL2023). They are configured to serve as the base images for Amazon

EKS nodes. The AMIs are configured to work with Amazon EKS and they include the following components:

- `kubelet`
- AWS IAM Authenticator
- Docker (Amazon EKS version `1.23` and earlier)
- `containerd`

> **Note**
>
> - You can track security or privacy events for Amazon Linux at the Amazon Linux security center by choosing the tab for your desired version. You can also subscribe to the applicable RSS feed. Security and privacy events include an overview of the issue, what packages are affected, and how to update your instances to correct the issue.
> - Before deploying an accelerated or Arm AMI, review the information in Amazon EKS optimized accelerated Amazon Linux AMIs and the section called "Amazon EKS optimized Arm Amazon Linux AMIs".
> - For Kubernetes version `1.23`, you can use an optional bootstrap flag to test migration from Docker to `containerd`. For more information, see the section called "Test Amazon Linux 2 migration from Docker to `containerd`".
> - Amazon EC2 P2 instances aren't supported on Amazon EKS because they require `NVIDIA` driver version 470 or earlier.
> - Any newly created managed node groups in clusters on version `1.30` or newer will automatically default to using AL2023 as the node operating system. Previously, new node groups would default to AL2. You can continue to use AL2 by choosing it as the AMI type when creating a new node group.
> - Support for AL2 will end on June 30th, 2025. For more information, see Amazon Linux 2 FAQs.

Amazon EKS optimized accelerated Amazon Linux AMIs

The Amazon EKS optimized accelerated Amazon Linux AMIs are built on top of the standard Amazon EKS optimized Amazon Linux AMIs. They are configured to serve as optional images for Amazon EKS nodes to support GPU, Inferentia, and Trainium based workloads.

In addition to the standard Amazon EKS optimized AMI configuration, the accelerated AMIs include the following:

- NVIDIA drivers
- `nvidia-container-toolkit`
- AWS Neuron driver

For a list of the latest components included in the accelerated AMIs, see the `amazon-eks-ami` Releases on GitHub.

Note

- Make sure to specify the applicable instance type in your node AWS CloudFormation template. By using the Amazon EKS optimized accelerated AMIs, you agree to NVIDIA's Cloud End User License Agreement (EULA).
- The Amazon EKS optimized accelerated AMIs were previously referred to as the *Amazon EKS optimized AMIs with GPU support*.
- Previous versions of the Amazon EKS optimized accelerated AMIs installed the `nvidia-docker` repository. The repository is no longer included in Amazon EKS AMI version `v20200529` and later.

For details on running workloads on Amazon EKS optimized accelerated Amazon Linux AMIs, see the section called "Run Linux GPU AMIs".

Amazon EKS optimized Arm Amazon Linux AMIs

Arm instances deliver significant cost savings for scale-out and Arm-based applications such as web servers, containerized microservices, caching fleets, and distributed data stores. When adding Arm nodes to your cluster, review the following considerations.

- If your cluster was deployed before August 17, 2020, you must do a one-time upgrade of critical cluster add-on manifests. This is so that Kubernetes can pull the correct image for each hardware architecture in use in your cluster. For more information about updating cluster add-ons, see the section called "Step 1: Prepare for upgrade". If you deployed your cluster on or after August 17, 2020, then your CoreDNS, `kube-proxy`, and Amazon VPC CNI plugin for Kubernetes add-ons are already multi-architecture capable.

- Applications deployed to Arm nodes must be compiled for Arm.
- If you have DaemonSets that are deployed in an existing cluster, or you want to deploy them to a new cluster that you also want to deploy Arm nodes in, then verify that your DaemonSet can run on all hardware architectures in your cluster.
- You can run Arm node groups and x86 node groups in the same cluster. If you do, consider deploying multi-architecture container images to a container repository such as Amazon Elastic Container Registry and then adding node selectors to your manifests so that Kubernetes knows what hardware architecture a Pod can be deployed to. For more information, see Pushing a multi-architecture image in the *Amazon ECR User Guide* and the Introducing multi-architecture container images for Amazon ECR blog post.

More information

For more information about using Amazon EKS optimized Amazon Linux AMIs, see the following sections:

- To use Amazon Linux with managed node groups, see the section called "Managed node groups".
- To launch self-managed Amazon Linux nodes, see the section called "Get latest IDs".
- For version information, see the section called "Get version information".
- To retrieve the latest IDs of the Amazon EKS optimized Amazon Linux AMIs, see the section called "Get latest IDs".
- For open-source scripts that are used to build the Amazon EKS optimized AMIs, see the section called "Custom builds".

Upgrade from Amazon Linux 2 to Amazon Linux 2023

The Amazon EKS optimized AMIs are available in two families based on AL2 and AL2023. AL2023 is a new Linux-based operating system designed to provide a secure, stable, and high-performance environment for your cloud applications. It's the next generation of Amazon Linux from Amazon Web Services and is available across all supported Amazon EKS versions, including versions `1.23` and `1.24` in extended support.

AL2023 offers several improvements over AL2. For a full comparison, see Comparing AL2 and Amazon Linux 2023 in the *Amazon Linux 2023 User Guide*. Several packages have been added, upgraded, and removed from AL2. It's highly recommended to test your applications with AL2023 before upgrading. For a list of all package changes in AL2023, see Package changes in Amazon Linux 2023 in the *Amazon Linux 2023 Release Notes*.

In addition to these changes, you should be aware of the following:

- AL2023 introduces a new node initialization process `nodeadm` that uses a YAML configuration schema. If you're using self-managed node groups or an AMI with a launch template, you'll now need to provide additional cluster metadata explicitly when creating a new node group. An example of the minimum required parameters is as follows, where `apiServerEndpoint`, `certificateAuthority`, and service `cidr` are now required:

```
---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    cidr: 10.100.0.0/16
```

  In AL2, the metadata from these parameters was discovered from the Amazon EKS `DescribeCluster` API call. With AL2023, this behavior has changed since the additional API call risks throttling during large node scale ups. This change doesn't affect you if you're using managed node groups without a launch template or if you're using Karpenter. For more information on `certificateAuthority` and service `cidr`, see `DescribeCluster` in the *Amazon EKS API Reference*.

- For AL2023, `nodeadm` also changes the format to apply parameters to the `kubelet` for each node using `NodeConfigSpec`. In AL2, this was done with the `--kubelet-extra-args` parameter. This is commonly used to add labels and taints to nodes. An example below shows applying `maxPods` and `--node-labels` to the node.

```
---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: test-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
```

```
    cidr: 10.100.0.0/16
  kubelet:
    config:
      maxPods: 110
    flags:
      - --node-labels=karpenter.sh/capacity-type=on-demand,karpenter.sh/nodepool=test
```

- Docker isn't supported in AL2023 for all supported Amazon EKS versions. Support for Docker has ended and been removed with Amazon EKS version `1.24` or greater in AL2. For more information on deprecation, see the section called "Dockershim deprecation".
- Amazon VPC CNI version `1.16.2` or greater is required for AL2023.
- AL2023 requires `IMDSv2` by default. `IMDSv2` has several benefits that help improve security posture. It uses a session-oriented authentication method that requires the creation of a secret token in a simple HTTP PUT request to start the session. A session's token can be valid for anywhere between 1 second and 6 hours. For more information on how to transition from `IMDSv1` to `IMDSv2`, see Transition to using Instance Metadata Service Version 2 and Get the full benefits of IMDSv2 and disable IMDSv1 across your AWS infrastructure. If you would like to use `IMDSv1`, you can still do so by manually overriding the settings using instance metadata option launch properties.

> **Note**
>
> For `IMDSv2`, the default hop count for managed node groups is set to 1. This means that containers won't have access to the node's credentials using IMDS. If you require container access to the node's credentials, you can still do so by manually overriding the `HttpPutResponseHopLimit` in a custom Amazon EC2 launch template, increasing it to 2.Alternatively, you can use Amazon EKS Pod Identity to provide credentials instead of `IMDSv2`.

- AL2023 features the next generation of unified control group hierarchy (`cgroupv2`). `cgroupv2` is used to implement a container runtime, and by `systemd`. While AL2023 still includes code that can make the system run using `cgroupv1`, this isn't a recommended or supported configuration. This configuration will be completely removed in a future major release of Amazon Linux.
- `eksctl` version `0.176.0` or greater is required for `eksctl` to support AL2023.

For previously existing managed node groups, you can either perform an in-place upgrade or a blue/green upgrade depending on how you're using a launch template:

- If you're using a custom AMI with a managed node group, you can perform an in-place upgrade by swapping the AMI ID in the launch template. You should ensure that your applications and any user data transfer over to AL2023 first before performing this upgrade strategy.
- If you're using managed node groups with either the standard launch template or with a custom launch template that doesn't specify the AMI ID, you're required to upgrade using a blue/green strategy. A blue/green upgrade is typically more complex and involves creating an entirely new node group where you would specify AL2023 as the AMI type. The new node group will need to then be carefully configured to ensure that all custom data from the AL2 node group is compatible with the new OS. Once the new node group has been tested and validated with your applications, Pods can be migrated from the old node group to the new node group. Once the migration is completed, you can delete the old node group.

If you're using Karpenter and want to use AL2023, you'll need to modify the `EC2NodeClass` `amiFamily` field with AL2023. By default, Drift is enabled in Karpenter. This means that once the `amiFamily` field has been changed, Karpenter will automatically update your worker nodes to the latest AMI when available.

Retrieve Amazon Linux AMI version information

Amazon EKS optimized Amazon Linux AMIs are versioned by Kubernetes version and the release date of the AMI in the following format:

```
k8s_major_version.k8s_minor_version.k8s_patch_version-release_date
```

Each AMI release includes various versions of kubelet, the Linux kernel, and containerd. The accelerated AMIs also include various versions of the NVIDIA driver. You can find this version information in the Changelog on GitHub.

Retrieve recommended Amazon Linux AMI IDs

When deploying nodes, you can specify an ID for a pre-built Amazon EKS optimized Amazon Machine Image (AMI). To retrieve an AMI ID that fits your desired configuration, query the AWS Systems Manager Parameter Store API. Using this API eliminates the need to manually look up Amazon EKS optimized AMI IDs. For more information, see GetParameter. The IAM principal that you use must have the `ssm:GetParameter` IAM permission to retrieve the Amazon EKS optimized AMI metadata.

You can retrieve the image ID of the latest recommended Amazon EKS optimized Amazon Linux AMI with the following command, which uses the sub-parameter `image_id`. Make the following modifications to the command as needed and then run the modified command:

- Replace `<kubernetes-version>` with a supported Amazon EKS version.
- Replace *`ami-type`* with one of the following options. For information about the types of Amazon EC2 instances, see Amazon EC2 instance types.
  - Use *`amazon-linux-2023/x86_64/standard`* for Amazon Linux 2023 (AL2023) x86 based instances.
  - Use *`amazon-linux-2023/arm64/standard`* for AL2023 ARM instances.
  - Use *`amazon-linux-2023/x86_64/nvidia`* for the latest approved AL2023 NVIDIA instances.
  - Use *`amazon-linux-2023/x86_64/neuron`* for the latest AL2023 AWS Neuron instances.
  - Use *`amazon-linux-2`* for Amazon Linux 2 (AL2) x86 based instances.
  - Use *`amazon-linux-2-arm64`* for AL2 ARM instances, such as AWS Graviton based instances.
  - Use *`amazon-linux-2-gpu`* for AL2 hardware accelerated x86 based instances for NVIDIA GPU, Inferentia, and Trainium based workloads.
- Replace `<region-code>` with an Amazon EKS supported AWS Region for which you want the AMI ID.

```
aws ssm get-parameter --name /aws/service/eks/optimized-ami/<kubernetes-version>/<ami-type>/recommended/image_id \
    --region <region-code> --query "Parameter.Value" --output text
```

Here's an example command after placeholder replacements have been made.

```
aws ssm get-parameter --name /aws/service/eks/optimized-ami/1.31/amazon-linux-2023/x86_64/standard/recommended/image_id \
    --region us-west-2 --query "Parameter.Value" --output text
```

An example output is as follows.

```
ami-1234567890abcdef0
```

Build a custom Amazon Linux AMI with a script

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

The Amazon EKS optimized Amazon Linux AMIs are built on top of Amazon Linux 2 (AL2) and Amazon Linux 2023 (AL2023), specifically for use as a node in Amazon EKS clusters. You can use this repository to view the specifics of how the Amazon EKS team configures `kubelet`, the runtime, the AWS IAM Authenticator for Kubernetes, and build your own Amazon Linux based AMI from scratch.

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

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

Additionally, the GitHub repository contains our Amazon EKS node AWS CloudFormation templates. These templates make it easier to spin up an instance running an 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 AL2 contains an optional bootstrap flag to enable the `containerd` runtime.

Create nodes with optimized Bottlerocket AMIs

Bottlerocket is an open source Linux distribution that's sponsored and supported by AWS. Bottlerocket is purpose-built for hosting container workloads. With Bottlerocket, you can improve the availability of containerized deployments and reduce operational costs by automating updates to your container infrastructure. Bottlerocket includes only the essential software to run containers, which improves resource usage, reduces security threats, and lowers management overhead. The Bottlerocket AMI includes `containerd`, `kubelet`, and AWS IAM Authenticator.

In addition to managed node groups and self-managed nodes, Bottlerocket is also supported by Karpenter.

Advantages

Using Bottlerocket with your Amazon EKS cluster has the following advantages:

- **Higher uptime with lower operational cost and lower management complexity –** Bottlerocket has a smaller resource footprint, shorter boot times, and is less vulnerable to security threats than other Linux distributions. Bottlerocket's smaller footprint helps to reduce costs by using less storage, compute, and networking resources.
- **Improved security from automatic OS updates –** Updates to Bottlerocket are applied as a single unit which can be rolled back, if necessary. This removes the risk of corrupted or failed updates that can leave the system in an unusable state. With Bottlerocket, security updates can be automatically applied as soon as they're available in a minimally disruptive manner and be rolled back if failures occur.
- **Premium support –** AWS provided builds of Bottlerocket on Amazon EC2 is covered under the same AWS Support plans that also cover AWS services such as Amazon EC2, Amazon EKS, and Amazon ECR.

Considerations

Consider the following when using Bottlerocket for your AMI type:

- Bottlerocket supports Amazon EC2 instances with `x86_64` and `arm64` processors. The Bottlerocket AMI isn't recommended for use with Amazon EC2 instances with an Inferentia chip.
- Bottlerocket images don't include an SSH server or a shell. You can employ out-of-band access methods to allow SSH. These approaches enable the admin container and to pass some bootstrapping configuration steps with user data. For more information, refer to the following sections in Bottlerocket OS on GitHub:
  - Exploration
  - Admin container
  - Kubernetes settings
- Bottlerocket uses different container types:
  - By default, a control container is enabled. This container runs the AWS Systems Manager agent that you can use to run commands or start shell sessions on Amazon EC2 Bottlerocket

instances. For more information, see Setting up Session Manager in the *AWS Systems Manager User Guide*.

- If an SSH key is given when creating the node group, an admin container is enabled. We recommend using the admin container only for development and testing scenarios. We don't recommend using it for production environments. For more information, see Admin container on GitHub.

More information

For more information about using Amazon EKS optimized Bottlerocket AMIs, see the following sections:

- For details about Bottlerocket, see the Bottlerocket Documentation.
- For version information resources, see the section called "Get version information".
- To use Bottlerocket with managed node groups, see the section called "Managed node groups".
- To launch self-managed Bottlerocket nodes, see the section called "Bottlerocket".
- To retrieve the latest IDs of the Amazon EKS optimized Bottlerocket AMIs, see the section called "Get latest IDs".
- For details on compliance support, see the section called "Compliance support".

Retrieve Bottlerocket AMI version information

Each Bottlerocket AMI release includes various versions of kubelet, the Bottlerocket kernel, and containerd. Accelerated AMI variants also include various versions of the NVIDIA driver. You can find this version information in the OS topic of the *Bottlerocket Documentation*. From this page, navigate to the applicable *Version Information* sub-topic.

The *Bottlerocket Documentation* can sometimes lag behind the versions that are available on GitHub. You can find a list of changes for the latest versions in the releases on GitHub.

Retrieve recommended Bottlerocket AMI IDs

When deploying nodes, you can specify an ID for a pre-built Amazon EKS optimized Amazon Machine Image (AMI). To retrieve an AMI ID that fits your desired configuration, query the AWS Systems Manager Parameter Store API. Using this API eliminates the need to manually look up Amazon EKS optimized AMI IDs. For more information, see GetParameter. The IAM principal that you use must have the `ssm:GetParameter` IAM permission to retrieve the Amazon EKS optimized AMI metadata.

You can retrieve the image ID of the latest recommended Amazon EKS optimized Bottlerocket AMI with the following AWS CLI command, which uses the sub-parameter `image_id`. Make the following modifications to the command as needed and then run the modified command:

- Replace *kubernetes-version* with a supported Amazon EKS version.
- Replace *-flavor* with one of the following options.
  - Remove *-flavor* for variants without a GPU.
  - Use *-nvidia* for GPU-enabled variants.
  - Use *-fips* for FIPS-enabled variants.
- Replace *architecture* with one of the following options.
  - Use *x86_64* for `x86` based instances.
  - Use *arm64* for ARM instances.
- Replace *region-code* with an Amazon EKS supported AWS Region for which you want the AMI ID.

```
aws ssm get-parameter --name /aws/service/bottlerocket/aws-k8s-kubernetes-version-flavor/architecture/latest/image_id \
    --region region-code --query "Parameter.Value" --output text
```

Here's an example command after placeholder replacements have been made.

```
aws ssm get-parameter --name /aws/service/bottlerocket/aws-k8s-1.31/x86_64/latest/image_id \
    --region us-west-2 --query "Parameter.Value" --output text
```

An example output is as follows.

```
ami-1234567890abcdef0
```

Meet compliance requirements with Bottlerocket

Bottlerocket complies with recommendations defined by various organizations:

- There is a CIS Benchmark defined for Bottlerocket. In a default configuration, Bottlerocket image has most of the controls required by CIS Level 1 configuration profile. You can implement the controls required for a CIS Level 2 configuration profile. For more information, see Validating Amazon EKS optimized Bottlerocket AMI against the CIS Benchmark on the AWS blog.
- The optimized feature set and reduced attack surface means that Bottlerocket instances require less configuration to satisfy PCI DSS requirements. The CIS Benchmark for Bottlerocket is an excellent resource for hardening guidance, and supports your requirements for secure configuration standards under PCI DSS requirement 2.2. You can also leverage Fluent Bit to support your requirements for operating system level audit logging under PCI DSS requirement 10.2. AWS publishes new (patched) Bottlerocket instances periodically to help you meet PCI DSS requirement 6.2 (for v3.2.1) and requirement 6.3.3 (for v4.0).
- Bottlerocket is an HIPAA-eligible feature authorized for use with regulated workloads for both Amazon EC2 and Amazon EKS. For more information, see the Architecting for HIPAA Security and Compliance on Amazon EKS whitepaper.
- Bottlerocket AMIs are available that are preconfigured to use FIPS 140-3 validated cryptographic modules. This includes the Amazon Linux 2023 Kernel Crypto API Cryptographic Module and the AWS-LC Cryptographic Module. For more information on selecting FIPS-enabled variants, see the section called “Get latest IDs”.

Create nodes with optimized Ubuntu Linux AMIs

Canonical has partnered with Amazon EKS to create node AMIs that you can use in your clusters.

Canonical delivers a built-for-purpose Kubernetes Node OS image. This minimized Ubuntu image is optimized for Amazon EKS and includes the custom AWS kernel that is jointly developed with AWS. For more information, see Ubuntu on Amazon Elastic Kubernetes Service (EKS) and the section called “Ubuntu Linux” . For information about support, see the Third-party software section of the *AWS Premium Support FAQs*.

Create nodes with optimized Windows AMIs

Windows Amazon EKS optimized AMIs are built on top of Windows Server 2019 and Windows Server 2022. They are configured to serve as the base image for Amazon EKS nodes. By default, the AMIs include the following components:

- kubelet
- kube-proxy

- AWS IAM Authenticator for Kubernetes
- csi-proxy
- containerd

Note

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

Amazon EKS offers AMIs that are optimized for Windows containers in the following variants:

- Amazon EKS-optimized Windows Server 2019 Core AMI
- Amazon EKS-optimized Windows Server 2019 Full AMI
- Amazon EKS-optimized Windows Server 2022 Core AMI
- Amazon EKS-optimized Windows Server 2022 Full AMI

Important

- The Amazon EKS-optimized Windows Server 20H2 Core AMI is deprecated. No new versions of this AMI will be released.
- To ensure that you have the latest security updates by default, Amazon EKS maintains optimized Windows AMIs for the last 4 months. Each new AMI will be available for 4 months from the time of initial release. After this period, older AMIs are made private and are no longer accessible. We encourage using the latest AMIs to avoid security vulnerabilities and losing access to older AMIs which have reached the end of their supported lifetime. While we can't guarantee that we can provide access to AMIs that have been made private, you can request access by filing a ticket with AWS Support.

Release calendar

The following table lists the release and end of support dates for Windows versions on Amazon EKS. If an end date is blank, it's because the version is still supported.

Windows version	Amazon EKS release	Amazon EKS end of support
Windows Server 2022 Core	10/17/2022	
Windows Server 2022 Full	10/17/2022	
Windows Server 20H2 Core	8/12/2021	8/9/2022
Windows Server 2004 Core	8/19/2020	12/14/2021
Windows Server 2019 Core	10/7/2019	
Windows Server 2019 Full	10/7/2019	
Windows Server 1909 Core	10/7/2019	12/8/2020

Bootstrap script configuration parameters

When you create a Windows node, there's a script on the node that allows for configuring different parameters. Depending on your setup, this script can be found on the node at a location similar to: `C:\Program Files\Amazon\EKS\Start-EKSBootstrap.ps1`. You can specify custom parameter values by specifying them as arguments to the bootstrap script. For example, you can update the user data in the launch template. For more information, see the section called "Amazon EC2 user data".

The script includes the following command-line parameters:

- `-EKSClusterName` – Specifies the Amazon EKS cluster name for this worker node to join.
- `-KubeletExtraArgs` – Specifies extra arguments for `kubelet` (optional).
- `-KubeProxyExtraArgs` – Specifies extra arguments for `kube-proxy` (optional).
- `-APIServerEndpoint` – Specifies the Amazon EKS cluster API server endpoint (optional). Only valid when used with `-Base64ClusterCA`. Bypasses calling `Get-EKSCluster`.
- `-Base64ClusterCA` – Specifies the base64 encoded cluster CA content (optional). Only valid when used with `-APIServerEndpoint`. Bypasses calling `Get-EKSCluster`.
- `-DNSClusterIP` – Overrides the IP address to use for DNS queries within the cluster (optional). Defaults to `10.100.0.10` or `172.20.0.10` based on the IP address of the primary interface.

- `-ServiceCIDR` – Overrides the Kubernetes service IP address range from which cluster services are addressed. Defaults to `172.20.0.0/16` or `10.100.0.0/16` based on the IP address of the primary interface.
- `-ExcludedSnatCIDRs` – A list of `IPv4` CIDRs to exclude from Source Network Address Translation (SNAT). This means that the pod private IP which is VPC addressable wouldn't be translated to the IP address of the instance ENI's primary `IPv4` address for outbound traffic. By default, the `IPv4` CIDR of the VPC for the Amazon EKS Windows node is added. Specifying CIDRs to this parameter also additionally excludes the specified CIDRs. For more information, see the section called "Outbound traffic".

In addition to the command line parameters, you can also specify some environment variable parameters. When specifying a command line parameter, it takes precedence over the respective environment variable. The environment variable(s) should be defined as machine (or system) scoped as the bootstrap script will only read machine-scoped variables.

The script takes into account the following environment variables:

- `SERVICE_IPV4_CIDR` – Refer to the `ServiceCIDR` command line parameter for the definition.
- `EXCLUDED_SNAT_CIDRS` – Should be a comma separated string. Refer to the `ExcludedSnatCIDRs` command line parameter for the definition.

gMSA authentication support

Amazon EKS Windows Pods allow different types of group Managed Service Account (gMSA) authentication.

- Amazon EKS supports Active Directory domain identities for authentication. For more information on domain-joined gMSA, see Windows Authentication on Amazon EKS Windowspods on the AWS blog.
- Amazon EKS offers a plugin that enables non-domain-joined Windows nodes to retrieve gMSA credentials with a portable user identity. For more information on domainless gMSA, see Domainless Windows Authentication for Amazon EKS Windowspods on the AWS blog.

Cached container images

Amazon EKS Windows optimized AMIs have certain container images cached for the `containerd` runtime. Container images are cached when building custom AMIs using Amazon-managed build components. For more information, see the section called “Using the Amazon-managed build component”.

The following cached container images are for the `containerd` runtime:

- `amazonaws.com/eks/pause-windows`
- `mcr.microsoft.com/windows/nanoserver`
- `mcr.microsoft.com/windows/servercore`

More information

For more information about using Amazon EKS optimized Windows AMIs, see the following sections:

- For details on running workloads on Amazon EKS optimized accelerated Windows AMIs, see the section called “Run Windows GPU AMIs”.
- To use Windows with managed node groups, see the section called “Managed node groups”.
- To launch self-managed Windows nodes, see the section called “Windows”.
- For version information, see the section called “Get version information”.
- To retrieve the latest IDs of the Amazon EKS optimized Windows AMIs, see the section called “Get latest IDs”.
- To use Amazon EC2 Image Builder to create custom Amazon EKS optimized Windows AMIs, see the section called “Custom builds”.
- For best practices, see Amazon EKS optimized Windows AMI management in the *EKS Best Practices Guide*.

Create self-managed Windows Server 2022 nodes with `eksctl`

You can use the following `test-windows-2022.yaml` as reference for creating self-managed Windows Server 2022 nodes. Replace every *`example value`* with your own values.

> **Note**
>
> You must use `eksctl` version 0.116.0 or later to run self-managed Windows Server 2022 nodes.

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

metadata:
  name: windows-2022-cluster
  region: region-code
  version: '1.32'

nodeGroups:
  - name: windows-ng
    instanceType: m5.2xlarge
    amiFamily: WindowsServer2022FullContainer
    volumeSize: 100
    minSize: 2
    maxSize: 3
  - name: linux-ng
    amiFamily: AmazonLinux2
    minSize: 2
    maxSize: 3
```

The node groups can then be created using the following command.

```
eksctl create cluster -f test-windows-2022.yaml
```

Retrieve Windows AMI version information

> **Important**
>
> Extended Support for Amazon EKS optimized Windows AMIs that are published by AWS isn't available for Kubernetes version `1.23` but is available for Kubernetes version `1.24` and higher.

This topic lists versions of the Amazon EKS optimized Windows AMIs and their corresponding versions of kubelet, containerd, and csi-proxy.

The Amazon EKS optimized AMI metadata, including the AMI ID, for each variant can be retrieved programmatically. For more information, see the section called "Get latest IDs".

AMIs are versioned by Kubernetes version and the release date of the AMI in the following format:

```
k8s_major_version.k8s_minor_version-release_date
```

Note

Amazon EKS managed node groups support the November 2022 and later releases of the Windows AMIs.

Amazon EKS optimized Windows Server 2022 Core AMI

The following tables list the current and previous versions of the Amazon EKS optimized Windows Server 2022 Core AMI.

Example

Kubernetes version 1.32

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.32-2025-02-18	1.32.1	1.7.20	1.1.3	
1.32-2025-01-15	1.32.0	1.7.20	1.1.3	

Kubernetes version 1.31

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.31-2025-02-15	1.31.5	1.7.20	1.1.3	
1.31-2025-01-15	1.31.4	1.7.20	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.31-2025-01-01	1.31.4	1.7.20	1.1.3	Includes patches for CVE-2024-9042 .
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes version 1.30

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.30-2025-02-15	1.30.9	1.7.14	1.1.3	
1.30-2025-01-15	1.30.8	1.7.14	1.1.3	
1.30-2025-01-01	1.30.8	1.7.14	1.1.3	Includes patches for

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
				CVE-2024-9042 .
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	Includes patches for CVE-2024-5321 .
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	Upgraded containerd to 1.7.14.
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes version 1.29

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.29-2025-02-15	1.29.13	1.7.14	1.1.3	
1.29-2025-01-15	1.29.12	1.7.14	1.1.3	
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	Includes patches for CVE-2024-9042 .
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	Upgraded containerd to 1.7.11. Upgraded kubelet to 1.29.3.
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	Upgraded containerd to 1.6.28. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.29-2024.03.12	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	Fixed a bug where the pause image was incorrectly deleted by kubelet garbage collection process.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.29-2024.01.11	1.29.0	1.6.18	1.1.2	Excluded Standalone Windows Update KB5034439 on Windows Server 2022 Core AMIs. The KB applies only to Windows installations with a separate WinRE partition, which aren't included with any of our Amazon EKS Optimized Windows AMIs.

Kubernetes version 1.28

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	
1.28-2025.01.15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-01	1.28.15	1.7.14	1.1.3	Includes patches for

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
				CVE-2024-9042 .
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	Upgraded containerd to 1.7.11.
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	Upgraded containerd to 1.6.28. Upgraded kubelet to 1.28.8.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.28-2024.03.12	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.11	1.28.5	1.6.18	1.1.2	Excluded Standalone Windows Update KB5034439 on Windows Server 2022 Core AMIs. The KB applies only to Windows installations with a separate WinRE partition, which aren't included with any of our Amazon EKS Optimized Windows AMIs.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	Fixed a security advisory in kubelet.
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Kubernetes version 1.27

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2025-02-15	1.27.16	1.7.14	1.1.3	
1.27-2025-01-15	1.27.16	1.7.14	1.1.3	
1.27-2025-01-01	1.27.16	1.7.14	1.1.3	Includes patches for CVE-2024-9042 .
1.27-2024.12.11	1.27.16	1.7.14	1.1.3	
1.27-2024.11.12	1.27.16	1.7.14	1.1.3	
1.27-2024.10.08	1.27.16	1.7.14	1.1.3	
1.27-2024.09.10	1.27.15	1.7.14	1.1.3	
1.27-2024.08.13	1.27.15	1.7.14	1.1.3	
1.27-2024.07.10	1.27.15	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.27-2024.06.17	1.27.12	1.7.11	1.1.2	Upgraded containerd to 1.7.11.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2024.05.14	1.27.12	1.6.28	1.1.2	Upgraded containerd to 1.6.28. Upgraded kubelet to 1.27.12.
1.27-2024.04.09	1.27.9	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.27-2024.03.12	1.27.9	1.6.18	1.1.2	
1.27-2024.02.13	1.27.9	1.6.18	1.1.2	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2024.01.11	1.27.9	1.6.18	1.1.2	Excluded Standalone Windows Update KB5034439 on Windows Server 2022 Core AMIs. The KB applies only to Windows installations with a separate WinRE partition, which aren't included with any of our Amazon EKS Optimized Windows AMIs.
1.27-2023.12.12	1.27.7	1.6.18	1.1.2	
1.27-2023.11.14	1.27.7	1.6.18	1.1.2	Includes patches for CVE-2023-5528.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2023.10.19	1.27.6	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).
1.27-2023-09.27	1.27.6	1.6.6	1.1.2	Fixed a security advisory in kubelet.
1.27-2023.09.12	1.27.4	1.6.6	1.1.2	Upgraded the Amazon VPC CNI plugin to use the Kubernetes connector binary, which gets the Pod IP address from the Kubernetes API server. Merged pull request #100.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2023.08.17	1.27.4	1.6.6	1.1.2	Includes patches for CVE-2023-3676 , CVE-2023-3893 , and CVE-2023-3955 .
1.27-2023.08.08	1.27.3	1.6.6	1.1.1	
1.27-2023.07.11	1.27.3	1.6.6	1.1.1	
1.27-2023.06.20	1.27.1	1.6.6	1.1.1	Resolved issue that was causing the DNS suffix search list to be incorrectly populated.
1.27-2023.06.14	1.27.1	1.6.6	1.1.1	Added support for host port mapping in CNI. Merged pull request #93.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2023.06.06	1.27.1	1.6.6	1.1.1	Fixed containers-roadmap issue #2042, which caused nodes to fail pulling private Amazon ECR images.
1.27-2023.05.17	1.27.1	1.6.6	1.1.1	

Kubernetes version 1.26

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2025-02-15	1.26.15	1.7.14	1.1.3	
1.26-2025-01-15	1.26.15	1.7.14	1.1.3	
1.26-2024.12.11	1.26.15	1.7.14	1.1.3	
1.26-2024.11.12	1.26.15	1.7.14	1.1.3	
1.26-2024.10.08	1.26.15	1.7.14	1.1.3	
1.26-2024.09.10	1.26.15	1.7.14	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2024.08.13	1.26.15	1.7.14	1.1.3	
1.26-2024.07.10	1.26.15	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.26-2024.06.17	1.26.15	1.7.11	1.1.2	Upgraded containerd to 1.7.11.
1.26-2024.05.14	1.26.15	1.6.28	1.1.2	Upgraded containerd to 1.6.28. Upgraded kubelet to 1.26.15.
1.26-2024.04.09	1.26.12	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.26-2024.03.12	1.26.12	1.6.18	1.1.2	
1.26-2024.02.13	1.26.12	1.6.18	1.1.2	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2024.01.11	1.26.12	1.6.18	1.1.2	Excluded Standalone Windows Update KB5034439 on Windows Server 2022 Core AMIs. The KB applies only to Windows installations with a separate WinRE partition, which aren't included with any of our Amazon EKS Optimized Windows AMIs.
1.26-2023.12.12	1.26.10	1.6.18	1.1.2	
1.26-2023.11.14	1.26.10	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2023.10.19	1.26.9	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Upgraded kubelet to 1.26.9. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).
1.26-2023.09.12	1.26.7	1.6.6	1.1.2	Upgraded the Amazon VPC CNI plugin to use the Kubernetes connector binary, which gets the Pod IP address from the Kubernetes API server. Merged pull request #100.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2023.08.17	1.26.7	1.6.6	1.1.2	Includes patches for CVE-2023-3676 , CVE-2023-3893 , and CVE-2023-3955 .
1.26-2023.08.08	1.26.6	1.6.6	1.1.1	
1.26-2023.07.11	1.26.6	1.6.6	1.1.1	
1.26-2023.06.20	1.26.4	1.6.6	1.1.1	Resolved issue that was causing the DNS suffix search list to be incorrectly populated.
1.26-2023.06.14	1.26.4	1.6.6	1.1.1	Upgraded Kubernetes to 1.26.4. Added support for host port mapping in CNI. Merged pull request #93.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2023.05.09	1.26.2	1.6.6	1.1.1	Fixed a bug causing network connectivity issue #1126 on pods after node restart. Introduced a new bootstrap script configuration parameter (ExcludedSnatCIDRs).
1.26-2023.04.26	1.26.2	1.6.6	1.1.1	
1.26-2023.04.11	1.26.2	1.6.6	1.1.1	Added recovery mechanism for kubelet and kube-proxy on service crash.
1.26-2023.03.24	1.26.2	1.6.6	1.1.1	

Kubernetes version 1.25

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2025.02.18	1.25.16	1.7.14	1.1.3	
1.25-2025.01.15	1.25.16	1.7.14	1.1.3	
1.25-2024.12.13	1.25.16	1.7.14	1.1.3	
1.25-2024.11.12	1.25.16	1.7.14	1.1.3	
1.25-2024.10.08	1.25.16	1.7.14	1.1.3	
1.25-2024.09.10	1.25.16	1.7.14	1.1.3	
1.25-2024.08.13	1.25.16	1.7.14	1.1.3	
1.25-2024.07.10	1.25.16	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.25-2024.06.17	1.25.16	1.7.11	1.1.2	Upgraded containerd to 1.7.11.
1.25-2024.05.14	1.25.16	1.6.28	1.1.2	Upgraded containerd to 1.6.28.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2024.04.09	1.25.16	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.25-2024.03.12	1.25.16	1.6.18	1.1.2	
1.25-2024.02.13	1.25.16	1.6.18	1.1.2	
1.25-2024.01.11	1.25.16	1.6.18	1.1.2	Excluded Standalone Windows Update KB5034439 on Windows Server 2022 Core AMIs. The KB applies only to Windows installations with a separate WinRE partition, which aren't included with any of our Amazon EKS Optimized Windows AMIs.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2023.12.12	1.25.15	1.6.18	1.1.2	
1.25-2023.11.14	1.25.15	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .
1.25-2023.10.19	1.25.14	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Upgraded kubelet to 1.25.14. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2023.09.12	1.25.12	1.6.6	1.1.2	Upgraded the Amazon VPC CNI plugin to use the Kubernetes connector binary, which gets the Pod IP address from the Kubernetes API server. Merged pull request #100.
1.25-2023.08.17	1.25.12	1.6.6	1.1.2	Includes patches for CVE-2023-3676, CVE-2023-3893, and CVE-2023-3955.
1.25-2023.08.08	1.25.9	1.6.6	1.1.1	
1.25-2023.07.11	1.25.9	1.6.6	1.1.1	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2023.06.20	1.25.9	1.6.6	1.1.1	Resolved issue that was causing the DNS suffix search list to be incorrectly populated.
1.25-2023.06.14	1.25.9	1.6.6	1.1.1	Upgraded Kubernetes to 1.25.9. Added support for host port mapping in CNI. Merged pull request #93.
1.25-2023.05.09	1.25.7	1.6.6	1.1.1	Fixed a bug causing network connectivity issue #1126 on pods after node restart. Introduced a new bootstrap script configuration parameter (ExcludedSnatCIDRs).

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2023.04.11	1.25.7	1.6.6	1.1.1	Added recovery mechanism for `kubelet` and `kube-proxy` on service crash.
1.25-2023.03.27	1.25.6	1.6.6	1.1.1	Installed a domainless gMSA plugin to facilitate gMSA authentication for Windows containers on Amazon EKS.
1.25-2023.03.20	1.25.6	1.6.6	1.1.1	
1.25-2023.02.14	1.25.6	1.6.6	1.1.1	

Kubernetes version 1.24

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2025-01-15	1.24.17	1.7.14	1.1.3	
1.24-2024.12.11	1.24.17	1.7.14	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2024.11.12	1.24.17	1.7.14	1.1.3	
1.24-2024.10.08	1.24.17	1.7.14	1.1.3	
1.24-2024.09.10	1.24.17	1.7.14	1.1.3	
1.24-2024.08.13	1.24.17	1.7.14	1.1.3	
1.24-2024.07.10	1.24.17	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.24-2024.06.17	1.24.17	1.7.11	1.1.2	Upgraded containerd to 1.7.11.
1.24-2024.05.14	1.24.17	1.6.28	1.1.2	Upgraded containerd to 1.6.28.
1.24-2024.04.09	1.24.17	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.24-2024.03.12	1.24.17	1.6.18	1.1.2	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2024.02.13	1.24.17	1.6.18	1.1.2	
1.24-2024.01.11	1.24.17	1.6.18	1.1.2	Excluded Standalone Windows Update KB5034439 on Windows Server 2022 Core AMIs. The KB applies only to Windows installations with a separate WinRE partition, which aren't included with any of our Amazon EKS Optimized Windows AMIs.
1.24-2023.12.12	1.24.17	1.6.18	1.1.2	
1.24-2023.11.14	1.24.17	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2023.10.19	1.24.17	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Upgraded kubelet to 1.24.17. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).
1.24-2023.09.12	1.24.16	1.6.6	1.1.2	Upgraded the Amazon VPC CNI plugin to use the Kubernetes connector binary, which gets the Pod IP address from the Kubernetes API server. Merged pull request #100.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2023.08.17	1.24.16	1.6.6	1.1.2	Includes patches for CVE-2023-3676 , CVE-2023-3893 , and CVE-2023-3955 .
1.24-2023.08.08	1.24.13	1.6.6	1.1.1	
1.24-2023.07.11	1.24.13	1.6.6	1.1.1	
1.24-2023.06.20	1.24.13	1.6.6	1.1.1	Resolved issue that was causing the DNS suffix search list to be incorrectly populated.
1.24-2023.06.14	1.24.13	1.6.6	1.1.1	Upgraded Kubernetes to 1.24.13. Added support for host port mapping in CNI. Merged pull request #93.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2023.05.09	1.24.7	1.6.6	1.1.1	Fixed a bug causing network connectivity issue #1126 on pods after node restart. Introduced a new bootstrap script configuration parameter (ExcludedSnatCIDRs).
1.24-2023.04.11	1.24.7	1.6.6	1.1.1	Added recovery mechanism for kubelet and kube-proxy on service crash.
1.24-2023.03.27	1.24.7	1.6.6	1.1.1	Installed a domainless gMSA plugin to facilitate gMSA authentication for Windows containers on Amazon EKS.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2023.03.20	1.24.7	1.6.6	1.1.1	Kubernetes version downgraded to 1.24.7 because 1.24.10 has a reported issue in kube-proxy .
1.24-2023.02.14	1.24.10	1.6.6	1.1.1	
1.24-2023.01.23	1.24.7	1.6.6	1.1.1	
1.24-2023.01.11	1.24.7	1.6.6	1.1.1	
1.24-2022.12.13	1.24.7	1.6.6	1.1.1	
1.24-2022.10.11	1.24.7	1.6.6	1.1.1	

Amazon EKS optimized Windows Server 2022 Full AMI

The following tables list the current and previous versions of the Amazon EKS optimized Windows Server 2022 Full AMI.

Example

Kubernetes version 1.32

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.32-2025-02-18	1.32.1	1.7.20	1.1.3	
1.32-2025-01-01	1.32.0	1.7.20	1.1.3	

Kubernetes version 1.31

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.31-2025-02-15	1.31.5	1.7.20	1.1.3	
1.31-2025-01-15	1.31.4	1.7.20	1.1.3	
1.31-2025-01-01	1.31.4	1.7.20	1.1.3	Includes patches for CVE-2024-9042 .
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes version 1.30

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.30-2025-02-15	1.30.9	1.7.14	1.1.3	
1.30-2025-01-15	1.30.8	1.7.14	1.1.3	
1.30-2025-01-01	1.30.8	1.7.14	1.1.3	Includes patches for CVE-2024-9042 .
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	Includes patches for CVE-2024-5321 .
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	Upgraded containerd to 1.7.14.
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes version 1.29

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.29-2025-02-15	1.29.13	1.7.14	1.1.3	
1.29-2025-01-15	1.29.12	1.7.14	1.1.3	
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	Includes patches for CVE-2024-9042 .
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	Upgraded containerd to 1.7.11. Upgraded kubelet to 1.29.3.
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	Upgraded containerd to 1.6.28. Rebuilt CNI and csi-proxy using golang 1.22.1.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.29-2024.03.12	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	Fixed a bug where the pause image was incorrectly deleted by `kubelet` garbage collection process.
1.29-2024.01.09	1.29.0	1.6.18	1.1.2	

Kubernetes version 1.28

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-01	1.28.15	1.7.14	1.1.3	Includes patches for `CVE-2024-9042` .

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	Upgraded containerd to 1.7.11.
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	Upgraded containerd to 1.6.28. Upgraded kubelet to 1.28.8.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.28-2024.03.12	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.09	1.28.5	1.6.18	1.1.2	
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	Fixed a security advisory in kubelet.
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Kubernetes version 1.27

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2025-02-15	1.27.16	1.7.14	1.1.3	
1.27-2025-01-15	1.27.16	1.7.14	1.1.3	
1.27-2025-01-01	1.27.16	1.7.14	1.1.3	Includes patches for

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
				CVE-2024-9042 .
1.27-2024.12.11	1.27.16	1.7.14	1.1.3	
1.27-2024.11.12	1.27.16	1.7.14	1.1.3	
1.27-2024.10.08	1.27.16	1.7.14	1.1.3	
1.27-2024.09.10	1.27.15	1.7.14	1.1.3	
1.27-2024.08.13	1.27.15	1.7.14	1.1.3	
1.27-2024.07.10	1.27.15	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.27-2024.06.17	1.27.12	1.7.11	1.1.2	Upgraded containerd to 1.7.11.
1.27-2024.05.14	1.27.12	1.6.28	1.1.2	Upgraded containerd to 1.6.28. Upgraded kubelet to 1.27.12.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2024.04.09	1.27.9	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.27-2024.03.12	1.27.9	1.6.18	1.1.2	
1.27-2024.02.13	1.27.9	1.6.18	1.1.2	
1.27-2024.01.09	1.27.9	1.6.18	1.1.2	
1.27-2023.12.12	1.27.7	1.6.18	1.1.2	
1.27-2023.11.14	1.27.7	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2023.10.19	1.27.6	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).
1.27-2023-09.27	1.27.6	1.6.6	1.1.2	Fixed a security advisory in kubelet.
1.27-2023.09.12	1.27.4	1.6.6	1.1.2	Upgraded the Amazon VPC CNI plugin to use the Kubernetes connector binary, which gets the Pod IP address from the Kubernetes API server. Merged pull request #100.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2023.08.17	1.27.4	1.6.6	1.1.2	Includes patches for CVE-2023-3676 , CVE-2023-3893 , and CVE-2023-3955 .
1.27-2023.08.08	1.27.3	1.6.6	1.1.1	
1.27-2023.07.11	1.27.3	1.6.6	1.1.1	
1.27-2023.06.20	1.27.1	1.6.6	1.1.1	Resolved issue that was causing the DNS suffix search list to be incorrectly populated.
1.27-2023.06.14	1.27.1	1.6.6	1.1.1	Added support for host port mapping in CNI. Merged pull request #93.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2023.06.06	1.27.1	1.6.6	1.1.1	Fixed containers-roadmap issue #2042, which caused nodes to fail pulling private Amazon ECR images.
1.27-2023.05.18	1.27.1	1.6.6	1.1.1	

Kubernetes version 1.26

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2025-02-15	1.26.15	1.7.14	1.1.3	
1.26-2025-01-15	1.26.15	1.7.14	1.1.3	
1.26-2024.12.11	1.26.15	1.7.14	1.1.3	
1.26-2024.11.12	1.26.15	1.7.14	1.1.3	
1.26-2024.10.08	1.26.15	1.7.14	1.1.3	
1.26-2024.09.10	1.26.15	1.7.14	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2024.08.13	1.26.15	1.7.14	1.1.3	
1.26-2024.07.10	1.26.15	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.26-2024.06.17	1.26.15	1.7.11	1.1.2	Upgraded containerd to 1.7.11.
1.26-2024.05.14	1.26.15	1.6.28	1.1.2	Upgraded containerd to 1.6.28. Upgraded kubelet to 1.26.15.
1.26-2024.04.09	1.26.12	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.26-2024.03.12	1.26.12	1.6.18	1.1.2	
1.26-2024.02.13	1.26.12	1.6.18	1.1.2	
1.26-2024.01.09	1.26.12	1.6.18	1.1.2	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2023.12.12	1.26.10	1.6.18	1.1.2	
1.26-2023.11.14	1.26.10	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .
1.26-2023.10.19	1.26.9	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Upgraded kubelet to 1.26.9. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2023.09.12	1.26.7	1.6.6	1.1.2	Upgraded the Amazon VPC CNI plugin to use the Kubernetes connector binary, which gets the Pod IP address from the Kubernetes API server. Merged pull request #100.
1.26-2023.08.17	1.26.7	1.6.6	1.1.2	Includes patches for CVE-2023-3676 , CVE-2023-3893 , and CVE-2023-3955 .
1.26-2023.08.08	1.26.6	1.6.6	1.1.1	
1.26-2023.07.11	1.26.6	1.6.6	1.1.1	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2023.06.20	1.26.4	1.6.6	1.1.1	Resolved issue that was causing the DNS suffix search list to be incorrectly populated.
1.26-2023.06.14	1.26.4	1.6.6	1.1.1	Upgraded Kubernetes to 1.26.4. Added support for host port mapping in CNI. Merged pull request #93.
1.26-2023.05.09	1.26.2	1.6.6	1.1.1	Fixed a bug causing network connectivity issue #1126 on pods after node restart. Introduced a new bootstrap script configuration parameter (ExcludedSnatCIDRs).
1.26-2023.04.26	1.26.2	1.6.6	1.1.1	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2023.04.11	1.26.2	1.6.6	1.1.1	Added recovery mechanism for `kubelet` and `kube-proxy` on service crash.
1.26-2023.03.24	1.26.2	1.6.6	1.1.1	

Kubernetes version 1.25

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2025.02.18	1.25.16	1.7.14	1.1.3	
1.25-2025-01-15	1.25.16	1.7.14	1.1.3	
1.25-2024.12.13	1.25.16	1.7.14	1.1.3	
1.25-2024.11.12	1.25.16	1.7.14	1.1.3	
1.25-2024.10.08	1.25.16	1.7.14	1.1.3	
1.25-2024.09.10	1.25.16	1.7.14	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2024.08.13	1.25.16	1.7.14	1.1.3	
1.25-2024.07.10	1.25.16	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.25-2024.06.17	1.25.16	1.7.11	1.1.2	Upgraded containerd to 1.7.11.
1.25-2024.05.14	1.25.16	1.6.28	1.1.2	Upgraded containerd to 1.6.28.
1.25-2024.04.09	1.25.16	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.25-2024.03.12	1.25.16	1.6.18	1.1.2	
1.25-2024.02.13	1.25.16	1.6.18	1.1.2	
1.25-2024.01.09	1.25.16	1.6.18	1.1.2	
1.25-2023.12.12	1.25.15	1.6.18	1.1.2	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2023.11.14	1.25.15	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .
1.25-2023.10.19	1.25.14	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Upgraded kubelet to 1.25.14. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).
1.25-2023.09.12	1.25.12	1.6.6	1.1.2	Upgraded the Amazon VPC CNI plugin to use the Kubernetes connector binary, which gets the Pod IP address from the Kubernetes API server. Merged pull request #100.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2023 .08.17	1.25.12	1.6.6	1.1.2	Includes patches for CVE-2023-3676 , CVE-2023-3893 , and CVE-2023-3955 .
1.25-2023 .08.08	1.25.9	1.6.6	1.1.1	
1.25-2023 .07.11	1.25.9	1.6.6	1.1.1	
1.25-2023 .06.20	1.25.9	1.6.6	1.1.1	Resolved issue that was causing the DNS suffix search list to be incorrectly populated.
1.25-2023 .06.14	1.25.9	1.6.6	1.1.1	Upgraded Kubernetes to 1.25.9. Added support for host port mapping in CNI. Merged pull request #93.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2023.05.09	1.25.7	1.6.6	1.1.1	Fixed a bug causing network connectivity issue #1126 on pods after node restart. Introduced a new bootstrap script configuration parameter (ExcludedSnatCIDRs).
1.25-2023.04.11	1.25.7	1.6.6	1.1.1	Added recovery mechanism for kubelet and kube-proxy on service crash.
1.25-2023.03.27	1.25.6	1.6.6	1.1.1	Installed a domainless gMSA plugin to facilitate gMSA authentication for Windows containers on Amazon EKS.
1.25-2023.03.20	1.25.6	1.6.6	1.1.1	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2023 .02.14	1.25.6	1.6.6	1.1.1	

Kubernetes version 1.24

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2025 -01-15	1.24.17	1.7.14	1.1.3	
1.24-2024 .12.11	1.24.17	1.7.14	1.1.3	
1.24-2024 .11.12	1.24.17	1.7.14	1.1.3	
1.24-2024 .10.08	1.24.17	1.7.14	1.1.3	
1.24-2024 .09.10	1.24.17	1.7.14	1.1.3	
1.24-2024 .08.13	1.24.17	1.7.14	1.1.3	
1.24-2024 .07.10	1.24.17	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.24-2024 .06.17	1.24.17	1.7.11	1.1.2	Upgraded containerd to 1.7.11.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2024.05.14	1.24.17	1.6.28	1.1.2	Upgraded containerd to 1.6.28.
1.24-2024.04.09	1.24.17	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.24-2024.03.12	1.24.17	1.6.18	1.1.2	
1.24-2024.02.13	1.24.17	1.6.18	1.1.2	
1.24-2024.01.09	1.24.17	1.6.18	1.1.2	
1.24-2023.12.12	1.24.17	1.6.18	1.1.2	
1.24-2023.11.14	1.24.17	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2023.10.19	1.24.17	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Upgraded kubelet to 1.24.17. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).
1.24-2023.09.12	1.24.16	1.6.6	1.1.2	Upgraded the Amazon VPC CNI plugin to use the Kubernetes connector binary, which gets the Pod IP address from the Kubernetes API server. Merged pull request #100.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2023.08.17	1.24.16	1.6.6	1.1.2	Includes patches for CVE-2023-3676 , CVE-2023-3893 , and CVE-2023-3955 .
1.24-2023.08.08	1.24.13	1.6.6	1.1.1	
1.24-2023.07.11	1.24.13	1.6.6	1.1.1	
1.24-2023.06.20	1.24.13	1.6.6	1.1.1	Resolved issue that was causing the DNS suffix search list to be incorrectly populated.
1.24-2023.06.14	1.24.13	1.6.6	1.1.1	Upgraded Kubernetes to 1.24.13. Added support for host port mapping in CNI. Merged pull request #93.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2023.05.09	1.24.7	1.6.6	1.1.1	Fixed a bug causing network connectivity issue #1126 on pods after node restart. Introduced a new bootstrap script configuration parameter (ExcludedSnatCIDRs).
1.24-2023.04.11	1.24.7	1.6.6	1.1.1	Added recovery mechanism for kubelet and kube-proxy on service crash.
1.24-2023.03.27	1.24.7	1.6.6	1.1.1	Installed a domainless gMSA plugin to facilitate gMSA authentication for Windows containers on Amazon EKS.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2023.03.20	1.24.7	1.6.6	1.1.1	Kubernetes version downgraded to 1.24.7 because 1.24.10 has a reported issue in kube-proxy .
1.24-2023.02.14	1.24.10	1.6.6	1.1.1	
1.24-2023.01.23	1.24.7	1.6.6	1.1.1	
1.24-2023.01.11	1.24.7	1.6.6	1.1.1	
1.24-2022.12.14	1.24.7	1.6.6	1.1.1	
1.24-2022.10.11	1.24.7	1.6.6	1.1.1	

Amazon EKS optimized Windows Server 2019 Core AMI

The following tables list the current and previous versions of the Amazon EKS optimized Windows Server 2019 Core AMI.

Example

Kubernetes version 1.32

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.32-2025-02-18	1.32.1	1.7.20	1.1.3	
1.32-2025-01-15	1.32.4	1.7.20	1.1.3	

Kubernetes version 1.31

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.31-2025-02-15	1.31.5	1.7.20	1.1.3	
1.31-2025-01-15	1.31.4	1.7.20	1.1.3	
1.31-2025-01-01	1.31.4	1.7.20	1.1.3	Includes patches for CVE-2024-9042 .
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes version 1.30

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.30-2025-02-15	1.30.9	1.7.14	1.1.3	
1.30-2025-01-15	1.30.8	1.7.14	1.1.3	
1.30-2025-01-01	1.30.8	1.7.14	1.1.3	Includes patches for `CVE-2024-9042` .
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	Includes patches for CVE-2024-5321 .
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	Upgraded containerd to 1.7.14.
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes version 1.29

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.29-2025-02-15	1.29.13	1.7.14	1.1.3	
1.29-2025-01-15	1.29.12	1.7.14	1.1.3	
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	Includes patches for CVE-2024-9042 .
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	Upgraded containerd to 1.7.11. Upgraded kubelet to 1.29.3.
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	Upgraded containerd to 1.6.28. Rebuilt CNI and csi-proxy using golang 1.22.1.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.29-2024.03.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	Fixed a bug where the pause image was incorrectly deleted by `kubelet` garbage collection process.
1.29-2024.01.09	1.29.0	1.6.18	1.1.2	

Kubernetes version 1.28

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-01	1.28.15	1.7.14	1.1.3	Includes patches for `CVE-2024-9042` .

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	Upgraded containerd to 1.7.11.
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	Upgraded containerd to 1.6.28. Upgraded kubelet to 1.28.8.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.28-2024.03.13	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.09	1.28.5	1.6.18	1.1.2	
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	Fixed a security advisory in kubelet.
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Kubernetes version 1.27

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2025-02-15	1.27.16	1.7.14	1.1.3	
1.27-2025-01-15	1.27.16	1.7.14	1.1.3	
1.27-2025-01-01	1.27.16	1.7.14	1.1.3	Includes patches for

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
				CVE-2024-9042 .
1.27-2024.12.11	1.27.16	1.7.14	1.1.3	
1.27-2024.11.12	1.27.16	1.7.14	1.1.3	
1.27-2024.10.08	1.27.16	1.7.14	1.1.3	
1.27-2024.09.10	1.27.15	1.7.14	1.1.3	
1.27-2024.08.13	1.27.15	1.7.14	1.1.3	
1.27-2024.07.10	1.27.15	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.27-2024.06.17	1.27.12	1.7.11	1.1.2	Upgraded containerd to 1.7.11.
1.27-2024.05.14	1.27.12	1.6.28	1.1.2	Upgraded containerd to 1.6.28. Upgraded kubelet to 1.27.12.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2024.04.09	1.27.9	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.27-2024.03.13	1.27.9	1.6.18	1.1.2	
1.27-2024.02.13	1.27.9	1.6.18	1.1.2	
1.27-2024.01.09	1.27.9	1.6.18	1.1.2	
1.27-2023.12.12	1.27.7	1.6.18	1.1.2	
1.27-2023.11.14	1.27.7	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2023.10.19	1.27.6	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).
1.27-2023-09.27	1.27.6	1.6.6	1.1.2	Fixed a security advisory in kubelet.
1.27-2023.09.12	1.27.4	1.6.6	1.1.2	Upgraded the Amazon VPC CNI plugin to use the Kubernetes connector binary, which gets the Pod IP address from the Kubernetes API server. Merged pull request #100.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2023.08.17	1.27.4	1.6.6	1.1.2	Includes patches for CVE-2023-3676 , CVE-2023-3893 , and CVE-2023-3955 .
1.27-2023.08.08	1.27.3	1.6.6	1.1.1	
1.27-2023.07.11	1.27.3	1.6.6	1.1.1	
1.27-2023.06.20	1.27.1	1.6.6	1.1.1	Resolved issue that was causing the DNS suffix search list to be incorrectly populated.
1.27-2023.06.14	1.27.1	1.6.6	1.1.1	Added support for host port mapping in CNI. Merged pull request #93.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2023.06.06	1.27.1	1.6.6	1.1.1	Fixed `containers-roadmap` [issue #2042](), which caused nodes to fail pulling private Amazon ECR images.
11.27-2023.05.18	1.27.1	1.6.6	1.1.1	

Kubernetes version 1.26

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2025-02-15	1.26.15	1.7.14	1.1.3	
1.26-2025-01-15	1.26.15	1.7.14	1.1.3	
1.26-2024.12.11	1.26.15	1.7.14	1.1.3	
1.26-2024.11.12	1.26.15	1.7.14	1.1.3	
1.26-2024.10.09	1.26.15	1.7.14	1.1.3	
1.26-2024.09.10	1.26.15	1.7.14	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2024.08.13	1.26.15	1.7.14	1.1.3	
1.26-2024.07.10	1.26.15	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.26-2024.06.17	1.26.15	1.7.11	1.1.2	Upgraded containerd to 1.7.11.
1.26-2024.05.14	1.26.15	1.6.28	1.1.2	Upgraded containerd to 1.6.28. Upgraded kubelet to 1.26.15.
1.26-2024.04.09	1.26.12	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.26-2024.03.13	1.26.12	1.6.18	1.1.2	
1.26-2024.02.13	1.26.12	1.6.18	1.1.2	
1.26-2024.01.09	1.26.12	1.6.18	1.1.2	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2023.12.12	1.26.10	1.6.18	1.1.2	
1.26-2023.11.14	1.26.10	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .
1.26-2023.10.19	1.26.9	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Upgraded kubelet to 1.26.9. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2023.09.12	1.26.7	1.6.6	1.1.2	Upgraded the Amazon VPC CNI plugin to use the Kubernetes connector binary, which gets the Pod IP address from the Kubernetes API server. Merged pull request #100.
1.26-2023.08.17	1.26.7	1.6.6	1.1.2	Includes patches for CVE-2023-3676 , CVE-2023-3893 , and CVE-2023-3955 .
1.26-2023.08.08	1.26.6	1.6.6	1.1.1	
1.26-2023.07.11	1.26.6	1.6.6	1.1.1	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2023.06.20	1.26.4	1.6.6	1.1.1	Resolved issue that was causing the DNS suffix search list to be incorrectly populated.
1.26-2023.06.14	1.26.4	1.6.6	1.1.1	Upgraded Kubernetes to 1.26.4. Added support for host port mapping in CNI. Merged pull request #93.
1.26-2023.05.09	1.26.2	1.6.6	1.1.1	Fixed a bug causing network connectivity issue #1126 on pods after node restart. Introduced a new bootstrap script configuration parameter (ExcludedSnatCIDRs).
1.26-2023.04.26	1.26.2	1.6.6	1.1.1	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2023.04.11	1.26.2	1.6.6	1.1.1	Added recovery mechanism for `kubelet` and `kube-proxy` on service crash.
1.26-2023.03.24	1.26.2	1.6.6	1.1.1	

Kubernetes version 1.25

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2025.02.18	1.25.16	1.7.14	1.1.3	
1.25-2025-01-15	1.25.16	1.7.14	1.1.3	
1.25-2024.12.13	1.25.16	1.7.14	1.1.3	
1.25-2024.11.12	1.25.16	1.7.14	1.1.3	
1.25-2024.10.08	1.25.16	1.7.14	1.1.3	
1.25-2024.09.10	1.25.16	1.7.14	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2024.08.13	1.25.16	1.7.14	1.1.3	
1.25-2024.07.10	1.25.16	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.25-2024.06.17	1.25.16	1.7.11	1.1.2	Upgraded containerd to 1.7.11.
1.25-2024.05.14	1.25.16	1.6.28	1.1.2	Upgraded containerd to 1.6.28.
1.25-2024.04.09	1.25.16	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.25-2024.03.13	1.25.16	1.6.18	1.1.2	
1.25-2024.02.13	1.25.16	1.6.18	1.1.2	
1.25-2024.01.09	1.25.16	1.6.18	1.1.2	
1.25-2023.12.12	1.25.15	1.6.18	1.1.2	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2023.11.14	1.25.15	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .
1.25-2023.10.19	1.25.14	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Upgraded kubelet to 1.25.14. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).
1.25-2023.09.12	1.25.12	1.6.6	1.1.2	Upgraded the Amazon VPC CNI plugin to use the Kubernetes connector binary, which gets the Pod IP address from the Kubernetes API server. Merged pull request #100.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2023.08.17	1.25.12	1.6.6	1.1.2	Includes patches for CVE-2023-3676 , CVE-2023-3893 , and CVE-2023-3955 .
1.25-2023.08.08	1.25.9	1.6.6	1.1.1	
1.25-2023.07.11	1.25.9	1.6.6	1.1.1	
1.25-2023.06.20	1.25.9	1.6.6	1.1.1	Resolved issue that was causing the DNS suffix search list to be incorrectly populated.
1.25-2023.06.14	1.25.9	1.6.6	1.1.1	Upgraded Kubernetes to 1.25.9. Added support for host port mapping in CNI. Merged pull request #93.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2023.05.09	1.25.7	1.6.6	1.1.1	Fixed a bug causing network connectivity issue #1126 on pods after node restart. Introduced a new bootstrap script configuration parameter (ExcludedSnatCIDRs).
1.25-2023.04.11	1.25.7	1.6.6	1.1.1	Added recovery mechanism for kubelet and kube-proxy on service crash.
1.25-2023.03.27	1.25.6	1.6.6	1.1.1	Installed a domainless gMSA plugin to facilitate gMSA authentication for Windows containers on Amazon EKS.
1.25-2023.03.20	1.25.6	1.6.6	1.1.1	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2023.02.14	1.25.6	1.6.6	1.1.1	

Kubernetes version 1.24

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2025-01-15	1.24.17	1.7.14	1.1.3	
1.24-2024.12.11	1.24.17	1.7.14	1.1.3	
1.24-2024.11.12	1.24.17	1.7.14	1.1.3	
1.24-2024.10.08	1.24.17	1.7.14	1.1.3	
1.24-2024.09.10	1.24.17	1.7.14	1.1.3	
1.24-2024.08.13	1.24.17	1.7.14	1.1.3	
1.24-2024.07.10	1.24.17	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.24-2024.06.17	1.24.17	1.7.11	1.1.2	Upgraded containerd to 1.7.11.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2024.05.14	1.24.17	1.6.28	1.1.2	Upgraded containerd to 1.6.28.
1.24-2024.04.09	1.24.17	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.24-2024.03.13	1.24.17	1.6.18	1.1.2	
1.24-2024.02.13	1.24.17	1.6.18	1.1.2	
1.24-2024.01.09	1.24.17	1.6.18	1.1.2	
1.24-2023.12.12	1.24.17	1.6.18	1.1.2	
1.24-2023.11.14	1.24.17	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2023.10.19	1.24.17	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Upgraded kubelet to 1.24.17. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).
1.24-2023.09.12	1.24.16	1.6.6	1.1.2	Upgraded the Amazon VPC CNI plugin to use the Kubernetes connector binary, which gets the Pod IP address from the Kubernetes API server. Merged pull request #100.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2023.08.17	1.24.16	1.6.6	1.1.2	Includes patches for CVE-2023-3676 , CVE-2023-3893 , and CVE-2023-3955 .
1.24-2023.08.08	1.24.13	1.6.6	1.1.1	
1.24-2023.07.11	1.24.13	1.6.6	1.1.1	
1.24-2023.06.20	1.24.13	1.6.6	1.1.1	Resolved issue that was causing the DNS suffix search list to be incorrectly populated.
1.24-2023.06.14	1.24.13	1.6.6	1.1.1	Upgraded Kubernetes to 1.24.13. Added support for host port mapping in CNI. Merged pull request #93.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2023.05.09	1.24.7	1.6.6	1.1.1	Fixed a bug causing network connectivity issue #1126 on pods after node restart. Introduced a new bootstrap script configuration parameter (ExcludedSnatCIDRs).
1.24-2023.04.11	1.24.7	1.6.6	1.1.1	Added recovery mechanism for kubelet and kube-proxy on service crash.
1.24-2023.03.27	1.24.7	1.6.6	1.1.1	Installed a domainless gMSA plugin to facilitate gMSA authentication for Windows containers on Amazon EKS.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2023.03.20	1.24.7	1.6.6	1.1.1	Kubernetes version downgraded to 1.24.7 because 1.24.10 has a reported issue in kube-proxy .
1.24-2023.02.14	1.24.10	1.6.6	1.1.1	
1.24-2023.01.23	1.24.7	1.6.6	1.1.1	
1.24-2023.01.11	1.24.7	1.6.6	1.1.1	
1.24-2022.12.13	1.24.7	1.6.6	1.1.1	
1.24-2022.11.08	1.24.7	1.6.6	1.1.1	

Amazon EKS optimized Windows Server 2019 Full AMI

The following tables list the current and previous versions of the Amazon EKS optimized Windows Server 2019 Full AMI.

Example

Kubernetes version 1.32

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.32-2025-02-18	1.32.1	1.7.20	1.1.3	
1.32-2025-01-15	1.32.0	1.7.20	1.1.3	

Kubernetes version 1.31

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.31-2025-02-15	1.31.5	1.7.20	1.1.3	
1.31-2025-01-15	1.31.4	1.7.20	1.1.3	
1.31-2025-01-01	1.31.4	1.7.20	1.1.3	Includes patches for CVE-2024-9042 .
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes version 1.30

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.30-2025-02-15	1.30.9	1.7.14	1.1.3	
1.30-2025-01-15	1.30.8	1.7.14	1.1.3	
1.30-2025-01-01	1.30.8	1.7.14	1.1.3	Includes patches for CVE-2024-9042 .
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	Includes patches for CVE-2024-5321 .
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	Upgraded containerd to 1.7.14.
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes version 1.29

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.29-2025-02-15	1.29.13	1.7.14	1.1.3	
1.29-2025-01-15	1.29.12	1.7.14	1.1.3	
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	Includes patches for CVE-2024-9042 .
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	Upgraded containerd to 1.7.11. Upgraded kubelet to 1.29.3.
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	Upgraded containerd to 1.6.28. Rebuilt CNI and csi-proxy using golang 1.22.1.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.29-2024 .03.13	1.29.0	1.6.25	1.1.2	
1.29-2024 .02.13	1.29.0	1.6.25	1.1.2	
1.29-2024 .02.06	1.29.0	1.6.25	1.1.2	Fixed a bug where the pause image was incorrectly deleted by `kubelet` garbage collection process.
1.29-2024 .01.09	1.29.0	1.6.18	1.1.2	

Kubernetes version 1.28

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.28-2025 .02.15	1.28.15	1.7.14	1.1.3	
1.28-2025 -01-15	1.28.15	1.7.14	1.1.3	
1.28-2025 -01-01	1.28.15	1.7.14	1.1.3	Includes patches for `CVE-2024-9042` .

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	Upgraded containerd to 1.7.11.
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	Upgraded containerd to 1.6.28. Upgraded kubelet to 1.28.8.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.28-2024.03.13	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.09	1.28.5	1.6.18	1.1.2	
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	Fixed a security advisory in kubelet.
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Kubernetes version 1.27

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2025-02-15	1.27.16	1.7.14	1.1.3	
1.27-2025-01-15	1.27.16	1.7.14	1.1.3	
1.27-2025-01-01	1.27.16	1.7.14	1.1.3	Includes patches for

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
				CVE-2024-9042 .
1.27-2024.12.11	1.27.16	1.7.14	1.1.3	
1.27-2024.11.12	1.27.16	1.7.14	1.1.3	
1.27-2024.10.08	1.27.16	1.7.14	1.1.3	
1.27-2024.09.10	1.27.15	1.7.14	1.1.3	
1.27-2024.08.13	1.27.15	1.7.14	1.1.3	
1.27-2024.07.10	1.27.15	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.27-2024.06.17	1.27.12	1.7.11	1.1.2	Upgraded containerd to 1.7.11.
1.27-2024.05.14	1.27.12	1.6.28	1.1.2	Upgraded containerd to 1.6.28. Upgraded kubelet to 1.27.12.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2024.04.09	1.27.9	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.27-2024.03.13	1.27.9	1.6.18	1.1.2	
1.27-2024.02.13	1.27.9	1.6.18	1.1.2	
1.27-2024.01.09	1.27.9	1.6.18	1.1.2	
1.27-2023.12.12	1.27.7	1.6.18	1.1.2	
1.27-2023.11.14	1.27.7	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2023.10.19	1.27.6	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).
1.27-2023-09.27	1.27.6	1.6.6	1.1.2	Fixed a security advisory in kubelet.
1.27-2023.09.12	1.27.4	1.6.6	1.1.2	Upgraded the Amazon VPC CNI plugin to use the Kubernetes connector binary, which gets the Pod IP address from the Kubernetes API server. Merged pull request #100.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2023.08.17	1.27.4	1.6.6	1.1.2	Includes patches for CVE-2023-3676 , CVE-2023-3893 , and CVE-2023-3955 .
1.27-2023.08.08	1.27.3	1.6.6	1.1.1	
1.27-2023.07.11	1.27.3	1.6.6	1.1.1	
1.27-2023.06.20	1.27.1	1.6.6	1.1.1	Resolved issue that was causing the DNS suffix search list to be incorrectly populated.
1.27-2023.06.14	1.27.1	1.6.6	1.1.1	Added support for host port mapping in CNI. Merged pull request #93.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.27-2023.06.06	1.27.1	1.6.6	1.1.1	Fixed containers-roadmap issue #2042, which caused nodes to fail pulling private Amazon ECR images.
1.27-2023.05.17	1.27.1	1.6.6	1.1.1	

Kubernetes version 1.26

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2025-02-15	1.26.15	1.7.14	1.1.3	
1.26-2025-01-15	1.26.15	1.7.14	1.1.3	
1.26-2024.12.11	1.26.15	1.7.14	1.1.3	
1.26-2024.11.12	1.26.15	1.7.14	1.1.3	
1.26-2024.10.08	1.26.15	1.7.14	1.1.3	
1.26-2024.09.10	1.26.15	1.7.14	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2024.08.13	1.26.15	1.7.14	1.1.3	
1.26-2024.07.10	1.26.15	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.26-2024.06.17	1.26.15	1.7.11	1.1.2	Upgraded containerd to 1.7.11.
1.26-2024.05.14	1.26.15	1.6.28	1.1.2	Upgraded containerd to 1.6.28. Upgraded kubelet to 1.26.15.
1.26-2024.04.09	1.26.12	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.26-2024.03.13	1.26.12	1.6.18	1.1.2	
1.26-2024.02.13	1.26.12	1.6.18	1.1.2	
1.26-2024.01.09	1.26.12	1.6.18	1.1.2	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2023.12.12	1.26.10	1.6.18	1.1.2	
1.26-2023.11.14	1.26.10	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .
1.26-2023.10.19	1.26.9	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Upgraded kubelet to 1.26.9. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2023.09.12	1.26.7	1.6.6	1.1.2	Upgraded the Amazon VPC CNI plugin to use the Kubernetes connector binary, which gets the Pod IP address from the Kubernetes API server. Merged pull request #100.
1.26-2023.08.17	1.26.7	1.6.6	1.1.2	Includes patches for CVE-2023-3676 , CVE-2023-3893 , and CVE-2023-3955 .
1.26-2023.08.08	1.26.6	1.6.6	1.1.1	
1.26-2023.07.11	1.26.6	1.6.6	1.1.1	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2023.06.20	1.26.4	1.6.6	1.1.1	Resolved issue that was causing the DNS suffix search list to be incorrectly populated.
1.26-2023.06.14	1.26.4	1.6.6	1.1.1	Upgraded Kubernetes to 1.26.4. Added support for host port mapping in CNI. Merged pull request #93.
1.26-2023.05.09	1.26.2	1.6.6	1.1.1	Fixed a bug causing network connectivity issue #1126 on pods after node restart. Introduced a new bootstrap script configuration parameter (ExcludedSnatCIDRs).
1.26-2023.04.26	1.26.2	1.6.6	1.1.1	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.26-2023.04.11	1.26.2	1.6.6	1.1.1	Added recovery mechanism for `kubelet` and `kube-proxy` on service crash.
1.26-2023.03.24	1.26.2	1.6.6	1.1.1	

Kubernetes version 1.25

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2025.02.18	1.25.16	1.7.14	1.1.3	
1.25-2025-01-15	1.25.16	1.7.14	1.1.3	
1.25-2024.12.13	1.25.16	1.7.14	1.1.3	
1.25-2024.11.12	1.25.16	1.7.14	1.1.3	
1.25-2024.10.08	1.25.16	1.7.14	1.1.3	
1.25-2024.09.10	1.25.16	1.7.14	1.1.3	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2024.08.13	1.25.16	1.7.14	1.1.3	
1.25-2024.07.10	1.25.16	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.25-2024.06.17	1.25.16	1.7.11	1.1.2	Upgraded containerd to 1.7.11.
1.25-2024.05.14	1.25.16	1.6.28	1.1.2	Upgraded containerd to 1.6.28.
1.25-2024.04.09	1.25.16	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.25-2024.03.13	1.25.16	1.6.18	1.1.2	
1.25-2024.02.13	1.25.16	1.6.18	1.1.2	
1.25-2024.01.09	1.25.16	1.6.18	1.1.2	
1.25-2023.12.12	1.25.15	1.6.18	1.1.2	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2023.11.14	1.25.15	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .
1.25-2023.10.19	1.25.14	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Upgraded kubelet to 1.25.14. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).
1.25-2023.09.12	1.25.12	1.6.6	1.1.2	Upgraded the Amazon VPC CNI plugin to use the Kubernetes connector binary, which gets the Pod IP address from the Kubernetes API server. Merged pull request #100.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2023.08.17	1.25.12	1.6.6	1.1.2	Includes patches for CVE-2023-3676 , CVE-2023-3893 , and CVE-2023-3955 .
1.25-2023.08.08	1.25.9	1.6.6	1.1.1	
1.25-2023.07.11	1.25.9	1.6.6	1.1.1	
1.25-2023.06.20	1.25.9	1.6.6	1.1.1	Resolved issue that was causing the DNS suffix search list to be incorrectly populated.
1.25-2023.06.14	1.25.9	1.6.6	1.1.1	Upgraded Kubernetes to 1.25.9. Added support for host port mapping in CNI. Merged pull request #93.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2023.05.09	1.25.7	1.6.6	1.1.1	Fixed a bug causing network connectivity issue #1126 on pods after node restart. Introduced a new bootstrap script configuration parameter (ExcludedSnatCIDRs).
1.25-2023.04.11	1.25.7	1.6.6	1.1.1	Added recovery mechanism for kubelet and kube-proxy on service crash.
1.25-2023.03.27	1.25.6	1.6.6	1.1.1	Installed a domainless gMSA plugin to facilitate gMSA authentication for Windows containers on Amazon EKS.
1.25-2023.03.20	1.25.6	1.6.6	1.1.1	

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.25-2023.02.14	1.25.6	1.6.6	1.1.1	

Kubernetes version 1.24

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2025-01-15	1.24.17	1.7.14	1.1.3	
1.24-2024.12.11	1.24.17	1.7.14	1.1.3	
1.24-2024.11.12	1.24.17	1.7.14	1.1.3	
1.24-2024.10.08	1.24.17	1.7.14	1.1.3	
1.24-2024.09.10	1.24.17	1.7.14	1.1.3	
1.24-2024.08.13	1.24.17	1.7.14	1.1.3	
1.24-2024.07.10	1.24.17	1.7.11	1.1.2	Includes patches for CVE-2024-5321 .
1.24-2024.06.17	1.24.17	1.7.11	1.1.2	Upgraded containerd to 1.7.11.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2024.05.14	1.24.17	1.6.28	1.1.2	Upgraded containerd to 1.6.28.
1.24-2024.04.09	1.24.17	1.6.25	1.1.2	Upgraded containerd to 1.6.25. Rebuilt CNI and csi-proxy using golang 1.22.1.
1.24-2024.03.13	1.24.17	1.6.18	1.1.2	
1.24-2024.02.13	1.24.17	1.6.18	1.1.2	
1.24-2024.01.09	1.24.17	1.6.18	1.1.2	
1.24-2023.12.12	1.24.17	1.6.18	1.1.2	
1.24-2023.11.14	1.24.17	1.6.18	1.1.2	Includes patches for CVE-2023-5528 .

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2023.10.19	1.24.17	1.6.18	1.1.2	Upgraded containerd to 1.6.18. Upgraded kubelet to 1.24.17. Added new bootstrap script environment variables (SERVICE_IPV4_CIDR and EXCLUDED_SNAT_CIDRS).
1.24-2023.09.12	1.24.16	1.6.6	1.1.2	Upgraded the Amazon VPC CNI plugin to use the Kubernetes connector binary, which gets the Pod IP address from the Kubernetes API server. Merged pull request #100.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2023.08.17	1.24.16	1.6.6	1.1.2	Includes patches for CVE-2023-3676 , CVE-2023-3893 , and CVE-2023-3955 .
1.24-2023.08.08	1.24.13	1.6.6	1.1.1	
1.24-2023.07.11	1.24.13	1.6.6	1.1.1	
1.24-2023.06.21	1.24.13	1.6.6	1.1.1	Resolved issue that was causing the DNS suffix search list to be incorrectly populated.
1.24-2023.06.14	1.24.13	1.6.6	1.1.1	Upgraded Kubernetes to 1.24.13. Added support for host port mapping in CNI. Merged pull request #93.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2023.05.09	1.24.7	1.6.6	1.1.1	Fixed a bug causing network connectivity issue #1126 on pods after node restart. Introduced a new bootstrap script configuration parameter (ExcludedSnatCIDRs).
1.24-2023.04.11	1.24.7	1.6.6	1.1.1	Added recovery mechanism for kubelet and kube-proxy on service crash.
1.24-2023.03.27	1.24.7	1.6.6	1.1.1	Installed a domainless gMSA plugin to facilitate gMSA authentication for Windows containers on Amazon EKS.

AMI version	kubelet version	containerd version	csi-proxy version	Release notes
1.24-2023.03.20	1.24.7	1.6.6	1.1.1	Kubernetes version downgraded to 1.24.7 because 1.24.10 has a reported issue in kube-proxy .
1.24-2023.02.14	1.24.10	1.6.6	1.1.1	
1.24-2023.01.23	1.24.7	1.6.6	1.1.1	
1.24-2023.01.11	1.24.7	1.6.6	1.1.1	
1.24-2022.12.14	1.24.7	1.6.6	1.1.1	
1.24-2022.10.12	1.24.7	1.6.6	1.1.1	

Retrieve recommended Microsoft Windows AMI IDs

When deploying nodes, you can specify an ID for a pre-built Amazon EKS optimized Amazon Machine Image (AMI). To retrieve an AMI ID that fits your desired configuration, query the AWS Systems Manager Parameter Store API. Using this API eliminates the need to manually look up Amazon EKS optimized AMI IDs. For more information, see GetParameter. The IAM principal that you use must have the `ssm:GetParameter` IAM permission to retrieve the Amazon EKS optimized AMI metadata.

You can retrieve the image ID of the latest recommended Amazon EKS optimized Windows AMI with the following command, which uses the sub-parameter `image_id`. Make the following modifications to the command as needed and then run the modified command:

- Replace *release* with one of the following options.
  - Use *2022* for Windows Server 2022, but only if you're using Kubernetes version `1.24` or later.
  - Use *2019* for Windows Server 2019.
- Replace *installation-option* with one of the following options. For more information, see [What is the Server Core installation option in Windows Server](#).
  - Use *Core* for a minimal installation with a smaller attack surface.
  - Use *Full* to include the Windows desktop experience.
- Replace *kubernetes-version* with a supported [Amazon EKS version](#).
- Replace *region-code* with an [Amazon EKS supported AWS Region](#) for which you want the AMI ID.

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-release-
English-installation-option-EKS_Optimized-kubernetes-version/image_id \
    --region region-code --query "Parameter.Value" --output text
```

Here's an example command after placeholder replacements have been made.

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-2022-
English-Core-EKS_Optimized-k8s-n-2/image_id \
    --region us-west-2 --query "Parameter.Value" --output text
```

An example output is as follows.

```
ami-1234567890abcdef0
```

Build a custom Windows AMI with Image Builder

You can use EC2 Image Builder to create custom Amazon EKS optimized Windows AMIs with one of the following options:

- [Using an Amazon EKS optimized Windows AMI as a base](#)

- Using the Amazon-managed build component

With both methods, you must create your own Image Builder recipe. For more information, see Create a new version of an image recipe in the Image Builder User Guide.

> **Important**
>
> The following **Amazon-managed** components for `eks` include patches for `CVE-2024-5321`.
>
> - `1.24.5` and higher
> - `1.25.4` and higher
> - `1.26.4` and higher
> - `1.27.2` and higher
> - `1.28.2` and higher
> - `1.29.2` and higher
> - `1.30.1` and higher

Using an Amazon EKS optimized Windows AMI as a base

This option is the recommended way to build your custom Windows AMIs. The Amazon EKS optimized Windows AMIs we provide are more frequently updated than the Amazon-managed build component.

1. Start a new Image Builder recipe.
    a. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder.
    b. In the left navigation pane, choose **Image recipes**.
    c. Choose **Create image recipe**.
2. In the **Recipe details** section, enter a **Name** and **Version**.
3. Specify the ID of the Amazon EKS optimized Windows AMI in the **Base image** section.
    a. Choose **Enter custom AMI ID**.
    b. Retrieve the AMI ID for the Windows OS version that you require. For more information, see the section called "Get latest IDs".

   c. Enter the custom **AMI ID**. If the AMI ID isn't found, make sure that the AWS Region for the AMI ID matches the AWS Region shown in the upper right of your console.

4. (Optional) To get the latest security updates, add the `update-windows` component in the **Build components -** section.

   a. From the dropdown list to the right of the **Find components by name** search box, choose **Amazon-managed**.

   b. In the **Find components by name** search box, enter `update-windows`.

   c. Select the check box of the **`update-windows`** search result. This component includes the latest Windows patches for the operating system.

5. Complete the remaining image recipe inputs with your required configurations. For more information, see [Create a new image recipe version (console)](#) in the Image Builder User Guide.

6. Choose **Create recipe**.

7. Use the new image recipe in a new or existing image pipeline. Once your image pipeline runs successfully, your custom AMI will be listed as an output image and is ready for use. For more information, see [Create an image pipeline using the EC2 Image Builder console wizard](#).

Using the Amazon-managed build component

When using an Amazon EKS optimized Windows AMI as a base isn't viable, you can use the Amazon-managed build component instead. This option may lag behind the most recent supported Kubernetes versions.

1. Start a new Image Builder recipe.

   a. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder.

   b. In the left navigation pane, choose **Image recipes**.

   c. Choose **Create image recipe**.

2. In the **Recipe details** section, enter a **Name** and **Version**.

3. Determine which option you will be using to create your custom AMI in the **Base image** section:

   - **Select managed images** – Choose **Windows** for your **Image Operating System (OS)**. Then choose one of the following options for **Image origin**.

     - **Quick start (Amazon-managed)** – In the **Image name** dropdown, choose an Amazon EKS supported Windows Server version. For more information, see the section called "Windows".

     - **Images owned by me** – For **Image name**, choose the ARN of your own image with your own license. The image that you provide can't already have Amazon EKS components installed.

- **Enter custom AMI ID** – For AMI ID, enter the ID for your AMI with your own license. The image that you provide can't already have Amazon EKS components installed.

4. In the **Build components - Windows** section, do the following:
   a. From the dropdown list to the right of the **Find components by name** search box, choose **Amazon-managed**.
   b. In the **Find components by name** search box, enter `eks`.
   c. Select the check box of the **`eks-optimized-ami-windows`** search result, even though the result returned may not be the version that you want.
   d. In the **Find components by name** search box, enter `update-windows` .
   e. Select the check box of the **update-windows** search result. This component includes the latest Windows patches for the operating system.
5. In the **Selected components** section, do the following:
   a. Choose **Versioning options** for **`eks-optimized-ami-windows`** .
   b. Choose **Specify component version**.
   c. In the **Component Version** field, enter *`version.x`*, replacing *`version`* with a supported Kubernetes version. Entering an *x* for part of the version number indicates to use the latest component version that also aligns with the part of the version you explicitly define. Pay attention to the console output as it will advise you on whether your desired version is available as a managed component. Keep in mind that the most recent Kubernetes versions may not be available for the build component. For more information about available versions, see the section called "Retrieving information about `eks-optimized-ami-windows` component versions".

      Note

      The `1.24.0` `eks-optimized-ami-windows` build component versions require `eksctl` version `0.129` or lower.
6. Complete the remaining image recipe inputs with your required configurations. For more information, see Create a new image recipe version (console) in the Image Builder User Guide.
7. Choose **Create recipe**.
8. Use the new image recipe in a new or existing image pipeline. Once your image pipeline runs successfully, your custom AMI will be listed as an output image and is ready for use. For more information, see Create an image pipeline using the EC2 Image Builder console wizard.

Retrieving information about `eks-optimized-ami-windows` component versions

You can retrieve specific information regarding what is installed with each component. For example, you can verify what `kubelet` version is installed. The components go through functional testing on the Amazon EKS supported Windows operating systems versions. For more information, see the section called "Release calendar". Any other Windows OS versions that aren't listed as supported or have reached end of support might not be compatible with the component.

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder.
2. In the left navigation pane, choose **Components**.
3. From the dropdown list to the right of the **Find components by name** search box, change **Owned by me** to **Quick start (Amazon-managed)**.
4. In the **Find components by name** box, enter `eks`.
5. (Optional) If you are using a recent version, sort the **Version** column in descending order by choosing it twice.
6. Choose the **`eks-optimized-ami-windows`** link with a desired version.

The **Description** in the resulting page shows the specific information.

Enable node auto repair and investigate node health issues

Node health refers to the operational status and capability of a node to effectively run workloads. A healthy node maintains expected connectivity, has sufficient resources, and can successfully run Pods without disruption. For information on getting details about your nodes, see the section called "View node health" and the section called "Get node logs".

To help with maintaining healthy nodes, Amazon EKS offers the node monitoring agent and node auto repair.

Node monitoring agent

The node monitoring agent automatically reads node logs to detect certain health issues. It parses through node logs to detect failures and surfaces various status information about worker nodes. A dedicated `NodeCondition` is applied on the worker nodes for each category of issues detected, such as storage and networking issues. Descriptions of detected health issues are made available in the observability dashboard. For more information, see the section called "Node health issues".

The node monitoring agent is included as a capability for all Amazon EKS Auto Mode clusters. For other cluster types, you can add the monitoring agent as an Amazon EKS add-on. For more information, see the section called "Create an add-on".

Node auto repair

Node auto repair is an additional feature that continuously monitors the health of nodes, automatically reacting to detected problems and replacing nodes when possible. This helps overall availability of the cluster with minimal manual intervention. If a health check fails, the node is automatically cordoned so that no new Pods are scheduled on the node.

By itself, node auto repair can react to the `Ready` condition of the `kubelet` and any node objects that are manually deleted. When paired with the node monitoring agent, node auto repair can react to more conditions that wouldn't be detected otherwise. These additional conditions include `KernelReady`, `NetworkingReady`, and `StorageReady`.

This automated node recovery automatically addresses intermittent node issues such as failures to join the cluster, unresponsive kubelets, and increased accelerator (device) errors. The improved reliability helps reduce application downtime and improve cluster operations. Node auto repair cannot handle certain problems that are reported such as `DiskPressure`, `MemoryPressure`, and `PIDPressure`. Amazon EKS waits 10 minutes before acting on the `AcceleratedHardwareReady` `NodeConditions`, and 30 minutes for all other conditions.

Managed node groups will also automatically disable node repairs for safety reasons under two scenarios. Any repair operations that are previously in progress will continue for both situations.

- If a zonal shift for your cluster has been triggered through the Application Recovery Controller (ARC), all subsequent repair operations are halted.
- If your node group has more than five nodes and more than 20% of the nodes in your node group are in an unhealthy state, repair operations are halted.

You can enable node auto repair when creating or editing a managed node group.

- When using the Amazon EKS console, activate the **Enable node auto repair** checkbox for the managed node group. For more information, see the section called "Create".
- When using the AWS CLI, add the `--node-repair-config enabled=true` to the `eks create nodegroup` or `eks update-nodegroup-config` command.

- For an example `eksctl` `ClusterConfig` that uses a managed node group with node auto repair, see 44-node-repair.yaml on GitHub.

Node health issues

The following tables describe node health issues that can be detected by the node monitoring agent. There are two types of issues:

- Condition – A terminal issue that warrants a remediation action like an instance replacement or reboot. When auto repair is enabled, Amazon EKS will do a repair action, either as a node replacement or reboot. For more information, see the section called “Node conditions”.
- Event – A temporary issue or sub-optimal node configuration. No auto repair action will take place. For more information, see the section called “Node events”.

Kernel node health issues

Name	Severity	Description
ForkFailedOutOfPID	Condition	A fork or exec call has failed due to the system being out of process IDs or memory, which may be caused by zombie processes or physical memory exhaustion.
AppBlocked	Event	The task has been blocked for a long period of time from scheduling, usually caused by being blocked on input or output.
AppCrash	Event	An application on the node has crashed.
ApproachingKernelPidMax	Event	The number of processes is approaching the maximum number of PIDs that are

Name	Severity	Description
		available per the current kernel.pid_max setting, after which no more processes can be launched.
ApproachingMaxOpenFiles	Event	The number of open files is approaching the maximum number of possible open files given the current kernel settings, after which opening new files will fail.
ConntrackExceededKernel	Event	Connection tracking exceeded the maximum for the kernel and new connections could not be established, which can result in packet loss.
ExcessiveZombieProcesses	Event	Processes which can’t be fully reclaimed are accumulat ing in large numbers, which indicates application issues and may lead to reaching system process limits.
KernelBug	Event	A kernel bug was detected and reported by the Linux kernel itself, though this may sometimes be caused by nodes with high CPU or memory usage leading to delayed event processing.

Name	Severity	Description
LargeEnvironment	Event	The number of environme nt variables for this process is larger than expected, potentially caused by many services with enableSer viceLinks set to true, which may cause performance issues.
RapidCron	Event	A cron job is running faster than every five minutes on this node, which may impact performance if the job consumes significant resources.
SoftLockup	Event	The CPU stalled for a given amount of time.

Networking node health issues

Name	Severity	Description
InterfaceNotRunning	Condition	This interface appears to not be running or there are network issues.
InterfaceNotUp	Condition	This interface appears to not be up or there are network issues.
IPAMDNotReady	Condition	IPAMD fails to connect to the API server.

Name	Severity	Description
IPAMDNotRunning	Condition	The `aws-k8s-agent` process was not found to be running.
MissingLoopbackInterface	Condition	The loopback interface is missing from this instance, causing failure of services depending on local connectiv ity.
BandwidthInExceeded	Event	Packets have been queued or dropped because the inbound aggregate bandwidth exceeded the maximum for the instance.
BandwidthOutExceeded	Event	Packets have been queued or dropped because the outbound aggregate bandwidth exceeded the maximum for the instance.
ConntrackExceeded	Event	Connection tracking exceeded the maximum for the instance and new connections could not be established, which can result in packet loss.
IPAMDNoIPs	Event	IPAM-D is out of IP addresses.
IPAMDRepeatedlyRestart	Event	Multiple restarts in the IPAMD service have occurred.
KubeProxyNotReady	Event	Kube-proxy failed to watch or list resources.

Name	Severity	Description
LinkLocalExceeded	Event	Packets were dropped because the PPS of traffic to local proxy services exceeded the network interface maximum.
MissingDefaultRoutes	Event	There are missing default route rules.
MissingIPRules, MissingIP Routes	Event	There are missing route rules for the following Pod IPs from the route table.
NetworkSysctl	Event	This node's network sysctl settings are potentially incorrect.
PortConflict	Event	If a Pod uses hostPort, it can write iptables rules that override the host's already bound ports, potentially preventing API server access to `kubelet`.
PPSExceeded	Event	Packets have been queued or dropped because the bidirecti onal PPS exceeded the maximum for the instance.
UnexpectedRejectRule	Event	An unexpected `REJECT` or `DROP` rule was found in the iptables, potentially blocking expected traffic.

Neuron node health issues

Name	Severity	Description
NeuronDMAError	Condition	A DMA engine encountered an unrecoverable error.
NeuronHBMUncorrect ableError	Condition	An HBM encountered an uncorrectable error and produced incorrect results.
NeuronNCUncorrectableError	Condition	A Neuron Core uncorrectable memory error was detected.
NeuronSRAMUncorrec tableError	Condition	An on-chip SRAM encounter ed a parity error and produced incorrect results.

NVIDIA node health issues

If auto repair is enabled, the repair actions that are listed start 10 minutes after the issue is detected. For more information on XID errors, see Xid Errors in the *NVIDIA GPU Deployment and Management Documentation*. For more information on the individual XID messages, see Understanding Xid Messages in the *NVIDIA GPU Deployment and Management Documentation*.

Name	Severity	Description	Repair action
NvidiaDoubleBitError	Condition	A double bit error was produced by the GPU driver.	Replace
NvidiaNVLinkError	Condition	NVLink errors were reported by the GPU driver.	Replace
NvidiaXID13Error	Condition	There is a graphics engine exception.	Reboot

Name	Severity	Description	Repair action
NvidiaXID31Error	Condition	There are suspected hardware problems.	Reboot
NvidiaXID48Error	Condition	Double bit ECC errors are reported by the driver.	Reboot
NvidiaXID63Error	Condition	There's a page retirement or row remap.	Reboot
NvidiaXID64Error	Condition	There are failures trying to retire a page or perform a node remap.	Reboot
NvidiaXID74Error	Condition	There is a problem with a connectio n from the GPU to another GPU or NVSwitch over NVLink. This may indicate a hardware failure with the link itself or may indicate a problem with the device at the remote end of the link.	Replace
NvidiaXID79Error	Condition	The GPU driver attempted to access the GPU over its PCI Express connection and found that the GPU is not accessible.	Replace

Name	Severity	Description	Repair action
NvidiaXID94Error	Condition	There are ECC memory errors.	Reboot
NvidiaXID95Error	Condition	There are ECC memory errors.	Reboot
NvidiaXID119Error	Condition	The GSP timed out responding to RPC requests from other bits in the driver.	Replace
NvidiaXID120Error	Condition	The GSP has responded in time, but with an error.	Replace
NvidiaXID121Error	Condition	C2C is chip interconn ect. It enables sharing memory between CPUs, accelerators, and more.	Replace
NvidiaXID140Error	Condition	The GPU driver may have observed uncorrectable errors in GPU memory, in such a way as to interrupt the GPU driver's ability to mark the pages for dynamic page offlining or row remapping.	Replace

Name	Severity	Description	Repair action
NvidiaPageRetirement	Event	The GPU driver has marked a memory page for retirement. This may occur if there is a single double bit error or two single bit errors are encountered at the same address.	None
NvidiaXID[Code]Warning	Event	Any occurrences of XIDs other than the ones defined in this list result in this event.	None

Runtime node health issues

Name	Severity	Description
PodStuckTerminating	Condition	A Pod is or was stuck terminating for an excessive amount of time, which can be caused by CRI errors preventing pod state progression.
%sRepeatedRestart	Event	Restarts of any systemd service on the node (formatted using the title-cased unit name).
ContainerRuntimeFailed	Event	The container runtime has failed to create a container,

Name	Severity	Description
		likely related to any reported issues if occurring repeatedly.
KubeletFailed	Event	The kubelet entered a failed state.
LivenessProbeFailures	Event	A liveness probe failure was detected, potentially indicatin g application code issues or insufficient timeout values if occurring repeatedly.
ReadinessProbeFailures	Event	A readiness probe failure was detected, potentially indicatin g application code issues or insufficient timeout values if occurring repeatedly.
ServiceFailedToStart	Event	A systemd unit failed to start.

Storage node health issues

Name	Severity	Description
XFSSmallAverageClusterSize	Condition	The XFS Average Cluster size is small, indicating excessive free space fragmentation that can prevent file creation despite available inodes or free space.
EtcHostsMountFailed	Event	Mounting of the kubelet generated `/etc/hosts` failed due to userdata remounting `/var/lib/`

Name	Severity	Description
		`kubelet/pods` during kubelet-container operation.
IODelays	Event	Input or output delay detected in a process, potentially indicating insuffici ent input-output provisioning if excessive.
KubeletDiskUsageSlow	Event	Kubelet is reporting slow disk usage while trying to access the filesystem, potentially indicating insufficient disk input-output or filesystem issues.

View the health status of your nodes

This topic explains the tools and methods available for monitoring node health status in Amazon EKS clusters. The information covers node conditions, events, and detection cases that help you identify and diagnose node-level issues. Use the commands and patterns described here to inspect node health resources, interpret status conditions, and analyze node events for operational troubleshooting.

You can get some node health information with Kubernetes commands for all nodes. And if you use the node monitoring agent through Amazon EKS Auto Mode or the Amazon EKS managed add-on, you will get a wider variety of node signals to help troubleshoot. Descriptions of detected health issues by the node monitoring agent are also made available in the observability dashboard. For more information, see the section called "Node health".

Node conditions

Node conditions represent terminal issues requiring remediation actions like instance replacement or reboot.

To get conditions for all nodes:

```
kubectl get nodes -o 'custom-
columns=NAME:.metadata.name,CONDITIONS:.status.conditions[*].type,STATUS:.status.conditions[*].
```

To get detailed conditions for a specific node

```
kubectl describe node node-name
```

Example condition output of a healthy node:

```
  - lastHeartbeatTime: "2024-11-21T19:07:40Z"
    lastTransitionTime: "2024-11-08T03:57:40Z"
    message: Monitoring for the Networking system is active
    reason: NetworkingIsReady
    status: "True"
    type: NetworkingReady
```

Example condition of a unhealthy node with a networking problem:

```
  - lastHeartbeatTime: "2024-11-21T19:12:29Z"
    lastTransitionTime: "2024-11-08T17:04:17Z"
    message: IPAM-D has failed to connect to API Server which could be an issue with
      IPTable rules or any other network configuration.
    reason: IPAMDNotReady
    status: "False"
    type: NetworkingReady
```

Node events

Node events indicate temporary issues or sub-optimal configurations.

To get all events reported by the node monitoring agent

When the node monitoring agent is available, you can run the following command.

```
kubectl get events --field-selector=reportingComponent=eks-node-monitoring-agent
```

Sample output:

```
LAST SEEN   TYPE      REASON       OBJECT
 MESSAGE
```

```
4s          Warning   SoftLockup   node/ip-192-168-71-251.us-west-2.compute.internal
 CPU stuck for 23s
```

To get events for all nodes

```
kubectl get events --field-selector involvedObject.kind=Node
```

To get events for a specific node

```
kubectl get events --field-selector involvedObject.kind=Node,involvedObject.name=node-
name
```

To watch events in real-time

```
kubectl get events -w --field-selector involvedObject.kind=Node
```

Example event output:

```
LAST SEEN   TYPE     REASON           OBJECT        MESSAGE
2m          Warning  MemoryPressure   Node/node-1   Node experiencing memory pressure
5m          Normal   NodeReady        Node/node-1   Node became ready
```

Common troubleshooting commands

```
# Get comprehensive node status
kubectl get node node-name -o yaml

# Watch node status changes
kubectl get nodes -w

# Get node metrics
kubectl top node
```

Retrieve node logs for a managed node using kubectl and S3

Learn how to retrieve node logs for an Amazon EKS managed node that has the node monitoring agent.

Prerequisites

Make sure you have the following:

- An existing Amazon EKS cluster with the node monitoring agent. For more information, see the section called "Node health".
- The `kubectl` command-line tool installed and configured to communicate with your cluster.
- The AWS CLI installed and logged in with sufficent permissions to create S3 buckets and objects.
- A recent version of Python 3 installed
- The AWS SDK for Python 3, Boto 3, installed.

Step 1: Create S3 bucket destination (optional)

If you don't already have an S3 bucket to store the logs, create one. Use the following AWS CLI command. The bucket defaults to the `private` access control list. Replace *bucket-name* with your chosen unique bucket name.

```
aws s3api create-bucket --bucket bucket-name
```

Step 2: Create pre-signed S3 URL for HTTP Put

Amazon EKS returns the node logs by doing a HTTP PUT operation to a URL you specify. In this tutorial, we will generate a pre-signed S3 HTTP PUT URL.

The logs will be returned as a gzip tarball, with the `.tar.gz` extension.

Note

You must use the AWS API or a SDK to create the pre-signed S3 upload URL for EKS to upload the log file. You cannot create a pre-signed S3 upload URL using the AWS CLI.

1. Determine where in the bucket you want to store the logs. For example, you might use `2024-11-12/logs1.tar.gz` as the key.
2. Save the following Python code to the file `presign-upload.py`. Replace `<bucket-name>` and `<key>`. The key should end with `.tar.gz`.

```
import boto3; print(boto3.client('s3').generate_presigned_url(
   ClientMethod='put_object',
   Params={'Bucket': '<bucket-name>', 'Key': '<key>'},
   ExpiresIn=1000
))
```

3. Run the script with

```
python presign-upload.py
```

4. Note the URL output. Use this value in the next step as the `http-put-destination`.

For more information, see Generate a presigned URL to upload a file in the AWS Boto3 SDK for Python Documentation.

Step 3: Create NodeDiagnostic resource

Identify the name of the node you want to collect logs from.

Create a `NodeDiagnostic` manifest that uses the name of the node as the resource's name, and providing a HTTP PUT URL destination.

```
apiVersion: eks.amazonaws.com/v1alpha1
kind: NodeDiagnostic
metadata:
    name: node-name
spec:
    logCapture:
        destination: http-put-destination
```

Apply the manifest to the cluster.

```
kubectl apply -f nodediagnostic.yaml
```

You can check on the Status of the collection by describing the `NodeDiagnostic` resource:

- A status of `Success` or `SuccessWithErrors` indicates that the task completed and the logs uploaded to the provided destination (`SuccessWithErrors` indicates that some logs might be missing)

- If the status is Failure, confirm the upload URL is well-formed and not expired.

```
kubectl describe nodediagnostics.eks.amazonaws.com/node-name
```

Step 4: Download logs from S3

Wait approximately one minute before attempting to download the logs. Then, use the S3 CLI to download the logs.

```
# Once NodeDiagnostic shows Success status, download the logs
aws s3 cp s3://bucket-name/key ./node-logs.tar.gz
```

Step 5: Clean up NodeDiagnostic resource

- `NodeDiagnostic` resources do not get automatically deleted. You should clean these up on your own after you have obtained your log artifacts

```
# Delete the NodeDiagnostic resource
kubectl delete nodediagnostics.eks.amazonaws.com/node-name
```

Amazon EKS Hybrid Nodes overview

With *Amazon EKS Hybrid Nodes*, you can use your on-premises and edge infrastructure as nodes in Amazon EKS clusters. AWS manages the AWS-hosted Kubernetes control plane of the Amazon EKS cluster, and you manage the hybrid nodes that run in your on-premises or edge environments. This unifies Kubernetes management across your environments and offloads Kubernetes control plane management to AWS for your on-premises and edge applications.

Amazon EKS Hybrid Nodes works with any on-premises hardware or virtual machines, bringing the efficiency, scalability, and availability of Amazon EKS to wherever your applications need to run. You can use a wide range of Amazon EKS features with Amazon EKS Hybrid Nodes including Amazon EKS add-ons, Amazon EKS Pod Identity, cluster access entries, cluster insights, and extended Kubernetes version support. Amazon EKS Hybrid Nodes natively integrates with AWS services including AWS Systems Manager, AWS IAM Roles Anywhere, Amazon Managed Service for

Prometheus, Amazon CloudWatch, and Amazon GuardDuty for centralized monitoring, logging, and identity management.

With Amazon EKS Hybrid Nodes, there are no upfront commitments or minimum fees, and you are charged per hour for the vCPU resources of your hybrid nodes when they are attached to your Amazon EKS clusters. For more pricing information, see Amazon EKS Pricing.

For an overview of the other Amazon EKS options for on-premises and edge deployments, see the section called “Deployment options”.

General concepts of Amazon EKS Hybrid Nodes

- Amazon EKS Hybrid Nodes must have a reliable connection between your on-premises environment and AWS. Amazon EKS Hybrid Nodes aren’t a fit for disconnected, disrupted, intermittent or limited (DDIL) environments. If you are running in a DDIL environment, consider Amazon EKS Anywhere.
- Running Amazon EKS Hybrid Nodes on cloud infrastructure, including AWS Regions, AWS Local Zones, AWS Outposts, or in other clouds, is not supported. Use Amazon EKS Auto Mode, Karpenter, Amazon EC2 managed node groups, self-managed nodes, or AWS Fargate when running in AWS Regions. Use Amazon EC2 managed node groups or Amazon EC2 self-managed nodes when running on AWS Local Zones. Only Amazon EC2 self-managed nodes can be used on AWS Outposts or AWS Wavelength Zones.
- A single Amazon EKS cluster can be used to run hybrid nodes and nodes in AWS Regions, AWS Local Zones, or AWS Outposts.
- Amazon EKS Hybrid Nodes is available in all AWS Regions, except the AWS GovCloud (US) Regions and the AWS China Regions.
- You will be charged the hybrid nodes fee if you run hybrid nodes on Amazon EC2 instances.
- Billing for hybrid nodes starts when the nodes join the Amazon EKS cluster and stops when the nodes are removed from the cluster. Be sure to remove your hybrid nodes from your Amazon EKS cluster if you are not using them.

Infrastructure Management

- Amazon EKS Hybrid Nodes follows a *bring your own infrastructure* approach where it is your responsibility to provision and manage the physical or virtual machines and the operating system you use for hybrid nodes.

- Amazon EKS Hybrid Nodes are agnostic to the infrastructure they run on. You can run hybrid nodes on physical or virtual machines, and x86 and ARM architectures.

Operating Systems for hybrid nodes

- **Amazon Linux 2023 (AL2023)**: You can use Amazon Linux 2023 (AL2023) as the node operating system for hybrid nodes, but only in virtualized environments such as VMWare, KVM, and Hyper-V. AWS supports the integration of hybrid nodes with AL2023, but AL2023 isn't covered by the AWS Support Plans when you run it outside of Amazon EC2.
- **Ubuntu**: You can use Ubuntu 20.04, Ubuntu 22.04, and Ubuntu 24.04 as the node operating system for hybrid nodes.
- **Red Hat Enterprise Linux (RHEL)**: You can use RHEL 8 and RHEL 9 as the node operating system for hybrid nodes.

Kubernetes and platform versions

- Amazon EKS Hybrid Nodes supports the same Kubernetes versions and deprecation schedule as Amazon EKS, including standard and extended Kubernetes version support. For more information on Kubernetes versions in Amazon EKS, see the section called "Kubernetes versions". For more information about Amazon EKS platform versions, see the section called "Platform versions".
- You must create new Amazon EKS clusters to use Amazon EKS Hybrid Nodes. Hybrid nodes can't be used with existing Amazon EKS clusters.

Networking

- The communication between the Amazon EKS control plane and hybrid nodes is routed through the VPC and subnets you pass during cluster creation, which builds on the existing mechanism in Amazon EKS for control plane to node networking.
- Amazon EKS Hybrid Nodes is flexible to your preferred method of connecting your on-premises networks to a VPC in AWS. There are several documented options available including AWS Site-to-Site VPN and AWS Direct Connect, and you can choose the method that best fits your use case.

- **IP address family**: Hybrid nodes can be used with Amazon EKS clusters configured with the `IPv4` IP address family only. You can't use Amazon EKS clusters configured with the `IPv6` IP address family. Similarly, your on-premises node and Pod CIDRs must be `IPv4` RFC1918 CIDR blocks.
- You must enable the required domains, protocols, and ports for Amazon EKS Hybrid Nodes in your on-premises environments and firewalls. For more information, including minimum networking requirements, see the section called "Prepare networking".
- **Cluster endpoint access**: You can use "Public" or "Private" cluster endpoint access. You should not use "Public and Private" cluster endpoint access, as the endpoint DNS resolution will always resolve to the public addresses for queries originating from your on-premises environment.
- For information and best practices during scenarios where there are network disconnections between hybrid nodes and the AWS Region, see the hybrid nodes section of the *Amazon EKS Best Practices Guide*.
- **Application load balancing**: Kubernetes has a Service object to define the names and domain names for your applications and resolve and load balance to them. By default, the `type:LoadBalancer` type of Service additionally creates an AWS Classic Load Balancer for traffic from outside the cluster. You can change this behavior with add-ons. Specifically, we recommend the AWS Application Load Balancer and AWS Network Load Balancer which are created by the AWS Load Balancer Controller, instead of the AWS Classic Load Balancer. For steps to install the AWS Load Balancer Controller in a hybrid environment, see the section called "AWS Load Balancer Controller".

Security for hybrid nodes

- Amazon EKS Hybrid Nodes use temporary IAM credentials to authenticate with your Amazon EKS cluster. You can use either AWS IAM Roles Anywhere or AWS Systems Manager (SSM) hybrid activations for provisioning the on-premises IAM credentials for hybrid nodes. It is recommended to use AWS SSM hybrid activations if you do not have existing Public Key Infrastructure (PKI) with a Certificate Authority (CA) and certificates for your on-premises environments. If you do have existing PKI and certificates on-premises, use AWS IAM Roles Anywhere.
- You can use `API` or `API_AND_CONFIG_MAP` cluster authentication modes for your hybrid nodes-enabled Amazon EKS clusters. Use the cluster access entry type called `HYBRID_LINUX` with your hybrid nodes IAM role to enable hybrid nodes to join the Amazon EKS cluster.
- OIDC authentication is supported for hybrid nodes-enabled Amazon EKS clusters.

- You can use Amazon EKS Pod Identities and IAM Roles for Service Accounts (IRSA) with applications running on hybrid nodes to enable granular access for your Pods running on hybrid nodes with other AWS services.
- You can use Amazon GuardDuty EKS Protection with hybrid nodes-enabled Amazon EKS clusters to analyze activities of users and applications accessing your cluster.

Add-ons for hybrid nodes

For detailed information, see the section called "Configure add-ons".

- **Container Networking Interface (CNI)**: The AWS VPC CNI can't be used with hybrid nodes. The core capabilities of Cilium and Calico are supported for use with hybrid nodes. You can manage your CNI with your choice of tooling such as Helm. For more information, see the section called "Configure CNI".
- **`kube-proxy` and CoreDNS**: `kube-proxy` and CoreDNS are installed automatically when hybrid nodes join the Amazon EKS cluster. These add-ons can be managed as Amazon EKS add-ons after cluster creation.
- **Ingress and Load Balancing**: You can use the AWS Load Balancer Controller and Application Load Balancer (ALB) or Network Load Balancer (NLB) with the target type `ip` for workloads on hybrid nodes connected with AWS Direct Connect or AWS Site-to-Site VPN. You can alternatively use your choice of Ingress controller or load balancer for application traffic that stays local to your on-premises environment.
- **Metrics**: You can use Amazon Managed Prometheus (AMP) agent-less scrapers, AWS Distro for Open Telemetry (ADOT), and the Amazon CloudWatch Observability Agent with hybrid nodes. To use AMP agent-less scrapers for Pod metrics on hybrid nodes, your Pods must be accessible from the VPC that you use for the Amazon EKS cluster.
- **Logs**: You can enable Amazon EKS control plane logging for hybrid nodes-enabled clusters. You can use the ADOT EKS add-on and the Amazon CloudWatch Observability Agent EKS add-on for hybrid node and Pod logging.

User interfaces

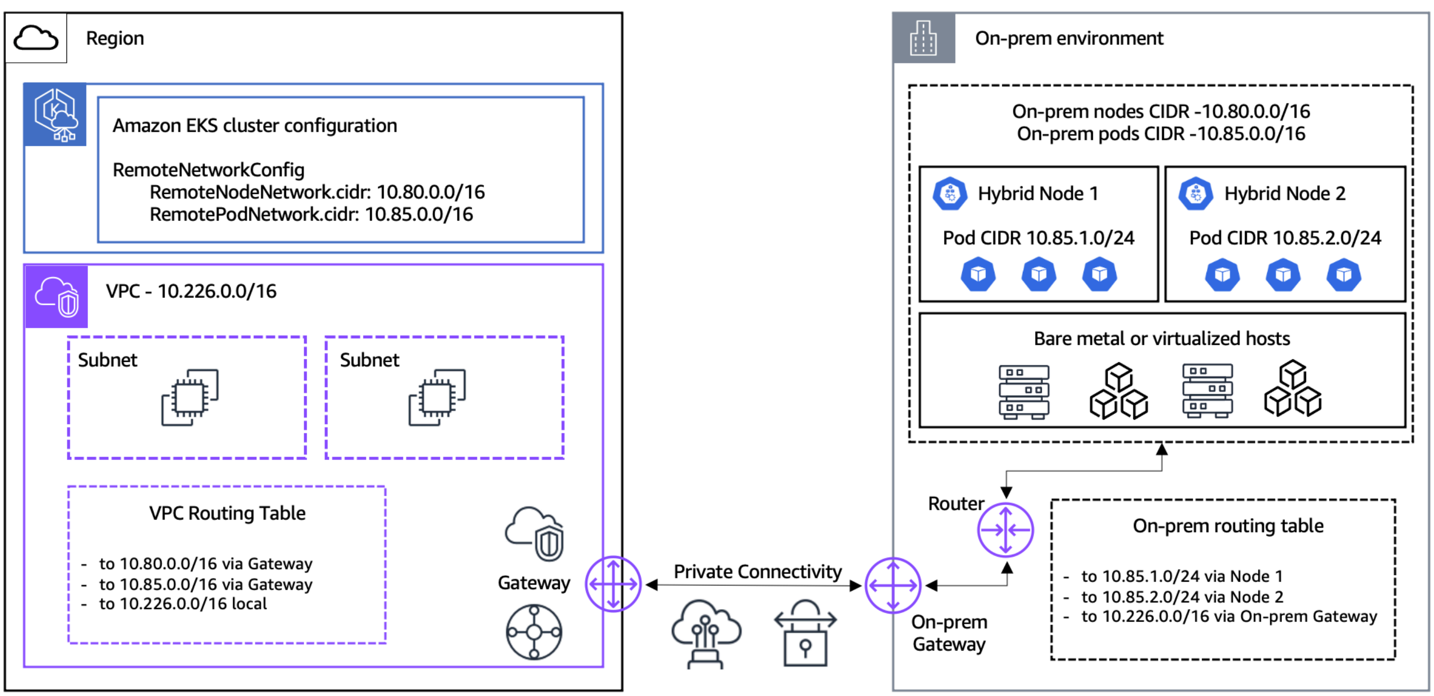
- **Node management**: The Amazon EKS Hybrid Nodes CLI is called `nodeadm` and is run on each on-premises host to simplify the installation, configuration, registration, and uninstall of the hybrid nodes components. The hybrid nodes `nodeadm` version is different than the `nodeadm` version used in the AL2023 Amazon EKS-optimized AMIs. You should not use the hybrid nodes `nodeadm` version for nodes running in Amazon EC2.

- **Cluster management**: The Amazon EKS user interfaces for cluster management are the same with hybrid nodes-enabled Amazon EKS clusters. This includes the AWS Management Console, AWS API, AWS SDKs, AWS CLI, eksctl CLI, AWS CloudFormation, and Terraform.

Prerequisite setup for hybrid nodes

To use Amazon EKS Hybrid Nodes, you must have private connectivity from your on-premises environment to/from AWS, bare metal servers or virtual machines with a supported operating system, and AWS IAM Roles Anywhere or AWS Systems Manager (SSM) hybrid activations configured. You are responsible for managing these prerequisites throughout the hybrid nodes lifecycle.

- Hybrid network connectivity from your on-premises environment to/from AWS
- Infrastructure in the form of physical or virtual machines
- Operating system that is compatible with hybrid nodes
- On-premises IAM credentials provider configured



Hybrid network connectivity

The communication between the Amazon EKS control plane and hybrid nodes is routed through the VPC and subnets you pass during cluster creation, which builds on the existing mechanism in Amazon EKS for control plane to node networking. There are several documented options available for you to connect your on-premises environment with your VPC including AWS Site-to-Site VPN, AWS Direct Connect, or your own VPN connection. Reference the AWS Site-to-Site VPN and AWS Direct Connect user guides for more information on how to use those solutions for your hybrid network connection.

For an optimal experience, AWS recommends reliable network connectivity of at least 100 Mbps and a maximum of 200ms round trip latency for the hybrid nodes connection to the AWS Region. The bandwidth and latency requirements can vary depending on the number of hybrid nodes and your workload characteristics, such as application image size, application elasticity, monitoring and logging configurations, and application dependencies on accessing data stored in other AWS services. We recommend that you test with your own applications and environments before deploying to production to validate that your networking setup meets the requirements for your workloads.

On-premises network configuration

You must enable inbound network access from the Amazon EKS control plane to your on-premises environment to allow the Amazon EKS control plane to communicate with the `kubelet` running on hybrid nodes and optionally with webhooks running on your hybrid nodes. Additionally, you must enable outbound network access for your hybrid nodes and components running on them to communicate with the Amazon EKS control plane. You can configure this communication to stay fully private to your AWS Direct Connect, AWS Site-to-Site VPN, or your own VPN connection. For a full list of the required ports and protocols that you must enable in your firewall and on-premises environment, see the section called “Prepare networking”.

The Classless Inter-Domain Routing (CIDR) ranges you use for your on-premises node and pod networks must use IPv4 RFC1918 address ranges. When you create your hybrid nodes-enabled Amazon EKS cluster, you pass your on-premises node and optionally pod CIDRs to enable communication from the Amazon EKS control plane to your hybrid nodes and the resources running on them. Your on-premises router must be configured with routes to your on-premises nodes and optionally pods. You can use Border Gateway Protocol (BGP) or static configurations to advertise pod IPs to your router.

EKS cluster configuration

To minimize latency, it is recommended to create your Amazon EKS cluster in the AWS Region closest to your on-premises or edge environment. You pass your on-premises node and pod CIDRs during Amazon EKS cluster creation via two API fields: `RemoteNodeNetwork` and `RemotePodNetwork`. You may need to discuss with your on-premises network team to identify your on-premises node and pod CIDRs. The node CIDR is allocated from your on-premises network and the pod CIDR is allocated from the Container Network Interface (CNI) you use if you are using an overlay network for your CNI.

The on-premises node and pod CIDRs are used to configure the Amazon EKS control plane to route traffic through your VPC to the `kubelet` and the pods running on your hybrid nodes. Your on-premises node and pod CIDRs cannot overlap with each other, the VPC CIDR you pass during cluster creation, or the service IPv4 configuration for your Amazon EKS cluster. The pod CIDR is optional. You must configure your pod CIDR if your CNI does not use Network Address Translation (NAT) or masquerading for pod IP addresses when pod traffic leaves your on-premises hosts. You additionally must configure your pod CIDR if you are running *Kubernetes webhooks* on hybrid nodes. For example, AWS Distro for Open Telemetry (ADOT) uses webhooks.

It is recommended to use either public or private endpoint access for the Amazon EKS Kubernetes API server endpoint. If you choose “Public and Private”, the Amazon EKS Kubernetes API server endpoint will always resolve to the public IPs for hybrid nodes running outside of your VPC, which can prevent your hybrid nodes from joining the cluster. You can use either public or private endpoint access for the Amazon EKS Kubernetes API server endpoint. You cannot choose “Public and Private”. When you use public endpoint access, the Kubernetes API server endpoint is resolved to public IPs and the communication from hybrid nodes to the Amazon EKS control plane will be routed over the internet. When you choose private endpoint access, the Kubernetes API server endpoint is resolved to private IPs and the communication from hybrid nodes to the Amazon EKS control plane will be routed over your private connectivity link, in most cases AWS Direct Connect or AWS Site-to-Site VPN.

VPC configuration

You must configure the VPC you pass during Amazon EKS cluster creation with routes in its routing table for your on-premises node and optionally pod networks with your virtual private gateway (VGW) or transit gateway (TGW) as the target. An example is shown below. Replace `REMOTE_NODE_CIDR` and `REMOTE_POD_CIDR` with the values for your on-premises network.

Destination	Target	Description
10.226.0.0/16	local	Traffic local to the VPC routes within the VPC
REMOTE_NODE_CIDR	tgw-abcdef123456	On-prem node CIDR, route traffic to the TGW
REMODE_POD_CIDR	tgw-abcdef123456	On-prem pod CIDR, route traffic to the TGW

Security group configuration

When you create a cluster, Amazon EKS creates a security group that's named `eks-cluster-sg-<cluster-name>-<uniqueID>`. You cannot alter the inbound rules of this Cluster Security Group but you can restrict the outbound rules. You must add an additional security group to your cluster to enable the kubelet and optionally webhooks running on your hybrid nodes to contact the Amazon EKS control plane. The required inbound rules for this additional security group are shown below. Replace `REMOTE_NODE_CIDR` and `REMOTE_POD_CIDR` with the values for your on-premises network.

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source
On-prem node inbound	sgr-abcdef123456	IPv4	HTTPS	TCP	443	REMOTE_NODE_CIDR
On-prem pod inbound	sgr-abcdef654321	IPv4	HTTPS	TCP	443	REMOTE_POD_CIDR

Infrastructure

You must have bare metal servers or virtual machines available to use as hybrid nodes. Hybrid nodes are agnostic to the underlying infrastructure and support x86 and ARM architectures. Amazon EKS Hybrid Nodes follows a "bring your own infrastructure" approach, where you are responsible for provisioning and managing the bare metal servers or virtual machines that you use for hybrid nodes. While there is not a strict minimum resource requirement, it is recommended to use hosts with at least 1 vCPU and 1GiB RAM for hybrid nodes.

Operating system

Amazon Linux 2023 (AL2023), Ubuntu, and RHEL are validated on an ongoing basis for use as the node operating system for hybrid nodes. AWS supports the hybrid nodes integration with these operating systems but does not provide support for the operating systems itself. AL2023 is not covered by AWS Support Plans when run outside of Amazon EC2. AL2023 can only be used in on-premises virtualized environments, see the Amazon Linux 2023 User Guide for more information.

You are responsible for operating system provisioning and management. When you are testing hybrid nodes for the first time, it is easiest to run the Amazon EKS Hybrid Nodes CLI (`nodeadm`) on an already provisioned host. For production deployments, it is recommended to include `nodeadm` in your golden operating system images with it configured to run as a systemd service to automatically join hosts to Amazon EKS clusters at host startup.

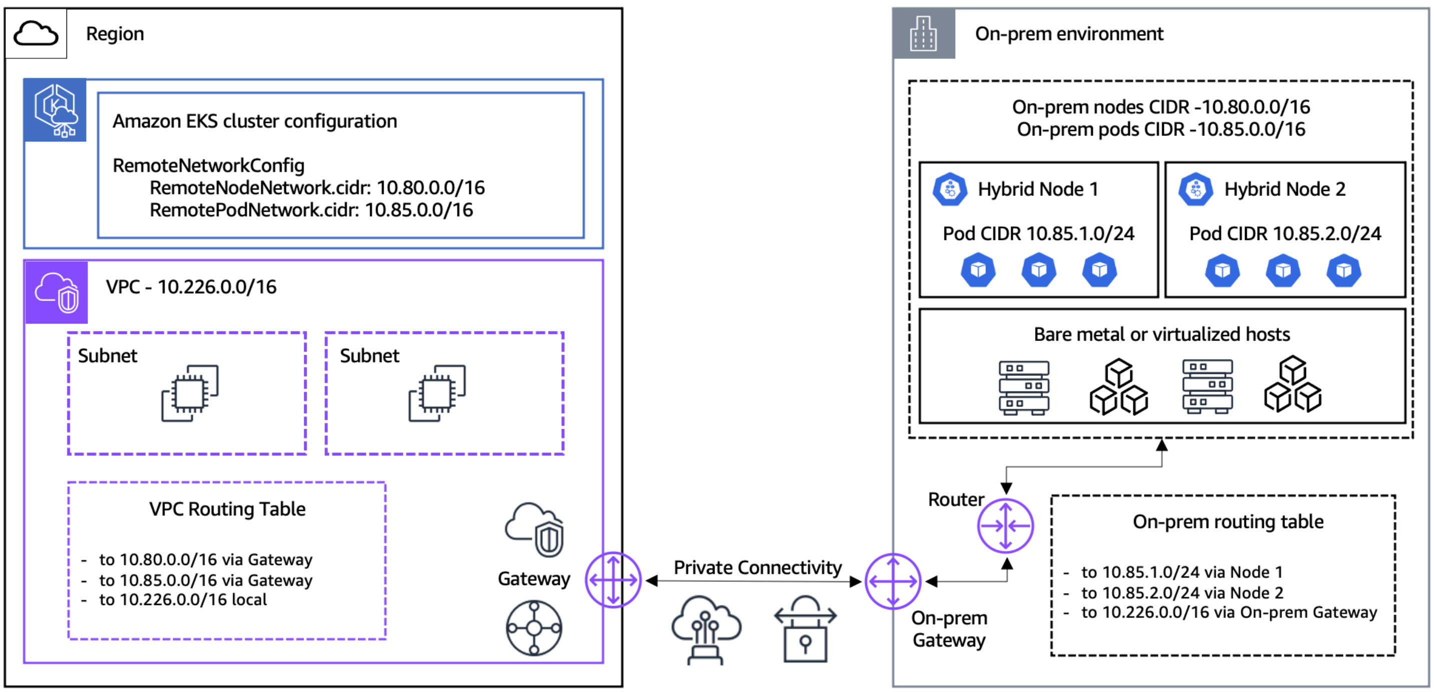
On-premises IAM credentials provider

Amazon EKS Hybrid Nodes use temporary IAM credentials provisioned by AWS SSM hybrid activations or AWS IAM Roles Anywhere to authenticate with the Amazon EKS cluster. You must use either AWS SSM hybrid activations or AWS IAM Roles Anywhere with the Amazon EKS Hybrid Nodes CLI (`nodeadm`). It is recommended to use AWS SSM hybrid activations if you do not have existing Public Key Infrastructure (PKI) with a Certificate Authority (CA) and certificates for your on-premises environments. If you do have existing PKI and certificates on-premises, use AWS IAM Roles Anywhere.

Similar to the the section called "Node IAM role" for nodes running on Amazon EC2, you will create a Hybrid Nodes IAM Role with the required permissions to join hybrid nodes to Amazon EKS clusters. If you are using AWS IAM Roles Anywhere, configure a trust policy that allows AWS IAM Roles Anywhere to assume the Hybrid Nodes IAM Role and configure your AWS IAM Roles Anywhere profile with the Hybrid Nodes IAM Role as an assumable role. If you are using AWS SSM, configure a trust policy that allows AWS SSM to assume the Hybrid Nodes IAM Role and create the hybrid activation with the Hybrid Nodes IAM Role. See the section called “Prepare credentials” for how to create the Hybrid Nodes IAM Role with the required permissions.

Prepare networking for hybrid nodes

This topic provides an overview of the networking setup you must have configured before creating your Amazon EKS cluster and attaching hybrid nodes. This guide assumes you have met the prerequisite requirements for hybrid network connectivity using AWS Site-to-Site VPN, AWS Direct Connect, or your own VPN solution.



On-premises networking configuration

Minimum network requirements

For an optimal experience, AWS recommends reliable network connectivity of at least 100 Mbps and a maximum of 200ms round trip latency for the hybrid nodes connection to the AWS Region. The bandwidth and latency requirements can vary depending on the number of hybrid nodes and your workload characteristics such as application image size, application elasticity, monitoring and logging configurations, and application dependencies on accessing data stored in other AWS services.

On-premises node and pod CIDRs

Identify the node and pod CIDRs you will use for your hybrid nodes and the workloads running on them. The node CIDR is allocated from your on-premises network and the pod CIDR is allocated from your Container Network Interface (CNI) if you are using an overlay network for your CNI. You pass your on-premises node CIDRs and optionally pod CIDRs as inputs when you create your Amazon EKS cluster with the `RemoteNodeNetwork` and `RemotePodNetwork` fields.

The on-premises node and pod CIDR blocks must meet the following requirements:

1. Be within one of the following `IPv4` RFC-1918 ranges: `10.0.0.0/8`, `172.16.0.0/12`, or `192.168.0.0/16`.
2. Not overlap with each other, the VPC CIDR for your Amazon EKS cluster, or your Kubernetes service `IPv4` CIDR.

If your CNI performs Network Address Translation (NAT) for pod traffic as it leaves your on-premises hosts, you do not need to advertise your pod CIDR to your on-premises network or configure your Amazon EKS cluster with your *remote pod network* for hybrid nodes to become ready to workloads. If your CNI does not use NAT for pod traffic as it leaves your on-premises hosts, you must advertise your pod CIDR with your on-premises network and you must configure your Amazon EKS cluster with your remote pod network for hybrid nodes to become ready to workloads. If you are running webhooks on your hybrid nodes, you must advertise your pod CIDR to your on-premises network and configure your Amazon EKS cluster with your remote pod network so the Amazon EKS control plane can directly connect to the webhooks running on hybrid nodes.

Access required during hybrid node installation and upgrade

You must have access to the following domains during the installation process where you install the hybrid nodes dependencies on your hosts. This process can be done once when you are building your operating system images or it can be done on each host at runtime. This includes initial installation and when you upgrade the Kubernetes version of your hybrid nodes.

Component	URL	Protocol	Port
EKS node artifacts (S3)	https://hybrid-assets.eks.amazonaws.com	HTTPS	443
EKS service endpoints	https://eks.*region*.amazonaws.com	HTTPS	443

Component	URL	Protocol	Port
EKS ECR endpoints	See the section called “View Amazon image registries” for regional endpoints.	HTTPS	443
SSM binary endpoint [1]	https://amazon-ssm-*region*.s3.*region*.amazonaws.com	HTTPS	443
SSM service endpoint [1]	https://ssm.*region*.amazonaws.com	HTTPS	443
IAM Anywhere binary endpoint [2]	https://rolesanywhere.amazonaws.com	HTTPS	443
IAM Anywhere service endpoint [2]	https://rolesanywhere.*region*.amazonaws.com	HTTPS	443

Note

[1] Access to the AWS SSM endpoints are only required if you are using AWS SSM hybrid activations for your on-premises IAM credential provider.
[2] Access to the AWS IAM endpoints are only required if you are using AWS IAM Roles Anywhere for your on-premises IAM credential provider.

Access required for ongoing cluster operations

The following network access for your on-premises firewall is required for ongoing cluster operations.

Important

Depending on your choice of CNI, you need to configure additional network access rules for the CNI ports. See the Cilium documentation and the Calico documentation for details.

Type	Protocol	Direction	Port	Source	Destination	Usage
HTTPS	TCP	Outbound	443	Remote Node CIDR(s)	EKS cluster IPs [1]	kubelet to Kubernetes API server
HTTPS	TCP	Outbound	443	Remote Pod CIDR(s)	EKS cluster IPs [1]	Pod to Kubernetes API server
HTTPS	TCP	Outbound	443	Remote Node CIDR(s)	SSM service endpoint	SSM hybrid activations credential refresh and SSM heartbeats every 5 minutes
HTTPS	TCP	Outbound	443	Remote Node CIDR(s)	IAM Anywhere service endpoint	IAM Roles Anywhere credential refresh
HTTPS	TCP	Outbound	443	Remote Pod CIDR(s)	STS Regional Endpoint	Pod to STS endpoint, only

Type	Protocol	Direction	Port	Source	Destination	Usage
						required for IRSA
HTTPS	TCP	Outbound	443	Remote Node CIDR(s)	[Amazon EKS Auth service endpoint](#)	Node to Amazon EKS Auth endpoint, only required for Amazon EKS Pod Identity
HTTPS	TCP	Inbound	10250	EKS cluster IPs [1]	Remote Node CIDR(s)	kubelet to Kubernetes API server
HTTPS	TCP	Inbound	Webhook ports	EKS cluster IPs [1]	Remote Pod CIDR(s)	Kubernetes API server to webhooks

Type	Protocol	Direction	Port	Source	Destination	Usage
HTTPS	TCP,UDP	Inbound,Outbound	53	Remote Pod CIDR(s)	Remote Pod CIDR(s)	Pod to CoreDNS. If you run at least 1 replica of CoreDNS in the cloud, you must allow DNS traffic to the VPC where CoreDNS is running.
User-defined	User-defined	Inbound,Outbound	App ports	Remote Pod CIDR(s)	Remote Pod CIDR(s)	Pod to Pod

Note

[1] The IPs of the Amazon EKS cluster. See the following section on Amazon EKS elastic network interfaces.

Amazon EKS network interfaces

Amazon EKS attaches network interfaces to the subnets in the VPC you pass during cluster creation to enable the communication between the Amazon EKS control plane and your VPC. The network interfaces that Amazon EKS creates can be found after cluster creation in the Amazon EC2 console or with the AWS CLI. The original network interfaces are deleted and new network interfaces are created when changes are applied on your Amazon EKS cluster, such as Kubernetes version upgrades. You can restrict the IP range for the Amazon EKS network interfaces by using constrained subnet sizes for the subnets you pass during cluster creation, which makes it easier to configure your on-premises firewall to allow inbound/outbound connectivity to this known, constrained set of IPs. To control which subnets network interfaces are created in, you can limit the number of subnets you specify when you create a cluster or you can update the subnets after creating the cluster.

The network interfaces provisioned by Amazon EKS have a description of the format `Amazon EKS` *`your-cluster-name`* . See the example below for an AWS CLI command you can use to find the IP addresses of the network interfaces that Amazon EKS provisions. Replace `VPC_ID` with the ID of the VPC you pass during cluster creation.

```
aws ec2 describe-network-interfaces \
--query 'NetworkInterfaces[?(VpcId == VPC_ID && contains(Description,Amazon
 EKS))].PrivateIpAddress'
```

AWS VPC and subnet setup

The existing VPC and subnet requirements for Amazon EKS apply to clusters with hybrid nodes. Additionally, your VPC CIDR can't overlap with your on-premises node and pod CIDRs. You must configure routes in your VPC routing table for your on-premises node and optionally pod CIDRs. These routes must be setup to route traffic to the gateway you are using for your hybrid network connectivity, which is commonly a virtual private gateway (VGW) or transit gateway (TGW). If you are using TGW or VGW to connect your VPC with your on-premises environment, you must create a TGW or VGW attachment for your VPC. Your VPC must have DNS hostname and DNS resolution support.

The following steps use the AWS CLI. You can also create these resources in the AWS Management Console or with other interfaces such as AWS CloudFormation, AWS CDK, or Terraform.

Step 1: Create VPC

1. Run the following command to create a VPC. Replace `VPC_CIDR` with an `IPv4` RFC-1918 (private) or non-RFC-1918 (public) CIDR range (for example `10.0.0.0/16`). Note: DNS resolution, which is an EKS requirement, is enabled for the VPC by default.

   ```
   aws ec2 create-vpc --cidr-block VPC_CIDR
   ```

2. Enable DNS hostnames for your VPC. Note, DNS resolution is enabled for the VPC by default. Replace `VPC_ID` with the ID of the VPC you created in the previous step.

```
aws ec2 modify-vpc-attribute --vpc-id VPC_ID --enable-dns-hostnames
```

Step 2: Create subnets

Create at least 2 subnets. Amazon EKS uses these subnets for the cluster network interfaces. For more information, see the Subnets requirements and considerations.

1. You can find the availability zones for an AWS Region with the following command. Replace `us-west-2` with your region.

```
aws ec2 describe-availability-zones \
     --query 'AvailabilityZones[?(RegionName == us-west-2)].ZoneName'
```

2. Create a subnet. Replace `VPC_ID` with the ID of the VPC. Replace `SUBNET_CIDR` with the CIDR block for your subnet (for example 10.0.1.0/24). Replace `AZ` with the availability zone where the subnet will be created (for example us-west-2a). The subnets you create must be in at least 2 different availability zones.

```
aws ec2 create-subnet \
    --vpc-id VPC_ID \
    --cidr-block SUBNET_CIDR \
    --availability-zone AZ
```

(Optional) Step 3: Attach VPC with Amazon VPC Transit Gateway (TGW) or AWS Direct Connect virtual private gateway (VGW)

If you are using a TGW or VGW, attach your VPC to the TGW or VGW. For more information, see Amazon VPC attachments in Amazon VPC Transit Gateways or AWS Direct Connect virtual private gateway associations.

Transit Gateway

Run the following command to attach a Transit Gateway. Replace `VPC_ID` with the ID of the VPC. Replace `SUBNET_ID1` and `SUBNET_ID2` with the IDs of the subnets you created in the previous step. Replace `TGW_ID` with the ID of your TGW.

```
aws ec2 create-transit-gateway-vpc-attachment \
```

```
    --vpc-id VPC_ID \
    --subnet-ids SUBNET_ID1 SUBNET_ID2 \
    --transit-gateway-id TGW_ID
```

Virtual Private Gateway

Run the following command to attach a Transit Gateway. Replace `VPN_ID` with the ID of your VGW. Replace `VPC_ID` with the ID of the VPC.

```
aws ec2 attach-vpn-gateway \
    --vpn-gateway-id VPN_ID \
    --vpc-id VPC_ID
```

(Optional) Step 4: Create route table

You can modify the main route table for the VPC or you can create a custom route table. The following steps create a custom route table with the routes to on-premises node and pod CIDRs. For more information, see Subnet route tables. Replace `VPC_ID` with the ID of the VPC.

```
aws ec2 create-route-table --vpc-id VPC_ID
```

Step 5: Create routes for on-premises nodes and pods

Create routes in the route table for each of your on-premises remote nodes. You can modify the main route table for the VPC or use the custom route table you created in the previous step.

The examples below show how to create routes for your on-premises node and pod CIDRs. In the examples, a transit gateway (TGW) is used to connect the VPC with the on-premises environment. If you have multiple on-premises node and pods CIDRs, repeat the steps for each CIDR.

- If you are using an internet gateway or a virtual private gateway (VGW) replace `--transit-gateway-id` with `--gateway-id`.
- Replace `RT_ID` with the ID of the route table you created in the previous step.
- Replace `REMOTE_NODE_CIDR` with the CIDR range you will use for your hybrid nodes.
- Replace `REMOTE_POD_CIDR` with the CIDR range you will use for the pods running on hybrid nodes. The pod CIDR range corresponds to the Container Networking Interface

(CNI) configuration, which most commonly uses an overlay network on-premises. For more information, see the section called "Configure CNI".

- Replace TGW_ID with the ID of your TGW.

Remote node network

```
aws ec2 create-route \
    --route-table-id RT_ID \
    --destination-cidr-block REMOTE_NODE_CIDR \
    --transit-gateway-id TGW_ID
```

Remote Pod network

```
aws ec2 create-route \
    --route-table-id RT_ID \
    --destination-cidr-block REMOTE_POD_CIDR \
    --transit-gateway-id TGW_ID
```

(Optional) Step 6: Associate subnets with route table

If you created a custom route table in the previous step, associate each of the subnets you created in the previous step with your custom route table. If you are modifying the VPC main route table, the subnets are automatically associated with the main route table of the VPC and you can skip this step.

Run the following command for each of the subnets you created in the previous steps. Replace RT_ID with the route table you created in the previous step. Replace SUBNET_ID with the ID of a subnet.

```
aws ec2 associate-route-table --route-table-id RT_ID --subnet-id SUBNET_ID
```

Cluster security group configuration

The following access for your Amazon EKS cluster security group is required for ongoing cluster operations.

Type	Protocol	Direction	Port	Source	Destination	Usage
HTTPS	TCP	Inbound	443	Remote Node CIDR(s)	N/A	Kubelet to Kubernetes API server
HTTPS	TCP	Inbound	443	Remote Pod CIDR(s)	N/A	Pods requiring access to K8s API server when the CNI is not using NAT for the pod traffic.
HTTPS	TCP	Outbound	10250	N/A	Remote Node CIDR(s)	Kubernetes API server to Kubelet
HTTPS	TCP	Outbound	Webhook ports	N/A	Remote Pod CIDR(s)	Kubernetes API server to webhook (if running webhooks on hybrid nodes)

To create a security group with the inbound access rules, run the following commands. This security group must be passed when you create your Amazon EKS cluster. By default, the command below creates a security group that allows all outbound access. You can restrict outbound access to include only the rules above. If you're considering limiting the outbound rules, we recommend that you thoroughly test all of your applications and pod connectivity before you apply your changed rules to a production cluster.

- In the first command, replace SG_NAME with a name for your security group
- In the first command, replace VPC_ID with the ID of the VPC you created in the previous step
- In the second command, replace SG_ID with the ID of the security group you create in the first command
- In the second command, replace REMOTE_NODE_CIDR and REMOTE_POD_CIDR with the values for your hybrid nodes and on-premises network.

```
aws ec2 create-security-group \
    --group-name SG_NAME \
    --description "security group for hybrid nodes" \
    --vpc-id VPC_ID
```

```
aws ec2 authorize-security-group-ingress \
    --group-id SG_ID \
    --ip-permissions '[{"IpProtocol": "tcp", "FromPort": 443, "ToPort": 443,
 "IpRanges": [{"CidrIp": "REMOTE_NODE_CIDR"}, {"CidrIp": "REMOTE_POD_CIDR"}]}]'
```

Prepare operating system for hybrid nodes

Amazon Linux 2023 (AL2023), Ubuntu, and Red Hat Enterprise Linux (RHEL) are validated on an ongoing basis for use as the node operating system for hybrid nodes. AWS supports the hybrid nodes integration with these operating systems but does not provide support for the operating systems itself. AL2023 is not covered by AWS Support Plans when run outside of Amazon EC2. AL2023 can only be used in on-premises virtualized environments, reference the Amazon Linux 2023 User Guide for more information.

You are responsible for operating system provisioning and management. When you are testing hybrid nodes for the first time, it is easiest to run the Amazon EKS Hybrid Nodes CLI (`nodeadm`) on an already provisioned host. For production deployments, it is recommended to include `nodeadm` in your operating system images with it configured to run as a systemd service to automatically join hosts to Amazon EKS clusters at host startup.

Version compatibility

The table below represents the operating system versions that are compatible and validated to use as the node operating system for hybrid nodes. If you are using other operating system variants or versions that are not included in this table, then the compatibility of hybrid nodes with your operating system variant or version is not covered by AWS Support. Hybrid nodes are agnostic to the underlying infrastructure and support x86 and ARM architectures.

Operating System	Versions
Amazon Linux	Amazon Linux 2023 (AL2023)
Ubuntu	Ubuntu 20.04, Ubuntu 22.04, Ubuntu 24.04
Red Hat Enterprise Linux	RHEL 8, RHEL 9

Operating system considerations

General

- The Amazon EKS Hybrid Nodes CLI (`nodeadm`) can be used to simplify the installation and configuration of the hybrid nodes components and dependencies. You can run the `nodeadm install` process during your operating system image build pipelines or at runtime on each on-premises host. For more information on the components that `nodeadm` installs, see the the section called “Hybrid nodes nodeadm”.
- If you are using a proxy in your on-premises environment to reach the internet, there is additional operating system configuration required for the install and upgrade processes to configure your package manager to use the proxy. See the section called “Configure proxy” for instructions.

Containerd

- Containerd is the standard Kubernetes container runtime and is a dependency for hybrid nodes, as well as all Amazon EKS node compute types. The Amazon EKS Hybrid Nodes CLI (`nodeadm`) attempts to install containerd during the `nodeadm install` process. You can configure the containerd installation at `nodeadm install` runtime with the `--containerd-source` command line option. Valid options are `none`, `distro`, and `docker`. If you are using RHEL, `distro` is not a valid option and you can either configure `nodeadm` to install the containerd build from Docker's repos or you can manually install containerd. When using AL2023 or Ubuntu, `nodeadm` defaults to installing containerd from the operating system distribution. If you do not want nodeadm to install containerd, use the `--containerd-source none` option.

Ubuntu

- If you are using Ubuntu 20.04, you must use AWS Systems Manager hybrid activations as your credential provider. AWS IAM Roles Anywhere is not supported on Ubuntu 20.04.
- If you are using Ubuntu 24.04, you may need to update your version of containerd or change your AppArmor configuration to adopt a fix that allows pods to properly terminate, see Ubuntu #2065423. A reboot is required to apply changes to the AppArmor profile. The latest version of Ubuntu 24.04 has an updated containerd version in its package manager with the fix (containerd version 1.7.19+).

RHEL

- If you are using RHEL 8, you must use AWS Systems Manager hybrid activations as your credential provider. AWS IAM Roles Anywhere isn't supported on RHEL 8.

ARM

- If you are using ARM hardware, an ARMv8.2 compliant processor with the Cryptography Extension (ARMv8.2+crypto) is required to run version 1.31 and above of the EKS kube-proxy add-on. All Raspberry Pi systems prior to the Raspberry Pi 5, as well as Cortex-A72 based processors, do not meet this requirement. As a workaround, you can continue to use version 1.30 of the EKS kube-proxy add-on until it reaches end of extended support in July of 2026, see the section called “Amazon EKS Kubernetes release calendar”, or use a custom kube-proxy image from upstream.
- The following error message in the kube-proxy log indicates this incompatibility:

```
Fatal glibc error: This version of Amazon Linux requires a newer ARM64 processor
 compliant with at least ARM architecture 8.2-a with Cryptographic extensions. On EC2
 this is Graviton 2 or later.
```

Building operating system images

Amazon EKS provides example Packer templates you can use to create operating system images that include `nodeadm` and configure it to run at host-startup. This process is recommended to avoid pulling the hybrid nodes dependencies individually on each host and to automate the hybrid nodes bootstrap process. You can use the example Packer templates with an Ubuntu 22.04, Ubuntu 24.04, RHEL 8 or RHEL 9 ISO image and can output images with these formats: OVA, Qcow2, or raw.

Prerequisites

Before using the example Packer templates, you must have the following installed on the machine from where you are running Packer.

- Packer version 1.11.0 or higher. For instructions on installing Packer, see Install Packer in the Packer documentation.
- If building OVAs, VMware vSphere plugin 1.4.0 or higher
- If building `Qcow2` or raw images, QEMU plugin version 1.x

Set Environment Variables

Before running the Packer build, set the following environment variables on the machine from where you are running Packer.

General

The following environment variables must be set for building images with all operating systems and output formats.

Environment Variable	Type	Description
PKR_SSH_PASSWORD	String	Packer uses the `ssh_username` and `ssh_password` variables to SSH into the created machine when provisioning. This needs to match the passwords used to create the initial user within

Environment Variable	Type	Description
		the respective OS's kickstart or user-data files. The default is set as "builder" or "ubuntu" depending on the OS. When setting your password, make sure to change it within the corresponding `ks.cfg` or `user-data` file to match.
ISO_URL	String	URL of the ISO to use. Can be a web link to download from a server, or an absolute path to a local file
ISO_CHECKSUM	String	Associated checksum for the supplied ISO.
CREDENTIAL_PROVIDER	String	Credential provider for hybrid nodes. Valid values are `ssm` (default) for SSM hybrid activations and `iam` for IAM Roles Anywhere
K8S_VERSION	String	Kubernetes version for hybrid nodes (for example `1.31`). For supported Kubernetes versions, see the section called "Kubernetes versions".
NODEADM_ARCH	String	Architecture for `nodeadm install`. Select `amd` or `arm`.

RHEL

If you are using RHEL, the following environment variables must be set.

Environment Variable	Type	Description
RH_USERNAME	String	RHEL subscription manager username
RH_PASSWORD	String	RHEL subscription manager password
RHEL_VERSION	String	Rhel iso version being used. Valid values are 8 or 9.

Ubuntu

There are no Ubuntu-specific environment variables required.

vSphere

If you are building a VMware vSphere OVA, the following environment variables must be set.

Environment Variable	Type	Description
VSPHERE_SERVER	String	vSphere server address
VSPHERE_USER	String	vSphere username
VSPHERE_PASSWORD	String	vSphere password
VSPHERE_DATACENTER	String	vSphere datacenter name
VSPHERE_CLUSTER	String	vSphere cluster name
VSPHERE_DATASTORE	String	vSphere datastore name
VSPHERE_NETWORK	String	vSphere network name
VSPHERE_OUTPUT_FOLDER	String	vSphere output folder for the templates

QEMU

Environment Variable	Type	Description
PACKER_OUTPUT_FORMAT	String	Output format for the QEMU builder. Valid values are `qcow2` and `raw`.

Validate template

Before running your build, validate your template with the following command after setting your environment variables. Replace `template.pkr.hcl` if you are using a different name for your template.

```
packer validate template.pkr.hcl
```

Build images

Build your images with the following commands and use the `-only` flag to specify the target and operating system for your images. Replace `template.pkr.hcl` if you are using a different name for your template.

vSphere OVAs

Note

If you are using RHEL with vSphere you need to convert the kickstart files to an OEMDRV image and pass it as an ISO to boot from. For more information, see the Packer Readme in the EKS Hybrid Nodes GitHub Repository.

Ubuntu 22.04 OVA

```
packer build -only=general-build.vsphere-iso.ubuntu22 template.pkr.hcl
```

Ubuntu 24.04 OVA

```
packer build -only=general-build.vsphere-iso.ubuntu24 template.pkr.hcl
```

RHEL 8 OVA

```
packer build -only=general-build.vsphere-iso.rhel8 template.pkr.hcl
```

RHEL 9 OVA

```
packer build -only=general-build.vsphere-iso.rhel9 template.pkr.hcl
```

QEMU

> **Note**
>
> If you are building an image for a specific host CPU that does not match your builder host, see the QEMU documentation for the name that matches your host CPU and use the `-cpu` flag with the name of the host CPU when you run the following commands.

Ubuntu 22.04 Qcow2 / Raw

```
packer build -only=general-build.qemu.ubuntu22 template.pkr.hcl
```

Ubuntu 24.04 Qcow2 / Raw

```
packer build -only=general-build.qemu.ubuntu24 template.pkr.hcl
```

RHEL 8 Qcow2 / Raw

```
packer build -only=general-build.qemu.rhel8 template.pkr.hcl
```

RHEL 9 Qcow2 / Raw

```
packer build -only=general-build.qemu.rhel9 template.pkr.hcl
```

Pass nodeadm configuration through user-data

You can pass configuration for `nodeadm` in your user-data through cloud-init to configure and automatically connect hybrid nodes to your EKS cluster at host startup. Below is an example for how to accomplish this when using VMware vSphere as the infrastructure for your hybrid nodes.

1. Install the the `govc` CLI following the instructions in the govc readme on GitHub.

2. After running the Packer build in the previous section and provisioning your template, you can clone your template to create multiple different nodes using the following. You must clone the template for each new VM you are creating that will be used for hybrid nodes. Replace the variables in the command below with the values for your environment. The `VM_NAME` in the command below is used as your `NODE_NAME` when you inject the names for your VMs via your `metadata.yaml` file.

```
govc vm.clone -vm "/PATH/TO/TEMPLATE" -ds="YOUR_DATASTORE" \
    -on=false -template=false -folder=/FOLDER/TO/SAVE/VM "VM_NAME"
```

3. After cloning the template for each of your new VMs, create a `userdata.yaml` and `metadata.yaml` for your VMs. Your VMs can share the same `userdata.yaml` and `metadata.yaml` and you will populate these on a per VM basis in the steps below. The `nodeadm` configuration is created and defined in the `write_files` section of your `userdata.yaml`. The example below uses AWS SSM hybrid activations as the on-premises credential provider for hybrid nodes. For more information on `nodeadm` configuration, see the the section called "Hybrid nodes nodeadm".

userdata.yaml:

```
#cloud-config
users:
  - name: # username for login. Use 'builder' for RHEL or 'ubuntu' for Ubuntu.
    passwd: # password to login. Default is 'builder' for RHEL.
    groups: [adm, cdrom, dip, plugdev, lxd, sudo]
    lock-passwd: false
    sudo: ALL=(ALL) NOPASSWD:ALL
    shell: /bin/bash

write_files:
  - path: /usr/local/bin/nodeConfig.yaml
    permissions: '0644'
    content: |
      apiVersion: node.eks.aws/v1alpha1
      kind: NodeConfig
      spec:
          cluster:
              name: # Cluster Name
              region: # AWS region
          hybrid:
              ssm:
```

```
                  activationCode: # Your ssm activation code
                  activationId: # Your ssm activation id

runcmd:
  - /usr/local/bin/nodeadm init -c file:///usr/local/bin/nodeConfig.yaml >> /var/log/
nodeadm-init.log 2>&1
```

metadata.yaml:

Create a `metadata.yaml` for your environment. Keep the `"$NODE_NAME"` variable format in the file as this will be populated with values in a subsequent step.

```
instance-id: "$NODE_NAME"
local-hostname: "$NODE_NAME"
network:
  version: 2
  ethernets:
    nics:
      match:
        name: ens*
      dhcp4: yes
```

4. Add the `userdata.yaml` and `metadata.yaml` files as `gzip+base64` strings with the following commands. The following commands should be run for each of the VMs you are creating. Replace `VM_NAME` with the name of the VM you are updating.

```
export NODE_NAME="VM_NAME"
export USER_DATA=$(gzip -c9 <userdata.yaml | base64)

govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.userdata="${USER_DATA}"
govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.userdata.encoding=gzip+base64

envsubst '$NODE_NAME' < metadata.yaml > metadata.yaml.tmp
export METADATA=$(gzip -c9 <metadata.yaml.tmp | base64)

govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.metadata="${METADATA}"
govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.metadata.encoding=gzip+base64
```

5. Power on your new VMs, which should automatically connect to the EKS cluster you configured.

```
govc vm.power -on "${NODE_NAME}"
```

Prepare credentials for hybrid nodes

Amazon EKS Hybrid Nodes use temporary IAM credentials provisioned by AWS SSM hybrid activations or AWS IAM Roles Anywhere to authenticate with the Amazon EKS cluster. You must use either AWS SSM hybrid activations or AWS IAM Roles Anywhere with the Amazon EKS Hybrid Nodes CLI (`nodeadm`). You should not use both AWS SSM hybrid activations and AWS IAM Roles Anywhere. It is recommended to use AWS SSM hybrid activations if you do not have existing Public Key Infrastructure (PKI) with a Certificate Authority (CA) and certificates for your on-premises environments. If you do have existing PKI and certificates on-premises, use AWS IAM Roles Anywhere.

Hybrid Nodes IAM Role

Before you can connect hybrid nodes to your Amazon EKS cluster, you must create an IAM role that will be used with AWS SSM hybrid activations or AWS IAM Roles Anywhere for your hybrid nodes credentials. After cluster creation, you will use this role with an Amazon EKS access entry or `aws-auth` ConfigMap entry to map the IAM role to Kubernetes Role-Based Access Control (RBAC). For more information on associating the Hybrid Nodes IAM role with Kubernetes RBAC, see the section called “Prepare cluster access”.

The Hybrid Nodes IAM role must have the following permissions.

- Permissions for `nodeadm` to use the `eks:DescribeCluster` action to gather information about the cluster used for connecting hybrid nodes to the cluster. If you do not enable the `eks:DescribeCluster` action, then you must pass your Kubernetes API endpoint, cluster CA bundle, and service IPv4 CIDR in the node configuration you pass to `nodeadm` when you run `nodeadm` init.
- Permissions for the kubelet to use container images from Amazon Elastic Container Registry (Amazon ECR) as defined in the AmazonEC2ContainerRegistryPullOnly policy.
- If using AWS SSM, permissions for `nodeadm` init to use AWS SSM hybrid activations as defined in the aws-managed-policy/latest/reference/AmazonSSMManagedInstanceCore.html policy.

- If using AWS SSM, permissions to use the `ssm:DeregisterManagedInstance` action and `ssm:DescribeInstanceInformation` action for `nodeadm uninstall` to deregister instances.
- (Optional) Permissions for the Amazon EKS Pod Identity Agent to use the `eks-auth:AssumeRoleForPodIdentity` action to retrieve credentials for pods.

Setup AWS SSM hybrid activations

Before setting up AWS SSM hybrid activations, you must have a Hybrid Nodes IAM role created and configured. For more information, see the section called "Create the Hybrid Nodes IAM role". Follow the instructions at Create a hybrid activation to register nodes with Systems Manager in the AWS Systems Manager User Guide to create an AWS SSM hybrid activation for your hybrid nodes. The Activation Code and ID you receive is used with `nodeadm` when you register your hosts as hybrid nodes with your Amazon EKS cluster. You can come back to this step at a later point after you have created and prepared your Amazon EKS clusters for hybrid nodes.

> **Important**
>
> Systems Manager immediately returns the Activation Code and ID to the console or the command window, depending on how you created the activation. Copy this information and store it in a safe place. If you navigate away from the console or close the command window, you might lose this information. If you lose it, you must create a new activation.

By default, AWS SSM hybrid activations are active for 24 hours. You can alternatively specify an `--expiration-date` when you create your hybrid activation in timestamp format, such as `2024-08-01T00:00:00`. When you use AWS SSM as your credential provider, the node name for your hybrid nodes is not configurable, and is auto-generated by AWS SSM. You can view and manage the AWS SSM Managed Instances in the AWS Systems Manager console under Fleet Manager. You can register up to 1,000 standard hybrid-activated nodes per account per AWS Region at no additional cost. However, registering more than 1,000 hybrid nodes requires that you activate the advanced-instances tier. There is a charge to use the advanced-instances tier that is not included in the Amazon EKS Hybrid Nodes pricing. For more information, see AWS Systems Manager Pricing.

See the example below for how to create an AWS SSM hybrid activation with your Hybrid Nodes IAM role. When you use AWS SSM hybrid activations for your hybrid nodes credentials, the names of your hybrid nodes will have the format `mi-012345678abcdefgh` and the temporary credentials provisioned by AWS SSM are valid for 1 hour. You cannot alter the node name or credential duration when using AWS SSM as your credential provider. The temporary credentials are automatically rotated by AWS SSM and the rotation does not impact the status of your nodes or applications.

It is recommended to use one AWS SSM hybrid activation per EKS cluster to scope the AWS SSM `ssm:DeregisterManagedInstance` permission of the Hybrid Nodes IAM role to only be able to deregister instances that are associated with your AWS SSM hybrid activation. In the example on this page, a tag with the EKS cluster ARN is used, which can be used to map your AWS SSM hybrid activation to the EKS cluster. You can alternatively use your preferred tag and method of scoping the AWS SSM permissions based on your permission boundaries and requirements. The `REGISTRATION_LIMIT` option in the command below is an integer used to limit the number of machines that can use the AWS SSM hybrid activation (for example `10`)

```
aws ssm create-activation \
     --region AWS_REGION \
     --default-instance-name eks-hybrid-nodes \
     --description "Activation for EKS hybrid nodes" \
     --iam-role AmazonEKSHybridNodesRole \
     --tags Key=EKSClusterARN,Value=arn:aws:eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/
CLUSTER_NAME \
     --registration-limit REGISTRATION_LIMIT
```

Review the instructions on Create a hybrid activation to register nodes with Systems Manager for more information about the available configuration settings for AWS SSM hybrid activations.

Setup AWS IAM Roles Anywhere

Follow the instructions at Getting started with IAM Roles Anywhere in the IAM Roles Anywhere User Guide to set up the trust anchor and profile you will use for temporary IAM credentials for your Hybrid Nodes IAM role. When you create your profile, you can create it without adding any roles. You can create this profile, return to these steps to create your Hybrid Nodes IAM role, and then add your role to your profile after it is created. You can alternatively use the AWS CloudFormation steps later on this page to complete your IAM Roles Anywhere setup for hybrid nodes.

When you add the Hybrid Nodes IAM role to your profile, select **Accept custom role session name** in the **Custom role** session name panel at the bottom of the **Edit profile** page in the

AWS IAM Roles Anywhere console. This corresponds to the acceptRoleSessionName field of the `CreateProfile` API. This allows you to supply a custom node name for your hybrid nodes in the configuration you pass to `nodeadm` during the bootstrap process. Passing a custom node name during the `nodeadm init` process is required. You can update your profile to accept a custom role session name after creating your profile.

You can configure the credential validity duration with AWS IAM Roles Anywhere through the durationSeconds field of your AWS IAM Roles Anywhere profile. The default duration is 1 hour with a maximum of 12 hours. The `MaxSessionDuration` setting on your Hybrid Nodes IAM role must be greater than the `durationSeconds` setting on your AWS IAM Roles Anywhere profile. For more information on `MaxSessionDuration`, see UpdateRole API documentation.

The per-machine certificates and keys you generate from your certificate authority (CA) must be placed in the `/etc/iam/pki` directory on each hybrid node with the file names `server.pem` for the certificate and `server.key` for the key.

Create the Hybrid Nodes IAM role

To run the steps in this section, the IAM principal using the AWS console or AWS CLI must have the following permissions.

- `iam:CreatePolicy`
- `iam:CreateRole`
- `iam:AttachRolePolicy`
- If using AWS IAM Roles Anywhere
  - `rolesanywhere:CreateTrustAnchor`
  - `rolesanywhere:CreateProfile`
  - `iam:PassRole`

AWS CloudFormation

Install and configure the AWS CLI, if you haven't already. See Installing or updating to the last version of the AWS CLI.

Steps for AWS SSM hybrid activations

The CloudFormation stack creates the Hybrid Nodes IAM Role with the permissions outlined above. The CloudFormation template does not create the AWS SSM hybrid activation.

1. Download the AWS SSM CloudFormation template for hybrid nodes:

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/
hybrid-ssm-cfn.yaml'
```

2. Create a `cfn-ssm-parameters.json` with the following options:

   a. Replace `ROLE_NAME` with the name for your Hybrid Nodes IAM role. By default, the CloudFormation template uses `AmazonEKSHybridNodesRole` as the name of the role it creates if you do not specify a name.

   b. Replace `TAG_KEY` with the AWS SSM resource tag key you used when creating your AWS SSM hybrid activation. The combination of the tag key and tag value is used in the condition for the `ssm:DeregisterManagedInstance` to only allow the Hybrid Nodes IAM role to deregister the AWS SSM managed instances that are associated with your AWS SSM hybrid activation. In the CloudFormation template, `TAG_KEY` defaults to `EKSClusterARN`.

   c. Replace `TAG_VALUE` with the AWS SSM resource tag value you used when creating your AWS SSM hybrid activation. The combination of the tag key and tag value is used in the condition for the `ssm:DeregisterManagedInstance` to only allow the Hybrid Nodes IAM role to deregister the AWS SSM managed instances that are associated with your AWS SSM hybrid activation. If you are using the default `TAG_KEY` of `EKSClusterARN`, then pass your EKS cluster ARN as the `TAG_VALUE`. EKS cluster ARNs have the format `arn:aws:eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/CLUSTER_NAME`.

   ```
   {
     "Parameters": {
       "RoleName": "ROLE_NAME",
       "SSMDeregisterConditionTagKey": "TAG_KEY",
       "SSMDeregisterConditionTagValue": "TAG_VALUE"
     }
   }
   ```

3. Deploy the CloudFormation stack. Replace `STACK_NAME` with your name for the CloudFormation stack.

```
aws cloudformation deploy \
    --stack-name STACK_NAME \
    --template-file hybrid-ssm-cfn.yaml \
    --parameter-overrides file://cfn-ssm-parameters.json \
    --capabilities CAPABILITY_NAMED_IAM
```

Steps for AWS IAM Roles Anywhere

The CloudFormation stack creates the AWS IAM Roles Anywhere trust anchor with the certificate authority (CA) you configure, creates the AWS IAM Roles Anywhere profile, and creates the Hybrid Nodes IAM role with the permissions outlined previously.

1. To set up a certificate authority (CA)
   a. To use an AWS Private CA resource, open the AWS Private Certificate Authority console. Follow the instructions in the AWS Private CA User Guide.
   b. To use an external CA, follow the instructions provided by the CA. You provide the certificate body in a later step.
   c. Certificates issued from public CAs cannot be used as trust anchors.
2. Download the AWS IAM Roles Anywhere CloudFormation template for hybrid nodes

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/
hybrid-ira-cfn.yaml'
```

3. Create a `cfn-iamra-parameters.json` with the following options:
   a. Replace `ROLE_NAME` with the name for your Hybrid Nodes IAM role. By default, the CloudFormation template uses `AmazonEKSHybridNodesRole` as the name of the role it creates if you do not specify a name.
   b. Replace `CERT_ATTRIBUTE` with the per-machine certificate attribute that uniquely identifies your host. The certificate attribute you use must match the nodeName you use for the `nodeadm` configuration when you connect hybrid nodes to your cluster. For more information, see the the section called "Hybrid nodes nodeadm". By default, the CloudFormation template uses `${aws:PrincipalTag/x509Subject/CN}` as the `CERT_ATTRIBUTE`, which corresponds to the CN field of your per-machine certificates. You can alternatively pass `$(aws:PrincipalTag/x509SAN/Name/CN}` as your `CERT_ATTRIBUTE`.
   c. Replace `CA_CERT_BODY` with the certificate body of your CA without line breaks. The `CA_CERT_BODY` must be in Privacy Enhanced Mail (PEM) format. If you have a CA certificate in PEM format, remove the line breaks and BEGIN CERTIFICATE and END CERTIFICATE lines before placing the CA certificate body in your `cfn-iamra-parameters.json` file.

```
{
  "Parameters": {
    "RoleName": "ROLE_NAME",
    "CertAttributeTrustPolicy": "CERT_ATTRIBUTE",
```

```
      "CABundleCert": "CA_CERT_BODY"
    }
  }
```

4. Deploy the CloudFormation template. Replace STACK_NAME with your name for the CloudFormation stack.

```
aws cloudformation deploy \
    --stack-name STACK_NAME \
    --template-file hybrid-ira-cfn.yaml \
    --parameter-overrides file://cfn-iamra-parameters.json
    --capabilities CAPABILITY_NAMED_IAM
```

AWS CLI

Install and configure the AWS CLI, if you haven't already. See Installing or updating to the last version of the AWS CLI.

Create EKS Describe Cluster Policy

1. Create a file named `eks-describe-cluster-policy.json` with the following contents:

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

2. Create the policy with the following command:

```
aws iam create-policy \
    --policy-name EKSDescribeClusterPolicy \
    --policy-document file://eks-describe-cluster-policy.json
```

Steps for AWS SSM hybrid activations

1. Create a file named `eks-hybrid-ssm-policy.json` with the following contents. The policy grants permission for two actions `ssm:DescribeInstanceInformation` and `ssm:DeregisterManagedInstance`. The policy restricts the `ssm:DeregisterManagedInstance` permission to AWS SSM managed instances associated with your AWS SSM hybrid activation based on the resource tag you specify in your trust policy.

   a. Replace `AWS_REGION` with the AWS Region for your AWS SSM hybrid activation.

   b. Replace `AWS_ACCOUNT_ID` with your AWS account ID.

   c. Replace `TAG_KEY` with the AWS SSM resource tag key you used when creating your AWS SSM hybrid activation. The combination of the tag key and tag value is used in the condition for the `ssm:DeregisterManagedInstance` to only allow the Hybrid Nodes IAM role to deregister the AWS SSM managed instances that are associated with your AWS SSM hybrid activation. In the CloudFormation template, `TAG_KEY` defaults to `EKSClusterARN`.

   d. Replace `TAG_VALUE` with the AWS SSM resource tag value you used when creating your AWS SSM hybrid activation. The combination of the tag key and tag value is used in the condition for the `ssm:DeregisterManagedInstance` to only allow the Hybrid Nodes IAM role to deregister the AWS SSM managed instances that are associated with your AWS SSM hybrid activation. If you are using the default `TAG_KEY` of `EKSClusterARN`, then pass your EKS cluster ARN as the `TAG_VALUE`. EKS cluster ARNs have the format `arn:aws:eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/CLUSTER_NAME`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ssm:DescribeInstanceInformation",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ssm:DeregisterManagedInstance",
            "Resource": "arn:aws:ssm:AWS_REGION:AWS_ACCOUNT_ID:managed-instance/
*",
            "Condition": {
                "StringEquals": {
                    "ssm:resourceTag/TAG_KEY": "TAG_VALUE"
                }
```

```
            }
        }
    ]
}
```

2. Create the policy with the following command

```
aws iam create-policy \
    --policy-name EKSHybridSSMPolicy \
    --policy-document file://eks-hybrid-ssm-policy.json
```

3. Create a file named `eks-hybrid-ssm-trust.json`. Replace `AWS_REGION` with the AWS Region of your AWS SSM hybrid activation and `AWS_ACCOUNT_ID` with your AWS account ID.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"",
         "Effect":"Allow",
         "Principal":{
            "Service":"ssm.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"AWS_ACCOUNT_ID"
            },
            "ArnEquals":{
               "aws:SourceArn":"arn:aws:ssm:AWS_REGION:AWS_ACCOUNT_ID:*"
            }
         }
      }
   ]
}
```

4. Create the role with the following command.

```
aws iam create-role \
    --role-name AmazonEKSHybridNodesRole \
    --assume-role-policy-document file://eks-hybrid-ssm-trust.json
```

5. Attach the EKSDescribeClusterPolicy and the EKSHybridSSMPolicy you created in the previous steps. Replace AWS_ACCOUNT_ID with your AWS account ID.

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws:iam::AWS_ACCOUNT_ID:policy/EKSDescribeClusterPolicy
```

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws:iam::AWS_ACCOUNT_ID:policy/EKSHybridSSMPolicy
```

6. Attach the AmazonEC2ContainerRegistryPullOnly and AmazonSSMManagedInstanceCore AWS managed policies.

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
```

Steps for AWS IAM Roles Anywhere

To use AWS IAM Roles Anywhere, you must set up your AWS IAM Roles Anywhere trust anchor before creating the Hybrid Nodes IAM Role. See the section called “Setup AWS IAM Roles Anywhere” for instructions.

1. Create a file named eks-hybrid-iamra-trust.json. Replace TRUST_ANCHOR ARN with the ARN of the trust anchor you created in the the section called “Setup AWS IAM Roles Anywhere” steps. The condition in this trust policy restricts the ability of AWS IAM Roles Anywhere to assume the Hybrid Nodes IAM role to exchange temporary IAM credentials only when the role session name matches the CN in the x509 certificate installed on your hybrid nodes. You can alternatively use other certificate attributes to uniquely identify your node. The certificate attribute that you use in the trust policy must correspond to the nodeName you set in your nodeadm configuration. For more information, see the the section called “Hybrid nodes nodeadm”.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": [
                "sts:TagSession",
                "sts:SetSourceIdentity"
            ],
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": "TRUST_ANCHOR_ARN"
                }
            }
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}"
                },
                "ArnEquals": {
                    "aws:SourceArn": "TRUST_ANCHOR_ARN"
                }
            }
        }
    ]
}
```

2. Create the role with the following command.

```
aws iam create-role \
    --role-name AmazonEKSHybridNodesRole \
    --assume-role-policy-document file://eks-hybrid-iamra-trust.json
```

3. Attach the `EKSDescribeClusterPolicy` you created in the previous steps. Replace `AWS_ACCOUNT_ID` with your AWS account ID.

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws:iam::AWS_ACCOUNT_ID:policy/EKSDescribeClusterPolicy
```

4. Attach the `AmazonEC2ContainerRegistryPullOnly` AWS managed policy

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

AWS Management Console

Create EKS Describe Cluster Policy

1. Open the [Amazon IAM console]
2. In the left navigation pane, choose **Policies**.
3. On the **Policies** page, choose **Create policy**.
4. On the Specify permissions page, in the Select a service panel, choose EKS.
   a. Filter actions for **DescribeCluster** and select the **DescribeCluster** Read action.
   b. Choose **Next**.
5. On the **Review and create** page
   a. Enter a **Policy name** for your policy such as `EKSDescribeClusterPolicy`.
   b. Choose **Create policy**.

Steps for AWS SSM hybrid activations

1. Open the [Amazon IAM console]
2. In the left navigation pane, choose **Policies**.
3. On the **Policies page**, choose **Create policy**.
4. On the **Specify permissions** page, in the **Policy editor** top right navigation, choose **JSON**. Paste the following snippet. Replace `AWS_REGION` with the AWS Region of your AWS SSM hybrid activation and replace `AWS_ACCOUNT_ID` with your AWS account ID. Replace `TAG_KEY` and

TAG_VALUE with the AWS SSM resource tag key you used when creating your AWS SSM hybrid activation.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ssm:DescribeInstanceInformation",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ssm:DeregisterManagedInstance",
            "Resource": "arn:aws:ssm:AWS_REGION:AWS_ACCOUNT_ID:managed-instance/*",
            "Condition": {
                "StringEquals": {
                    "ssm:resourceTag/TAG_KEY": "TAG_VALUE"
                }
            }
        }
    ]
}
```

a. Choose **Next**.

5. On the **Review and Create** page.

   a. Enter a **Policy** name for your policy such as `EKSHybridSSMPolicy`

   b. Choose **Create Policy**.

6. In the left navigation pane, choose **Roles**.
7. On the **Roles** page, choose **Create role**.
8. On the **Select trusted entity** page, do the following:

   a. In the **Trusted entity** type section, choose **Custom trust policy**. Paste the following into the Custom trust policy editor. Replace `AWS_REGION` with the AWS Region of your AWS SSM hybrid activation and `AWS_ACCOUNT_ID` with your AWS account ID.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
```

```
            "Sid":"",
            "Effect":"Allow",
            "Principal":{
               "Service":"ssm.amazonaws.com"
            },
            "Action":"sts:AssumeRole",
            "Condition":{
               "StringEquals":{
                  "aws:SourceAccount":"AWS_ACCOUNT_ID"
               },
               "ArnEquals":{
                  "aws:SourceArn":"arn:aws:ssm:AWS_REGION:AWS_ACCOUNT_ID:*"
               }
            }
         }
      ]
}
```

   b. Choose Next.

9. On the **Add permissions** page, attach a custom policy or do the following:

   a. In the **Filter policies** box, enter `EKSDescribeClusterPolicy`, or the name of the policy you created above. Select the check box to the left of your policy name in the search results.

   b. In the **Filter policies** box, enter `EKSHybridSSMPolicy`, or the name of the policy you created above. Select the check box to the left of your policy name in the search results.

   c. In the **Filter policies** box, enter `AmazonEC2ContainerRegistryPullOnly`. Select the check box to the left of `AmazonEC2ContainerRegistryPullOnly` in the search results.

   d. In the **Filter policies** box, enter `AmazonSSMManagedInstanceCore`. Select the check box to the left of `AmazonSSMManagedInstanceCore` in the search results.

   e. Choose **Next**.

10. On the **Name, review, and create** page, do the following:

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

   b. For **Description**, replace the current text with descriptive text such as `Amazon EKS - Hybrid Nodes role`.

   c. Choose **Create role**.

Steps for AWS IAM Roles Anywhere

To use AWS IAM Roles Anywhere, you must set up your AWS IAM Roles Anywhere trust anchor before creating the Hybrid Nodes IAM Role. See the section called "Setup AWS IAM Roles Anywhere" for instructions.

1. Open the Amazon IAM console
2. In the left navigation pane, choose **Roles**.
3. On the **Roles** page, choose **Create role**.
4. On the **Select trusted entity** page, do the following:
   a. In the **Trusted entity type section**, choose **Custom trust policy**. Paste the following into the Custom trust policy editor. Replace `TRUST_ANCHOR ARN` with the ARN of the trust anchor you created in the the section called "Setup AWS IAM Roles Anywhere" steps. The condition in this trust policy restricts the ability of AWS IAM Roles Anywhere to assume the Hybrid Nodes IAM role to exchange temporary IAM credentials only when the role session name matches the CN in the x509 certificate installed on your hybrid nodes. You can alternatively use other certificate attributes to uniquely identify your node. The certificate attribute that you use in the trust policy must correspond to the nodeName you set in your nodeadm configuration. For more information, see the the section called "Hybrid nodes nodeadm".

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": [
                "sts:TagSession",
                "sts:SetSourceIdentity"
            ],
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": "TRUST_ANCHOR_ARN"
                }
            }
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
```

```
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}"
                },
                "ArnEquals": {
                    "aws:SourceArn": "TRUST_ANCHOR_ARN"
                }
            }
        }
    ]
}
```

b. Choose Next.

5. On the **Add permissions** page, attach a custom policy or do the following:

   a. In the **Filter policies** box, enter `EKSDescribeClusterPolicy`, or the name of the policy you created above. Select the check box to the left of your policy name in the search results.

   b. In the **Filter policies** box, enter `AmazonEC2ContainerRegistryPullOnly`. Select the check box to the left of `AmazonEC2ContainerRegistryPullOnly` in the search results.

   c. Choose **Next**.

6. On the **Name, review, and create** page, do the following:

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

   b. For **Description**, replace the current text with descriptive text such as `Amazon EKS - Hybrid Nodes role`.

   c. Choose **Create role**.

Create an Amazon EKS cluster with hybrid nodes

This topic provides an overview of the available options and describes what to consider when you create a hybrid nodes-enabled Amazon EKS cluster. If you are not planning to use hybrid nodes, see the section called “Create a cluster”.

Prerequisites

- The the section called “Prerequisites” completed. Before you create your hybrid nodes-enabled cluster, you must have your on-premises node and optionally pod CIDRs identified, your VPC and subnets created according to the EKS requirements, and hybrid nodes requirements, and your security group with inbound rules for your on-premises and optionally pod CIDRs. For more information on these prerequisites, see the section called "Prepare networking".

- The latest version of the AWS Command Line Interface (AWS CLI) installed and configured on your device. To check your current version, use `aws --version`. Package managers such yum, apt-get, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing or updating to the last version of the AWS CLI and Configuring settings for the AWS CLI in the AWS Command Line Interface User Guide.
- An IAM principal with permissions to create IAM roles and attach policies, and create and describe EKS clusters

Considerations

- Your cluster must use either `API` or `API_AND_CONFIG_MAP` for the cluster authentication mode.
- Your cluster must use IPv4 address family.
- Your cluster must use either Public or Private cluster endpoint connectivity. Your cluster cannot use "Public and Private" cluster endpoint connectivity, because the Amazon EKS Kubernetes API server endpoint will resolve to the public IPs for hybrid nodes running outside of your VPC.
- Currently, hybrid nodes must be enabled during cluster creation. You cannot change your `RemoteNodeNetwork` or `RemotePodNetwork` after cluster creation.

Step 1: Create cluster IAM role

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

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

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

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

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

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

3. You can assign either the Amazon EKS managed policy or create your own custom policy. For the minimum permissions that you must use in your custom policy, see the section called "Node IAM role". Attach the Amazon EKS managed policy named `AmazonEKSClusterPolicy` to the role. To attach an IAM policy to an IAM principal, the principal that is attaching the policy must be assigned one of the following IAM actions (permissions): `iam:AttachUserPolicy` or `iam:AttachRolePolicy`.

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

Step 2: Create hybrid nodes-enabled cluster

You can create a cluster by using:

- eksctl
- AWS CloudFormation
- AWS CLI
- AWS Management Console

Create hybrid nodes-enabled cluster - eksctl

You need to install the latest version of the `eksctl` command line tool. To install or update `eksctl`, see Installation in the `eksctl` documentation.

1. Create `cluster-config.yaml` to define a hybrid nodes-enabled Amazon EKS IPv4 cluster. Make the following replacements in your `cluster-config.yaml`. For a full list of settings, see the eksctl documentation.

   a. Replace `CLUSTER_NAME` with a name for your cluster. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.

   b. Replace `AWS_REGION` with the AWS Region that you want to create your cluster in.

   c. Replace `K8S_VERSION` with any Amazon EKS supported version.

   d. Replace `CREDS_PROVIDER` with `ssm` or `ira` based on the credential provider you configured in the steps for the section called "Prepare credentials".

   e. Replace `CA_BUNDLE_CERT` if your credential provider is set to `ira`, which uses AWS IAM Roles Anywhere as the credential provider. The CA_BUNDLE_CERT is the certificate authority (CA) certificate body and depends on your choice of CA. The certificate must be in Privacy Enhanced Mail (PEM) format.

   f. Replace `GATEWAY_ID` with the ID of your virtual private gateway or transit gateway to be attached to your VPC.

   g. Replace `REMOTE_NODE_CIDRS` with the on-premises node CIDR for your hybrid nodes.

   h. Replace `REMOTE_POD_CIDRS` with the on-premises pod CIDR for workloads running on hybrid nodes or remove the line from your configuration if you are not running webhooks on hybrid nodes. You must configure your `REMOTE_POD_CIDRS` if your CNI does not use Network Address Translation (NAT) or masquerading for pod IP addresses when pod traffic leaves your on-premises hosts. You must configure `REMOTE_POD_CIDRS` if you are running webhooks on hybrid nodes.

   i. Your on-premises node and pod CIDR blocks must meet the following requirements:

      i. Be within one of the IPv4 RFC-1918 ranges: `10.0.0.0/8`, `172.16.0.0/12`, or `192.168.0.0/16`.

      ii. Not overlap with each other, the `VPC CIDR` for your cluster, or your Kubernetes service IPv4 CIDR

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

metadata:
  name: CLUSTER_NAME
  region: AWS_REGION
  version: "K8S_VERSION"

remoteNetworkConfig:
  iam:
    provider: CREDS_PROVIDER # default SSM, can also be set to IRA
    # caBundleCert: CA_BUNDLE_CERT
  vpcGatewayID: GATEWAY_ID
  remoteNodeNetworks:
  - cidrs: ["REMOTE_NODE_CIDRS"]
  remotePodNetworks:
  - cidrs: ["REMOTE_POD_CIDRS"]
```

2. Run the following command:

```
eksctl create cluster -f cluster-config.yaml
```

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

```
[#]  EKS cluster "CLUSTER_NAME" in "REGION" region is ready
```

3. Continue with the section called “Step 3: Update kubeconfig”.

Create hybrid nodes-enabled cluster - AWS CloudFormation

The CloudFormation stack creates the EKS cluster IAM role and an EKS cluster with the `RemoteNodeNetwork` and `RemotePodNetwork` you specify. Modify the CloudFormation template If you need to customize settings for your EKS cluster that are not exposed in the CloudFormation template.

1. Download the CloudFormation template.

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/
hybrid-eks-cfn.yaml'
```

2. Create a `cfn-eks-parameters.json` and specify your configuration for each value.

   a. `CLUSTER_NAME`: name of the EKS cluster to be created

   b. `CLUSTER_ROLE_NAME`: name of the EKS cluster IAM role to be created. The default in the template is “EKSClusterRole”.

   c. `SUBNET1_ID`: the ID of the first subnet you created in the prerequisite steps

   d. `SUBNET2_ID`: the ID of the second subnet you created in the prerequisite steps

   e. `SG_ID`: the security group ID you created in the prerequisite steps

   f. `REMOTE_NODE_CIDRS`: the on-premises node CIDR for your hybrid nodes

   g. `REMOTE_POD_CIDRS`: the on-premises pod CIDR for workloads running on hybrid nodes. You must configure your `REMOTE_POD_CIDRS` if your CNI does not use Network Address Translation (NAT) or masquerading for pod IP addresses when pod traffic leaves your on-premises hosts. You must configure `REMOTE_POD_CIDRS` if you are running webhooks on hybrid nodes.

   h. Your on-premises node and pod CIDR blocks must meet the following requirements:

      i. Be within one of the IPv4 RFC-1918 ranges: `10.0.0.0/8`, `172.16.0.0/12`, or `192.168.0.0/16`.

      ii. Not overlap with each other, the `VPC CIDR` for your cluster, or your Kubernetes service IPv4 CIDR.

   i. `CLUSTER_AUTH`: the cluster authentication mode for your cluster. Valid values are `API` and `API_AND_CONFIG_MAP`. The default in the template is `API_AND_CONFIG_MAP`.

   j. `CLUSTER_ENDPOINT`: the cluster endpoint connectivity for your cluster. Valid values are “Public” and “Private”. The default in the template is Private, which means you will only be able to connect to the Kubernetes API endpoint from within your VPC.

   k. `K8S_VERSION`: the Kubernetes version to use for your cluster. See the section called “Kubernetes versions”.

```
{
  "Parameters": {
    "ClusterName": "CLUSTER_NAME",
    "ClusterRoleName": "CLUSTER_ROLE_NAME",
    "SubnetId1": "SUBNET1_ID",
```

```
    "SubnetId2": "SUBNET2_ID",
    "SecurityGroupId" "SG_ID",
    "RemoteNodeCIDR": "REMOTE_NODE_CIDRS",
    "RemotePodCIDR": "REMOTE_POD_CIDRS",
    "ClusterAuthMode": "CLUSTER_AUTH",
    "ClusterEndpointConnectivity": "CLUSTER_ENDPOINT",
    "K8sVersion": "K8S_VERSION"
  }
}
```

3. Deploy the CloudFormation stack. Replace STACK_NAME with your name for the CloudFormation stack and AWS_REGION with your desired AWS Region where the cluster will be created.

```
aws cloudformation deploy \
    --stack-name STACK_NAME \
    --region AWS_REGION \
    --template-file hybrid-eks-cfn.yaml \
    --parameter-overrides file://cfn-eks-parameters.json \
    --capabilities CAPABILITY_NAMED_IAM
```

   Cluster provisioning takes several minutes. You can check the status of your stack with the following command. Replace STACK_NAME with your name for the CloudFormation stack and AWS_REGION with your desired AWS Region where the cluster will be created.

```
aws cloudformation describe-stacks \
    --stack-name STACK_NAME \
    --region AWS_REGION \
    --query 'Stacks[].StackStatus'
```

4. Continue with the section called "Step 3: Update kubeconfig".

Create hybrid nodes-enabled cluster - AWS CLI

1. Run the following command to create a hybrid nodes-enabled EKS cluster. Before running the command, replace the following with your desired settings. For a full list of settings, see the the section called "Create a cluster" documentation.

   a. CLUSTER_NAME: name of the EKS cluster to be created

   b. AWS_REGION: AWS Region where the cluster will be created.

c. K8S_VERSION: the Kubernetes version to use for your cluster. See Amazon EKS supported versions.

d. ROLE_ARN: the Amazon EKS cluster role you configured for your cluster. See Amazon EKS cluster IAM role for more information.

e. SUBNET1_ID: the ID of the first subnet you created in the prerequisite steps

f. SUBNET2_ID: the ID of the second subnet you created in the prerequisite steps

g. SG_ID: the security group ID you created in the prerequisite steps

h. You can use API and API_AND_CONFIG_MAP for your cluster access authentication mode. In the command below, the cluster access authentication mode is set to API_AND_CONFIG_MAP.

i. You can use the endpointPublicAccess and endpointPrivateAccess parameters to enable or disable public and private access to your cluster's Kubernetes API server endpoint. In the command below endpointPublicAccess is set to false and endpointPrivateAccess is set to true.

j. REMOTE_NODE_CIDRS: the on-premises node CIDR for your hybrid nodes.

k. REMOTE_POD_CIDRS (optional): the on-premises pod CIDR for workloads running on hybrid nodes.

l. Your on-premises node and pod CIDR blocks must meet the following requirements:

   i. Be within one of the IPv4 RFC-1918 ranges: 10.0.0.0/8, 172.16.0.0/12, or 192.168.0.0/16.

   ii. Not overlap with each other, the VPC CIDR for your Amazon EKS cluster, or your Kubernetes service IPv4 CIDR.

```
aws eks create-cluster \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --kubernetes-version K8S_VERSION \
    --role-arn ROLE_ARN \
    --resources-vpc-config
 subnetIds=SUBNET1_ID,SUBNET2_ID,securityGroupIds=SG_ID,endpointPrivateAccess=true,endpo
 \
    --access-config authenticationMode=API_AND_CONFIG_MAP \
    --remote-network-config '{"remoteNodeNetworks":[{"cidrs":
["REMOTE_NODE_CIDRS"]}],"remotePodNetworks":[{"cidrs":["REMOTE_POD_CIDRS"]}]}'
```

2. It takes several minutes to provision the cluster. You can query the status of your cluster with the following command. Replace CLUSTER_NAME with the name of the cluster you are creating and

AWS_REGION with the AWS Region where the cluster is creating. Don't proceed to the next step until the output returned is ACTIVE.

```
aws eks describe-cluster \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --query "cluster.status"
```

3. Continue with the section called "Step 3: Update kubeconfig".

Create hybrid nodes-enabled cluster - AWS Management Console

1. Open the Amazon EKS console at Amazon EKS console.
2. Choose Add cluster and then choose Create.
3. On the Configure cluster page, enter the following fields:
   a. **Name** – A name for your cluster. The name can contain only alphanumeric characters (case-sensitive), hyphens, and underscores. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.
   b. **Cluster IAM role** – Choose the Amazon EKS cluster IAM role that you created to allow the Kubernetes control plane to manage AWS resources on your behalf.
   c. **Kubernetes version** – The version of Kubernetes to use for your cluster. We recommend selecting the latest version, unless you need an earlier version.
   d. **Upgrade policy** - Choose either Extended or Standard.
      i. **Extended:** This option supports the Kubernetes version for 26 months after the release date. The extended support period has an additional hourly cost that begins after the standard support period ends. When extended support ends, your cluster will be auto upgraded to the next version.
      ii. **Standard:** This option supports the Kubernetes version for 14 months after the release date. There is no additional cost. When standard support ends, your cluster will be auto upgraded to the next version.
   e. **Cluster access** - choose to allow or disallow cluster administrator access and select an authentication mode. The following authentication modes are supported for hybrid nodes-enabled clusters.

   i. **EKS API**: The cluster will source authenticated IAM principals only from EKS access entry APIs.

   ii. **EKS API and ConfigMap**: The cluster will source authenticated IAM principals from both EKS access entry APIs and the `aws-auth` ConfigMap.

f. **Secrets encryption** – (Optional) Choose to enable secrets encryption of Kubernetes secrets using a KMS key. You can also enable this after you create your cluster. Before you enable this capability, make sure that you're familiar with the information in the section called "Enable secret encryption".

g. **ARC Zonal Shift** - If enabled, EKS will register your cluster with ARC zonal shift to enable you to use zonal shift to shift application traffic away from an AZ.

h. **Tags** – (Optional) Add any tags to your cluster. For more information, see the section called "Tagging your resources".

i. When you're done with this page, choose **Next**.

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

   a. **VPC** – Choose an existing VPC that meets the section called "VPC and subnet requirements" and Amazon EKS Hybrid Nodes requirements. Before choosing a VPC, we recommend that you're familiar with all of the requirements and considerations in View Amazon EKS networking requirements for VPC, subnets, and hybrid nodes. You can't change which VPC you want to use after cluster creation. If no VPCs are listed, then you need to create one first. For more information, see the section called "Create a VPC" and the Amazon EKS Hybrid Nodes networking requirements.

   b. **Subnets** – By default, all available subnets in the VPC specified in the previous field are preselected. You must select at least two.

   c. **Security groups** – (Optional) Specify one or more security groups that you want Amazon EKS to associate to the network interfaces that it creates. At least one of the security groups you specify must have inbound rules for your on-premises node and optionally pod CIDRs. See the Amazon EKS Hybrid Nodes networking requirements for more information. Whether you choose any security groups or not, Amazon EKS creates a security group that enables communication between your cluster and your VPC. Amazon EKS associates this security group, and any that you choose, to the network interfaces that it creates. For more information about the cluster security group that Amazon EKS creates, see the section called "Security group requirements" You can modify the rules in the cluster security group that Amazon EKS creates.

   d. **Choose cluster IP address family** – You must choose IPv4 for hybrid nodes-enabled clusters.

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

f. **Choose Configure remote networks to enable hybrid nodes** and specify your on-premises node and pod CIDRs for hybrid nodes.

g. You must configure your remote pod CIDR if your CNI does not use Network Address Translation (NAT) or masquerading for pod IP addresses when pod traffic leaves your on-premises hosts. You must configure the remote pod CIDR if you are running webhooks on hybrid nodes.

h. Your on-premises node and pod CIDR blocks must meet the following requirements:

   i. Be within one of the IPv4 RFC-1918 ranges: `10.0.0.0/8`, `172.16.0.0/12`, or `192.168.0.0/16`.

   ii. Not overlap with each other, the `VPC CIDR` for your cluster, or your Kubernetes service IPv4 CIDR

i. For **Cluster endpoint access**, select an option. After your cluster is created, you can change this option. For hybrid nodes-enabled clusters, you must choose either Public or Private. Before selecting a non-default option, make sure to familiarize yourself with the options and their implications. For more information, see the section called “Configure endpoint access”.

j. When you’re done with this page, choose **Next**.

5. (Optional) On the **Configure** observability page, choose which Metrics and Control plane logging options to turn on. By default, each log type is turned off.

   a. For more information about the Prometheus metrics option, see the section called “Prometheus metrics”.

   b. For more information about the EKS control logging options, see the section called “Control plane logs”.

   c. When you’re done with this page, choose **Next**.

6. On the **Select add-ons** page, choose the add-ons that you want to add to your cluster.

   a. You can choose as many **Amazon EKS add-ons** and **AWS Marketplace add-ons** as you require. Amazon EKS add-ons that are not compatible with hybrid nodes are marked with “Not compatible with Hybrid Nodes” and the add-ons have an anti-affinity rule to prevent them from running on hybrid nodes. See Configuring add-ons for hybrid nodes for more information. If the **AWS Marketplace** add-ons that you want to install isn’t listed, you can search for available **AWS Marketplace add-ons** by entering text in the search box. You can also search by **category**, **vendor**, or **pricing model** and then choose the add-ons from the search results.

b. Some add-ons, such as CoreDNS and kube-proxy, are installed by default. If you disable any of the default add-ons, this may affect your ability to run Kubernetes applications.

c. When you're done with this page, choose `Next`.

7. On the **Configure selected add-ons settings** page, select the version that you want to install.

   a. You can always update to a later version after cluster creation. You can update the configuration of each add-on after cluster creation. For more information about configuring add-ons, see the section called "Update an add-on". For the add-ons versions that are compatible with hybrid nodes, see the section called "Configure add-ons".

   b. When you're done with this page, choose Next.

8. On the **Review and create** page, review the information that you entered or selected on the previous pages. If you need to make changes, choose **Edit**. When you're satisfied, choose **Create**. The **Status** field shows **CREATING** while the cluster is provisioned. Cluster provisioning takes several minutes.

9. Continue with the section called "Step 3: Update kubeconfig".

Step 3: Update kubeconfig

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

```
aws eks update-kubeconfig --name CLUSTER_NAME --region AWS_REGION
```

An example output is as follows.

```
Added new context arn:aws:eks:AWS_REGION:111122223333:cluster/CLUSTER_NAME to /home/
username/.kube/config
```

Confirm communication with your cluster by running the following command.

```
kubectl get svc
```

An example output is as follows.

```
NAME          TYPE         CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
```

```
kubernetes    ClusterIP    10.100.0.1    <none>         443/TCP    28h
```

Step 4: Cluster setup

As a next step, see the section called "Prepare cluster access" to enable access for your hybrid nodes to join your cluster.

Prepare cluster access for hybrid nodes

Before connecting hybrid nodes to your Amazon EKS cluster, you must enable your Hybrid Nodes IAM Role with Kubernetes permissions to join the cluster. See the section called "Prepare credentials" for information on how to create the Hybrid Nodes IAM role. Amazon EKS supports two ways to associate IAM principals with Kubernetes Role-Based Access Control (RBAC), Amazon EKS access entries and the `aws-auth` ConfigMap. For more information on Amazon EKS access management, see the section called "Kubernetes API access".

Use the procedures below to associate your Hybrid Nodes IAM role with Kubernetes permissions. To use Amazon EKS access entries, your cluster must have been created with the `API` or `API_AND_CONFIG_MAP` authentication modes. To use the `aws-auth` ConfigMap, your cluster must have been created with the `API_AND_CONFIG_MAP` authentication mode. The `CONFIG_MAP`-only authentication mode is not supported for hybrid nodes-enabled Amazon EKS clusters.

Using Amazon EKS access entries for Hybrid Nodes IAM role

There is an Amazon EKS access entry type for hybrid nodes named HYBRID_LINUX that can be used with an IAM role. With this access entry type, the username is automatically set to system:node:{{SessionName}}. For more information on creating access entries, see the section called "Create access entries".

AWS CLI

1. You must have the latest version of the AWS CLI installed and configured on your device. To check your current version, use `aws --version`. Package managers such yum, apt-get, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the AWS Command Line Interface User Guide.
2. Create your access entry with the following command. Replace CLUSTER_NAME with the name of your cluster and HYBRID_NODES_ROLE_ARN with the ARN of the role you created in the steps for the section called "Prepare credentials".

```
aws eks create-access-entry --cluster-name CLUSTER_NAME \
    --principal-arn HYBRID_NODES_ROLE_ARN \
    --type HYBRID_LINUX
```

AWS Management Console

1. Open the Amazon EKS console at Amazon EKS console.
2. Choose the name of your hybrid nodes-enabled cluster.
3. Choose the **Access** tab.
4. Choose **Create access entry**.
5. For **IAM principal**, select the Hybrid Nodes IAM role you created in the steps for the section called "Prepare credentials".
6. For **Type**, select **Hybrid Linux**.
7. (Optional) For **Tags**, assign labels to the access entry. For example, to make it easier to find all resources with the same tag.
8. Choose **Skip to review and create**. You cannot add policies to the Hybrid Linux access entry or change its access scope.
9. Review the configuration for your access entry. If anything looks incorrect, choose **Previous** to go back through the steps and correct the error. If the configuration is correct, choose **Create**.

Using aws-auth ConfigMap for Hybrid Nodes IAM role

In the following steps, you will create or update the `aws-auth` ConfigMap with the ARN of the Hybrid Nodes IAM Role you created in the steps for the section called "Prepare credentials".

1. Check to see if you have an existing `aws-auth` ConfigMap for your cluster. Note that if you are using a specific `kubeconfig` file, use the `--kubeconfig` flag.

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

2. If you are shown an `aws-auth` ConfigMap, then update it as needed.

   a. Open the ConfigMap for editing.

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

b. Add a new `mapRoles` entry as needed. Replace `HYBRID_NODES_ROLE_ARN` with the ARN of your Hybrid Nodes IAM role. Note, `{{SessionName}}` is the correct template format to save in the ConfigMap. Do not replace it with other values.

```
data:
  mapRoles: |
  - groups:
    - system:bootstrappers
    - system:nodes
    rolearn: HYBRID_NODES_ROLE_ARN
    username: system:node:{{SessionName}}
```

c. Save the file and exit your text editor.

3. If there is not an existing `aws-auth` ConfigMap for your cluster, create it with the following command. Replace `HYBRID_NODES_ROLE_ARN` with the ARN of your Hybrid Nodes IAM role. Note that `{{SessionName}}` is the correct template format to save in the ConfigMap. Do not replace it with other values.

```
kubectl apply -f=/dev/stdin <<-EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
  - groups:
    - system:bootstrappers
    - system:nodes
    rolearn: HYBRID_NODES_ROLE_ARN
    username: system:node:{{SessionName}}
EOF
```

Run on-premises workloads on hybrid nodes

In an EKS cluster with hybrid nodes enabled, you can run on-premises and edge applications on your own infrastructure with the same Amazon EKS clusters, features, and tools that you use in AWS Cloud.

The following sections contain step-by-step instructions for using hybrid nodes.

Topics

- Connect hybrid nodes
- Upgrade hybrid nodes for your cluster
- Remove hybrid nodes

Connect hybrid nodes

This topic describes how to connect hybrid nodes to an Amazon EKS cluster. After your hybrid nodes join the cluster, they will appear with status Not Ready in the Amazon EKS console and in Kubernetes-compatible tooling such as kubectl. After completing the steps on this page, proceed to the section called "Configure CNI" to make your hybrid nodes ready to run applications.

Prerequisites

Before connecting hybrid nodes to your Amazon EKS cluster, make sure you have completed the prerequisite steps.

- You have network connectivity from your on-premises environment to the AWS Region hosting your Amazon EKS cluster. See the section called "Prepare networking" for more information.
- You have a compatible operating system for hybrid nodes installed on your on-premises hosts. See the section called "Prepare operating system" for more information.
- You have created your Hybrid Nodes IAM role and set up your on-premises credential provider (AWS Systems Manager hybrid activations or AWS IAM Roles Anywhere). See the section called "Prepare credentials" for more information.
- You have created your hybrid nodes-enabled Amazon EKS cluster. See the section called "Create cluster" for more information.
- You have associated your Hybrid Nodes IAM role with Kubernetes Role-Based Access Control (RBAC) permissions. See the section called "Prepare cluster access" for more information.

Step 1: Install the hybrid nodes CLI (`nodeadm`) on each on-premises host

If you are including the Amazon EKS Hybrid Nodes CLI (`nodeadm`) in your pre-built operating system images, you can skip this step. For more information on the hybrid nodes version of `nodeadm`, see the section called "Hybrid nodes nodeadm".

The hybrid nodes version of nodeadm is hosted in Amazon S3 fronted by Amazon CloudFront. To install nodeadm on each on-premises host, you can run the following command from your on-premises hosts.

For x86_64 hosts:

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/amd64/
nodeadm'
```

For ARM hosts

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/arm64/
nodeadm'
```

Add executable file permission to the downloaded binary on each host.

```
chmod +x nodeadm
```

Step 2: Install the hybrid nodes dependencies with nodeadm

If you are installing the hybrid nodes dependencies in pre-built operating system images, you can skip this step. The `nodeadm install` command can be used to install all dependencies required for hybrid nodes. The hybrid nodes dependencies include containerd, kubelet, kubectl, and AWS SSM or AWS IAM Roles Anywhere components. See the section called "Hybrid nodes nodeadm" for more information on the components and file locations installed by `nodeadm install`. See the section called "Prepare networking" for hybrid nodes for more information on the domains that must be allowed in your on-premises firewall for the `nodeadm install` process.

Run the command below to install the hybrid nodes dependencies on your on-premises host. The command below must be run with a user that has sudo/root access on your host.

> **Important**
>
> The hybrid nodes CLI (nodeadm) must be run with a user that has sudo/root access on your host.

- Replace `K8S_VERSION` with the Kubernetes minor version of your Amazon EKS cluster, for example `1.31`. See the section called "Kubernetes versions" for a list of the supported Kubernetes versions.

- Replace CREDS_PROVIDER with the on-premises credential provider you are using. Valid values are ssm for AWS SSM and iam-ra for AWS IAM Roles Anywhere.

```
nodeadm install K8S_VERSION --credential-provider CREDS_PROVIDER
```

Step 3: Connect hybrid nodes to your cluster

Before connecting your hybrid nodes to your cluster, make sure you have allowed the required access in your on-premises firewall and in the security group for your cluster for the Amazon EKS control plane to/from hybrid node communication. Most issues at this step are related to the firewall configuration, security group configuration, or Hybrid Nodes IAM role configuration.

> **Important**
>
> The hybrid nodes CLI (nodeadm) must be run with a user that has sudo/root access on your host.

1. Create a nodeConfig.yaml file on each host with the values for your deployment. For a full description of the available configuration settings, see the section called “Hybrid nodes nodeadm”. If your Hybrid Nodes IAM role does not have permission for the eks:DescribeCluster action, you must pass your Kubernetes API endpoint, cluster CA bundle, and Kubernetes service IPv4 CIDR in the cluster section of your nodeConfig.yaml.
   a. Use the nodeConfig.yaml example below if you are using AWS SSM hybrid activations for your on-premises credentials provider.
      i. Replace CLUSTER_NAME with the name of your cluster.
      ii. Replace AWS_REGION with the AWS Region hosting your cluster. For example, us-west-2.
      iii. Replace ACTIVATION_CODE with the activation code you received when creating your AWS SSM hybrid activation. See the section called “Prepare credentials” for more information.
      iv. Replace ACTIVATION_ID with the activation ID you received when creating your AWS SSM hybrid activation. You can retrieve this information from the AWS Systems Manager console or from the AWS CLI aws ssm describe-activations command.

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
```

```
spec:
  cluster:
    name: CLUSTER_NAME
    region: AWS_REGION
  hybrid:
    ssm:
      activationCode: ACTIVATION_CODE
      activationId: ACTIVATION_ID
```

b. Use the `nodeConfig.yaml` example below if you are using AWS IAM Roles Anywhere for your on-premises credentials provider.

   i. Replace `CLUSTER_NAME` with the name of your cluster.

   ii. Replace `AWS_REGION` with the AWS Region hosting your cluster. For example, `us-west-2`.

   iii. Replace `NODE_NAME` with the name of your node. The node name must match the CN of the certificate on the host if you configured the trust policy of your Hybrid Nodes IAM role with the `"sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}"` resource condition. The `nodeName` you use must not be longer than 64 characters.

   iv. Replace `TRUST_ANCHOR_ARN` with the ARN of the trust anchor you configured in the steps for Prepare credentials for hybrid nodes.

   v. Replace `PROFILE_ARN` with the ARN of the trust anchor you configured in the steps for the section called “Prepare credentials”.

   vi. Replace `ROLE_ARN` with the ARN of your Hybrid Nodes IAM role.

   vii. Replace `CERTIFICATE_PATH` with the path in disk to your node certificate. If you don’t specify it, the default is `/etc/iam/pki/server.pem`.

   viii. Replace `KEY_PATH` with the path in disk to your certificate private key. If you don’t specify it, the default is `/etc/iam/pki/server.key`.

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: CLUSTER_NAME
    region: AWS_REGION
  hybrid:
    iamRolesAnywhere:
      nodeName: NODE_NAME
      trustAnchorArn: TRUST_ANCHOR_ARN
      profileArn: PROFILE_ARN
```

```
        roleArn: ROLE_ARN
        certificatePath: CERTIFICATE_PATH
        privateKeyPath: KEY_PATH
```

2. Run the `nodeadm init` command with your `nodeConfig.yaml` to connect your hybrid nodes to your Amazon EKS cluster.

```
nodeadm init -c file://nodeConfig.yaml
```

If the above command completes successfully, your hybrid node has joined your Amazon EKS cluster. You can verify this in the Amazon EKS console by navigating to the Compute tab for your cluster (ensure IAM principal has permissions to view) or with `kubectl get nodes`.

> **Important**
>
> Your nodes will have status `Not Ready`, which is expected and is due to the lack of a CNI running on your hybrid nodes. If your nodes did not join the cluster, see the section called "Troubleshooting".

Step 4: Configure a CNI for hybrid nodes

To make your hybrid nodes ready to run applications, continue with the steps on the section called "Configure CNI".

Upgrade hybrid nodes for your cluster

The guidance for upgrading hybrid nodes is similar to self-managed Amazon EKS nodes that run in Amazon EC2. It is recommended to can create new hybrid nodes on your target Kubernetes version, gracefully migrate your existing applications to the hybrid nodes on the new Kubernetes version, and remove the hybrid nodes on the old Kubernetes version from your cluster. Be sure to review the Amazon EKS Best Practices for upgrades before initiating an upgrade. Amazon EKS Hybrid Nodes have the same Kubernetes version support for Amazon EKS clusters with cloud nodes, including standard and extended support.

Amazon EKS Hybrid Nodes follow the same version skew policy for nodes as upstream Kubernetes. Amazon EKS Hybrid Nodes cannot be on a newer version than the Amazon EKS control plane, and hybrid nodes may be up to three Kubernetes minor versions older than the Amazon EKS control plane minor version.

If you do not have spare capacity to create new hybrid nodes on your target Kubernetes version for a cutover migration upgrade strategy, you can alternatively use the Amazon EKS Hybrid Nodes CLI (`nodeadm`) to upgrade the Kubernetes version of your hybrid nodes in-place.

> **Important**
>
> If you are upgrading your hybrid nodes in-place with `nodeadm`, there is downtime for the node during the process where the older version of the Kubernetes components are shut down and the new Kubernetes version components are installed and started.

Prerequisites

Before upgrading, make sure you have completed the following prerequisites.

- The target Kubernetes version for your hybrid nodes upgrade must be equal to or less than the Amazon EKS control plane version.
- If you are following a cutover migration upgrade strategy, the new hybrid nodes you are installing on your target Kubernetes version must meet the the section called "Prerequisites" requirements. This includes having IP addresses within the Remote Node Network CIDR you passed during Amazon EKS cluster creation.
- For both cutover migration and in-place upgrades, the hybrid nodes must have access to the required domains to pull the new versions of the hybrid nodes dependencies.
- You must have kubectl installed on your local machine or instance you are using to interact with your Amazon EKS Kubernetes API endpoint.
- The version of your CNI must support the Kubernetes version you are upgrading to. If it does not, upgrade your CNI version before upgrading your hybrid nodes. See the section called "Configure CNI" for more information.

Cutover migration upgrades

Cutover migration upgrades refer to the process of creating new hybrid nodes on new hosts with your target Kubernetes version, gracefully migrating your existing applications to the new hybrid nodes on your target Kubernetes version, and removing the hybrid nodes on the old Kubernetes version from your cluster.

1. Connect your new hosts as hybrid nodes following the the section called "Connect hybrid nodes" steps. When running the `nodeadm install` command, use your target Kubernetes version.
2. Enable communication between the new hybrid nodes on the target Kubernetes version and your hybrid nodes on the old Kubernetes version. This configuration allows pods to communicate with each other while you are migrating your workload to the hybrid nodes on the target Kubernetes version.
3. Confirm your hybrid nodes on your target Kubernetes version successfully joined your cluster and have status Ready.
4. Use the following command to taint each of the nodes that you want to remove with `NoSchedule`. This is so that new pods aren't scheduled or rescheduled on the nodes that you are replacing. For more information, see Taints and Tolerations in the Kubernetes documentation. Replace `NODE_NAME` with the name of the hybrid nodes on the old Kubernetes version.

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

   You can identify and taint all of the nodes of a particular Kubernetes version (in this case, `1.28`) with the following code snippet.

```
K8S_VERSION=1.28
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
```

5. If your current deployment is running fewer than two CoreDNS replicas on your hybrid nodes, scale out the deployment to at least two replicas. It is recommended to run at least two CoreDNS replicas on hybrid nodes for resiliency during normal operations.

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

6. Drain each of the hybrid nodes on the old Kubernetes version that you want to remove from your cluster with the following command. For more information on draining nodes, see Safely Drain a Node in the Kubernetes documentation. Replace `NODE_NAME` with the name of the hybrid nodes on the old Kubernetes version.

```
kubectl drain NODE_NAME --ignore-daemonsets --delete-emptydir-data
```

You can identify and drain all of the nodes of a particular Kubernetes version (in this case, `1.28`) with the following code snippet.

```
K8S_VERSION=1.28
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-emptydir-data
done
```

7. You can use `nodeadm` to stop and remove the hybrid nodes artifacts from the host. You must run `nodeadm` with a user that has root/sudo privileges. By default, `nodeadm uninstall` will not proceed if there are pods remaining on the node. For more information see the section called “Hybrid nodes nodeadm”.

```
nodeadm uninstall
```

8. With the hybrid nodes artifacts stopped and uninstalled, remove the node resource from your cluster.

```
kubectl delete node node-name
```

You can identify and delete all of the nodes of a particular Kubernetes version (in this case, `1.28`) with the following code snippet.

```
K8S_VERSION=1.28
nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==
\"v$K8S_VERSION\")].metadata.name}")
for node in ${nodes[@]}
do
    echo "Deleting $node"
    kubectl delete node $node
done
```

9. Depending on your choice of CNI, there may be artifacts remaining on your hybrid nodes after running the above steps. See the section called "Configure CNI" for more information.

In-place upgrades

The in-place upgrade process refers to using `nodeadm upgrade` to upgrade the Kubernetes version for hybrid nodes without using new physical or virtual hosts and a cutover migration strategy. The `nodeadm upgrade` process shuts down the existing older Kubernetes components running on the hybrid node, uninstalls the existing older Kubernetes components, installs the new target Kubernetes components, and starts the new target Kubernetes components. It is strongly recommend to upgrade one node at a time to minimize impact to applications running on the hybrid nodes. The duration of this process depends on your network bandwidth and latency.

1. Use the following command to taint the node you are upgrading with `NoSchedule`. This is so that new pods aren't scheduled or rescheduled on the node that you are upgrading. For more information, see Taints and Tolerations in the Kubernetes documentation. Replace `NODE_NAME` with the name of the hybrid node you are upgrading

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

2. Drain the node you are upgrading with the following command. For more information on draining nodes, see Safely Drain a Node in the Kubernetes documentation. Replace `NODE_NAME` with the name of the hybrid node you are upgrading.

```
kubectl drain NODE_NAME --ignore-daemonsets --delete-emptydir-data
```

3. Run `nodeadm upgrade` on the hybrid node you are upgrading. You must run `nodeadm` with a user that has root/sudo privileges. The name of the node is preserved through upgrade for both AWS SSM and AWS IAM Roles Anywhere credential providers. You cannot change credentials providers during the upgrade process. See the section called "Hybrid nodes nodeadm" for configuration values for `nodeConfig.yaml`. Replace `K8S_VERSION` with the target Kubernetes version you upgrading to.

```
nodeadm upgrade K8S_VERSION -c file://nodeConfig.yaml
```

4. To allow pods to be scheduled on the node after you have upgraded, type the following. Replace `NODE_NAME` with the name of the node.

```
kubectl taint nodes NODE_NAME key=value:NoSchedule-
kubectl uncordon NODE_NAME
```

5. Watch the status of your hybrid nodes and wait for your nodes to shutdown and restart on the new Kubernetes version with the Ready status.

```
kubectl get nodes -o -w
```

Remove hybrid nodes

This topic describes how to delete hybrid nodes from your Amazon EKS cluster. You must delete your hybrid nodes with your choice of Kubernetes-compatible tooling such as kubectl. Charges for hybrid nodes stop when the node object is removed from the Amazon EKS cluster. For more information on hybrid nodes pricing, see Amazon EKS Pricing.

> **Important**
>
> Removing nodes is disruptive to workloads running on the node. Before deleting hybrid nodes, it is recommended to first drain the node to move pods to another active node. For more information on draining nodes, see Safely Drain a Node in the Kubernetes documentation.

Run the kubectl steps below from your local machine or instance that you use to interact with the Amazon EKS cluster's Kubernetes API endpoint. If you are using a specific `kubeconfig` file, use the `--kubeconfig` flag.

Step 1: List your nodes

```
kubectl get nodes
```

Step 2: Drain your node

See kubectl drain in the Kubernetes documentation for more information on the `kubectl drain` command.

```
kubectl drain --ignore-daemonsets <node-name>
```

Step 3: Stop and uninstall hybrid nodes artifacts

You can use the Amazon EKS Hybrid Nodes CLI (nodeadm) to stop and remove the hybrid nodes artifacts from the host. You must run nodeadm with a user that has root/sudo privileges. By default, nodeadm uninstall will not proceed if there are pods remaining on the node. If you are using AWS Systems Manager (SSM) as your credentials provider, the nodeadm uninstall command deregisters the host as an AWS SSM managed instance. For more information, see the section called "Hybrid nodes nodeadm".

```
nodeadm uninstall
```

Step 4: Delete your node from the cluster

With the hybrid nodes artifacts stopped and uninstalled, remove the node resource from your cluster.

```
kubectl delete node <node-name>
```

Step 5: Check for remaining artifacts

Depending on your choice of CNI, there may be artifacts remaining on your hybrid nodes after running the above steps. See the section called "Configure CNI" for more information.

Configure a CNI for hybrid nodes

Cilium and Calico are supported as the Container Networking Interfaces (CNIs) for Amazon EKS Hybrid Nodes. You must install a CNI for hybrid nodes to become ready to serve workloads. Hybrid nodes appear with status Not Ready until a CNI is running. You can manage these CNIs with your choice of tools such as Helm. The Amazon VPC CNI is not compatible with hybrid nodes and the VPC CNI is configured with anti-affinity for the eks.amazonaws.com/compute-type: hybrid label.

CNI version compatibility

Calico version 3.29.x is supported and recommended for EKS Hybrid Nodes for every Kubernetes version supported in Amazon EKS.

Cilium version 1.16.x is supported and recommended for EKS Hybrid Nodes for every Kubernetes version supported in Amazon EKS.

Supported capabilities

AWS supports the following capabilities of Cilium and Calico for use with hybrid nodes. If you plan to use functionality outside the scope of AWS support, we recommend that you obtain commercial support for the plugin or have the in-house expertise to troubleshoot and contribute fixes to the CNI plugin project.

Feature	Cilium	Calico
Kubernetes network conformance	Yes	Yes
Control plane to node connectivity	Yes	Yes
Control plane to pod connectivity	Yes	Yes
Lifecycle Management	Install, Upgrade, Delete	Install, Upgrade, Delete
Networking Mode	VXLAN	VXLAN
IP Address Management (IPAM)	Cluster Scope (Cilium IPAM)	Calico IPAM
IP family	IPv4	IPv4
BGP	Yes (Cilium Control Plane)	Yes

Install Cilium on hybrid nodes

1. Ensure that you have installed the helm CLI on your command-line environment. See the Helm documentation for installation instructions.
2. Install the Cilium Helm repo.

```
helm repo add cilium https://helm.cilium.io/
```

3. Create a YAML file called `cilium-values.yaml`. If you configured at least one *remote pod network*, configure the same pod CIDRs for your `clusterPoolIPv4PodCIDRList`. You

shouldn't change your clusterPoolIPv4PodCIDRList after deploying Cilium on your cluster. You can configure clusterPoolIPv4MaskSize based on your required pods per node, see Expanding the cluster pool in the Cilium documentation. For a full list of Helm values for Cilium, see the Helm reference in the Cilium documentation. The following example configures all of the Cilium components to run on only the hybrid nodes, since they have the eks.amazonaws.com/compute-type: hybrid label.

By default, Cilium masquerades the source IP address of all pod traffic leaving the cluster to the IP address of the node. This makes it possible for Cilium to run with Amazon EKS clusters that have remote pod networks configured and with clusters that don't have remote pod networks configured. If you disable masquerading for your Cilium deployment, then you must configure your Amazon EKS cluster with your remote pod networks and you must advertise your pod addresses with your on-premises network. If you are running webhooks on your hybrid nodes, you must configure your cluster with your remote pod networks and you must advertise your pod addresses with your on-premises network.

A common way to advertise pod addresses with your on-premises network is by using BGP. To use BGP with Cilium, you must set bgpControlPlane.enabled: true. For more information on Cilium's BGP support, see Cilium BGP Control Plane in the Cilium documentation.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: In
          values:
          - hybrid
ipam:
  mode: cluster-pool
  operator:
    clusterPoolIPv4MaskSize: 25
    clusterPoolIPv4PodCIDRList:
    - POD_CIDR
operator:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
```

```
          - key: eks.amazonaws.com/compute-type
            operator: In
            values:
              - hybrid
  unmanagedPodWatcher:
    restart: false
envoy:
  enabled: false
```

4. Install Cilium on your cluster. Replace `CILIUM_VERSION` with your desired Cilium version. It is recommended to run the latest patch version for your Cilium minor version. You can find the latest patch release for a given minor Cilium release in the [Stable Releases section](https://github.com/cilium/cilium#stable-releases) of the Cilium documentation. If you are enabling BGP for your deployment, add the `--set bgpControlPlane.enabled=true` flag in the command below. If you are using a specific kubeconfig file, use the `--kubeconfig` flag with the Helm install command.

```
helm install cilium cilium/cilium \
    --version CILIUM_VERSION \
    --namespace kube-system \
    --values cilium-values.yaml
```

5. You can confirm your Cilium installation was successful with the following commands. You should see the `cilium-operator` deployment and the `cilium-agent` running on each of your hybrid nodes. Additionally, your hybrid nodes should now have status `Ready`. For information on how to configure BGP for Cilium, proceed to the next step.

```
kubectl get pods -n kube-system
```

```
NAME                              READY   STATUS    RESTARTS   AGE
cilium-jjjn8                      1/1     Running   0          11m
cilium-operator-d4f4d7fcb-sc5xn   1/1     Running   0          11m
```

```
kubectl get nodes
```

```
NAME                   STATUS   ROLES    AGE   VERSION
mi-04a2cf999b7112233   Ready    <none>   19m   v1.31.0-eks-a737599
```

6. To use BGP with Cilium to advertise your pod addresses with your on-premises network, you must have installed Cilium with `bgpControlPlane.enabled: true`. To

configure BGP in Cilium, first create a file called `cilium-bgp-cluster.yaml` with a `CiliumBGPClusterConfig` with the `peerAddress` set to your on-premises router IP that you are peering with. Configure the `localASN` and `peerASN` based on your on-premises router configuration.

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPClusterConfig
metadata:
  name: cilium-bgp
spec:
  nodeSelector:
    matchExpressions:
    - key: eks.amazonaws.com/compute-type
      operator: In
      values:
      - hybrid
  bgpInstances:
  - name: "rack0"
    localASN: ONPREM_ROUTER_ASN
    peers:
    - name: "onprem-router"
      peerASN: PEER_ASN
      peerAddress: ONPREM_ROUTER_IP
      peerConfigRef:
        name: "cilium-peer"
```

7. Apply the Cilium BGP Cluster configuration to your cluster.

```
kubectl apply -f cilium-bgp-cluster.yaml
```

8. The `CiliumBGPPeerConfig` resource defines a BGP peer configuration. Multiple peers can share the same configuration and provide reference to the common `CiliumBGPPeerConfig` resource. Create a file named `cilium-bgp-peer.yaml` to configure the peer configuration for your on-premises network. See the BGP Peer Configuration in the Cilium documentation for a full list of configuration options.

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPPeerConfig
metadata:
  name: cilium-peer
spec:
  timers:
```

```
    holdTimeSeconds: 30
    keepAliveTimeSeconds: 10
  gracefulRestart:
    enabled: true
    restartTimeSeconds: 120
  families:
    - afi: ipv4
      safi: unicast
      advertisements:
        matchLabels:
          advertise: "bgp"
```

9. Apply the Cilium BGP Peer configuration to your cluster.

```
kubectl apply -f cilium-bgp-peer.yaml
```

10. The `CiliumBGPAdvertisement` resource is used to define various advertisement types and attributes associated with them. Create a file named `cilium-bgp-advertisement.yaml` and configure the `CiliumBGPAdvertisement` resource with your desired settings.

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: bgp-advertisements
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "PodCIDR"
    - advertisementType: "Service"
      service:
        addresses:
          - ClusterIP
          - ExternalIP
          - LoadBalancerIP
```

11. Apply the Cilium BGP Advertisement configuration to your cluster.

```
kubectl apply -f cilium-bgp-advertisement.yaml
```

You can confirm the BGP peering worked with the [Cilium CLI](https://github.com/cilium/cilium-cli) by using the `cilium bgp peers` command. You should see the correct values in the output for your environment and the

Session State as `established`. See the Troubleshooting and Operations Guide in the Cilium documentation for more information on troubleshooting.

Upgrade Cilium on hybrid nodes

Before upgrading your Cilium deployment, carefully review the Cilium upgrade documentation and the upgrade notes to understand the changes in the target Cilium version.

1. Ensure that you have installed the `helm` CLI on your command-line environment. See the Helm documentation for installation instructions.
2. Install the Cilium Helm repo.

```
helm repo add cilium https://helm.cilium.io/
```

3. Run the Cilium upgrade pre-flight check. Replace `CILIUM_VERSION` with your target Cilium version. We recommend that you run the latest patch version for your Cilium minor version. You can find the latest patch release for a given minor Cilium release in the Stable Releases section of the Cilium documentation.

```
helm install cilium-preflight cilium/cilium --version CILIUM_VERSION \
  --namespace=kube-system \
  --set preflight.enabled=true \
  --set agent=false \
  --set operator.enabled=false
```

4. After applying the `cilium-preflight.yaml`, ensure that the number of `READY` pods is the same number of Cilium pods running.

```
kubectl get ds -n kube-system | sed -n '1p;/cilium/p'
```

```
NAME                      DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
 SELECTOR   AGE
cilium                    2         2         2       2            2           <none>
          1h20m
cilium-pre-flight-check   2         2         2       2            2           <none>
          7m15s
```

5. Once the number of READY pods are equal, make sure the Cilium pre-flight deployment is also marked as READY 1/1. If it shows READY 0/1, consult the CNP Validation section and resolve issues with the deployment before continuing with the upgrade.

```
kubectl get deployment -n kube-system cilium-pre-flight-check -w
```

```
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
cilium-pre-flight-check   1/1     1            0           12s
```

6. Delete the preflight

```
helm uninstall cilium-preflight --namespace kube-system
```

7. During normal cluster operations, all Cilium components should run the same version. The following steps describe how to upgrade all of the components from one stable release to a later stable release. When upgrading from one minor release to another minor release, it is recommended to upgrade to the latest patch release for the existing Cilium minor version first. To minimize disruption, set the `upgradeCompatibility` option to the initial Cilium version that you installed in this cluster.

   Before running the helm upgrade command, preserve the values for your deployment in a `cilium-values.yaml` or use `--set` command line options for your settings. The upgrade operation overwrites the Cilium ConfigMap, so it is critical that your configuration values are passed when you upgrade. If you are using BGP, it is recommended to use the `--set bgpControlPlane=true` command line option instead of supplying this information in your values file.

```
helm upgrade cilium cilium/cilium --version CILIUM_VERSION \
  --namespace kube-system \
  --set upgradeCompatibility=1.X \
  -f cilium-values.yaml
```

8. (Optional) If you need to rollback your upgrade due to issues, run the following commands.

```
helm history cilium --namespace kube-system
helm rollback cilium [REVISION] --namespace kube-system
```

Delete Cilium from hybrid nodes

1. Run the following command to uninstall all Cilium components from your cluster. Note, uninstalling the CNI may impact the health of nodes and pods and shouldn't be performed on production clusters.

```
helm uninstall cilium --namespace kube-system
```

   The interfaces and routes configured by Cilium are not removed by default when the CNI is removed from the cluster, see the GitHub issue for more information.

2. To clean up the on-disk configuration files and resources, if you are using the standard configuration directories, you can remove the files as shown by the `cni-uninstall.sh` script in the Cilium repository on GitHub.

3. To remove the Cilium Custom Resource Definitions (CRDs) from your cluster, you can run the following commands.

```
kubectl get crds -oname | grep "cilium" | xargs kubectl delete
```

Install Calico on hybrid nodes

1. Ensure that you have installed the helm CLI on your command-line environment. See the Helm documentation for installation instructions.

2. Install the Cilium Helm repo.

```
helm repo add projectcalico https://docs.tigera.io/calico/charts
```

3. Create a YAML file called `calico-values.yaml` that configures Calico with affinity to run on hybrid nodes. For more information on the different Calico networking modes, see Determining the best networking option in the Calico documentation.

   a. Replace `POD_CIDR` with the CIDR ranges for your pods. If you configured your Amazon EKS cluster with remote pod networks, the `POD_CIDR` that you specify for Calico should be the same as the remote pod networks. For example, `10.100.0.0/24`.

   b. Replace `CIDR_SIZE` with the size of the CIDR segment you want to allocate to each node. For example, 25 for a /25 segment size. For more information on CIDR `blockSize` and changing the `blockSize`, see Change IP pool block size in the Calico documentation.

c. In the example below, `natOutgoing` is enabled and `bgp` is disabled. In this configuration, Calico can run on Amazon EKS clusters that have Remote Pod Network configured and can run on clusters that do not have Remote Pod Network configured. If you have `natOutgoing` set to disabled, you must configure your cluster with your remote pod networks and your on-premises network must be able to properly route traffic destined for your pod CIDRs. A common way to advertise pod addresses with your on-premises network is by using BGP. To use BGP with Calico, you must enable `bgp`. The example below configures all of the Calico components to run on only the hybrid nodes, since they have the `eks.amazonaws.com/compute-type: hybrid` label. If you are running webhooks on your hybrid nodes, you must configure your cluster with your Remote Pod Networks and you must advertise your pod addresses with your on-premises network. The example below configures `controlPlaneReplicas: 1`, increase the value if you have multiple hybrid nodes and want to run the Calico control plane components in a highly available fashion.

```
installation:
  enabled: true
  cni:
    type: Calico
    ipam:
      type: Calico
  calicoNetwork:
    bgp: Disabled
    ipPools:
    - cidr: POD_CIDR
      blockSize: CIDR_SIZE
      encapsulation: VXLAN
      natOutgoing: Enabled
      nodeSelector: eks.amazonaws.com/compute-type == "hybrid"
  controlPlaneReplicas: 1
  controlPlaneNodeSelector:
    eks.amazonaws.com/compute-type: hybrid
  calicoNodeDaemonSet:
    spec:
      template:
        spec:
          nodeSelector:
            eks.amazonaws.com/compute-type: hybrid
  csiNodeDriverDaemonSet:
    spec:
      template:
        spec:
```

```
          nodeSelector:
            eks.amazonaws.com/compute-type: hybrid
  calicoKubeControllersDeployment:
    spec:
      template:
        spec:
          nodeSelector:
            eks.amazonaws.com/compute-type: hybrid
  typhaDeployment:
    spec:
      template:
        spec:
          nodeSelector:
            eks.amazonaws.com/compute-type: hybrid
```

4. Install Calico on your cluster. Replace `CALICO_VERSION` with your desired Calico version (for example 3.29.0), see the [Calico releases](#) to find the latest patch release for your Calico minor version. It is recommended to run the latest patch version for the Calico minor version. If you are using a specific `kubeconfig` file, use the `--kubeconfig` flag.

```
helm install calico projectcalico/tigera-operator \
    --version CALICO_VERSION \
    --namespace kube-system \
    -f calico-values.yaml
```

5. You can confirm your Calico installation was successful with the following commands. You should see the `tigera-operator` deployment, the `calico-node` agent running on each of your hybrid nodes, the `calico-apiserver`, `csi-node-driver`, and `calico-kube-controllers` deployed. Additionally, your hybrid nodes should now have status `Ready`. If you are using `natOutgoing: Disabled`, then all of the Calico components will not be able to start successfully until you advertise your pod addresses with your on-premises network. For information on how to configure BGP for Calico, proceed to the next step.

```
kubectl get pods -A
```

```
NAMESPACE          NAME                                          READY   STATUS
 RESTARTS   AGE
calico-apiserver   calico-apiserver-6c77bb6d46-2n8mq             1/1     Running    0
      69s
calico-system      calico-kube-controllers-7c5f8556b5-7h267      1/1     Running    0
      68s
```

```
calico-system      calico-node-s5nnk                    1/1     Running    0
      68s
calico-system      calico-typha-6487cc9d8c-wc5jm        1/1     Running    0
      69s
calico-system      csi-node-driver-cv42d                2/2     Running    0
      68s
kube-system        coredns-7bb495d866-2lc9v             1/1     Running    0
      6m27s
kube-system        coredns-7bb495d866-2t8ln             1/1     Running    0
      157m
kube-system        kube-proxy-lxzxh                     1/1     Running    0
      18m
kube-system        tigera-operator-f8bc97d4c-28b4d      1/1     Running    0
      90s
```

```
kubectl get nodes
```

```
NAME                   STATUS   ROLES    AGE    VERSION
mi-0c6ec2f6f79176565   Ready    <none>   5h13m  v1.31.0-eks-a737599
```

6. If you installed Calico without BGP, skip this step. To configure BGP, create a file called `calico-bgp.yaml` with a `BGPPeer` configuration and a `BGPConfiguration`. It is important to distinguish `BGPPeer` and `BGPConfiguration`. The `BGPPeer` is the BGP-enabled router or remote resource with which the nodes in a Calico cluster will peer. The `asNumber` in the `BGPPeer` configuration is similar to the Cilium setting `peerASN` . The `BGPConfiguration` is applied to each Calico node and the `asNumber` for the `BGPConfiguration` is equivalent to the Cilium setting `localASN`. Replace `ONPREM_ROUTER_IP`, `ONPREM_ROUTER_ASN`, and `LOCAL_ASN` in the example below with the values for your on-premises environment. The `keepOriginalNextHop: true` setting is used to ensure each node advertises only the pod network CIDR that it owns.

```
apiVersion: projectcalico.org/v3
kind: BGPPeer
metadata:
  name: calico-hybrid-nodes
spec:
  peerIP: ONPREM_ROUTER_IP
  asNumber: ONPREM_ROUTER_ASN
  keepOriginalNextHop: true
---
```

```
apiVersion: projectcalico.org/v3
kind: BGPConfiguration
metadata:
  name: default
spec:
  nodeToNodeMeshEnabled: false
  asNumber: LOCAL_ASN
```

7. Apply the file to your cluster.

```
kubectl apply -f calico-bgp.yaml
```

8. Confirm the Calico pods are running with the following command.

```
kubectl get pods -n calico-system -w
```

```
NAMESPACE          NAME                                        READY   STATUS
 RESTARTS       AGE
calico-apiserver   calico-apiserver-598bf99b6c-2vltk           1/1     Running   0
          3h24m
calico-system      calico-kube-controllers-75f84bbfd6-zwmnx    1/1     Running   31
 (59m ago)   3h20m
calico-system      calico-node-9b2pg                           1/1     Running   0
          5h17m
calico-system      calico-typha-7d55c76584-kxtnq               1/1     Running   0
          5h18m
calico-system      csi-node-driver-dmnmm                       2/2     Running   0
          5h18m
kube-system        coredns-7bb495d866-dtn4z                    1/1     Running   0
          6h23m
kube-system        coredns-7bb495d866-mk7j4                    1/1     Running   0
          6h19m
kube-system        kube-proxy-vms28                            1/1     Running   0
          6h12m
kube-system        tigera-operator-55f9d9d565-jj9bg            1/1     Running   0
          73m
```

If you encountered issues during these steps, see the troubleshooting guidance in the Calico documentation.

Upgrade Calico on hybrid nodes

Before upgrading your Calico deployment, carefully review the Calico upgrade documentation and the release notes to understand the changes in the target Calico version. The upgrade steps vary based on whether you are using Helm, the Calico operator, and the type of datastore. The steps below assume use of Helm.

1. Download the operator manifest for the version of Calico you are upgrading to. Replace `CALICO_VERSION` with the version you are upgrading to, for example `v3.29.0`. Make sure to prepend the `v` to the major.minor.patch.

```
kubectl apply --server-side --force-conflicts \
    -f https://raw.githubusercontent.com/projectcalico/calico/CALICO_VERSION/
manifests/operator-crds.yaml
```

2. Run `helm upgrade` to upgrade your Calico deployment. Replace CALICO_VERSION with the version you are upgrading to, for example `v3.29.0`. Create the `calico-values.yaml` file from the configuration values that you used to install Calico.

```
helm upgrade calico projectcalico/tigera-operator \
    --version CALICO_VERSION \
    --namespace kube-system \
    -f calico-values.yaml
```

Delete Calico from hybrid nodes

1. Run the following command to uninstall Calico components from your cluster. Note that uninstalling the CNI might impact the health of nodes and pods and shouldn't be performed on production clusters. If you installed Calico in a namespace other than `kube-system` change the namespace in the command below.

```
helm uninstall calico --namespace kube-system
```

   Note that the interfaces and routes configured by Calico are not removed by default when you remove the CNI from the cluster.

2. To clean up the on-disk configuration files and resources, remove the Calico files from the `/opt/cni` and `/etc/cni` directories.

3. To remove the Calico CRDs from your cluster, run the following commands.

```
kubectl get crds -oname | grep "calico" | xargs kubectl delete
```

```
kubectl get crds -oname | grep "tigera" | xargs kubectl delete
```

Configure add-ons for hybrid nodes

This page describes considerations for running Amazon EKS add-ons from AWS on Amazon EKS Hybrid Nodes. To learn more about the Amazon EKS add-ons from AWS and the processes for creating, upgrading, and removing add-ons from your cluster, see the section called "Amazon EKS add-ons". The processes for creating, upgrading, and removing Amazon EKS add-ons is the same for Amazon EKS clusters with hybrid nodes as it is for Amazon EKS clusters with nodes running in AWS Cloud unless otherwise noted on this page.

The following Amazon EKS add-ons from AWS are compatible with Amazon EKS Hybrid Nodes.

EKS add-on	Compatible add-on versions
kube-proxy	v1.25.14-eksbuild.2 and above
CoreDNS	v1.9.3-eksbuild.7 and above
AWS Distro for OpenTelemetry (ADOT)	v0.102.1-eksbuild.2 and above
CloudWatch Observability Agent	v2.2.1-eksbuild.1 and above
EKS Pod Identity Agent	v1.3.3-eksbuild.1 and above
CSI snapshot controller	v8.1.0-eksbuild.1 and above

In addition to the Amazon EKS add-ons in the table above, the Amazon Managed Service for Prometheus Collector, and the AWS Load Balancer Controller for application ingress (HTTP) and load balancing (TCP/UDP) are compatible with hybrid nodes.

Amazon EKS add-ons from AWS that are not compatible with Amazon EKS Hybrid Nodes have been updated with an affinity rule for the default eks.amazonaws.com/compute-type: hybrid label applied to hybrid nodes. This prevents them from running on hybrid nodes when deployed in your clusters. If you have clusters with both hybrid nodes and nodes running in AWS Cloud, Amazon EKS add-ons that are not compatible with hybrid nodes can still be deployed in your cluster to nodes running in AWS Cloud. The Amazon VPC CNI is not compatible with hybrid nodes, and Cilium and Calico are supported as the Container Networking Interfaces (CNIs) for Amazon EKS Hybrid Nodes. See the section called "Configure CNI" for more information.

The rest of this page describes differences between running compatible Amazon EKS add-ons from AWS on hybrid nodes, compared to the other Amazon EKS compute types.

kube-proxy and CoreDNS

Kube-proxy and CoreDNS are installed as unmanaged add-ons by default when an EKS cluster is created. These add-ons can be managed as Amazon EKS add-ons after cluster creation. Reference the EKS documentation for details on the section called "kube-proxy" and the section called "CoreDNS". If you are running a cluster with hybrid nodes and nodes in AWS Cloud, it is recommended to have at least one CoreDNS replica on hybrid nodes and at least one CoreDNS replica on your nodes in AWS Cloud.

CloudWatch Observability Agent add-on

Node-level metrics are not available for hybrid nodes because CloudWatch Container Insights depends on the availability of Instance Metadata Service (IMDS) for node-level metrics. Cluster, workload, pod, and container-level metrics are available for hybrid nodes.

After installing the add-on by following the steps described in Install the CloudWatch agent with the Amazon CloudWatch Observability, the add-on manifest must be updated before the agent can run successfully on hybrid nodes. Edit the `amazoncloudwatchagents` resource on the cluster to add the `RUN_WITH_IRSA` environment variable as shown below.

```
kubectl edit amazoncloudwatchagents -n amazon-cloudwatch cloudwatch-agent
```

```
apiVersion: v1
items:
- apiVersion: cloudwatch.aws.amazon.com/v1alpha1
  kind: AmazonCloudWatchAgent
  metadata:
    ...
    name: cloudwatch-agent
    namespace: amazon-cloudwatch
    ...
```

```
spec:
  ...
  env:
  - name: RUN_WITH_IRSA # <-- Add this
    value: "True" # <-- Add this
  - name: K8S_NODE_NAME
    valueFrom:
      fieldRef:
        fieldPath: spec.nodeName
        ...
```

Amazon Managed Prometheus managed collector for hybrid nodes

An Amazon Managed Service for Prometheus (AMP) managed collector consists of a scraper that discovers and collects metrics from the resources in an Amazon EKS cluster. AMP manages the scraper for you, removing the need to manage any instances, agents, or scrapers yourself.

You can use AMP managed collectors without any additional configuration specific to hybrid nodes. However the metric endpoints for your applications on the hybrid nodes must be reachable from the VPC, including routes from the VPC to remote pod network CIDRs and the ports open in your on-premises firewall. Additionally, your cluster must have private cluster endpoint access.

Follow the steps in Using an AWS managed collector in the Amazon Managed Service for Prometheus User Guide.

AWS Distro for OpenTelemetry (ADOT) add-on

You can use the AWS Distro for OpenTelemetry (ADOT) Amazon EKS add-on to collect metrics, logs, and tracing data from your applications running on hybrid nodes. Note, ADOT uses admission webhooks to mutate and validate the Collector Custom Resource requests and you must configure your remote pod network when creating your Amazon EKS cluster.

Follow the steps in Getting Started with AWS Distro for OpenTelemetry using EKS Add-Ons in the *AWS Distro for OpenTelemetry* documentation.

AWS Load Balancer Controller

You can use the AWS Load Balancer Controller and Application Load Balancer (ALB) or Network Load Balancer (NLB) with the target type ip for workloads on hybrid nodes connected with AWS Direct Connect or AWS Site-to-Site VPN. As the AWS Load Balancer Controller uses webhooks, you must configure your remote pod network when creating your Amazon EKS cluster.

To install the AWS Load Balancer Controller, follow the steps at the section called "Install with Helm" or the section called "Install with manifests".

For ingress with ALB, you must specify the annotations below. See the section called "Application load balancing" for instructions.

```
alb.ingress.kubernetes.io/target-type: ip
```

For load balancing with NLB, you must specify the annotations below. See the section called "Network load balancing" for instructions.

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip"
```

EKS Pod Identity Agent add-on

The original Amazon EKS Pod Identity Agent DaemonSet relies on the availability of EC2 IMDS on the node to obtain the required AWS credentials. As IMDS isn't available on hybrid nodes, starting in add-on version `1.3.3-eksbuild.1`, the Pod Identity Agent add-on optionally deploys a second DaemonSet that specifically targets hybrid nodes. This DaemonSet mounts the required credentials to the pods created by the Pod Identity Agent add-on.

1. To use the Pod Identity agent on hybrid nodes, set `enableCredentialsFile: true` in the hybrid section of `nodeadm` config as shown below:

   ```
   apiVersion: node.eks.aws/v1alpha1
   kind: NodeConfig
   spec:
       hybrid:
           enableCredentialsFile: true # <-- Add this
   ```

   This will configure `nodeadm` to create a credentials file to be configured on the node under `/eks-hybrid/.aws/credentials`, which will be used by `eks-pod-identity-agent` pods. This credentials file will contain temporary AWS credentials that will be refreshed periodically.

2. After you update the `nodeadm` config on *each* node, run the following `nodeadm init` command with your `nodeConfig.yaml` to join your hybrid nodes to your Amazon EKS cluster. If your nodes have joined the cluster previous, still run the `init` command again.

```
nodeadm init -c file://nodeConfig.yaml
```

3. Install `eks-pod-identity-agent` with support for hybrid nodes enabled, by either using the AWS CLI or AWS Management Console.

   a. AWS CLI: From the machine that you're using to administer the cluster, run the following command to install `eks-pod-identity-agent` with support for hybrid nodes enabled. Replace `my-cluster` with the name of your cluster.

   ```
   aws eks create-addon \
       --cluster-name my-cluster \
       --addon-name eks-pod-identity-agent \
       --configuration-values '{"daemonsets":{"hybrid":{"create": true}}}'
   ```

   b. AWS Management Console: If you are installing the Pod Identity Agent add-on through the AWS console, add the following to the optional configuration to deploy the daemonset that targets hybrid nodes.

   ```
   {"daemonsets":{"hybrid":{"create": true}}}
   ```

CSI snapshot controller add-on

Starting with version `v8.1.0-eksbuild.2`, the CSI snapshot controller add-on applies a soft anti-affinity rule for hybrid nodes, preferring the controller `deployment` to run on EC2 in the same AWS Region as the Amazon EKS control plane. Co-locating the `deployment` in the same AWS Region as the Amazon EKS control plane improves latency.

Configure proxy for hybrid nodes

If you are using a proxy server in your on-premises environment for traffic leaving your data center or edge environment, you need to configure your operating system, `containerd`, `kubelet`, and `kube-proxy` to use your proxy server. You must configure `kube-proxy` after creating your Amazon EKS cluster. You can make the changes for your operating system, `containerd`, and the `kubelet` during the build process for your operating system images or before you run `nodeadm init` on each hybrid node.

Node-level configuration

The configurations in this section must be applied in your operating system images or before running `nodeadm init` on each hybrid node.

`containerd` proxy configuration

`containerd` is the default container management runtime for Kubernetes. If you are using a proxy for internet access, you must configure `containerd` so it can pull the container images required by Kubernetes and Amazon EKS.

Create a file on each hybrid node called `http-proxy.conf` in the `/etc/systemd/system/containerd.service.d` directory with the following contents. Replace `proxy-domain` and `port` with the values for your environment.

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

`containerd` configuration from user data

The `containerd.service.d` directory will need to be created for this file. You will need to reload systemd to pick up the configuration file without a reboot. In AL2023, the service will likely already be running when your script executes, so you will also need to restart it.

```
mkdir -p /etc/systemd/system/containerd.service.d
echo '[Service]' > /etc/systemd/system/containerd.service.d
echo 'Environment="HTTP_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
containerd.service.d
echo 'Environment="HTTPS_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
containerd.service.d
echo 'Environment="NO_PROXY=localhost"' >> /etc/systemd/system/containerd.service.d
systemctl daemon-reload
systemctl restart containerd
```

`kubelet` proxy configuration

`kubelet` is the Kubernetes node agent that runs on each Kubernetes node and is responsible for managing the node and pods running on it. If you are using a proxy in your on-premises environment, you must configure the `kubelet` so it can communicate with your Amazon EKS cluster's public or private endpoints.

Create a file on each hybrid node called `http-proxy.conf` in the `/etc/systemd/system/kubelet.service.d/` directory with the following content. Replace `proxy-domain` and `port` with the values for your environment.

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

`kubelet` configuration from user data

The `kubelet.service.d` directory must be created for this file. You will need to reload systemd to pick up the configuration file without a reboot. In AL2023, the service will likely already be running when your script executes, so you will also need to restart it.

```
mkdir -p /etc/systemd/system/kubelet.service.d
echo '[Service]' > /etc/systemd/system/kubelet.service.d
echo 'Environment="HTTP_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
kubelet.service.d
echo 'Environment="HTTPS_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
kubelet.service.d
echo 'Environment="NO_PROXY=localhost"' >> /etc/systemd/system/kubelet.service.d
systemctl daemon-reload
systemctl restart containerd
```

Operating system proxy configuration

If you are using a proxy for internet access, you must configure your operating system to be able to pull the hybrid nodes dependencies from your operating systems' package manager.

Ubuntu

1. Configure `snap` to use your proxy with the following commands:

```
sudo snap set system proxy.https=http://proxy-domain:port
sudo snap set system proxy.http=http://proxy-domain:port
```

2. To enable proxy for apt, create a file called apt.conf in the /etc/apt/ directory. Replace proxy-domain and port with the values for your environment.

```
Acquire::http::Proxy "http://proxy-domain:port";
Acquire::https::Proxy "http://proxy-domain:port";
```

Amazon Linux 2023 and Red Hat Enterprise Linux

1. Configure yum to use your proxy. Create a file /etc/yum.conf with the proxy-domain and port values for your environment.

```
proxy=http://proxy-domain:port
```

Cluster wide configuration

The configurations in this section must be applied after you create your Amazon EKS cluster and before running nodeadm init on each hybrid node.

kube-proxy proxy configuration

Amazon EKS automatically installs kube-proxy on each hybrid node as a DaemonSet when your hybrid nodes join the cluster. kube-proxy enables routing across services that are backed by pods on Amazon EKS clusters. To configure each host, kube-proxy requires DNS resolution for your Amazon EKS cluster endpoint.

1. Edit the kube-proxy DaemonSet with the following command

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

   This will open the kube-proxy DaemonSet definition on your configured editor.

2. Add the environment variables for HTTP_PROXY and HTTPS_PROXY. Note the NODE_NAME environment variable should already exist in your configuration. Replace proxy-domain and port with values for your environment.

```
containers:
  - command:
    - kube-proxy
```

```
    - --v=2
    - --config=/var/lib/kube-proxy-config/config - --hostname-override=$(NODE_NAME)
    env:
    - name: HTTP_PROXY
      value: http://proxy-domain:port
    - name: HTTPS_PROXY
      value: http://proxy-domain:port
    - name: NODE_NAME
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: spec.nodeName
```

Hybrid nodes nodeadm reference

The Amazon EKS Hybrid Nodes CLI (nodeadm) used for hybrid nodes lifecycle management differs from the nodeadm version used for bootstrapping Amazon EC2 instances as nodes in Amazon EKS clusters. Follow the documentation and references for the appropriate nodeadm version. This documentation page is for the hybrid nodes nodeadm version and the hybrid nodes nodeadm version is available in the eks-hybrid repository on GitHub. See the nodeadm - Amazon EKS AMI documentation for the nodeadm version used for Amazon EC2 instances.

Download nodeadm

The hybrid nodes version of nodeadm is hosted in Amazon S3 fronted by Amazon CloudFront. To install nodeadm on each on-premises host, you can run the following command from your on-premises hosts.

For x86_64 hosts:

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/amd64/
nodeadm'
```

For ARM hosts

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/arm64/
nodeadm'
```

Add executable file permission to the downloaded binary on each host.

```
chmod +x nodeadm
```

Commands

Important

You must run `nodeadm` with a user that has root/sudo privileges.

Install

The `install` command is used to install the artifacts and dependencies required to run and join hybrid nodes to an Amazon EKS cluster. The `install` command can be run individually on each hybrid node or can be run during image build pipelines to preinstall the hybrid nodes dependencies in operating system images.

Usage

`nodeadm install [KUBERNETES_VERSION] [flags]`

Positional Arguments

(Required) `KUBERNETES_VERSION` The major.minor version of EKS Kubernetes to install, for example `1.31`

Flags

Name	Required	Description
`-p,` `--credential-provider`	TRUE	Credential provider to install. Supported values are `iam-ra` and `ssm`. See the section called "Prepare credentials" for more information.
`-s,` `--containerd-source`	FALSE	Source for `containerd` . `nodeadm` supports installing `containerd` from the OS distro, Docker packages,

Name	Required	Description
		and skipping `containerd` install. **Values** `distro` - This is the default value. `nodeadm` will install `containerd` package distributed by the node OS. `distro` is not a supported value for Red Hat Enterprise Linux (RHEL) operating systems. `docker` - `nodeadm` will install `containerd` package built and distributed by Docker. `docker` is not a supported value for Amazon Linux 2023 `none` - `nodeadm` will not install `containerd` package. You must manually install `containerd` before running `nodeadm init`.
`-t`, `--timeout`	FALSE	Maximum install command duration. The input follows duration format. For example `1h23m`. Default download timeout for install command is set to 20 minutes.

Name	Required	Description
`-h, --help`	FALSE	Displays help message with available flag, subcomman d and positional value parameters.

Examples

Install Kubernetes version `1.31` with AWS Systems Manager (SSM) as the credential provider

```
nodeadm install 1.31 --credential-provider ssm
```

Install Kubernetes version `1.31` with AWS Systems Manager (SSM) as the credential provider, Docker as the containerd source, with a download timeout of 20 minutes.

```
nodeadm install 1.31 --credential-provider ssm --containerd-source docker --timeout 20m
```

Install Kubernetes version `1.31` with AWS IAM Roles Anywhere as the credential provider

```
nodeadm install 1.31 --credential-provider iam-ra
```

Files installed

Artifact	Path
IAM Roles Anywhere CLI	/usr/local/bin/aws_signing_helper
Kubelet binary	/usr/bin/kubelet
Kubectl binary	usr/local/bin/kubectl
ECR Credentials Provider	/etc/eks/image-credential-provider/ecr-credential-provider
AWS IAM Authenticator	/usr/local/bin/aws-iam-authenticator
SSM Setup CLI	/opt/ssm/ssm-setup-cli

Artifact	Path
SSM Agent	On Ubuntu - /snap/amazon-ssm-agent/current/amazon-ssm-agent On RHEL & AL2023 - /usr/bin/amazon-ssm-agent
Containerd	On Ubuntu & AL2023 - /usr/bin/containerd On RHEL - /bin/containerd
Iptables	On Ubuntu & AL2023 - /usr/sbin/iptables On RHEL - /sbin/iptables
CNI plugins	/opt/cni/bin
installed artifacts tracker	/opt/nodeadm/tracker

Config check

The `config check` command checks the provided node configuration for errors. This command can be used to verify and validate the correctness of a hybrid node configuration file.

Usage

```
nodeadm config check [flags]
```

Flags

Name	Required	Description
`-c`, `--config-source`	TRUE	Source of nodeadm configuration. For hybrid nodes the input should follow a URI with file scheme.
`-h`, `--help`	FALSE	Displays help message with available flag, subcomman

Name	Required	Description
		d and positional value parameters.

Examples

```
nodeadm config check --config-source file:///root/nodeConfig.yaml
```

Init

The `init` command starts and connects the hybrid node with the configured Amazon EKS cluster.

Usage

```
nodeadm init [flags]
```

Flags

Name	Required	Description
`-c,` `--config-source`	TRUE	Source of `nodeadm` configura tion. For hybrid nodes the input should follow a URI with file scheme.
`-s,` `--skip`	FALSE	Phases of `init` to be skipped. It is not recommended to skip any of the phases unless it helps to fix an issue. **Values** `install-validation` skips checking if the proceding install command ran successfully.

Name	Required	Description
`-h, --help`	FALSE	Displays help message with available flag, subcommand and positional value parameters.

Examples

```
nodeadm init --config-source file://root/nodeConfig.yaml
```

Files installed

Name	Path
Kubelet kubeconfig	/var/lib/kubelet/kubeconfig
Kubelet config	/etc/kubernetes/kubelet/config.json
Kubelet systemd unit	/etc/systemd/system/kubelet.service
Image credentials provider config	/etc/eks/image-credential-provider/config.json
Kubelet env file	/etc/eks/kubelet/environment
Kubelet Certs	/etc/kubernetes/pki/ca.crt
Containerd config	/etc/containerd/config.toml
Containerd kernel modules config	/etc/modules-load.d/contianerd.conf
AWS config file	/etc/aws/hybrid/config
AWS credentials file (if enable credentials file)	/eks-hybrid/.aws/credentials
AWS signing helper system unit	/etc/systemd/system/aws_signing_helper_update.service

Name	Path
Sysctl conf file	/etc/sysctl.d/99-nodeadm.conf
Apt manager files for docker repo (if container d source is docker)	
Ca-certificates	/etc/ssl/certs/ca-certificates.crt
Gpg key file	/etc/apt/keyrings/docker.asc
Docker repo source file	/etc/apt/sources.list.d/docker.list

Upgrade

The `nodeadm upgrade` command upgrades all the installed artifacts to the latest version and bootstraps the node to configure the upgraded artifacts and join the EKS cluster on AWS. Upgrade is a disruptive command to the workloads running on the node. Please move your workloads to another node before running upgrade.

Usage

```
nodeadm upgrade [KUBERNETES_VERSION] [flags]
```

Positional Arguments

(Required) `KUBERNETES_VERSION` The major.minor version of EKS Kubernetes to install, for example `1.31`

Flags

Name	Required	Description
`-c,` `--config-source`	TRUE	Source of `nodeadm` configura tion. For hybrid nodes the input should follow a URI with file scheme.
`-t,`	FALSE	Timeout for downloadi ng artifacts. The input

Name	Required	Description
`--timeout`		follows duration format. For example 1h23m. Default download timeout for upgrade command is set to 10 minutes.
`-s`, `--skip`	FALSE	Phases of upgrade to be skipped. It is not recommend ed to skip any of the phase unless it helps to fix an issue. **Values** `pod-validation` skips checking if all the no pods are running on the node, except daemon sets and static pods. `node-validation` skips checking if the node has been cordoned. `init-validation` skips checking if the node has been initialized successfully before running upgrade.
`-h`, `--help`	FALSE	Displays help message with available flag, subcomman d and positional value parameters.

Examples

```
nodeadm upgrade 1.31 --config-source file:///root/nodeConfig.yaml
```

```
nodeadm upgrade 1.31 --config-source file:///root/nodeConfig.yaml --timeout 20m
```

Uninstall

The `nodeadm uninstall` command stops and removes the artifacts `nodeadm` installs during `nodeadm install`, including the kubelet and containerd. Note, the uninstall command does not drain or delete your hybrid nodes from your cluster. You must run the drain and delete operations separately, see the section called "Delete hybrid nodes" for more information. By default, `nodeadm uninstall` will not proceed if there are pods remaining on the node. Similarly, `nodeadm uninstall` does not remove CNI dependencies or dependencies of other Kubernetes add-ons you run on your cluster. To fully remove the CNI installation from your host, see the instructions at the section called "Configure CNI". If you are using AWS SSM hybrid activations as your on-premises credentials provider, the `nodeadm uninstall` command deregisters your hosts as AWS SSM managed instances.

Usage

```
nodeadm uninstall [flags]
```

Flags

Name	Required	Description
`-s`, `--skip`	FALSE	Phases of upgrade to be skipped. It is not recommend ed to skip any of the phase unless it helps to fix an issue. **Values** `pod-validation` skips checking if all the no pods are running on the node, except daemon sets and static pods. `node-validation` skips checking if the node has been cordoned.

Name	Required	Description
		`init-validation` skips checking if the node has been initialized successfully before running upgrade.
`-h`, `--help`	FALSE	Displays help message with available flag, subcomman d and positional value parameters.

Examples

```
nodeadm uninstall
```

```
nodeadm uninstall --skip node-validation,pod-validation
```

Debug

The `nodeadm debug` command can be used to troubleshoot unhealthy or misconfigured hybrid nodes. It validates the following requirements are in-place.

- The node has network access to the required AWS APIs for obtaining credentials,
- The node is able to get AWS credentials for the configured Hybrid Nodes IAM role,
- The node has network access to the EKS Kubernetes API endpoint and the validity of the EKS Kubernetes API endpoint certificate,
- The node is able to authenticate with the EKS cluster, its identity in the cluster is valid, and that the node has access to the EKS cluster through the VPC configured for the EKS cluster.

If errors are found, the command's output suggests troubleshooting steps. Certain validation steps show child processes. If these fail, the output is showed in a stderr section under the validation error.

Usage

```
nodeadm debug [flags]
```

Flags

Name	Required	Description
`-c`, `--config-source`	TRUE	Source of `nodeadm` configuration. For hybrid nodes the input should follow a URI with file scheme.
`-h`, `--help`	FALSE	Displays help message with available flag, subcommand and positional value parameters.

Examples

```
nodeadm debug --config-source file://nodeConfig.yaml
```

Node Config API Reference

AWS SSM hybrid activations

The following is a sample `nodeConfig.yaml` when using AWS SSM hybrid activations for hybrid nodes credentials.

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:             # Name of the EKS cluster
    region:           # AWS Region where the EKS cluster resides
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id
```

AWS IAM Roles Anywhere

The following is a sample `nodeConfig.yaml` for AWS IAM Roles Anywhere for hybrid nodes credentials.

When using AWS IAM Roles Anywhere as your on-premises credentials provider, the `nodeName` you use in your `nodeadm` configuration must align with the permissions you scoped for your Hybrid Nodes IAM role. For example, if your permissions for the Hybrid Nodes IAM role only allow AWS IAM Roles Anywhere to assume the role when the role session name is equal to the CN of the host certificate, then the `nodeName` in your `nodeadm` configuration must be the same as the CN of your certificates. The `nodeName` that you use can't be longer than 64 characters. For more information, see the section called "Prepare credentials".

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:              # Name of the EKS cluster
    region:            # AWS Region where the EKS cluster resides
  hybrid:
    iamRolesAnywhere:
      nodeName:        # Name of the node
      trustAnchorArn:  # ARN of the IAM Roles Anywhere trust anchor
      profileArn:      # ARN of the IAM Roles Anywhere profile
      roleArn:         # ARN of the Hybrid Nodes IAM role
      certificatePath: # Path to the certificate file to authenticate with the IAM
 Roles Anywhere trust anchor
      privateKeyPath:  # Path to the private key file for the certificate
```

(Optional) Kubelet configuration

You can pass kubelet configuration and flags in your `nodeadm` configuration. See the example below for how to add an additional node label `abc.amazonaws.com/test-label` and config for setting `shutdownGracePeriod` to 30 seconds.

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:             # Name of the EKS cluster
    region:           # AWS Region where the EKS cluster resides
  kubelet:
    config:           # Map of kubelet config and values
```

```
      shutdownGracePeriod: 30s
    flags:            # List of kubelet flags
      - --node-labels=abc.company.com/test-label=true
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id
```

(Optional) Containerd configuration

You can pass custom containerd configuration in your `nodeadm` configuration. The containerd configuration for `nodeadm` accepts in-line TOML. See the example below for how to configure containerd to disable deletion of unpacked image layers in the containerd content store.

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:             # Name of the EKS cluster
    region:           # AWS Region where the EKS cluster resides
  containerd:
    config: |         # Inline TOML containerd additional configuration
       [plugins."io.containerd.grpc.v1.cri".containerd]
       discard_unpacked_layers = false
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id
```

You can also use the containerd configuration to enable SELinux support. With SELinux enabled on containerd, ensure pods scheduled on the node have the proper securityContext and seLinuxOptions enabled. More information on configuring a security context can be found on the Kubernetes documentation.

Note

Red Hat Enterprise Linux (RHEL) 8 and RHEL 9 have SELinux enabled by default and set to strict on the host. Amazon Linux 2023 has SELinux enabled by default and set to permissive mode. When SELinux is set to permissive mode on the host, enabling it on containerd will not block requests but will log it according to the SELinux configuration on the host.

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:             # Name of the EKS cluster
    region:           # AWS Region where the EKS cluster resides
  containerd:
    config: |         # Inline TOML containerd additional configuration
       [plugins."io.containerd.grpc.v1.cri"]
       enable_selinux = true
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id
```

Troubleshooting hybrid nodes

This topic covers some common errors that you may see while using Amazon EKS Hybrid Nodes and how to fix them. For other troubleshooting information, see *Troubleshooting* and Knowledge Center tag for Amazon EKS on *AWS re:Post*. If you cannot resolve the issue, contact AWS Support.

You can run the `nodeadm debug` command from your hybrid nodes to validate networking and credential requirements are met. For more information on the `nodeadm debug` command, see the section called "Hybrid nodes nodeadm".

Installing hybrid nodes troubleshooting

The troubleshooting topics in this section are related to installing the hybrid nodes dependencies on hosts with the `nodeadm install` command.

nodeadm command failed `must run as root`

The `nodeadm install` command must be run with a user that has root/sudo privileges on your host. If you run `nodeadm install` with a user that does not have root/sudo privileges, you will see the following error in the nodeadm output.

```
"msg":"Command failed","error":"must run as root"
```

Unable to connect to dependencies

The `nodeadm install` command installs the dependencies required for hybrid nodes. The hybrid nodes dependencies include `containerd`, `kubelet`, `kubectl`, and AWS SSM or AWS IAM Roles Anywhere components. You must have access from where you are running nodeadm install to download these dependencies. For more information on the list of locations that you must be able to access, see the section called "Prepare networking". If you do not have access, you will see errors similar to the following in the `nodeadm install` output.

```
"msg":"Command failed","error":"failed reading file from url: ...: max retries achieved
 for http request"
```

Failed to update package manager

The `nodeadm install` command runs apt update or yum/dnf update before installing the hybrid nodes dependencies. If this step does not succeed you may see errors similar to the following. To remediate, you can run apt update or yum/dnf update before running `nodeadm install` or you can attempt to re-run `nodeadm install`.

```
failed to run update using package manager
```

Timeout or context deadline exceeded

When running `nodeadm install`, if you see issues at various stages of the install process with errors that indicate there was a timeout or context deadline exceeded, you may have a slow connection that is preventing the installation of the hybrid nodes dependencies before timeouts are met. To work around these issues, you can attempt to use the `--timeout` flag in `nodeadm` to extend the duration of the timeouts for downloading the dependencies.

```
nodeadm install K8S_VERSION --credential-provider CREDS_PROVIDER --timeout 20m0s
```

Connecting hybrid nodes troubleshooting

The troubleshooting topics in this section are related to the process of connecting hybrid nodes to EKS clusters with the `nodeadm init` command.

Operation errors or unsupported scheme

When running `nodeadm init`, if you see errors related to `operation error` or `unsupported scheme`, check your `nodeConfig.yaml` to make sure it is properly formatted and passed to nodeadm. For more information on the format and options for nodeConfig.yaml, see the section called "Hybrid nodes nodeadm".

```
"msg":"Command failed","error":"operation error ec2imds: GetRegion, request canceled,
 context deadline exceeded"
```

Hybrid Nodes IAM role missing permissions for the eks:DescribeCluster action

When running nodeadm init, nodeadm attempts to gather information about your EKS cluster by calling Describe Cluster. If your Hybrid Nodes IAM role does not have permission for the eks:DescribeCluster action. For more information on the required permissions for the Hybrid Nodes IAM role, see the section called "Prepare credentials".

```
"msg":"Command failed","error":"operation error EKS: DescribeCluster, https response
 error StatusCode: 403 ... AccessDeniedException"
```

Hybrid nodes are not appearing in EKS cluster

If you ran nodeadm init and it completed but your hybrid nodes do not appear in your cluster, there may be issues with the network connection between your hybrid nodes and the EKS control plane, you may not have the required security group permissions configured, or you may not have the required mapping of your Hybrid Nodes IAM role to Kubernetes Role-Based Access Control (RBAC). You can start the debugging process by checking the status of kubelet and the kubelet logs with the following commands. Run the following commands from the hybrid nodes that failed to join your cluster.

```
systemctl status kubelet
```

```
journalctl -u kubelet -f
```

Unable to communicate with cluster

If your hybrid node was unable to communicate with the cluster control plane, you may see logs similar to the following.

```
"Failed to ensure lease exists, will retry" err="Get ..."
```

```
"Unable to register node with API server" err="Post ..."
```

```
Failed to contact API server when waiting for CSINode publishing ... dial tcp <ip
 address>: i/o timeout
```

If you see these messages, check the following to ensure it meets the hybrid nodes requirements detailed in the section called “Prepare networking”.

- Confirm the VPC passed to EKS cluster has routes to your Transit Gateway (TGW) or Virtual Private Gateway (VGW) for your on-premises node and optionally pod CIDRs.
- Confirm you have an additional security group for your EKS cluster has inbound rules for your on-premises node CIDRs and optionally pod CIDRs.
- Confirm your on-premises router is configured to allow traffic to and from the EKS control plane.

Unauthorized

If your hybrid node was able to communicate with the EKS control plane but was not able to register, you may see logs similar to the following. Note the key difference in the log messages below is the `Unauthorized` error. This signals that the node was not able to perform its tasks because it does not have the required permissions.

```
"Failed to ensure lease exists, will retry" err="Unauthorized"
```

```
"Unable to register node with API server" err="Unauthorized"
```

```
Failed to contact API server when waiting for CSINode publishing: Unauthorized
```

If you see these messages, check the following to ensure it meets the hybrid nodes requirements details in the section called “Prepare credentials” and the section called “Prepare cluster access”.

- Confirm the identity of the hybrid nodes matches your expected Hybrid Nodes IAM role. This can be done by running `sudo aws sts get-caller-identity` from your hybrid nodes.
- Confirm your Hybrid Nodes IAM role has the required permissions.
- Confirm that in your cluster you have an EKS access entry for your Hybrid Nodes IAM role or confirm that your `aws-auth` ConfigMap has an entry for your Hybrid Nodes IAM role. If you are using EKS access entries, confirm your access entry for your Hybrid Nodes IAM role has the `HYBRID_LINUX` access type. If you are using the `aws-auth` ConfigMap, confirm your entry for

the Hybrid Nodes IAM role meets the requirements and formatting detailed in the section called "Prepare cluster access".

Hybrid nodes registered with EKS cluster but show status `Not Ready`

If your hybrid nodes successfully registered with your EKS cluster, but the hybrid nodes show status `Not Ready`, the first thing to check is your Container Networking Interface (CNI) status. If you have not installed a CNI, then it is expected that your hybrid nodes have status `Not Ready`. Once a CNI is installed and running successfully, nodes transition to have the status Ready. If you attempted to install a CNI but it is not running successfully, see the section called "Hybrid nodes CNI troubleshooting" on this page.

Certificate Signing Requests (CSRs) are stuck Pending

After connecting hybrid nodes to your EKS cluster, if you see that there are pending CSRs for your hybrid nodes, your hybrid nodes are not meeting the requirements for automatic approval. CSRs for hybrid nodes are automatically approved and issued if the CSRs for hybrid nodes were created by a node with `eks.amazonaws.com/compute-type: hybrid` label, and the CSR has the following Subject Alternative Names (SANs): at least one DNS SAN equal to the node name and the IP SANs belong to the remote node network CIDRs.

Hybrid profile already exists

If you changed your `nodeadm` configuration and attempt to re-register the node with the new configuration, you may see an error that states that the hybrid profile already exists but its contents have changed. Instead of running `nodeadm init` in between configuration changes, run `nodeadm uninstall` followed by a `nodeadm install` and `nodeadm init`. This ensures a proper clean up with the changes in configuration.

```
"msg":"Command failed","error":"hybrid profile already exists at /etc/aws/hybrid/config
 but its contents do not align with the expected configuration"
```

Hybrid node failed to resolve Private API

After running `nodeadm init`, if you see an error in the `kubelet` logs that shows failures to contact the EKS Kubernetes API server because there is `no such host`, you may have to change your DNS entry for the EKS Kubernetes API endpoint in your on-premises network or at the host level. See Forwarding inbound DNS queries to your VPC in the *AWS Route53 documentation*.

```
Failed to contact API server when waiting for CSINode publishing: Get ... no such host
```

Can’t view hybrid nodes in the EKS console

If you have registered your hybrid nodes but are unable to view them in the EKS console, check the permissions of the IAM principal you are using to view the console. The IAM principal you’re using must have specific minimum IAM and Kubernetes permissions to view resources in the console. For more information, see the section called “Access cluster resources”.

Running hybrid nodes troubleshooting

If your hybrid nodes registered with your EKS cluster, had status `Ready`, and then transitioned to status `Not Ready`, there are a wide range of issues that may have contributed to the unhealthy status such as the node lacking sufficient resources for CPU, memory, or available disk space, or the node is disconnected from the EKS control plane. You can use the steps below to troubleshoot your nodes, and if you cannot resolve the issue, contact AWS Support.

Run `nodeadm debug` from your hybrid nodes to validate networking and credential requirements are met. For more information on the `nodeadm debug` command, see the section called “Hybrid nodes nodeadm”.

Get node status

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

Check node conditions and events

```
kubectl describe node NODE_NAME
```

Get pod status

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

Check pod conditions and events

```
kubectl describe pod POD_NAME
```

Check pod logs

```
kubectl logs POD_NAME
```

Check kubectl logs

```
systemctl status kubelet
```

```
journalctl -u kubelet -f
```

Pod liveness probes failing or webhooks are not working

If applications, add-ons, or webhooks running on your hybrid nodes are not starting properly, you may have networking issues that block the communication to the pods. For the EKS control plane to contact webhooks running on hybrid nodes, you must configure your EKS cluster with a remote pod network and have routes for your on-premises pod CIDR in your VPC routing table with the target as your Transit Gateway (TGW), virtual private gateway (VPW), or other gateway you are using to connect your VPC with your on-premises network. For more information on the networking requirements for hybrid nodes, see the section called "Prepare networking". You additionally must allow this traffic in your on-premises firewall and ensure your router can properly route to your pods.

A common pod log message for this scenario is shown below the following where ip-address is the Cluster IP for the Kubernetes service.

```
dial tcp <ip-address>:443: connect: no route to host
```

Hybrid nodes CNI troubleshooting

If you run into issues with initially starting Cilium or Calico with hybrid nodes, it is most often due to networking issues between hybrid nodes or the CNI pods running on hybrid nodes, and the EKS control plane. Make sure your environment meets the requirements in Prepare networking for hybrid nodes. It's useful to break down the problem into parts.

EKS cluster configuration

Are the RemoteNodeNetwork and RemotePodNetwork configurations correct?

VPC configuration

Are there routes for the RemoteNodeNetwork and RemotePodNetwork in the VPC routing table with the target of the Transit Gateway or Virtual Private Gateway?

Security group configuration

Are there inbound and outbound rules for the RemoteNodeNetwork and RemotePodNetwork ?

On-premises network

Are there routes and access to/from the EKS control plane and to/from the hybrid nodes and the pods running on hybrid nodes?

CNI configuration

If using an overlay network, does the IP pool configuration for the CNI match the RemotePodNetwork configured for the EKS cluster if using webhooks?

Hybrid node has status Ready without a CNI installed

If your hybrid nodes are showing status `Ready`, but you have not installed a CNI on your cluster, it is possible that there are old CNI artifacts on your hybrid nodes. By default, when you uninstall Cilium and Calico with tools such as Helm, the on-disk resources are not removed from your physical or virtual machines. Additionally, the Custom Resource Definitions (CRDs) for these CNIs may still be present on your cluster from an old installation. For more information, see the Delete Cilium and Delete Calico sections of the section called “Configure CNI”.

Cilium troubleshooting

If you are having issues running Cilium on hybrid nodes, see the troubleshooting steps in the Cilium documentation. The sections below cover issues that may be unique to deploying Cilium on hybrid nodes.

Cilium isn’t starting

If the Cilium agents that run on each hybrid node are not starting, check the logs of the Cilium agent pods for errors. The Cilium agent requires connectivity to the EKS Kubernetes API endpoint to start. Cilium agent startup will fail if this connectivity is not correctly configured. In this case, you will see log messages similar to the following in the Cilium agent pod logs.

```
msg="Unable to contact k8s api-server"
```

```
level=fatal msg="failed to start: Get \"https://<k8s-cluster-ip>:443/api/v1/namespaces/
kube-system\": dial tcp <k8s-cluster-ip>:443: i/o timeout"
```

The Cilium agent runs on the host network. Your EKS cluster must be configured with `RemoteNodeNetwork` for the Cilium connectivity. Confirm you have an additional security group for your EKS cluster with an inbound rule for your `RemoteNodeNetwork`, that you have routes in your VPC for your `RemodeNodeNetwork`, and that your on-premises network is configured correctly to allow connectivity to the EKS control plane.

If the Cilium operator is running and some of your Cilium agents are running but not all, confirm that you have available pod IPs to allocate for all nodes in your cluster. You configure the size of your allocatable Pod CIDRs when using cluster pool IPAM with `clusterPoolIPv4PodCIDRList` in your Cilium configuration. The per-node CIDR size is configured with the `clusterPoolIPv4MaskSize` setting in your Cilium configuration. See [Expanding the cluster pool] in the Cilium documentation for more information.

Cilium BGP is not working

If you are using Cilium BGP Control Plane to advertise your pod or service addresses to your on-premises network, you can use the following Cilium CLI commands to check if BGP is advertising the routes to your resources. For steps to install the Cilium CLI, see [Install the Cilium CLI] in the Cilium documentation.

If BGP is working correctly, you should your hybrid nodes with Session State `established` in the output. You may need to work with your networking team to identify the correct values for your environment's Local AS, Peer AS, and Peer Address.

```
cilium bgp peers
```

```
cilium bgp routes
```

If you are using Cilium BGP to advertise the IPs of Services with type `LoadBalancer`, you must have the same label on both your `CiliumLoadBalancerIPPool` and Service, which should be used in the selector of your `CiliumBGPAdvertisement`. An example is shown below. Note, if you are using Cilium BGP to advertise the IPs of Services with type LoadBalancer, the BGP routes may be disrupted during Cilium agent restart. For more information, see [Failure Scenarios] in the Cilium documentation.

Service

```
kind: Service
apiVersion: v1
metadata:
  name: guestbook
  labels:
    app: guestbook
spec:
  ports:
  - port: 3000
    targetPort: http-server
  selector:
    app: guestbook
  type: LoadBalancer
```

CiliumLoadBalancerIPPool

```
apiVersion: cilium.io/v2alpha1
kind: CiliumLoadBalancerIPPool
metadata:
  name: guestbook-pool
  labels:
    app: guestbook
spec:
  blocks:
  - cidr: <CIDR to advertise>
  serviceSelector:
    matchExpressions:
      - { key: app, operator: In, values: [ guestbook ] }
```

CiliumBGPAdvertisement

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: bgp-advertisements-guestbook
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "Service"
      service:
        addresses:
```

```
          - ExternalIP
          - LoadBalancerIP
      selector:
        matchExpressions:
          - { key: app, operator: In, values: [ guestbook ] }
```

Calico troubleshooting

If you are having issues running Calico on hybrid nodes, see the troubleshooting steps in the Calico documentation. The sections below cover issues that may be unique to deploying Calico on hybrid nodes.

The table below summarizes the Calico components and whether they run on the node or pod network by default. If you configured Calico to use NAT for outgoing pod traffic, your on-premises network must be configured to route traffic to your on-premises node CIDR and your VPC routing tables must be configured with a route for your on-premises node CIDR with your transit gateway (TGW) or virtual private gateway (VGW) as the target. If you are not configuring Calico to use NAT for outgoing pod traffic, your on-premises network must be configured to route traffic to your on-premises pod CIDR and your VPC routing tables must be configured with a route for your on-premises pod CIDR with your transit gateway (TGW) or virtual private gateway (VGW) as the target.

Component	Network
Calico API server	Node
Calico kube controllers	Pod
Calico node agent	Node
Calico typha	Node
Calico CSI node driver	Pod
Calico operator	Node

Calico resources are scheduled or running on cordoned nodes

The Calico resources that don't run as a DaemonSet have flexible tolerations by default that enable them to be scheduled on cordoned nodes that are not ready for scheduling or running pods. You can tighten the tolerations for the non-DaemonSet Calico resources by changing your operator installation to include the following.

```
installation:
  ...
  controlPlaneTolerations:
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  calicoKubeControllersDeployment:
    spec:
      template:
        spec:
          tolerations:
          - effect: NoExecute
            key: node.kubernetes.io/unreachable
            operator: Exists
            tolerationSeconds: 300
          - effect: NoExecute
            key: node.kubernetes.io/not-ready
            operator: Exists
            tolerationSeconds: 300
  typhaDeployment:
    spec:
      template:
        spec:
          tolerations:
          - effect: NoExecute
            key: node.kubernetes.io/unreachable
            operator: Exists
            tolerationSeconds: 300
          - effect: NoExecute
            key: node.kubernetes.io/not-ready
            operator: Exists
            tolerationSeconds: 300
```

Credentials troubleshooting

For both AWS SSM hybrid activations and AWS IAM Roles Anywhere, you can validate that credentials for the Hybrid Nodes IAM role are correctly configured on your hybrid nodes by running the following command from your hybrid nodes. Confirm the node name and Hybrid Nodes IAM Role name are what you expect.

```
sudo aws sts get-caller-identity
```

```
{
    "UserId": "ABCDEFGHIJKLM12345678910:<node-name>",
    "Account": "<aws-account-id>",
    "Arn": "arn:aws:sts::<aws-account-id>:assumed-role/<hybrid-nodes-iam-role/<node-
name>"
}
```

AWS Systems Manager (SSM) troubleshooting

If you are using AWS SSM hybrid activations for your hybrid nodes credentials, be aware of the following SSM directories and artifacts that are installed on your hybrid nodes by nodeadm. For more information on the SSM agent, see Working with the SSM agent in the *AWS Systems Manager User Guide*.

Description	Location
SSM agent	Ubuntu - `/snap/amazon-ssm-agent/current/amazon-ssm-agent` RHEL & AL2023 - `/usr/bin/amazon-ssm-agent`
SSM agent logs	`/var/log/amazon/ssm`
AWS credentials	`/root/.aws/credentials`
SSM Setup CLI	`/opt/ssm/ssm-setup-cli`

Restarting the SSM agent

Some issues can be resolved by restarting the SSM agent. You can use the commands below to restart it.

AL2023 and other operating systems

```
systemctl restart amazon-ssm-agent
```

Ubuntu

```
systemctl restart snap.amazon-ssm-agent.amazon-ssm-agent
```

Check connectivity to SSM endpoints

Confirm you can connect to the SSM endpoints from your hybrid nodes. For a list of the SSM endpoints, see AWS Systems Manager endpoints and quotas. Replace `us-west-2` in the command below with the AWS Region for your AWS SSM hybrid activation.

```
ping ssm.us-west-2.amazonaws.com
```

View connection status of registered SSM instances

You can check the connection status of the instances that are registered with SSM hybrid activations with the following AWS CLI command. Replace the machine ID with the machine ID of your instance.

```
aws ssm get-connection-status --target mi-012345678abcdefgh
```

SSM Setup CLI checksum mismatch

When running `nodeadm install` if you see an issue with the `ssm-setup-cli` checksum mismatch you should confirm there are not older existing SSM installations on your host. If there are older SSM installations on your host, remove them and re-run `nodeadm install` to resolve the issue.

```
Failed to perform agent-installation/on-prem registration: error while verifying
 installed ssm-setup-cli checksum: checksum mismatch with latest ssm-setup-cli.
```

SSM `InvalidActivation`

If you see an error registering your instance with AWS SSM, confirm the `region`, `activationCode`, and `activationId` in your `nodeConfig.yaml` are correct. The AWS Region for your EKS cluster must match the region of your SSM hybrid activation. If these values are misconfigured, you may see an error similar to the following.

```
ERROR Registration failed due to error registering the instance with AWS SSM.
 InvalidActivation
```

SSM `ExpiredTokenException`: The security token included in the request is expired

If the SSM agent is not able to refresh credentials, you may see an `ExpiredTokenException`. In this scenario, if you are able to connect to the SSM endpoints from your hybrid nodes, you may need to restart the SSM agent to force a credential refresh.

```
"msg":"Command failed","error":"operation error SSM: DescribeInstanceInformation, https
 response error StatusCode: 400, RequestID: eee03a9e-f7cc-470a-9647-73d47e4cf0be, api
 error ExpiredTokenException: The security token included in the request is expired"
```

SSM error in running register machine command

If you see an error registering the machine with SSM, you may need to re-run `nodeadm install` to make sure all of the SSM dependencies are properly installed.

```
"error":"running register machine command: , error: fork/exec /opt/aws/ssm-setup-cli:
 no such file or directory"
```

SSM `ActivationExpired`

When running `nodeadm init`, if you see an error registering the instance with SSM due to an expired activation, you need to create a new SSM hybrid activation, update your `nodeConfig.yaml` with the `activationCode` and `activationId` of your new SSM hybrid activation, and re-run `nodeadm init`.

```
"msg":"Command failed","error":"SSM activation expired. Please use a valid activation"
```

```
ERROR Registration failed due to error registering the instance with AWS SSM.
 ActivationExpired
```

SSM failed to refresh cached credentials

If you see a failure to refresh cached credentials, the `/root/.aws/credentials` file may have been deleted on your host. First check your SSM hybrid activation and ensure it is active and your hybrid nodes are configured correctly to use the activation. Check the SSM agent logs at `/var/log/amazon/ssm` and re-run the nodeadm init command once you have resolved the issue on the SSM side.

```
"Command failed","error":"operation error SSM: DescribeInstanceInformation, get
 identity: get credentials: failed to refresh cached credentials"
```

Clean up SSM

To remove the SSM agent from your host, you can run the following commands.

```
dnf remove -y amazon-ssm-agent
sudo apt remove --purge amazon-ssm-agent
snap remove amazon-ssm-agent
rm -rf /var/lib/amazon/ssm/Vault/Store/RegistrationKey
```

AWS IAM Roles Anywhere troubleshooting

If you are using AWS IAM Roles Anywhere for your hybrid nodes credentials, be aware of the following directories and artifacts that are installed on your hybrid nodes by nodeadm. For more information on the troubleshooting IAM Roles Anywhere, see Troubleshooting AWS IAM Roles Anywhere identity and access in the *AWS IAM Roles Anywhere User Guide*.

Description	Location
IAM Roles Anywhere CLI	`/usr/local/bin/aws_signing_helper`
Default certificate location and name	`/etc/iam/pki/server.pem`
Default key location and name	`/etc/iam/pki/server.key`

IAM Roles Anywhere failed to refresh cached credentials

If you see a failure to refresh cached credentials, review the contents of `/etc/aws/hybrid/config` and confirm that IAM Roles Anywhere was configured correctly in your `nodeadm` configuration. Confirm that `/etc/iam/pki` exists. Each node must have a unique certificate and key. By default, when using IAM Roles Anywhere as the credential provider, `nodeadm` uses `/etc/`

iam/pki/server.pem for the certificate location and name, and /etc/iam/pki/server.key for the private key. You may need to create the directories before placing the certificates and keys in the directories with sudo mkdir -p /etc/iam/pki. You can verify the content of your certificate with the command below.

```
openssl x509 -text -noout -in server.pem
```

```
open /etc/iam/pki/server.pem: no such file or directory
could not parse PEM data
Command failed {"error": "... get identity: get credentials: failed to refresh cached
 credentials, process provider error: error in credential_process: exit status 1"}
```

IAM Roles Anywhere not authorized to perform sts:AssumeRole

In the kubelet logs, if you see an access denied issue for the sts:AssumeRole operation when using IAM Roles Anywhere, check the trust policy of your Hybrid Nodes IAM role to confirm the IAM Roles Anywhere service principal is allowed to assume the Hybrid Nodes IAM Role. Additionally confirm that the trust anchor ARN is configured properly in your Hybrid Nodes IAM role trust policy and that your Hybrid Nodes IAM role is added to your IAM Roles Anywhere profile.

```
could not get token: AccessDenied: User: ... is not authorized to perform:
 sts:AssumeRole on resource: ...
```

IAM Roles Anywhere not authorized to set roleSessionName

In the kubelet logs, if you see an access denied issue for setting the roleSessionName, confirm you have set acceptRoleSessionName to true for your IAM Roles Anywhere profile.

```
AccessDeniedException: Not authorized to set roleSessionName
```

Operating system troubleshooting

RHEL

Entitlement or subscription manager registration failures

If you are running nodeadm install and encounter a failure to install the hybrid nodes dependencies due to entitlement registration issues, ensure you have properly set your Red Hat username and password on your host.

```
This system is not registered with an entitlement server
```

GLIBC not found

If you are using Ubuntu for your operating system and IAM Roles Anywhere for your credential provider with hybrid nodes and see an issue with GLIBC not found, you can install that dependency manually to resolve the issue.

```
GLIBC_2.32 not found (required by /usr/local/bin/aws_signing_helper)
```

Run the following commands to install the dependency:

```
ldd --version
sudo apt update && apt install libc6
sudo apt install glibc-source
```

Store application data for your cluster

This chapter covers storage options for Amazon EKS clusters.

Topics

Store Kubernetes volumes with Amazon EBS

> **Note**
>
> **New:** Amazon EKS Auto Mode automates routine tasks for block storage. Learn how to the section called "Deploy stateful workload".

The Amazon Elastic Block Store (Amazon EBS) Container Storage Interface (CSI) driver manages the lifecycle of Amazon EBS volumes as storage for the Kubernetes Volumes that you create. The Amazon EBS CSI driver makes Amazon EBS volumes for these types of Kubernetes volumes: generic ephemeral volumes and persistent volumes.

Considerations

- You do not need to install the Amazon EBS CSI controller on EKS Auto Mode clusters.
- You can't mount Amazon EBS volumes to Fargate Pods.
- You can run the Amazon EBS CSI controller on Fargate nodes, but the Amazon EBS CSI node `DaemonSet` can only run on Amazon EC2 instances.

- Amazon EBS volumes and the Amazon EBS CSI driver are not compatible with Amazon EKS Hybrid Nodes.
- Support will be provided for the latest add-on version and one prior version. Bugs or vulnerabilities found in the latest version will be backported to the previous release in a new minor version.
- Only platform versions created from a storage class using `ebs.csi.eks.amazonaws.com` as the provisioner can be mounted on nodes created by EKS Auto Mode. Existing platform versions must be migrated to the new storage class using a volume snapshot.

> **Important**
>
> To use the snapshot functionality of the Amazon EBS CSI driver, you must first install the CSI snapshot controller. For more information, see the section called "CSI snapshot controller".

Prerequisites

- An existing cluster. To see the required platform version, run the following command.

```
aws eks describe-addon-versions --addon-name aws-ebs-csi-driver
```

- The EBS CSI driver needs AWS IAM Permissions.
  - AWS suggests using EKS Pod Identities. For more information, see the section called "Overview of setting up EKS Pod Identities".
  - For information about IAM Roles for Service Accounts, see the section called "IAM OIDC provider".
- If you're using a cluster wide restricted PodSecurityPolicy, make sure that the add-on is granted sufficient permissions to be deployed. For the permissions required by each add-on Pod, see the relevant add-on manifest definition on GitHub.

Step 1: Create an IAM role

The Amazon EBS CSI plugin requires IAM permissions to make calls to AWS APIs on your behalf. If you don't do these steps, attempting to install the add-on and running `kubectl describe pvc` will show `failed to provision volume with StorageClass` along with a `could not` `create volume in EC2: UnauthorizedOperation` error. For more information, see Set up driver permission on GitHub.

> **Note**
>
> Pods will have access to the permissions that are assigned to the IAM role unless you block access to IMDS. For more information, see the section called "Best practices".

The following procedure shows you how to create an IAM role and attach the AWS managed policy to it. To implement this procedure, you can use one of these tools:

- the section called "`eksctl`"
- the section called "AWS Management Console"
- the section called "AWS CLI"

> **Note**
>
> The specific steps in this procedure are written for using the driver as an Amazon EKS add-on. Different steps are needed to use the driver as a self-managed add-on. For more information, see Set up driver permissions on GitHub.

`eksctl`

1. Create an IAM role and attach a policy. AWS maintains an AWS managed policy or you can create your own custom policy. You can create an IAM role and attach the AWS managed policy with the following command. Replace *`my-cluster`* with the name of your cluster. The command deploys an AWS CloudFormation stack that creates an IAM role and attaches the IAM policy to it.

```
eksctl create iamserviceaccount \
        --name ebs-csi-controller-sa \
        --namespace kube-system \
        --cluster my-cluster \
        --role-name AmazonEKS_EBS_CSI_DriverRole \
        --role-only \
        --attach-policy-arn arn:aws:iam::aws:policy/service-role/
AmazonEBSCSIDriverPolicy \
```

```
        --approve
```

2. If you use a custom KMS key for encryption on your Amazon EBS volumes, customize the IAM role as needed. For example, do the following:

   a. Copy and paste the following code into a new `kms-key-for-encryption-on-ebs.json` file. Replace *custom-key-arn* with the custom KMS key ARN.

   ```
   {
         "Version": "2012-10-17",
         "Statement": [
           {
             "Effect": "Allow",
             "Action": [
               "kms:CreateGrant",
               "kms:ListGrants",
               "kms:RevokeGrant"
             ],
             "Resource": ["custom-key-arn"],
             "Condition": {
               "Bool": {
                 "kms:GrantIsForAWSResource": "true"
               }
             }
           },
           {
             "Effect": "Allow",
             "Action": [
               "kms:Encrypt",
               "kms:Decrypt",
               "kms:ReEncrypt*",
               "kms:GenerateDataKey*",
               "kms:DescribeKey"
             ],
             "Resource": ["custom-key-arn"]
           }
         ]
       }
   ```

   b. Create the policy. You can change *KMS_Key_For_Encryption_On_EBS_Policy* to a different name. However, if you do, make sure to change it in later steps, too.

   ```
   aws iam create-policy \
   ```

```
        --policy-name KMS_Key_For_Encryption_On_EBS_Policy \
        --policy-document file://kms-key-for-encryption-on-ebs.json
```

c. Attach the IAM policy to the role with the following command. Replace *111122223333* with your account ID.

```
aws iam attach-role-policy \
      --policy-arn arn:aws:iam::111122223333:policy/
KMS_Key_For_Encryption_On_EBS_Policy \
      --role-name AmazonEKS_EBS_CSI_DriverRole
```

AWS Management Console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Roles**.
3. On the **Roles** page, choose **Create role**.
4. On the **Select trusted entity** page, do the following:
   a. In the **Trusted entity type** section, choose **Web identity**.
   b. For **Identity provider**, choose the **OpenID Connect provider URL** for your cluster (as shown under **Overview** in Amazon EKS).
   c. For **Audience**, choose `sts.amazonaws.com`.
   d. Choose **Next**.
5. On the **Add permissions** page, do the following:
   a. In the **Filter policies** box, enter `AmazonEBSCSIDriverPolicy`.
   b. Select the check box to the left of the `AmazonEBSCSIDriverPolicy` returned in the search.
   c. Choose **Next**.
6. On the **Name, review, and create** page, do the following:
   a. For **Role name**, enter a unique name for your role, such as *AmazonEKS_EBS_CSI_DriverRole*.
   b. Under **Add tags (Optional)**, add metadata to the role by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM resources](#) in the *IAM User Guide*.
   c. Choose **Create role**.
7. After the role is created, choose the role in the console to open it for editing.
8. Choose the **Trust relationships** tab, and then choose **Edit trust policy**.

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

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

   Add a comma to the end of the previous line, and then add the following line after the previous line. Replace *region-code* with the AWS Region that your cluster is in. Replace *EXAMPLED539D4633E53DE1B71EXAMPLE* with your cluster's OIDC provider ID.

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":
 "system:serviceaccount:kube-system:ebs-csi-controller-sa"
```

10. Choose **Update policy** to finish.

11. If you use a custom KMS key for encryption on your Amazon EBS volumes, customize the IAM role as needed. For example, do the following:

    a. In the left navigation pane, choose **Policies**.

    b. On the **Policies** page, choose **Create Policy**.

    c. On the **Create policy** page, choose the **JSON** tab.

    d. Copy and paste the following code into the editor, replacing *custom-key-arn* with the custom KMS key ARN.

```
{
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "kms:CreateGrant",
            "kms:ListGrants",
            "kms:RevokeGrant"
          ],
          "Resource": ["custom-key-arn"],
          "Condition": {
            "Bool": {
              "kms:GrantIsForAWSResource": "true"
            }
          }
        },
        {
```

```
          "Effect": "Allow",
          "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:DescribeKey"
          ],
          "Resource": ["custom-key-arn"]
        }
      ]
    }
```

e. Choose **Next: Tags**.

f. On the **Add tags (Optional)** page, choose **Next: Review**.

g. For **Name**, enter a unique name for your policy (for example, *KMS_Key_For_Encryption_On_EBS_Policy*).

h. Choose **Create policy**.

i. In the left navigation pane, choose **Roles**.

j. Choose the ***AmazonEKS_EBS_CSI_DriverRole*** in the console to open it for editing.

k. From the **Add permissions** dropdown list, choose **Attach policies**.

l. In the **Filter policies** box, enter *KMS_Key_For_Encryption_On_EBS_Policy*.

m. Select the check box to the left of the *KMS_Key_For_Encryption_On_EBS_Policy* that was returned in the search.

n. Choose **Attach policies**.

AWS CLI

1. View your cluster's OIDC provider URL. Replace *my-cluster* with your cluster name. If the output from the command is None, review the **Prerequisites**.

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer" --
output text
```

An example output is as follows.

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

2. Create the IAM role, granting the `AssumeRoleWithWebIdentity` action.

   a. Copy the following contents to a file that's named `aws-ebs-csi-driver-trust-policy.json`. Replace *111122223333* with your account ID. Replace *EXAMPLED539D4633E53DE1B71EXAMPLE* and *region-code* with the values returned in the previous step.

```
{
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
          },
          "Action": "sts:AssumeRoleWithWebIdentity",
          "Condition": {
            "StringEquals": {
              "oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
              "oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:ebs-csi-controller-sa"
            }
          }
        }
      ]
    }
```

   b. Create the role. You can change *AmazonEKS_EBS_CSI_DriverRole* to a different name. If you change it, make sure to change it in later steps.

```
aws iam create-role \
      --role-name AmazonEKS_EBS_CSI_DriverRole \
      --assume-role-policy-document file://"aws-ebs-csi-driver-trust-policy.json"
```

3. Attach a policy. AWS maintains an AWS managed policy or you can create your own custom policy. Attach the AWS managed policy to the role with the following command.

```
aws iam attach-role-policy \
      --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
```

```
    --role-name AmazonEKS_EBS_CSI_DriverRole
```

4. If you use a custom KMS key for encryption on your Amazon EBS volumes, customize the IAM role as needed. For example, do the following:

   a. Copy and paste the following code into a new `kms-key-for-encryption-on-ebs.json` file. Replace *`custom-key-arn`* with the custom KMS key ARN.

   ```
   {
         "Version": "2012-10-17",
         "Statement": [
           {
             "Effect": "Allow",
             "Action": [
               "kms:CreateGrant",
               "kms:ListGrants",
               "kms:RevokeGrant"
             ],
             "Resource": ["custom-key-arn"],
             "Condition": {
               "Bool": {
                 "kms:GrantIsForAWSResource": "true"
               }
             }
           },
           {
             "Effect": "Allow",
             "Action": [
               "kms:Encrypt",
               "kms:Decrypt",
               "kms:ReEncrypt*",
               "kms:GenerateDataKey*",
               "kms:DescribeKey"
             ],
             "Resource": ["custom-key-arn"]
           }
         ]
       }
   ```

   b. Create the policy. You can change *`KMS_Key_For_Encryption_On_EBS_Policy`* to a different name. However, if you do, make sure to change it in later steps, too.

   ```
   aws iam create-policy \
   ```

```
      --policy-name KMS_Key_For_Encryption_On_EBS_Policy \
      --policy-document file://kms-key-for-encryption-on-ebs.json
```

c. Attach the IAM policy to the role with the following command. Replace *111122223333* with your account ID.

```
aws iam attach-role-policy \
      --policy-arn arn:aws:iam::111122223333:policy/
KMS_Key_For_Encryption_On_EBS_Policy \
      --role-name AmazonEKS_EBS_CSI_DriverRole
```

Now that you have created the Amazon EBS CSI driver IAM role, you can continue to the next section. When you deploy the add-on with this IAM role, it creates and is configured to use a service account that's named `ebs-csi-controller-sa`. The service account is bound to a Kubernetes `clusterrole` that's assigned the required Kubernetes permissions.

Step 2: Get the Amazon EBS CSI driver

We recommend that you install the Amazon EBS CSI driver through the Amazon EKS add-on to improve security and reduce the amount of work. To add an Amazon EKS add-on to your cluster, see the section called "Create an add-on". For more information about add-ons, see the section called "Amazon EKS add-ons".

> **Important**
>
> Before adding the Amazon EBS driver as an Amazon EKS add-on, confirm that you don't have a self-managed version of the driver installed on your cluster. If so, see Uninstalling a self-managed Amazon EBS CSI driver on GitHub.

Alternatively, if you want a self-managed installation of the Amazon EBS CSI driver, see Installation on GitHub.

Step 3: Deploy a sample application

You can deploy a variety of sample apps and modify them as needed. For more information, see Kubernetes Examples on GitHub.

Store an elastic file system with Amazon EFS

Amazon Elastic File System (Amazon EFS) provides serverless, fully elastic file storage so that you can share file data without provisioning or managing storage capacity and performance. The Amazon EFS Container Storage Interface (CSI) driver provides a CSI interface that allows Kubernetes clusters running on AWS to manage the lifecycle of Amazon EFS file systems. This topic shows you how to deploy the Amazon EFS CSI driver to your Amazon EKS cluster.

Considerations

- The Amazon EFS CSI driver isn't compatible with Windows-based container images.
- You can't use dynamic provisioning for persistent volumes with Fargate nodes, but you can use static provisioning.
- Dynamic provisioning requires 1.2 or later of the driver. You can use static provisioning for persistent volumes using version `1.1` of the driver on any supported Amazon EKS cluster version (see the section called "Kubernetes versions").
- Version 1.3.2 or later of this driver supports the Arm64 architecture, including Amazon EC2 Graviton-based instances.
- Version 1.4.2 or later of this driver supports using FIPS for mounting file systems.
- Take note of the resource quotas for Amazon EFS. For example, there's a quota of 1000 access points that can be created for each Amazon EFS file system. For more information, see Amazon EFS resource quotas that you cannot change.
- Starting in version 2.0.0, this driver switched from using `stunnel` to `efs-proxy` for TLS connections. When `efs-proxy` is used, it will open a number of threads equal to one plus the number of cores for the node it's running on.
- The Amazon EFS CSI driver isn't compatible with Amazon EKS Hybrid Nodes.

Prerequisites

- An existing AWS Identity and Access Management (IAM) OpenID Connect (OIDC) provider for your cluster. To determine whether you already have one, or to create one, see the section called "IAM OIDC provider".
- Version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version, use `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`,

apt-get, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*. The AWS CLI version that is installed in AWS CloudShell might also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the *AWS CloudShell User Guide*.

- The `kubectl` command line tool is installed on your device or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. For example, if your cluster version is `1.29`, you can use `kubectl` version `1.28`, `1.29`, or `1.30` with it. To install or upgrade `kubectl`, see the section called "Set up kubectl and eksctl".

> **Note**
>
> A Pod running on Fargate automatically mounts an Amazon EFS file system, without needing manual driver installation steps.

Step 1: Create an IAM role

The Amazon EFS CSI driver requires IAM permissions to interact with your file system. Create an IAM role and attach the required AWS managed policy to it. To implement this procedure, you can use one of these tools:

- the section called "`eksctl`"
- the section called "AWS Management Console"
- the section called "AWS CLI"

> **Note**
>
> The specific steps in this procedure are written for using the driver as an Amazon EKS add-on. For details on self-managed installations, see Set up driver permission on GitHub.

`eksctl`

Run the following commands to create an IAM role with `eksctl`. Replace *`my-cluster`* with your cluster name and *`AmazonEKS_EFS_CSI_DriverRole`* with the name for your role.

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
eksctl create iamserviceaccount \
    --name efs-csi-controller-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name $role_name \
    --role-only \
    --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
    --approve
TRUST_POLICY=$(aws iam get-role --role-name $role_name --query
 'Role.AssumeRolePolicyDocument' | \
    sed -e 's/efs-csi-controller-sa/efs-csi-*/' -e 's/StringEquals/StringLike/')
aws iam update-assume-role-policy --role-name $role_name --policy-document
 "$TRUST_POLICY"
```

AWS Management Console

Run the following to create an IAM role with AWS Management Console.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Roles**.
3. On the **Roles** page, choose **Create role**.
4. On the **Select trusted entity** page, do the following:
   a. In the **Trusted entity type** section, choose **Web identity**.
   b. For **Identity provider**, choose the **OpenID Connect provider URL** for your cluster (as shown under **Overview** in Amazon EKS).
   c. For **Audience**, choose `sts.amazonaws.com`.
   d. Choose **Next**.
5. On the **Add permissions** page, do the following:
   a. In the **Filter policies** box, enter *`AmazonEFSCSIDriverPolicy`*.
   b. Select the check box to the left of the *`AmazonEFSCSIDriverPolicy`* returned in the search.
   c. Choose **Next**.
6. On the **Name, review, and create** page, do the following:
   a. For **Role name**, enter a unique name for your role, such as *`AmazonEKS_EFS_CSI_DriverRole`*.

b. Under **Add tags (Optional)**, add metadata to the role by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM resources in the *IAM User Guide*.

c. Choose **Create role**.

7. After the role is created, choose the role in the console to open it for editing.
8. Choose the **Trust relationships** tab, and then choose **Edit trust policy**.
9. Find the line that looks similar to the following line:

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

Add the following line above the previous line. Replace *region-code* with the AWS Region that your cluster is in. Replace *EXAMPLED539D4633E53DE1B71EXAMPLE* with your cluster's OIDC provider ID.

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":
 "system:serviceaccount:kube-system:efs-csi-*",
```

10. Modify the `Condition` operator from `"StringEquals"` to `"StringLike"`.
11. Choose **Update policy** to finish.

AWS CLI

Run the following commands to create an IAM role with AWS CLI.

1. View your cluster's OIDC provider URL. Replace *my-cluster* with your cluster name. If the output from the command is `None`, review the **Prerequisites**.

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer" --
output text
```

An example output is as follows.

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

2. Create the IAM role that grants the `AssumeRoleWithWebIdentity` action.

a. Copy the following contents to a file named `aws-efs-csi-driver-trust-policy.json`. Replace *111122223333* with your account ID. Replace

EXAMPLED539D4633E53DE1B71EXAMPLE and *region-code* with the values returned in the previous step.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.region-
code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringLike": {
          "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:efs-csi-
*",
          "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
        }
      }
    }
  ]
}
```

b. Create the role. You can change *AmazonEKS_EFS_CSI_DriverRole* to a different name, but if you do, make sure to change it in later steps too.

```
aws iam create-role \
  --role-name AmazonEKS_EFS_CSI_DriverRole \
  --assume-role-policy-document file://"aws-efs-csi-driver-trust-policy.json"
```

3. Attach the required AWS managed policy to the role with the following command.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
  --role-name AmazonEKS_EFS_CSI_DriverRole
```

Step 2: Get the Amazon EFS CSI driver

We recommend that you install the Amazon EFS CSI driver through the Amazon EKS add-on. To add an Amazon EKS add-on to your cluster, see the section called "Create an add-on". For more information about add-ons, see the section called "Amazon EKS add-ons". If you're unable to use the Amazon EKS add-on, we encourage you to submit an issue about why you can't to the Containers roadmap GitHub repository.

Alternatively, if you want a self-managed installation of the Amazon EFS CSI driver, see Installation on GitHub.

Step 3: Create an Amazon EFS file system

To create an Amazon EFS file system, see Create an Amazon EFS file system for Amazon EKS on GitHub.

Step 4: Deploy a sample application

You can deploy a variety of sample apps and modify them as needed. For more information, see Examples on GitHub.

Store high-performance apps with FSx for Lustre

The FSx for Lustre Container Storage Interface (CSI) driver provides a CSI interface that allows Amazon EKS clusters to manage the lifecycle of FSx for Lustre file systems. For more information, see the FSx for Lustre User Guide.

This topic shows you how to deploy the FSx for Lustre CSI driver to your Amazon EKS cluster and verify that it works. We recommend using the latest version of the driver. For available versions, see CSI Specification Compatibility Matrix on GitHub.

> **Note**
>
> The driver isn't supported on Fargate or Amazon EKS Hybrid Nodes.

For detailed descriptions of the available parameters and complete examples that demonstrate the driver's features, see the FSx for Lustre Container Storage Interface (CSI) driver project on GitHub.

You must have:

- Version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version, use `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*. The AWS CLI version that is installed in AWS CloudShell might also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the *AWS CloudShell User Guide*.
- Version `0.204.0` or later of the `eksctl` command line tool installed on your device or AWS CloudShell. To install or update `eksctl`, see Installation in the `eksctl` documentation.
- The `kubectl` command line tool is installed on your device or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. For example, if your cluster version is `1.29`, you can use `kubectl` version `1.28`, `1.29`, or `1.30` with it. To install or upgrade `kubectl`, see the section called "Set up kubectl and eksctl".

The following procedures help you create a simple test cluster with the FSx for Lustre CSI driver so that you can see how it works. We don't recommend using the testing cluster for production workloads. For this tutorial, we recommend using the *`example values`*, except where it's noted to replace them. You can replace any *`example value`* when completing the steps for your production cluster. We recommend completing all steps in the same terminal because variables are set and used throughout the steps and won't exist in different terminals.

1. Set a few variables to use in the remaining steps. Replace *`my-csi-fsx-cluster`* with the name of the test cluster you want to createand *`region-code`* with the AWS Region that you want to create your test cluster in.

```
export cluster_name=my-csi-fsx-cluster
export region_code=region-code
```

2. Create a test cluster.

```
eksctl create cluster \
  --name $cluster_name \
  --region $region_code \
  --with-oidc \
  --ssh-access \
```

```
  --ssh-public-key my-key
```

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

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

3. Create a Kubernetes service account for the driver and attach the `AmazonFSxFullAccess` AWS-managed policy to the service account with the following command.

```
eksctl create iamserviceaccount \
  --name fsx-csi-controller-sa \
  --namespace kube-system \
  --cluster $cluster_name \
  --attach-policy-arn arn:aws:iam::aws:policy/AmazonFSxFullAccess \
  --approve \
  --role-name AmazonEKSFSxLustreCSIDriverFullAccess \
  --region $region_code
```

You'll see several lines of output as the service account is created. The last lines of output are similar to the following.

```
[#]  1 task: {
    2 sequential sub-tasks: {
        create IAM role for serviceaccount "kube-system/fsx-csi-controller-sa",
        create serviceaccount "kube-system/fsx-csi-controller-sa",
    } }
[#]  building iamserviceaccount stack "eksctl-my-csi-fsx-cluster-addon-
iamserviceaccount-kube-system-fsx-csi-controller-sa"
[#]  deploying stack "eksctl-my-csi-fsx-cluster-addon-iamserviceaccount-kube-system-
fsx-csi-controller-sa"
[#]  waiting for CloudFormation stack "eksctl-my-csi-fsx-cluster-addon-
iamserviceaccount-kube-system-fsx-csi-controller-sa"
[#]  created serviceaccount "kube-system/fsx-csi-controller-sa"
```

Note the name of the AWS CloudFormation stack that was deployed. In the previous example output, the stack is named `eksctl-my-csi-fsx-cluster-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa`.

4. Deploy the driver with the following command. Replace *`release-X.XX`* with your desired branch. The master branch isn't supported because it may contain upcoming features

incompatible with the currently released stable version of the driver. We recommend using the latest released version. For a list of branches, see `aws-fsx-csi-driver` Branches on GitHub.

Note

You can view the content being applied in aws-fsx-csi-driver/deploy/kubernetes/overlays/stable on GitHub.

```
kubectl apply -k "github.com/kubernetes-sigs/aws-fsx-csi-driver/deploy/kubernetes/
overlays/stable/?ref=release-X.XX"
```

An example output is as follows.

```
serviceaccount/fsx-csi-controller-sa created
serviceaccount/fsx-csi-node-sa created
clusterrole.rbac.authorization.k8s.io/fsx-csi-external-provisioner-role created
clusterrole.rbac.authorization.k8s.io/fsx-external-resizer-role created
clusterrolebinding.rbac.authorization.k8s.io/fsx-csi-external-provisioner-binding
 created
clusterrolebinding.rbac.authorization.k8s.io/fsx-csi-resizer-binding created
deployment.apps/fsx-csi-controller created
daemonset.apps/fsx-csi-node created
csidriver.storage.k8s.io/fsx.csi.aws.com created
```

5. Note the ARN for the role that was created. If you didn't note it earlier and don't have it available anymore in the AWS CLI output, you can do the following to see it in the AWS Management Console.

   a. Open the AWS CloudFormation console.

   b. Ensure that the console is set to the AWS Region that you created your IAM role in and then select **Stacks**.

   c. Select the stack named `eksctl-my-csi-fsx-cluster-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa`.

   d. Select the **Outputs** tab. The **Role1** ARN is listed on the **Outputs (1)** page.

6. Patch the driver deployment to add the service account that you created earlier with the following command. Replace the ARN with the ARN that you noted. Replace *111122223333* with your account ID.

```
kubectl annotate serviceaccount -n kube-system fsx-csi-controller-sa \
  eks.amazonaws.com/role-arn=arn:aws:iam::111122223333:role/
AmazonEKSFSxLustreCSIDriverFullAccess --overwrite=true
```

An example output is as follows.

```
serviceaccount/fsx-csi-controller-sa annotated
```

This procedure uses the [FSx for Lustre Container Storage Interface (CSI) driver]GitHub repository to consume a dynamically-provisioned FSx for Lustre volume.

1. Note the security group for your cluster. You can see it in the AWS Management Console under the **Networking** section or by using the following AWS CLI command.

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

2. Create a security group for your Amazon FSx file system according to the criteria shown in [Amazon VPC Security Groups] in the Amazon FSx for Lustre User Guide. For the **VPC**, select the VPC of your cluster as shown under the **Networking** section. For "the security groups associated with your Lustre clients", use your cluster security group. You can leave the outbound rules alone to allow **All traffic**.

3. Download the storage class manifest with the following command.

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/
examples/kubernetes/dynamic_provisioning/specs/storageclass.yaml
```

4. Edit the parameters section of the `storageclass.yaml` file. Replace every *`example value`* with your own values.

```
parameters:
  subnetId: subnet-0eabfaa81fb22bcaf
  securityGroupIds: sg-068000ccf82dfba88
  deploymentType: PERSISTENT_1
  automaticBackupRetentionDays: "1"
  dailyAutomaticBackupStartTime: "00:00"
  copyTagsToBackups: "true"
  perUnitStorageThroughput: "200"
```

```
  dataCompressionType: "NONE"
  weeklyMaintenanceStartTime: "7:09:00"
  fileSystemTypeVersion: "2.12"
```

- **`subnetId`** – The subnet ID that the Amazon FSx for Lustre file system should be created in. Amazon FSx for Lustre isn't supported in all Availability Zones. Open the Amazon FSx for Lustre console at https://console.aws.amazon.com/fsx/ to confirm that the subnet that you want to use is in a supported Availability Zone. The subnet can include your nodes, or can be a different subnet or VPC:
  - You can check for the node subnets in the AWS Management Console by selecting the node group under the **Compute** section.
  - If the subnet that you specify isn't the same subnet that you have nodes in, then your VPCs must be connected, and you must ensure that you have the necessary ports open in your security groups.
- **`securityGroupIds`** – The ID of the security group you created for the file system.
- **`deploymentType` (optional)** – The file system deployment type. Valid values are `SCRATCH_1`, `SCRATCH_2`, `PERSISTENT_1`, and `PERSISTENT_2`. For more information about deployment types, see Create your Amazon FSx for Lustre file system.
- **other parameters (optional)** – For information about the other parameters, see Edit StorageClass on GitHub.

5. Create the storage class manifest.

```
kubectl apply -f storageclass.yaml
```

An example output is as follows.

```
storageclass.storage.k8s.io/fsx-sc created
```

6. Download the persistent volume claim manifest.

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/
examples/kubernetes/dynamic_provisioning/specs/claim.yaml
```

7. (Optional) Edit the `claim.yaml` file. Change *`1200Gi`* to one of the following increment values, based on your storage requirements and the `deploymentType` that you selected in a previous step.

```
storage: 1200Gi
```

- SCRATCH_2 and PERSISTENT – `1.2 TiB`, `2.4 TiB`, or increments of 2.4 TiB over 2.4 TiB.
- SCRATCH_1 – `1.2 TiB`, `2.4 TiB`, `3.6 TiB`, or increments of 3.6 TiB over 3.6 TiB.

8. Create the persistent volume claim.

```
kubectl apply -f claim.yaml
```

An example output is as follows.

```
persistentvolumeclaim/fsx-claim created
```

9. Confirm that the file system is provisioned.

```
kubectl describe pvc
```

An example output is as follows.

```
Name:          fsx-claim
Namespace:     default
StorageClass:  fsx-sc
Status:        Bound
[...]
```

> **Note**
>
> The `Status` may show as `Pending` for 5-10 minutes, before changing to `Bound`. Don't continue with the next step until the `Status` is `Bound`. If the `Status` shows `Pending` for more than 10 minutes, use warning messages in the `Events` as reference for addressing any problems.

10. Deploy the sample application.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-
driver/master/examples/kubernetes/dynamic_provisioning/specs/pod.yaml
```

11. Verify that the sample application is running.

```
kubectl get pods
```

An example output is as follows.

```
NAME      READY   STATUS              RESTARTS   AGE
fsx-app   1/1     Running             0          8s
```

12. Verify that the file system is mounted correctly by the application.

```
kubectl exec -ti fsx-app -- df -h
```

An example output is as follows.

```
Filesystem                     Size  Used Avail Use% Mounted on
overlay                         80G  4.0G   77G   5% /
tmpfs                           64M     0   64M   0% /dev
tmpfs                          3.8G     0  3.8G   0% /sys/fs/cgroup
192.0.2.0@tcp:/abcdef01        1.1T  7.8M  1.1T   1% /data
/dev/nvme0n1p1                  80G  4.0G   77G   5% /etc/hosts
shm                             64M     0   64M   0% /dev/shm
tmpfs                          6.9G   12K  6.9G   1% /run/secrets/kubernetes.io/
serviceaccount
tmpfs                          3.8G     0  3.8G   0% /proc/acpi
tmpfs                          3.8G     0  3.8G   0% /sys/firmware
```

13. Verify that data was written to the FSx for Lustre file system by the sample app.

```
kubectl exec -it fsx-app -- ls /data
```

An example output is as follows.

```
out.txt
```

This example output shows that the sample app successfully wrote the `out.txt` file to the file system.

> **Note**
>
> Before deleting the cluster, make sure to delete the FSx for Lustre file system. For more information, see Clean up resources in the *FSx for Lustre User Guide*.

Store high-performance apps with FSx for NetApp ONTAP

The NetApp Trident provides dynamic storage orchestration using a Container Storage Interface (CSI) compliant driver. This allows Amazon EKS clusters to manage the lifecycle of persistent volumes (PVs) backed by Amazon FSx for NetApp ONTAP file systems. Note that the Amazon FSx for NetApp ONTAP CSI driver is not compatible with Amazon EKS Hybrid Nodes. To get started, see Use Trident with Amazon FSx for NetApp ONTAP in the NetApp Trident documentation.

Amazon FSx for NetApp ONTAP is a storage service that allows you to launch and run fully managed ONTAP file systems in the cloud. ONTAP is NetApp's file system technology that provides a widely adopted set of data access and data management capabilities. FSx for ONTAP provides the features, performance, and APIs of on-premises NetApp file systems with the agility, scalability, and simplicity of a fully managed AWS service. For more information, see the FSx for ONTAP User Guide.

Store data using Amazon FSx for OpenZFS

Amazon FSx for OpenZFS is a fully managed file storage service that makes it easy to move data to AWS from on-premises ZFS or other Linux-based file servers. You can do this without changing your application code or how you manage data. It offers highly reliable, scalable, efficient, and feature-rich file storage built on the open-source OpenZFS file system. It combines these capabilities with the agility, scalability, and simplicity of a fully managed AWS service. For more information, see the Amazon FSx for OpenZFS User Guide.

The FSx for OpenZFS Container Storage Interface (CSI) driver provides a CSI interface that allows Amazon EKS clusters to manage the life cycle of FSx for OpenZFS volumes. Note that the Amazon FSx for OpenZFS CSI driver is not compatible with Amazon EKS Hybrid Nodes. To deploy the FSx for OpenZFS CSI driver to your Amazon EKS cluster, see aws-fsx-openzfs-csi-driver on GitHub.

Minimize latency with Amazon File Cache

Amazon File Cache is a fully managed, high-speed cache on AWS that's used to process file data, regardless of where the data is stored. Amazon File Cache automatically loads data into the cache when it's accessed for the first time and releases data when it's not used. For more information, see the Amazon File Cache User Guide.

The Amazon File Cache Container Storage Interface (CSI) driver provides a CSI interface that allows Amazon EKS clusters to manage the life cycle of Amazon file caches. Note that the Amazon File Cache CSI driver is not compatible with Amazon EKS Hybrid Nodes. To deploy the Amazon File Cache CSI driver to your Amazon EKS cluster, see aws-file-cache-csi-driver on GitHub.

Access Amazon S3 objects with Mountpoint for Amazon S3 CSI driver

With the Mountpoint for Amazon S3 Container Storage Interface (CSI) driver, your Kubernetes applications can access Amazon S3 objects through a file system interface, achieving high aggregate throughput without changing any application code. Built on Mountpoint for Amazon S3, the CSI driver presents an Amazon S3 bucket as a volume that can be accessed by containers in Amazon EKS and self-managed Kubernetes clusters. This topic shows you how to deploy the Mountpoint for Amazon S3 CSI driver to your Amazon EKS cluster.

Considerations

- The Mountpoint for Amazon S3 CSI driver isn't presently compatible with Windows-based container images.
- The Mountpoint for Amazon S3 CSI driver isn't presently compatible with Amazon EKS Hybrid Nodes.
- The Mountpoint for Amazon S3 CSI driver doesn't support AWS Fargate. However, containers that are running in Amazon EC2 (either with Amazon EKS or a custom Kubernetes installation) are supported.
- The Mountpoint for Amazon S3 CSI driver supports only static provisioning. Dynamic provisioning, or creation of new buckets, isn't supported.

> **Note**
>
> Static provisioning refers to using an existing Amazon S3 bucket that is specified as the `bucketName` in the `volumeAttributes` in the `PersistentVolume` object. For more information, see Static Provisioning on GitHub.

- Volumes mounted with the Mountpoint for Amazon S3 CSI driver don't support all POSIX file-system features. For details about file-system behavior, see Mountpoint for Amazon S3 file system behavior on GitHub.

Prerequisites

- An existing AWS Identity and Access Management (IAM) OpenID Connect (OIDC) provider for your cluster. To determine whether you already have one, or to create one, see the section called "IAM OIDC provider".
- Version 2.12.3 or later of the AWS CLI installed and configured on your device or AWS CloudShell.
- The `kubectl` command line tool is installed on your device or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. For example, if your cluster version is `1.29`, you can use `kubectl` version `1.28`, `1.29`, or `1.30` with it. To install or upgrade `kubectl`, see the section called "Set up kubectl and eksctl".

Create an IAM policy

The Mountpoint for Amazon S3 CSI driver requires Amazon S3 permissions to interact with your file system. This section shows how to create an IAM policy that grants the necessary permissions.

The following example policy follows the IAM permission recommendations for Mountpoint. Alternatively, you can use the AWS managed policy AmazonS3FullAccess, but this managed policy grants more permissions than are needed for Mountpoint.

For more information about the recommended permissions for Mountpoint, see Mountpoint IAM permissions on GitHub.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Policies**.

3. On the **Policies** page, choose **Create policy**.
4. For **Policy editor**, choose **JSON**.
5. Under **Policy editor**, copy and paste the following:

> **Important**
>
> Replace `amzn-s3-demo-bucket1` with your own Amazon S3 bucket name.

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
            "Sid": "MountpointFullBucketAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket1"
            ]
        },
        {
            "Sid": "MountpointFullObjectAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:AbortMultipartUpload",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket1/*"
            ]
        }
   ]
}
```

Directory buckets, introduced with the Amazon S3 Express One Zone storage class, use a different authentication mechanism from general purpose buckets. Instead of using `s3:*`

actions, you should use the `s3express:CreateSession` action. For information about directory buckets, see Directory buckets in the *Amazon S3 User Guide*.

Below is an example of least-privilege policy that you would use for a directory bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3express:CreateSession",
            "Resource": "arn:aws:s3express:us-west-2:111122223333:bucket/amzn-s3-
demo-bucket1--usw2-az1--x-s3"
        }
    ]
}
```

6. Choose **Next**.
7. On the **Review and create** page, name your policy. This example walkthrough uses the name `AmazonS3CSIDriverPolicy`.
8. Choose **Create policy**.

Create an IAM role

The Mountpoint for Amazon S3 CSI driver requires Amazon S3 permissions to interact with your file system. This section shows how to create an IAM role to delegate these permissions. To create this role, you can use one of these tools:

- the section called “eksctl”
- the section called “AWS Management Console”
- the section called “AWS CLI”

> **Note**
>
> The IAM policy `AmazonS3CSIDriverPolicy` was created in the previous section.

eksctl

To create your Mountpoint for Amazon S3 CSI driver IAM role with `eksctl`

To create the IAM role and the Kubernetes service account, run the following commands. These commands also attach the `AmazonS3CSIDriverPolicy` IAM policy to the role, annotate the Kubernetes service account (`s3-csi-controller-sa`) with the IAM role's Amazon Resource Name (ARN), and add the Kubernetes service account name to the trust policy for the IAM role.

```
CLUSTER_NAME=my-cluster
REGION=region-code
ROLE_NAME=AmazonEKS_S3_CSI_DriverRole
POLICY_ARN=AmazonEKS_S3_CSI_DriverRole_ARN
eksctl create iamserviceaccount \
    --name s3-csi-driver-sa \
    --namespace kube-system \
    --cluster $CLUSTER_NAME \
    --attach-policy-arn $POLICY_ARN \
    --approve \
    --role-name $ROLE_NAME \
    --region $REGION \
    --role-only
```

AWS Management Console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Roles**.
3. On the **Roles** page, choose **Create role**.
4. On the **Select trusted entity** page, do the following:
   a. In the **Trusted entity type** section, choose **Web identity**.
   b. For **Identity provider**, choose the **OpenID Connect provider URL** for your cluster (as shown under **Overview** in Amazon EKS).

      If no URLs are shown, review the Prerequisites.
   c. For **Audience**, choose `sts.amazonaws.com`.
   d. Choose **Next**.
5. On the **Add permissions** page, do the following:
   a. In the **Filter policies** box, enter AmazonS3CSIDriverPolicy.

> **Note**
>
> This policy was created in the previous section.

   b. Select the check box to the left of the `AmazonS3CSIDriverPolicy` result that was returned in the search.

   c. Choose **Next**.

6. On the **Name, review, and create** page, do the following:

   a. For **Role name**, enter a unique name for your role, such as AmazonEKS_S3_CSI_DriverRole.

   b. Under **Add tags (Optional)**, add metadata to the role by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM resources in the *IAM User Guide*.

   c. Choose **Create role**.

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

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

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

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

   Add a comma to the end of the previous line, and then add the following line after it. Replace *`region-code`* with the AWS Region that your cluster is in. Replace *`EXAMPLED539D4633E53DE1B71EXAMPLE`* with your cluster's OIDC provider ID.

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":
 "system:serviceaccount:kube-system:s3-csi-driver-sa"
```

10. Ensure that the `Condition` operator is set to `"StringEquals"`.

11. Choose **Update policy** to finish.

AWS CLI

1. View the OIDC provider URL for your cluster. Replace *`my-cluster`* with the name of your cluster. If the output from the command is `None`, review the Prerequisites.

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer" --
output text
```

An example output is as follows.

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

2. Create the IAM role, granting the Kubernetes service account the `AssumeRoleWithWebIdentity` action.

   a. Copy the following contents to a file named `aws-s3-csi-driver-trust-policy.json`. Replace *`111122223333`* with your account ID. Replace *`EXAMPLED539D4633E53DE1B71EXAMPLE`* and *`region-code`* with the values returned in the previous step.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.region-
code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:s3-csi-
driver-sa",
          "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
        }
      }
    }
  ]
}
```

   b. Create the role. You can change *`AmazonEKS_S3_CSI_DriverRole`* to a different name, but if you do, make sure to change it in later steps too.

```
aws iam create-role \
  --role-name AmazonEKS_S3_CSI_DriverRole \
  --assume-role-policy-document file://"aws-s3-csi-driver-trust-policy.json"
```

3. Attach the previously created IAM policy to the role with the following command.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonS3CSIDriverPolicy \
  --role-name AmazonEKS_S3_CSI_DriverRole
```

Note

The IAM policy `AmazonS3CSIDriverPolicy` was created in the previous section.

4. Skip this step if you're installing the driver as an Amazon EKS add-on. For self-managed installations of the driver, create Kubernetes service accounts that are annotated with the ARN of the IAM role that you created.

   a. Save the following contents to a file named `mountpoint-s3-service-account.yaml`. Replace *111122223333* with your account ID.

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/name: aws-mountpoint-s3-csi-driver
  name: mountpoint-s3-csi-controller-sa
  namespace: kube-system
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/
AmazonEKS_S3_CSI_DriverRole
```

   b. Create the Kubernetes service account on your cluster. The Kubernetes service account (`mountpoint-s3-csi-controller-sa`) is annotated with the IAM role that you created named *AmazonEKS_S3_CSI_DriverRole*.

```
kubectl apply -f mountpoint-s3-service-account.yaml
```

> **Note**
>
> When you deploy the plugin in this procedure, it creates and is configured to use a service account named `s3-csi-driver-sa`.

Install the Mountpoint for Amazon S3 CSI driver

You may install the Mountpoint for Amazon S3 CSI driver through the Amazon EKS add-on. You can use the following tools to add the add-on to your cluster:

- the section called "eksctl"
- the section called "AWS Management Console"
- the section called "AWS CLI"

Alternatively, you may install Mountpoint for Amazon S3 CSI driver as a self-managed installation. For instructions on doing a self-managed installation, see Installation on GitHub.

Starting from `v1.8.0`, you can configure taints to tolerate for the CSI driver's Pods. To do this, either specify a custom set of taints to tolerate with `node.tolerations` or tolorate all taints with `node.tolerateAllTaints`. For more information, see Taints and Tolerations in the Kubernetes documentation.

eksctl

To add the Amazon S3 CSI add-on using `eksctl`

Run the following command. Replace *`my-cluster`* with the name of your cluster, *`111122223333`* with your account ID, and *`AmazonEKS_S3_CSI_DriverRole`* with the name of the IAM role created earlier.

```
eksctl create addon --name aws-mountpoint-s3-csi-driver --cluster my-cluster \
  --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole
 --force
```

If you remove the *`--force`* option and any of the Amazon EKS add-on settings conflict with your existing settings, then updating the Amazon EKS add-on fails, and you receive an error message to help you resolve the conflict. Before specifying this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to manage, because those settings are overwritten with this option. For more information about other options for this setting, see Addons in the `eksctl` documentation. For more information about Amazon EKS Kubernetes field management, see the section called "Fields you can customize".

You can customize `eksctl` through configuration files. For more information, see Working with configuration values in the `eksctl` documentation. The following example shows how to tolerate all taints.

```
# config.yaml
...

addons:
- name: aws-mountpoint-s3-csi-driver
  serviceAccountRoleARN: arn:aws:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole
  configurationValues: |-
    node:
      tolerateAllTaints: true
```

AWS Management Console

1. Open the Amazon EKS console.
2. In the left navigation pane, choose **Clusters**.
3. Choose the name of the cluster that you want to configure the Mountpoint for Amazon S3 CSI add-on for.
4. Choose the **Add-ons** tab.
5. Choose **Get more add-ons**.
6. On the **Select add-ons** page, do the following:
   a. In the **Amazon EKS-addons** section, select the **Mountpoint for Amazon S3 CSI Driver** check box.
   b. Choose **Next**.
7. On the **Configure selected add-ons settings** page, do the following:
   a. Select the **Version** you'd like to use.
   b. For **Select IAM role**, select the name of an IAM role that you attached the Mountpoint for Amazon S3 CSI driver IAM policy to.

c. (Optional) Update the **Conflict resolution method** after expanding the **Optional configuration settings**. If you select **Override**, one or more of the settings for the existing add-on can be overwritten with the Amazon EKS add-on settings. If you don't enable this option and there's a conflict with your existing settings, the operation fails. You can use the resulting error message to troubleshoot the conflict. Before selecting this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to self-manage.

d. (Optional) Configure tolerations in the **Configuration values** field after expanding the **Optional configuration settings**.

e. Choose **Next**.

8. On the **Review and add** page, choose **Create**. After the add-on installation is complete, you see your installed add-on.

AWS CLI

To add the Mountpoint for Amazon S3 CSI add-on using the AWS CLI

Run the following command. Replace *my-cluster* with the name of your cluster, *111122223333* with your account ID, and *AmazonEKS_S3_CSI_DriverRole* with the name of the role that was created earlier.

```
aws eks create-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-
driver \
  --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole
```

You can customize the command with the `--configuration-values` flag. The following alternative example shows how to tolerate all taints.

```
aws eks create-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-
driver \
  --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole
 \
  --configuration-values '{"node":{"tolerateAllTaints":true}}'
```

Configure Mountpoint for Amazon S3

In most cases, you can configure Mountpoint for Amazon S3 with only a bucket name. For instructions on configuring Mountpoint for Amazon S3, see Configuring Mountpoint for Amazon S3 on GitHub.

Deploy a sample application

You can deploy static provisioning to the driver on an existing Amazon S3 bucket. For more information, see Static provisioning on GitHub.

Remove Mountpoint for Amazon S3 CSI Driver

You have two options for removing an Amazon EKS add-on.

- **Preserve add-on software on your cluster** – This option removes Amazon EKS management of any settings. It also removes the ability for Amazon EKS to notify you of updates and automatically update the Amazon EKS add-on after you initiate an update. However, it preserves the add-on software on your cluster. This option makes the add-on a self-managed installation, rather than an Amazon EKS add-on. With this option, there's no downtime for the add-on. The commands in this procedure use this option.
- **Remove add-on software entirely from your cluster** – We recommend that you remove the Amazon EKS add-on from your cluster only if there are no resources on your cluster that are dependent on it. To do this option, delete `--preserve` from the command you use in this procedure.

If the add-on has an IAM account associated with it, the IAM account isn't removed.

You can use the following tools to remove the Amazon S3 CSI add-on:

- the section called "eksctl"
- the section called "AWS Management Console"
- the section called "AWS CLI"

eksctl

To remove the Amazon S3 CSI add-on using `eksctl`

Replace *`my-cluster`* with the name of your cluster, and then run the following command.

```
eksctl delete addon --cluster my-cluster --name aws-mountpoint-s3-csi-driver --preserve
```

AWS Management Console

1. Open the [Amazon EKS console](https://console.aws.amazon.com/eks/home#/clusters).
2. In the left navigation pane, choose **Clusters**.
3. Choose the name of the cluster that you want to remove the Amazon EBS CSI add-on for.
4. Choose the **Add-ons** tab.
5. Choose **Mountpoint for Amazon S3 CSI Driver**.
6. Choose **Remove**.
7. In the **Remove: aws-mountpoint-s3-csi-driver** confirmation dialog box, do the following:
   a. If you want Amazon EKS to stop managing settings for the add-on, select **Preserve on cluster**. Do this if you want to retain the add-on software on your cluster. This is so that you can manage all of the settings of the add-on on your own.
   b. Enter `aws-mountpoint-s3-csi-driver`.
   c. Select **Remove**.

AWS CLI

To remove the Amazon S3 CSI add-on using the AWS CLI

Replace *`my-cluster`* with the name of your cluster, and then run the following command.

```
aws eks delete-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-driver --preserve
```

Enable snapshot functionality for CSI volumes

Snapshot functionality allows for point-in-time copies of your data. For this capability to work in Kubernetes, you need both a CSI driver with snapshot support (such as the Amazon EBS CSI driver) and a CSI snapshot controller. The snapshot controller is available either as an Amazon EKS managed add-on or as a self-managed installation.

Here are some things to consider when using the CSI snapshot controller.

- The snapshot controller must be installed alongside a CSI driver with snapshot functionality. For installation instructions of the Amazon EBS CSI driver, see the section called “Amazon EBS”.

- Kubernetes doesn't support snapshots of volumes being served via CSI migration, such as Amazon EBS volumes using a `StorageClass` with provisioner `kubernetes.io/aws-ebs`. Volumes must be created with a `StorageClass` that references the CSI driver provisioner, `ebs.csi.aws.com`.
- Amazon EKS Auto Mode does not include the snapshot controller. The storage capability of EKS Auto Mode is compatible with the snapshot controller.

We recommend that you install the CSI snapshot controller through the Amazon EKS managed add-on. This add-on includes the custom resource definitions (CRDs) that are needed to create and manage snapshots on Amazon EKS. To add an Amazon EKS add-on to your cluster, see the section called "Create an add-on". For more information about add-ons, see the section called "Amazon EKS add-ons".

Alternatively, if you want a self-managed installation of the CSI snapshot controller, see Usage in the upstream Kubernetes `external-snapshotter` on GitHub.

Configure networking for Amazon EKS clusters

Your Amazon EKS cluster is created in a VPC. Pod networking is provided by the Amazon VPC Container Network Interface (CNI) plugin for nodes that run on AWS infrastructure. If you are running nodes on your own infrastructure, see the section called "Configure CNI". This chapter includes the following topics for learning more about networking for your cluster.

Topics

- View Amazon EKS networking requirements for VPC and subnets
- Create an Amazon VPC for your Amazon EKS cluster
- View Amazon EKS security group requirements for clusters
- Manage networking add-ons for Amazon EKS clusters

View Amazon EKS networking requirements for VPC and subnets

When you create a cluster, you specify a VPC and at least two subnets that are in different Availability Zones. This topic provides an overview of Amazon EKS specific requirements and considerations for the VPC and subnets that you use with your cluster. If you don't have a VPC to use with Amazon EKS, see the section called "Create a VPC". If you're creating a local or extended cluster on AWS Outposts, see the section called "Create a VPC and subnets" instead of this topic. The content in this topic applies for Amazon EKS clusters with hybrid nodes. For additional networking requirements for hybrid nodes, see the section called "Prepare networking".

VPC requirements and considerations

When you create a cluster, the VPC that you specify must meet the following requirements and considerations:

- The VPC must have a sufficient number of IP addresses available for the cluster, any nodes, and other Kubernetes resources that you want to create. If the VPC that you want to use doesn't have a sufficient number of IP addresses, try to increase the number of available IP addresses.

  You can do this by updating the cluster configuration to change which subnets and security groups the cluster uses. You can update from the AWS Management Console, the latest version of the AWS CLI, AWS CloudFormation, and `eksctl` version `v0.164.0-rc.0` or later. You might need to do this to provide subnets with more available IP addresses to successfully upgrade a cluster version.

> **Important**
>
> All subnets that you add must be in the same set of AZs as originally provided when you created the cluster. New subnets must satisfy all of the other requirements, for example they must have sufficient IP addresses.
> For example, assume that you made a cluster and specified four subnets. In the order that you specified them, the first subnet is in the `us-west-2a` Availability Zone, the second and third subnets are in `us-west-2b` Availability Zone, and the fourth subnet is in `us-west-2c` Availability Zone. If you want to change the subnets, you must provide at least one subnet in each of the three Availability Zones, and the subnets must be in the same VPC as the original subnets.

If you need more IP addresses than the CIDR blocks in the VPC have, you can add additional CIDR blocks by associating additional Classless Inter-Domain Routing (CIDR) blocks with your VPC. You can associate private (RFC 1918) and public (non-RFC 1918) CIDR blocks to your VPC either before or after you create your cluster. It can take a cluster up to five hours for a CIDR block that you associated with a VPC to be recognized.

You can conserve IP address utilization by using a transit gateway with a shared services VPC. For more information, see Isolated VPCs with shared services and Amazon EKS VPC routable IP address conservation patterns in a hybrid network.

- If you want Kubernetes to assign `IPv6` addresses to Pods and services, associate an `IPv6` CIDR block with your VPC. For more information, see Associate an IPv6 CIDR block with your VPC in the Amazon VPC User Guide. You cannot use `IPv6` addresses with Pods and services running on hybrid nodes and you cannot use hybrid nodes with clusters configured with the `IPv6` IP address family.
- The VPC must have `DNS` hostname and `DNS` resolution support. Otherwise, nodes can't register to your cluster. For more information, see DNS attributes for your VPC in the Amazon VPC User Guide.
- The VPC might require VPC endpoints using AWS PrivateLink. For more information, see the section called “Subnet requirements and considerations”.

If you created a cluster with Kubernetes 1.14 or earlier, Amazon EKS added the following tag to your VPC:

Key	Value
kubernetes.io/cluster/ *my-cluster*	owned

This tag was only used by Amazon EKS. You can remove the tag without impacting your services. It's not used with clusters that are version 1.15 or later.

Subnet requirements and considerations

When you create a cluster, Amazon EKS creates 2–4 elastic network interfaces in the subnets that you specify. These network interfaces enable communication between your cluster and your VPC. These network interfaces also enable Kubernetes features such as `kubectl exec` and `kubectl logs`. Each Amazon EKS created network interface has the text `Amazon EKS` *cluster-name* in its description.

Amazon EKS can create its network interfaces in any subnet that you specify when you create a cluster. You can change which subnets Amazon EKS creates its network interfaces in after your cluster is created. When you update the Kubernetes version of a cluster, Amazon EKS deletes the original network interfaces that it created, and creates new network interfaces. These network interfaces might be created in the same subnets as the original network interfaces or in different subnets than the original network interfaces. To control which subnets network interfaces are created in, you can limit the number of subnets you specify to only two when you create a cluster or update the subnets after creating the cluster.

Subnet requirements for clusters

The subnets that you specify when you create or update a cluster must meet the following requirements:

- The subnets must each have at least six IP addresses for use by Amazon EKS. However, we recommend at least 16 IP addresses.
- The subnets must be in at least two different Availability Zones.
- The subnets can't reside in AWS Outposts or AWS Wavelength. However, if you have them in your VPC, you can deploy self-managed nodes and Kubernetes resources to these types of subnets. For more information about self-managed nodes, see the section called "Self-managed nodes".

- The subnets can be a public or private. However, we recommend that you specify private subnets, if possible. A public subnet is a subnet with a route table that includes a route to an internet gateway, whereas a private subnet is a subnet with a route table that doesn't include a route to an internet gateway.
- The subnets can't reside in the following Availability Zones:

AWS Region	Region name	Disallowed Availability Zone IDs
`us-east-1`	US East (N. Virginia)	`use1-az3`
`us-west-1`	US West (N. California)	`usw1-az2`
`ca-central-1`	Canada (Central)	`cac1-az3`

IP address family usage by component

The following table contains the IP address family used by each component of Amazon EKS. You can use a network address translation (NAT) or other compatibility system to connect to these components from source IP addresses in families with the "No" value for a table entry.

Functionality can differ depending on the IP family (`ipFamily`) setting of the cluster. This setting changes the type of IP addresses used for the CIDR block that Kubernetes assigns to Services. A cluster with the setting value of IPv4 is referred to as an *IPv4 cluster*, and a cluster with the setting value of IPv6 is referred to as an *IPv6 cluster*.

Component	IPv4 addresses	IPv6 addresses	Dual stack addresses
EKS API public endpoint	Yes[1,3]	Yes[1,3]	Yes[1,3]
EKS API VPC endpoint	Yes	No	No
EKS Auth API public endpoint (EKS Pod Identity)	Yes[1]	Yes[1]	Yes[1]

Component	IPv4 addresses	IPv6 addresses	Dual stack addresses
EKS Auth API VPC endpoint (EKS Pod Identity)	Yes[1]	Yes[1]	Yes[1]
IPv4 Kubernetes cluster public endpoint[2]	Yes	No	No
IPv4 Kubernetes cluster private endpoint[2]	Yes	No	No
IPv6 Kubernetes cluster public endpoint[2]	Yes[1,4]	Yes[1,4]	Yes[4]
IPv6 Kubernetes cluster private endpoint[2]	Yes[1,4]	Yes[1,4]	Yes[4]
Kubernetes cluster subnets	Yes[2]	No	Yes[2]
Node Primary IP addresses	Yes[2]	No	Yes[2]
Cluster CIDR range for Service IP addresses	Yes[2]	Yes[2]	No
Pod IP addresses from the VPC CNI	Yes[2]	Yes[2]	No
IRSA OIDC Issuer URLs	Yes[1,3]	Yes[1,3]	Yes[1,3]

> **Note**
>
> [1] The endpoint is dual stack with both `IPv4` and `IPv6` addresses. Your applications outside of AWS, your nodes for the cluster, and your pods inside the cluster can reach this endpoint by either `IPv4` or `IPv6`.
> [2] You choose between an `IPv4` cluster and `IPv6` cluster in the IP family (`ipFamily`) setting of the cluster when you create a cluster and this can't be changed. Instead, you must choose a different setting when you create another cluster and migrate your workloads.
> [3] The dual-stack endpoint was introduced in August 2024. To use the dual-stack endpoints with the AWS CLI, see the Dual-stack and FIPS endpoints configuration in the *AWS SDKs and Tools Reference Guide*. The following lists the new endpoints:
>
> **EKS API public endpoint**
>
> ```
> eks.region.api.aws
> ```
>
> **IRSA OIDC Issuer URLs**
>
> ```
> oidc-eks.region.api.aws
> ```
>
> [4] The dual-stack cluster endpoint was introduced in October 2024. EKS creates the following endpoint for new clusters that are made after this date and that select `IPv6` in the IP family (ipFamily) setting of the cluster:
>
> **EKS cluster public/private endpoint**
>
> ```
> eks-cluster.region.api.aws
> ```

Subnet requirements for nodes

You can deploy nodes and Kubernetes resources to the same subnets that you specify when you create your cluster. However, this isn't necessary. This is because you can also deploy nodes and Kubernetes resources to subnets that you didn't specify when you created the cluster. If you deploy nodes to different subnets, Amazon EKS doesn't create cluster network interfaces in those subnets. Any subnet that you deploy nodes and Kubernetes resources to must meet the following requirements:

- The subnets must have enough available IP addresses to deploy all of your nodes and Kubernetes resources to.
- If you want Kubernetes to assign IPv6 addresses to Pods and services, then you must have one IPv6 CIDR block and one IPv4 CIDR block that are associated with your subnet. For more information, see Associate an IPv6 CIDR block with your subnet in the Amazon VPC User Guide. The route tables that are associated with the subnets must include routes to IPv4 and IPv6 addresses. For more information, see Routes in the Amazon VPC User Guide. Pods are assigned only an IPv6 address. However the network interfaces that Amazon EKS creates for your cluster and your nodes are assigned an IPv4 and an IPv6 address.
- If you need inbound access from the internet to your Pods, make sure to have at least one public subnet with enough available IP addresses to deploy load balancers and ingresses to. You can deploy load balancers to public subnets. Load balancers can load balance to Pods in private or public subnets. We recommend deploying your nodes to private subnets, if possible.
- If you plan to deploy nodes to a public subnet, the subnet must auto-assign IPv4 public addresses or IPv6 addresses. If you deploy nodes to a private subnet that has an associated IPv6 CIDR block, the private subnet must also auto-assign IPv6 addresses. If you used the AWS CloudFormation template provided by Amazon EKS to deploy your VPC after March 26, 2020, this setting is enabled. If you used the templates to deploy your VPC before this date or you use your own VPC, you must enable this setting manually. For the template, see the section called "Create a VPC". For more information, see Modify the public IPv4 addressing attribute for your subnet and Modify the IPv6 addressing attribute for your subnet in the Amazon VPC User Guide.
- If the subnet that you deploy a node to is a private subnet and its route table doesn't include a route to a network address translation (NAT) device (IPv4) or an egress-only gateway (IPv6), add VPC endpoints using AWS PrivateLink to your VPC. VPC endpoints are needed for all the AWS services that your nodes and Pods need to communicate with. Examples include Amazon ECR, Elastic Load Balancing, Amazon CloudWatch, AWS Security Token Service, and Amazon Simple Storage Service (Amazon S3). The endpoint must include the subnet that the nodes are in. Not all AWS services support VPC endpoints. For more information, see What is AWS PrivateLink? and AWS services that integrate with AWS PrivateLink. For a list of more Amazon EKS requirements, see the section called "Private clusters".
- If you want to deploy load balancers to a subnet, the subnet must have the following tag:
  - Private subnets

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

- Public subnets

Key	Value
kubernetes.io/role/elb	1

When a Kubernetes cluster that's version 1.18 and earlier was created, Amazon EKS added the following tag to all of the subnets that were specified.

Key	Value
kubernetes.io/cluster/ *my-cluster*	shared

When you create a new Kubernetes cluster now, Amazon EKS doesn't add the tag to your subnets. If the tag was on subnets that were used by a cluster that was previously a version earlier than 1.19, the tag wasn't automatically removed from the subnets when the cluster was updated to a newer version. Version 2.1.1 or earlier of the AWS Load Balancer Controller requires this tag. If you are using a newer version of the Load Balancer Controller, you can remove the tag without interrupting your services. For more information about the controller, see the section called " AWS Load Balancer Controller".

If you deployed a VPC by using eksctl or any of the Amazon EKS AWS CloudFormation VPC templates, the following applies:

- **On or after March 26, 2020 –** Public IPv4 addresses are automatically assigned by public subnets to new nodes that are deployed to public subnets.
- **Before March 26, 2020 –** Public IPv4 addresses aren't automatically assigned by public subnets to new nodes that are deployed to public subnets.

This change impacts new node groups that are deployed to public subnets in the following ways:

- **Managed node groups** – If the node group is deployed to a public subnet on or after April 22, 2020, automatic assignment of public IP addresses must be enabled for the public subnet. For more information, see Modifying the public IPv4 addressing attribute for your subnet.
- **Linux, Windows, or Arm self-managed node groups** – If the node group is deployed to a public subnet on or after March 26, 2020, automatic assignment of public IP addresses must be enabled for the public subnet. Otherwise, the nodes must be launched with a public IP address instead. For more information, see Modifying the public IPv4 addressing attribute for your subnet or Assigning a public IPv4 address during instance launch.

Shared subnet requirements and considerations

You can use *VPC sharing* to share subnets with other AWS accounts within the same AWS Organizations. You can create Amazon EKS clusters in shared subnets, with the following considerations:

- The owner of the VPC subnet must share a subnet with a participant account before that account can create an Amazon EKS cluster in it.
- You can't launch resources using the default security group for the VPC because it belongs to the owner. Additionally, participants can't launch resources using security groups that are owned by other participants or the owner.
- In a shared subnet, the participant and the owner separately controls the security groups within each respective account. The subnet owner can see security groups that are created by the participants but cannot perform any actions on them. If the subnet owner wants to remove or modify these security groups, the participant that created the security group must take the action.
- If a cluster is created by a participant, the following considerations apply:
  - Cluster IAM role and Node IAM roles must be created in that account. For more information, see the section called "Cluster IAM role" and the section called "Node IAM role".
  - All nodes must be made by the same participant, including managed node groups.
- The shared VPC owner cannot view, update or delete a cluster that a participant creates in the shared subnet. This is in addition to the VPC resources that each account has different access to. For more information, see Responsibilities and permissions for owners and participants in the *Amazon VPC User Guide*.

- If you use the *custom networking* feature of the Amazon VPC CNI plugin for Kubernetes, you need to use the Availability Zone ID mappings listed in the owner account to create each `ENIConfig`. For more information, see the section called "Custom networking".

For more information about VPC subnet sharing, see Share your VPC with other accounts in the *Amazon VPC User Guide*.

Create an Amazon VPC for your Amazon EKS cluster

You can use Amazon Virtual Private Cloud (Amazon VPC) to launch AWS resources into a virtual network that you've defined. This virtual network closely resembles a traditional network that you might operate in your own data center. However, it comes with the benefits of using the scalable infrastructure of Amazon Web Services. We recommend that you have a thorough understanding of the Amazon VPC service before deploying production Amazon EKS clusters. For more information, see the Amazon VPC User Guide.

An Amazon EKS cluster, nodes, and Kubernetes resources are deployed to a VPC. If you want to use an existing VPC with Amazon EKS, that VPC must meet the requirements that are described in the section called "VPC and subnet requirements". This topic describes how to create a VPC that meets Amazon EKS requirements using an Amazon EKS provided AWS CloudFormation template. Once you've deployed a template, you can view the resources created by the template to know exactly what resources it created, and the configuration of those resources. If you are using hybrid nodes, your VPC must have routes in its route table for your on-premises network. For more information about the network requirements for hybrid nodes, see the section called "Prepare networking".

Prerequisites

To create a VPC for Amazon EKS, you must have the necessary IAM permissions to create Amazon VPC resources. These resources are VPCs, subnets, security groups, route tables and routes, and internet and NAT gateways. For more information, see Create a VPC with a public subnet example policy in the Amazon VPC User Guide and the full list of Actions in the Service Authorization Reference.

You can create a VPC with public and private subnets, only public subnets, or only private subnets.

Public and private subnets

This VPC has two public and two private subnets. A public subnet's associated route table has a route to an internet gateway. However, the route table of a private subnet doesn't have a route to an internet gateway. One public and one private subnet are deployed to the same Availability Zone. The other public and private subnets are deployed to a second Availability Zone in the same AWS Region. We recommend this option for most deployments.

With this option, you can deploy your nodes to private subnets. This option allows Kubernetes to deploy load balancers to the public subnets that can load balance traffic to Pods that run on nodes in the private subnets. Public `IPv4` addresses are automatically assigned to nodes that are deployed to public subnets, but public `IPv4` addresses aren't assigned to nodes deployed to private subnets.

You can also assign `IPv6` addresses to nodes in public and private subnets. The nodes in private subnets can communicate with the cluster and other AWS services. Pods can communicate to the internet through a NAT gateway using `IPv4` addresses or outbound-only Internet gateway using `IPv6` addresses deployed in each Availability Zone. A security group is deployed that has rules that deny all inbound traffic from sources other than the cluster or nodes but allows all outbound traffic. The subnets are tagged so that Kubernetes can deploy load balancers to them.

a. Open the [AWS CloudFormation console](https://console.aws.amazon.com/cloudformation/).

b. From the navigation bar, select an AWS Region that supports Amazon EKS.

c. Choose **Create stack**, **With new resources (standard)**.

d. Under **Prerequisite - Prepare template**, make sure that **Template is ready** is selected and then under **Specify template**, select **Amazon S3 URL**.

e. You can create a VPC that supports only `IPv4`, or a VPC that supports `IPv4` and `IPv6`. Paste one of the following URLs into the text area under **Amazon S3 URL** and choose **Next**:

- `IPv4`

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

- `IPv4` and `IPv6`

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-
ipv6-vpc-public-private-subnets.yaml
```

a. On the **Specify stack details** page, enter the parameters, and then choose **Next**.
   - **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can use the template name you used in the previous step. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.
   - **VpcBlock**: Choose an `IPv4` CIDR range for your VPC. Each node, Pod, and load balancer that you deploy is assigned an `IPv4` address from this block. The default `IPv4` values provide enough IP addresses for most implementations, but if it doesn't, then you can change it. For more information, see VPC and subnet sizing in the Amazon VPC User Guide. You can also add additional CIDR blocks to the VPC once it's created. If you're creating an `IPv6` VPC, `IPv6` CIDR ranges are automatically assigned for you from Amazon's Global Unicast Address space.
   - **PublicSubnet01Block**: Specify an `IPv4` CIDR block for public subnet 1. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it. If you're creating an `IPv6` VPC, this block is specified for you within the template.
   - **PublicSubnet02Block**: Specify an `IPv4` CIDR block for public subnet 2. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it. If you're creating an `IPv6` VPC, this block is specified for you within the template.
   - **PrivateSubnet01Block**: Specify an `IPv4` CIDR block for private subnet 1. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it. If you're creating an `IPv6` VPC, this block is specified for you within the template.
   - **PrivateSubnet02Block**: Specify an `IPv4` CIDR block for private subnet 2. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it. If you're creating an `IPv6` VPC, this block is specified for you within the template.

b. (Optional) On the **Configure stack options** page, tag your stack resources and then choose **Next**.

c. On the **Review** page, choose **Create stack**.

d. When your stack is created, select it in the console and choose **Outputs**.

e. Record the **VpcId** for the VPC that was created. You need this when you create your cluster and nodes.

f. Record the **SubnetIds** for the subnets that were created and whether you created them as public or private subnets. You need at least two of these when you create your cluster and nodes.

g. If you created an `IPv4` VPC, skip this step. If you created an `IPv6` VPC, you must enable the auto-assign `IPv6` address option for the public subnets that were created by the template. That setting is already enabled for the private subnets. To enable the setting, complete the following steps:

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

   ii. In the left navigation pane, choose **Subnets**

   iii. Select one of your public subnets (***stack-name*/SubnetPublic01** or ***stack-name*/SubnetPublic02** contains the word **public**) and choose **Actions**, **Edit subnet settings**.

   iv. Choose the **Enable auto-assign IPv6 address** check box and then choose **Save**.

   v. Complete the previous steps again for your other public subnet.

Only public subnets

This VPC has three public subnets that are deployed into different Availability Zones in an AWS Region. All nodes are automatically assigned public `IPv4` addresses and can send and receive internet traffic through an internet gateway. A security group is deployed that denies all inbound traffic and allows all outbound traffic. The subnets are tagged so that Kubernetes can deploy load balancers to them.

a. Open the AWS CloudFormation console.

b. From the navigation bar, select an AWS Region that supports Amazon EKS.

c. Choose **Create stack**, **With new resources (standard)**.

d. Under **Prepare template**, make sure that **Template is ready** is selected and then under **Template source**, select **Amazon S3 URL**.

e. Paste the following URL into the text area under **Amazon S3 URL** and choose **Next**:

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

a. On the **Specify Details** page, enter the parameters, and then choose **Next**.

   - **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it *amazon-eks-vpc-sample*. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.

- **VpcBlock**: Choose a CIDR block for your VPC. Each node, Pod, and load balancer that you deploy is assigned an `IPv4` address from this block. The default `IPv4` values provide enough IP addresses for most implementations, but if it doesn't, then you can change it. For more information, see VPC and subnet sizing in the Amazon VPC User Guide. You can also add additional CIDR blocks to the VPC once it's created.
- **Subnet01Block**: Specify a CIDR block for subnet 1. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it.
- **Subnet02Block**: Specify a CIDR block for subnet 2. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it.
- **Subnet03Block**: Specify a CIDR block for subnet 3. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it.

b. (Optional) On the **Options** page, tag your stack resources. Choose **Next**.

c. On the **Review** page, choose **Create**.

d. When your stack is created, select it in the console and choose **Outputs**.

e. Record the **VpcId** for the VPC that was created. You need this when you create your cluster and nodes.

f. Record the **SubnetIds** for the subnets that were created. You need at least two of these when you create your cluster and nodes.

g. (Optional) Any cluster that you deploy to this VPC can assign private `IPv4` addresses to your Pods and services. If you want to deploy clusters to this VPC to assign private `IPv6` addresses to your Pods and services, make updates to your VPC, subnet, route tables, and security groups. For more information, see Migrate existing VPCs from IPv4 to IPv6 in the Amazon VPC User Guide. Amazon EKS requires that your subnets have the `Auto-assign IPv6` addresses option enabled. By default, it's disabled.

Only private subnets

This VPC has three private subnets that are deployed into different Availability Zones in the AWS Region. Resources that are deployed to the subnets can't access the internet, nor can the internet access resources in the subnets. The template creates VPC endpoints using AWS PrivateLink for several AWS services that nodes typically need to access. If your nodes need outbound internet access, you can add a public NAT gateway in the Availability Zone of each subnet after the VPC is created. A security group is created that denies all inbound traffic, except from resources deployed into the subnets. A security group also allows all outbound traffic. The subnets are tagged so that Kubernetes can deploy internal load balancers to them. If you're creating a VPC with this configuration, see the section called "Private clusters" for additional requirements and considerations.

a. Open the AWS CloudFormation console.

b. From the navigation bar, select an AWS Region that supports Amazon EKS.

c. Choose **Create stack**, **With new resources (standard)**.

d. Under **Prepare template**, make sure that **Template is ready** is selected and then under **Template source**, select **Amazon S3 URL**.

e. Paste the following URL into the text area under **Amazon S3 URL** and choose **Next**:

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

a. On the **Specify Details** page, enter the parameters and then choose **Next**.

- **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it *amazon-eks-fully-private-vpc*. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.
- **VpcBlock**: Choose a CIDR block for your VPC. Each node, Pod, and load balancer that you deploy is assigned an `IPv4` address from this block. The default `IPv4` values provide enough IP addresses for most implementations, but if it doesn't, then you can change it. For more information, see VPC and subnet sizing in the Amazon VPC User Guide. You can also add additional CIDR blocks to the VPC once it's created.
- **PrivateSubnet01Block**: Specify a CIDR block for subnet 1. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it.
- **PrivateSubnet02Block**: Specify a CIDR block for subnet 2. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it.
- **PrivateSubnet03Block**: Specify a CIDR block for subnet 3. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it.

b. (Optional) On the **Options** page, tag your stack resources. Choose **Next**.

c. On the **Review** page, choose **Create**.

d. When your stack is created, select it in the console and choose **Outputs**.

e. Record the **VpcId** for the VPC that was created. You need this when you create your cluster and nodes.

f. Record the **SubnetIds** for the subnets that were created. You need at least two of these when you create your cluster and nodes.

g. (Optional) Any cluster that you deploy to this VPC can assign private `IPv4` addresses to your Pods and services. If you want deploy clusters to this VPC to assign private `IPv6` addresses to your Pods and services, make updates to your VPC, subnet, route tables, and security groups. For more information, see [Migrate existing VPCs from IPv4 to IPv6](#) in the Amazon VPC User Guide. Amazon EKS requires that your subnets have the `Auto-assign IPv6` addresses option enabled (it's disabled by default).

View Amazon EKS security group requirements for clusters

This topic describes the security group requirements of an Amazon EKS cluster.

Default cluster security group

When you create a cluster, Amazon EKS creates a security group that's named `eks-cluster-sg-my-cluster-uniqueID` . This security group has the following default rules:

Rule type	Protocol	Ports	Source	Destination
Inbound	All	All	Self	
Outbound	All	All		0.0.0.0/0(`IPv4`) or ::/0 (`IPv6`)

> **Important**
>
> If your cluster doesn't need the outbound rule, you can remove it. If you remove it, you must still have the minimum rules listed in [Restricting cluster traffic](#). If you remove the inbound rule, Amazon EKS recreates it whenever the cluster is updated.

Amazon EKS adds the following tags to the security group. If you remove the tags, Amazon EKS adds them back to the security group whenever your cluster is updated.

Key	Value
`kubernetes.io/cluster/ my-cluster`	`owned`
`aws:eks:cluster-name`	`my-cluster`
`Name`	`eks-cluster-sg- my-cluster -uniqueid`

Amazon EKS automatically associates this security group to the following resources that it also creates:

- 2–4 elastic network interfaces (referred to for the rest of this document as *network interface*) that are created when you create your cluster.
- Network interfaces of the nodes in any managed node group that you create.

The default rules allow all traffic to flow freely between your cluster and nodes, and allows all outbound traffic to any destination. When you create a cluster, you can (optionally) specify your own security groups. If you do, then Amazon EKS also associates the security groups that you specify to the network interfaces that it creates for your cluster. However, it doesn't associate them to any node groups that you create.

You can determine the ID of your cluster security group in the AWS Management Console under the cluster's **Networking** section. Or, you can do so by running the following AWS CLI command.

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

Restricting cluster traffic

If you need to limit the open ports between the cluster and nodes, you can remove the default outbound rule and add the following minimum rules that are required for the cluster. If you remove the default inbound rule, Amazon EKS recreates it whenever the cluster is updated.

Rule type	Protocol	Port	Destination
Outbound	TCP	443	Cluster security group
Outbound	TCP	10250	Cluster security group
Outbound (DNS)	TCP and UDP	53	Cluster security group

You must also add rules for the following traffic:

- Any protocol and ports that you expect your nodes to use for inter-node communication.
- Outbound internet access so that nodes can access the Amazon EKS APIs for cluster introspection and node registration at launch time. If your nodes don't have internet access, review Deploy private clusters with limited internet access for additional considerations.
- Node access to pull container images from Amazon ECR or other container registries APIs that they need to pull images from, such as DockerHub. For more information, see AWS IP address ranges in the AWS General Reference.
- Node access to Amazon S3.
- Separate rules are required for `IPv4` and `IPv6` addresses.
- If you are using hybrid nodes, you must add an additional security group to your cluster to allow communication with your on-premises nodes and pods. For more information, see the section called "Prepare networking".

If you're considering limiting the rules, we recommend that you thoroughly test all of your Pods before you apply your changed rules to a production cluster.

If you originally deployed a cluster with Kubernetes `1.14` and a platform version of `eks.3` or earlier, then consider the following:

- You might also have control plane and node security groups. When these groups were created, they included the restricted rules listed in the previous table. These security groups are no longer required and can be removed. However, you need to make sure your cluster security group contains the rules that those groups contain.
- If you deployed the cluster using the API directly or you used a tool such as the AWS CLI or AWS CloudFormation to create the cluster and you didn't specify a security group at cluster creation,

then the default security group for the VPC was applied to the cluster network interfaces that Amazon EKS created.

Shared security groups

Amazon EKS supports shared security groups.

- **Security Group VPC Associations** associate security groups with multiple VPCs in the same account and region.
  - Learn how to [Associate security groups with multiple VPCs] in the *Amazon VPC User Guide*.
- **Shared security groups** enable you to share security groups with other AWS accounts. The accounts must be in the same AWS organization.
  - Learn how to [Share security groups with organizations] in the *Amazon VPC User Guide*.
- Security groups are always limited to a single AWS region.

Considerations for Amazon EKS

- EKS has the same requirements of shared or multi-VPC security groups as standard security groups.

Manage networking add-ons for Amazon EKS clusters

Several networking add-ons are available for your Amazon EKS cluster.

Built-in add-ons

> **Note**
>
> If you create clusters in any way except by using the console, each cluster comes with the self-managed versions of the built-in add-ons. The self-managed versions can't be managed from the AWS Management Console, AWS Command Line Interface, or SDKs. You manage the configuration and upgrades of self-managed add-ons.
> We recommend adding the Amazon EKS type of the add-on to your cluster instead of using the self-managed type of the add-on. If you create clusters in the console, the Amazon EKS type of these add-ons is installed.

Amazon VPC CNI plugin for Kubernetes

This CNI add-on creates elastic network interfaces and attaches them to your Amazon EC2 nodes. The add-on also assigns a private `IPv4` or `IPv6` address from your VPC to each Pod and service. This add-on is installed, by default, on your cluster. For more information, see the section called "Amazon VPC CNI". If you are using hybrid nodes, the VPC CNI is still installed by default but it is prevented from running on your hybrid nodes with an anti-affinity rule. For more information about your CNI options for hybrid nodes, see the section called "Configure CNI".

CoreDNS

CoreDNS is a flexible, extensible DNS server that can serve as the Kubernetes cluster DNS. CoreDNS provides name resolution for all Pods in the cluster. This add-on is installed, by default, on your cluster. For more information, see the section called "CoreDNS".

`kube-proxy`

This add-on maintains network rules on your Amazon EC2 nodes and enables network communication to your Pods. This add-on is installed, by default, on your cluster. For more information, see the section called "kube-proxy".

Optional AWS networking add-ons

AWS Load Balancer Controller

When you deploy Kubernetes service objects of type `loadbalancer`, the controller creates AWS Network Load Balancers . When you create Kubernetes ingress objects, the controller creates AWS Application Load Balancers. We recommend using this controller to provision Network Load Balancers, rather than using the legacy Cloud Provider controller built-in to Kubernetes. For more information, see the AWS Load Balancer Controller documentation.

AWS Gateway API Controller

This controller lets you connect services across multiple Kubernetes clusters using the Kubernetes gateway API. The controller connects Kubernetes services running on Amazon EC2 instances, containers, and serverless functions by using the Amazon VPC Lattice service. For more information, see the AWS Gateway API Controller documentation.

For more information about add-ons, see the section called "Amazon EKS add-ons".

Assign IPs to Pods with the Amazon VPC CNI

Tip

With Amazon EKS Auto Mode, you don't need to install or upgrade networking add-ons. Auto Mode includes pod networking and load balancing capabilities.

For more information, see *EKS Auto Mode*.

The Amazon VPC CNI plugin for Kubernetes add-on is deployed on each Amazon EC2 node in your Amazon EKS cluster. The add-on creates elastic network interfaces and attaches them to your Amazon EC2 nodes. The add-on also assigns a private `IPv4` or `IPv6` address from your VPC to each Pod.

A version of the add-on is deployed with each Fargate node in your cluster, but you don't update it on Fargate nodes. Other compatible CNI plugins are available for use on Amazon EKS clusters, but this is the only CNI plugin supported by Amazon EKS for nodes that run on AWS infrastructure. For more information about the other compatible CNI plugins, see the section called "Alternate CNI plugins". The VPC CNI isn't supported for use with hybrid nodes. For more information about your CNI options for hybrid nodes, see the section called "Configure CNI".

The following table lists the latest available version of the Amazon EKS add-on type for each Kubernetes version.

Amazon VPC CNI versions

Kubernetes version	Amazon EKS type of VPC CNI version
1.32	v1.19.2-eksbuild.5
1.31	v1.19.2-eksbuild.5
1.30	v1.19.2-eksbuild.5
1.29	v1.19.2-eksbuild.5
1.28	v1.19.2-eksbuild.5
1.27	v1.19.2-eksbuild.5

Kubernetes version	Amazon EKS type of VPC CNI version
1.26	v1.19.2-eksbuild.5
1.25	v1.19.2-eksbuild.5
1.24	v1.19.2-eksbuild.5

Important

If you're self-managing this add-on, the versions in the table might not be the same as the available self-managed versions. For more information about updating the self-managed type of this add-on, see the section called "Update (self-managed)".

Important

To upgrade to VPC CNI v1.12.0 or later, you must upgrade to VPC CNI v1.7.0 first. We recommend that you update one minor version at a time.

Considerations

The following are considerations for using the feature.

- Versions are specified as `major-version.minor-version.patch-version-eksbuild.build-number`.
- Check version compatibility for each feature. Some features of each release of the Amazon VPC CNI plugin for Kubernetes require certain Kubernetes versions. When using different Amazon EKS features, if a specific version of the add-on is required, then it's noted in the feature documentation. Unless you have a specific reason for running an earlier version, we recommend running the latest version.

Create the Amazon VPC CNI (Amazon EKS add-on)

Use the following steps to create the Amazon VPC CNI plugin for Kubernetes Amazon EKS add-on.

Before you begin, review the considerations. For more information, see the section called "Considerations".

Prerequisites

The following are prerequisites for the Amazon VPC CNI plugin for Kubernetes Amazon EKS add-on.

- An existing Amazon EKS cluster. To deploy one, see *Get started*.
- An existing AWS Identity and Access Management (IAM) OpenID Connect (OIDC) provider for your cluster. To determine whether you already have one, or to create one, see the section called "IAM OIDC provider".
- An IAM role with the AmazonEKS_CNI_Policy IAM policy (if your cluster uses the `IPv4` family) or an IPv6 policy (if your cluster uses the `IPv6` family) attached to it. For more information about the VPC CNI role, see the section called "Configure for IRSA". For information about the IPv6 policy, see the section called "Create IAM policy for clusters that use the `IPv6` family".
- If you're using version `1.7.0` or later of the Amazon VPC CNI plugin for Kubernetes and you use custom Pod security policies, see the section called "Delete the default Amazon EKS Pod security policy" and the section called "Legacy default PSP".

> **Important**
>
> Amazon VPC CNI plugin for Kubernetes versions `v1.16.0` to `v1.16.1` removed compatibility with Kubernetes versions `1.23` and earlier. VPC CNI version `v1.16.2` restores compatibility with Kubernetes versions `1.23` and earlier and CNI spec `v0.4.0`.
> Amazon VPC CNI plugin for Kubernetes versions `v1.16.0` to `v1.16.1` implement CNI specification version `v1.0.0`. CNI spec `v1.0.0` is supported on EKS clusters that run the Kubernetes versions `v1.24` or later. VPC CNI version `v1.16.0` to `v1.16.1` and CNI spec `v1.0.0` aren't supported on Kubernetes version `v1.23` or earlier. For more information about `v1.0.0` of the CNI spec, see Container Network Interface (CNI) Specification on GitHub.

Procedure

After you complete the prerequisites, use the following steps to create the add-on.

1. See which version of the add-on is installed on your cluster.

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

An example output is as follows.

```
v1.16.4-eksbuild.2
```

2. See which type of the add-on is installed on your cluster. Depending on the tool that you created your cluster with, you might not currently have the Amazon EKS add-on type installed on your cluster. Replace *my-cluster* with the name of your cluster.

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
 addon.addonVersion --output text
```

If a version number is returned, you have the Amazon EKS type of the add-on installed on your cluster and don't need to complete the remaining steps in this procedure. If an error is returned, you don't have the Amazon EKS type of the add-on installed on your cluster. Complete the remaining steps of this procedure to install it.

3. Save the configuration of your currently installed add-on.

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

4. Create the add-on using the AWS CLI. If you want to use the AWS Management Console or `eksctl` to create the add-on, see the section called "Create an add-on" and specify `vpc-cni` for the add-on name. Copy the command that follows to your device. Make the following modifications to the command, as needed, and then run the modified command.

   - Replace *my-cluster* with the name of your cluster.
   - Replace *v1.19.2-eksbuild.1* with the latest version listed in the latest version table for your cluster version. For the latest version table, see the section called "Amazon VPC CNI versions".
   - Replace *111122223333* with your account ID and *AmazonEKSVPCCNIRole* with the name of an existing IAM role that you've created. Specifying a role requires that you have an IAM OpenID Connect (OIDC) provider for your cluster. To determine whether you have one for your cluster, or to create one, see the section called "IAM OIDC provider".

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 v1.19.2-eksbuild.1 \
    --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole
```

If you've applied custom settings to your current add-on that conflict with the default settings of the Amazon EKS add-on, creation might fail. If creation fails, you receive an error that can help you resolve the issue. Alternatively, you can add `--resolve-conflicts OVERWRITE` to the previous command. This allows the add-on to overwrite any existing custom settings. Once you've created the add-on, you can update it with your custom settings.

5. Confirm that the latest version of the add-on for your cluster's Kubernetes version was added to your cluster. Replace *`my-cluster`* with the name of your cluster.

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
 addon.addonVersion --output text
```

It might take several seconds for add-on creation to complete.

An example output is as follows.

```
v1.19.2-eksbuild.1
```

6. If you made custom settings to your original add-on, before you created the Amazon EKS add-on, use the configuration that you saved in a previous step to update the EKS add-on with your custom settings. Follow the steps in the section called "Update (EKS add-on)".
7. (Optional) Install the `cni-metrics-helper` to your cluster. It scrapes elastic network interface and IP address information, aggregates it at a cluster level, and publishes the metrics to Amazon CloudWatch. For more information, see cni-metrics-helper on GitHub.

Update the Amazon VPC CNI (Amazon EKS add-on)

Update the Amazon EKS type of the Amazon VPC CNI plugin for Kubernetes add-on. If you haven't added the Amazon EKS type of the add-on to your cluster, you can install it by following the section called "Create". Or, update the other type of VPC CNI installation by following the section called "Update (self-managed)".

1. See which version of the add-on is installed on your cluster. Replace *`my-cluster`* with your cluster name.

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
 "addon.addonVersion" --output text
```

An example output is as follows.

```
v1.16.4-eksbuild.2
```

Compare the version with the table of latest versions at the section called "Amazon VPC CNI versions". If the version returned is the same as the version for your cluster's Kubernetes version in the latest version table, then you already have the latest version installed on your cluster and don't need to complete the rest of this procedure. If you receive an error, instead of a version number in your output, then you don't have the Amazon EKS type of the add-on installed on your cluster. You need to create the add-on before you can update it with this procedure. To create the Amazon EKS type of the VPC CNI add-on, you can follow the section called "Create".

2. Save the configuration of your currently installed add-on.

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

3. Update your add-on using the AWS CLI. If you want to use the AWS Management Console or `eksctl` to update the add-on, see the section called "Update an add-on". Copy the command that follows to your device. Make the following modifications to the command, as needed, and then run the modified command.

   - Replace *`my-cluster`* with the name of your cluster.
   - Replace *`v1.19.2-eksbuild.1`* with the latest version listed in the latest version table for your cluster version.
   - Replace *`111122223333`* with your account ID and *`AmazonEKSVPCCNIRole`* with the name of an existing IAM role that you've created. To create an IAM role for the VPC CNI, see the section called "Step 1: Create the Amazon VPC CNI plugin for Kubernetes IAM role". Specifying a role requires that you have an IAM OpenID Connect (OIDC) provider for your cluster. To determine whether you have one for your cluster, or to create one, see the section called "IAM OIDC provider".
   - The `--resolve-conflicts PRESERVE` option preserves existing configuration values for the add-on. If you've set custom values for add-on settings, and you don't use this option, Amazon EKS overwrites your values with its default values. If you use this option, then we recommend testing any field and value changes on a non-production cluster before updating

the add-on on your production cluster. If you change this value to `OVERWRITE`, all settings are changed to Amazon EKS default values. If you've set custom values for any settings, they might be overwritten with Amazon EKS default values. If you change this value to `none`, Amazon EKS doesn't change the value of any settings, but the update might fail. If the update fails, you receive an error message to help you resolve the conflict.

- If you're not updating a configuration setting, remove `--configuration-values '{"env":{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}}'` from the command. If you're updating a configuration setting, replace *`"env":{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}`* with the setting that you want to set. In this example, the `AWS_VPC_K8S_CNI_EXTERNALSNAT` environment variable is set to `true`. The value that you specify must be valid for the configuration schema. If you don't know the configuration schema, run `aws eks describe-addon-configuration --addon-name vpc-cni --addon-version v1.19.2-eksbuild.1` , replacing *`v1.19.2-eksbuild.1`* with the version number of the add-on that you want to see the configuration for. The schema is returned in the output. If you have any existing custom configuration, want to remove it all, and set the values for all settings back to Amazon EKS defaults, remove *`"env":{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}`* from the command, so that you have empty `{}`. For an explanation of each setting, see [CNI Configuration Variables] on GitHub.

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 v1.19.2-eksbuild.1 \
    --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole \
    --resolve-conflicts PRESERVE --configuration-values '{"env":
{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}}'
```

It might take several seconds for the update to complete.

4. Confirm that the add-on version was updated. Replace *`my-cluster`* with the name of your cluster.

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni
```

It might take several seconds for the update to complete.

An example output is as follows.

```
{
    "addon": {
```

```
        "addonName": "vpc-cni",
        "clusterName": "my-cluster",
        "status": "ACTIVE",
        "addonVersion": "v1.19.2-eksbuild.1",
        "health": {
            "issues": []
        },
        "addonArn": "arn:aws:eks:region:111122223333:addon/my-cluster/vpc-
cni/74c33d2f-b4dc-8718-56e7-9fdfa65d14a9",
        "createdAt": "2023-04-12T18:25:19.319000+00:00",
        "modifiedAt": "2023-04-12T18:40:28.683000+00:00",
        "serviceAccountRoleArn": "arn:aws:iam::111122223333:role/
AmazonEKSVPCCNIRole",
        "tags": {},
        "configurationValues": "{\"env\":{\"AWS_VPC_K8S_CNI_EXTERNALSNAT\":\"true
\"}}"
    }
}
```

Update the Amazon VPC CNI (self-managed add-on)

> **Important**
>
> We recommend adding the Amazon EKS type of the add-on to your cluster instead of using the self-managed type of the add-on. If you're not familiar with the difference between the types, see the section called "Amazon EKS add-ons". For more information about adding an Amazon EKS add-on to your cluster, see the section called "Create an add-on". If you're unable to use the Amazon EKS add-on, we encourage you to submit an issue about why you can't to the Containers roadmap GitHub repository.

1. Confirm that you don't have the Amazon EKS type of the add-on installed on your cluster. Replace *my-cluster* with the name of your cluster.

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
    addon.addonVersion --output text
   ```

   If an error message is returned, you don't have the Amazon EKS type of the add-on installed on your cluster. To self-manage the add-on, complete the remaining steps in this procedure to update the add-on. If a version number is returned, you have the Amazon EKS type of the add-on installed on your cluster. To update it, use the procedure in the section called “Update an add-on”, rather than using this procedure. If you're not familiar with the differences between the add-on types, see the section called “Amazon EKS add-ons”.

2. See which version of the container image is currently installed on your cluster.

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

An example output is as follows.

```
v1.19.2-eksbuild.1
```

Your output might not include the build number.

3. Backup your current settings so you can configure the same settings once you've updated your version.

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

To review the available versions and familiarize yourself with the changes in the version that you want to update to, see releases on GitHub. Note that we recommend updating to the same `major.minor.patch` version listed in the latest available versions table, even if later versions are available on GitHub. For the latest available version table, see the section called “Amazon VPC CNI versions”. The build versions listed in the table aren't specified in the self-managed versions listed on GitHub. Update your version by completing the tasks in one of the following options:

- If you don't have any custom settings for the add-on, then run the command under the `To apply this release:` heading on GitHub for the release that you're updating to.
- If you have custom settings, download the manifest file with the following command. Change *`https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.19.2/config/master/aws-k8s-cni.yaml`* to the URL for the release on GitHub that you're updating to.

```
curl -O https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.19.2/config/
master/aws-k8s-cni.yaml
```

If necessary, modify the manifest with the custom settings from the backup you made in a previous step and then apply the modified manifest to your cluster. If your nodes don't have access to the private Amazon EKS Amazon ECR repositories that the images are pulled from (see the lines that start with `image:` in the manifest), then you'll have to download the images, copy them to your own repository, and modify the manifest to pull the images from your repository. For more information, see the section called "Copy an image to a repository".

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

4. Confirm that the new version is now installed on your cluster.

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

An example output is as follows.

```
v1.19.2
```

5. (Optional) Install the `cni-metrics-helper` to your cluster. It scrapes elastic network interface and IP address information, aggregates it at a cluster level, and publishes the metrics to Amazon CloudWatch. For more information, see cni-metrics-helper on GitHub.

Configure Amazon VPC CNI plugin to use IRSA

The Amazon VPC CNI plugin for Kubernetes is the networking plugin for Pod networking in Amazon EKS clusters. The plugin is responsible for allocating VPC IP addresses to Kubernetes nodes and configuring the necessary networking for Pods on each node. The plugin:

- Requires AWS Identity and Access Management (IAM) permissions. If your cluster uses the `IPv4` family, the permissions are specified in the `AmazonEKS_CNI_Policy` AWS managed policy. If your cluster uses the `IPv6` family, then the permissions must be added to an IAM policy that you create; for instructions, see the section called "Create IAM policy for clusters that use the IPv6 family". You can attach the policy to the Amazon EKS node IAM role, or to a separate IAM role. For instructions to attach the policy to the Amazon EKS node IAM role, see the section called "Node IAM role". We recommend that you assign it to a separate role, as detailed in this topic.

- Creates and is configured to use a Kubernetes service account named `aws-node` when it's deployed. The service account is bound to a Kubernetes `clusterrole` named `aws-node`, which is assigned the required Kubernetes permissions.

> **Note**
>
> The Pods for the Amazon VPC CNI plugin for Kubernetes have access to the permissions assigned to the Amazon EKS node IAM role, unless you block access to IMDS. For more information, see Restrict access to the instance profile assigned to the worker node.

- An existing Amazon EKS cluster. To deploy one, see *Get started*.
- An existing AWS Identity and Access Management (IAM) OpenID Connect (OIDC) provider for your cluster. To determine whether you already have one, or to create one, see the section called "IAM OIDC provider".

Step 1: Create the Amazon VPC CNI plugin for Kubernetes IAM role

1. Determine the IP family of your cluster.

```
aws eks describe-cluster --name my-cluster | grep ipFamily
```

   An example output is as follows.

```
"ipFamily": "ipv4"
```

   The output may return `ipv6` instead.

2. Create the IAM role. You can use `eksctl` or `kubectl` and the AWS CLI to create your IAM role.

   eksctl

   - Create an IAM role and attach the IAM policy to the role with the command that matches the IP family of your cluster. The command creates and deploys an AWS CloudFormation stack that creates an IAM role, attaches the policy that you specify to it, and annotates the existing `aws-node` Kubernetes service account with the ARN of the IAM role that is created.

     - `IPv4`

Replace *my-cluster* with your own value.

```
eksctl create iamserviceaccount \
    --name aws-node \
    --namespace kube-system \
    --cluster my-cluster \
    --role-name AmazonEKSVPCCNIRole \
    --attach-policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
    --override-existing-serviceaccounts \
    --approve
```

- `IPv6`

  Replace *my-cluster* with your own value. Replace *111122223333* with your account ID and replace *AmazonEKS_CNI_IPv6_Policy* with the name of your `IPv6` policy. If you don't have an `IPv6` policy, see the section called "Create IAM policy for clusters that use the IPv6 family" to create one. To use `IPv6` with your cluster, it must meet several requirements. For more information, see the section called "IPv6".

```
eksctl create iamserviceaccount \
    --name aws-node \
    --namespace kube-system \
    --cluster my-cluster \
    --role-name AmazonEKSVPCCNIRole \
    --attach-policy-arn arn:aws:iam::111122223333:policy/
AmazonEKS_CNI_IPv6_Policy \
    --override-existing-serviceaccounts \
    --approve
```

kubectl and the AWS CLI

i. View your cluster's OIDC provider URL.

```
aws eks describe-cluster --name my-cluster --query
 "cluster.identity.oidc.issuer" --output text
```

An example output is as follows.

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

If no output is returned, then you must create an IAM OIDC provider for your cluster.

ii. Copy the following contents to a file named *vpc-cni-trust-policy.json*. Replace *111122223333* with your account ID and *EXAMPLED539D4633E53DE1B71EXAMPLE* with the output returned in the previous step. Replace *region-code* with the AWS Region that your cluster is in.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
                    "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:aws-
node"
                }
            }
        }
    ]
}
```

iii. Create the role. You can replace *AmazonEKSVPCCNIRole* with any name that you choose.

```
aws iam create-role \
  --role-name AmazonEKSVPCCNIRole \
  --assume-role-policy-document file://"vpc-cni-trust-policy.json"
```

iv. Attach the required IAM policy to the role. Run the command that matches the IP family of your cluster.

- `IPv4`

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

- IPv6

  Replace *111122223333* with your account ID and *AmazonEKS_CNI_IPv6_Policy* with the name of your IPv6 policy. If you don't have an IPv6 policy, see the section called "Create IAM policy for clusters that use the IPv6 family" to create one. To use IPv6 with your cluster, it must meet several requirements. For more information, see the section called "IPv6".

  ```
  aws iam attach-role-policy \
    --policy-arn arn:aws:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy \
    --role-name AmazonEKSVPCCNIRole
  ```

v. Run the following command to annotate the aws-node service account with the ARN of the IAM role that you created previously. Replace the *example values* with your own values.

```
kubectl annotate serviceaccount \
    -n kube-system aws-node \
    eks.amazonaws.com/role-arn=arn:aws:iam::111122223333:role/
AmazonEKSVPCCNIRole
```

3. (Optional) Configure the AWS Security Token Service endpoint type used by your Kubernetes service account. For more information, see the section called "STS endpoints".

Step 2: Re-deploy Amazon VPC CNI plugin for Kubernetes Pods

1. Delete and re-create any existing Pods that are associated with the service account to apply the credential environment variables. The annotation is not applied to Pods that are currently running without the annotation. The following command deletes the existing aws-node DaemonSet Pods and deploys them with the service account annotation.

```
kubectl delete Pods -n kube-system -l k8s-app=aws-node
```

2. Confirm that the Pods all restarted.

```
kubectl get pods -n kube-system -l k8s-app=aws-node
```

3. Describe one of the Pods and verify that the `AWS_WEB_IDENTITY_TOKEN_FILE` and `AWS_ROLE_ARN` environment variables exist. Replace *cpjw7* with the name of one of your Pods returned in the output of the previous step.

```
kubectl describe pod -n kube-system aws-node-cpjw7 | grep 'AWS_ROLE_ARN:\|
AWS_WEB_IDENTITY_TOKEN_FILE:'
```

An example output is as follows.

```
AWS_ROLE_ARN:                    arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole
      AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
      AWS_ROLE_ARN:                            arn:aws:iam::111122223333:role/
AmazonEKSVPCCNIRole
      AWS_WEB_IDENTITY_TOKEN_FILE:            /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
```

Two sets of duplicate results are returned because the Pod contains two containers. Both containers have the same values.

If your Pod is using the AWS Regional endpoint, then the following line is also returned in the previous output.

```
AWS_STS_REGIONAL_ENDPOINTS=regional
```

Step 3: Remove the CNI policy from the node IAM role

If your Amazon EKS node IAM role currently has the `AmazonEKS_CNI_Policy` IAM (`IPv4`) policyor an IPv6 policyattached to it, and you've created a separate IAM role, attached the policy to it instead, and assigned it to the `aws-node` Kubernetes service account, then we recommend that you remove the policy from your node role with the AWS CLI command that matches the IP family of your cluster. Replace *AmazonEKSNodeRole* with the name of your node role.

- `IPv4`

```
aws iam detach-role-policy --role-name AmazonEKSNodeRole --policy-arn
 arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
```

- `IPv6`

Replace *111122223333* with your account ID and *AmazonEKS_CNI_IPv6_Policy* with the name of your IPv6 policy.

```
aws iam detach-role-policy --role-name AmazonEKSNodeRole --policy-arn
 arn:aws:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy
```

Create IAM policy for clusters that use the IPv6 family

If you created a cluster that uses the IPv6 family and the cluster has version 1.10.1 or later of the Amazon VPC CNI plugin for Kubernetes add-on configured, then you need to create an IAM policy that you can assign to an IAM role. If you have an existing cluster that you didn't configure with the IPv6 family when you created it, then to use IPv6, you must create a new cluster. For more information about using IPv6 with your cluster, see the section called "IPv6".

1. Copy the following text and save it to a file named vpc-cni-ipv6-policy.json.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AssignIpv6Addresses",
                "ec2:DescribeInstances",
                "ec2:DescribeTags",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeInstanceTypes"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*"
            ]
        }
    ]
```

```
}
```

2. Create the IAM policy.

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-document
 file://vpc-cni-ipv6-policy.json
```

Learn about VPC CNI modes and configuration

The Amazon VPC CNI plugin for Kubernetes provides networking for Pods. Use the following table to learn more about the available networking features.

Networking feature	Learn more
Configure your cluster to assign IPv6 addresses to clusters, Pods, and services	the section called “IPv6”
Use IPv4 Source Network Address Translation for Pods	the section called “Outbound traffic”
Restrict network traffic to and from your Pods	the section called “Restrict traffic”
Customize the secondary network interface in nodes	the section called “Custom networking”
Increase IP addresses for your node	the section called “Increase IP addresses”
Use security groups for Pod network traffic	the section called “Security groups for Pods”
Use multiple network interfaces for Pods	the section called “Multiple interfaces”

Learn about IPv6 addresses to clusters, Pods, and services

Applies to: Pods with Amazon EC2 instances and Fargate Pods

By default, Kubernetes assigns `IPv4` addresses to your Pods and services. Instead of assigning `IPv4` addresses to your Pods and services, you can configure your cluster to assign `IPv6` addresses to them. Amazon EKS doesn’t support dual-stacked Pods or services, even though Kubernetes does in version 1.23 and later. As a result, you can't assign both IPv4 and IPv6 addresses to your Pods and services.

You select which IP family you want to use for your cluster when you create it. You can't change the family after you create the cluster.

For a tutorial to deploy an Amazon EKS IPv6 cluster, see the section called "Deploy".

The following are considerations for using the feature:

IPv6 Feature support

- **No Windows support**: Windows Pods and services aren't supported.
- **Nitro-based EC2 nodes required**: You can only use IPv6 with AWS Nitro-based Amazon EC2 or Fargate nodes.
- **EC2 and Fargate nodes supported**: You can use IPv6 with the section called "Security groups for Pods" with Amazon EC2 nodes and Fargate nodes.
- **Outposts not supported**: You can't use IPv6 with *Amazon EKS on AWS Outposts*.
- **FSx for Lustre is not supported**: The the section called "Amazon FSx for Lustre" is not supported.
- **Instance Metadata Service not supported**: Use of the Amazon EC2 Instance Metadata Service IPv6 endpoint is not supported with Amazon EKS.
- **Custom networking not supported**: If you previously used the section called "Custom networking" to help alleviate IP address exhaustion, you can use IPv6 instead. You can't use custom networking with IPv6. If you use custom networking for network isolation, then you might need to continue to use custom networking and the IPv4 family for your clusters.

IP address assignments

- **Kubernetes services**: Kubernetes services are only assigned an IPv6 addresses. They aren't assigned IPv4 addresses.
- **Pods**: Pods are assigned an IPv6 address and a host-local IPv4 address. The host-local IPv4 address is assigned by using a host-local CNI plugin chained with VPC CNI and the address is not reported to the Kubernetes control plane. It is only used when a pod needs to communicate with an external IPv4 resources in another Amazon VPC or the internet. The host-local IPv4 address gets SNATed (by VPC CNI) to the primary IPv4 address of the primary ENI of the worker node.

- **Pods and services**: Pods and services are only assigned an IPv6 address. They aren't assigned an IPv4 address. Because Pods are able to communicate to IPv4 endpoints through NAT on the instance itself, DNS64 and NAT64 aren't needed. If the traffic needs a public IP address, the traffic is then source network address translated to a public IP.
- **Routing addresses**: The source IPv6 address of a Pod isn't source network address translated to the IPv6 address of the node when communicating outside of the VPC. It is routed using an internet gateway or egress-only internet gateway.
- **Nodes**: All nodes are assigned an IPv4 and IPv6 address.
- **Fargate Pods**: Each Fargate Pod receives an IPv6 address from the CIDR that's specified for the subnet that it's deployed in. The underlying hardware unit that runs Fargate Pods gets a unique IPv4 and IPv6 address from the CIDRs that are assigned to the subnet that the hardware unit is deployed in.

How to use IPv6 with EKS

- **Create new cluster**: You must create a new cluster and specify that you want to use the IPv6 family for that cluster. You can't enable the IPv6 family for a cluster that you updated from a previous version. For instructions on how to create a new cluster, see Considerations .
- **Use recent VPC CNI**: Deploy Amazon VPC CNI version 1.10.1 or later. This version or later is deployed by default. After you deploy the add-on, you can't downgrade your Amazon VPC CNI add-on to a version lower than 1.10.1 without first removing all nodes in all node groups in your cluster.
- **Configure VPC CNI for IPv6** : If you use Amazon EC2 nodes, you must configure the Amazon VPC CNI add-on with IP prefix delegation and IPv6. If you choose the IPv6 family when creating your cluster, the 1.10.1 version of the add-on defaults to this configuration. This is the case for both a self-managed or Amazon EKS add-on. For more information about IP prefix delegation, see the section called "Increase IP addresses".
- **Configure IPv4 and IPv6 addresses**: When you create a cluster, the VPC and subnets that you specify must have an IPv6 CIDR block that's assigned to the VPC and subnets that you specify. They must also have an IPv4 CIDR block assigned to them. This is because, even if you only want to use IPv6, a VPC still requires an IPv4 CIDR block to function. For more information, see Associate an IPv6 CIDR block with your VPC in the Amazon VPC User Guide.
- **Auto-assign IPv6 addresses to nodes:** When you create your nodes, you must specify subnets that are configured to auto-assign IPv6 addresses. Otherwise, you can't deploy your nodes. By

default, this configuration is disabled. For more information, see Modify the IPv6 addressing attribute for your subnet in the Amazon VPC User Guide.

- **Set route tables to use `IPv6`** : The route tables that are assigned to your subnets must have routes for `IPv6` addresses. For more information, see Migrate to IPv6 in the Amazon VPC User Guide.
- **Set security groups for `IPv6`** : Your security groups must allow `IPv6` addresses. For more information, see Migrate to IPv6 in the Amazon VPC User Guide.
- **Set up load balancer**: Use version `2.3.1` or later of the AWS Load Balancer Controller to load balance HTTP applications using the the section called "Application load balancing" or network traffic using the the section called "Network load balancing" to `IPv6` Pods with either load balancer in IP mode, but not instance mode. For more information, see the section called " AWS Load Balancer Controller".
- **Add `IPv6` IAM policy**: You must attach an `IPv6` IAM policy to your node IAM or CNI IAM role. Between the two, we recommend that you attach it to a CNI IAM role. For more information, see the section called "Create IAM policy for clusters that use the `IPv6` family" and the section called "Step 1: Create the Amazon VPC CNI plugin for Kubernetes IAM role".
- **Evaluate all components**: Perform a thorough evaluation of your applications, Amazon EKS add-ons, and AWS services that you integrate with before deploying `IPv6` clusters. This is to ensure that everything works as expected with `IPv6`.
- **Add `BootstrapArguments` self-managed node groups**: When creating a self-managed node group in a cluster that uses the `IPv6` family, user-data must include the following `BootstrapArguments` for the bootstrap.sh file that runs at node start up. Replace *`your-cidr`* with the `IPv6` CIDR range of your cluster's VPC.

```
--ip-family ipv6 --service-ipv6-cidr your-cidr
```

If you don't know the `IPv6 CIDR` range for your cluster, you can see it with the following command (requires the AWS CLI version `2.4.9` or later).

```
aws eks describe-cluster --name my-cluster --query
 cluster.kubernetesNetworkConfig.serviceIpv6Cidr --output text
```

Deploying an Amazon EKS IPv6 cluster and managed Amazon Linux nodes

In this tutorial, you deploy an `IPv6` Amazon VPC, an Amazon EKS cluster with the `IPv6` family, and a managed node group with Amazon EC2 Amazon Linux nodes. You can't deploy Amazon EC2 Windows nodes in an `IPv6` cluster. You can also deploy Fargate nodes to your cluster, though those instructions aren't provided in this topic for simplicity.

Prerequisites

Complete the following before you start the tutorial:

Install and configure the following tools and resources that you need to create and manage an Amazon EKS cluster.

- We recommend that you familiarize yourself with all settings and deploy a cluster with the settings that meet your requirements. For more information, see the section called "Create a cluster", the section called "Managed node groups", and the Considerations for this topic. You can only enable some settings when creating your cluster.
- The `kubectl` command line tool is installed on your device or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. For example, if your cluster version is `1.29`, you can use `kubectl` version `1.28`, `1.29`, or `1.30` with it. To install or upgrade `kubectl`, see the section called "Set up kubectl and eksctl".
- The IAM security principal that you're using must have permissions to work with Amazon EKS IAM roles, service linked roles, AWS CloudFormation, a VPC, and related resources. For more information, see Actions and Using service-linked roles in the IAM User Guide.
- If you use the eksctl, install version `0.204.0` or later on your computer. To install or update to it, see Installation in the `eksctl` documentation.
- Version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version, use `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*. The AWS CLI version that is installed in AWS CloudShell might also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the *AWS CloudShell User Guide*. If you use the AWS CloudShell, you may need to install version 2.12.3 or later or 1.27.160 or later of the AWS CLI, because the default AWS CLI version installed in the AWS CloudShell may be an earlier version.

You can use the eksctl or CLI to deploy an `IPv6` cluster.

Deploy an IPv6 cluster with eksctl

a. Create the `ipv6-cluster.yaml` file. Copy the command that follows to your device. Make the following modifications to the command as needed and then run the modified command:

   - Replace *`my-cluster`* with a name for your cluster. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.
   - Replace *`region-code`* with any AWS Region that is supported by Amazon EKS. For a list of AWS Regions, see Amazon EKS endpoints and quotas in the AWS General Reference guide.
   - The value for `version` with the version of your cluster. For more information, see the section called "Kubernetes versions".
   - Replace *`my-nodegroup`* with a name for your node group. The node group name can't be longer than 63 characters. It must start with letter or digit, but can also include hyphens and underscores for the remaining characters.
   - Replace *`t3.medium`* with any AWS Nitro System instance type.

```
cat >ipv6-cluster.yaml <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: "X.XX"

kubernetesNetworkConfig:
  ipFamily: IPv6

addons:
  - name: vpc-cni
    version: latest
  - name: coredns
    version: latest
  - name: kube-proxy
    version: latest
```

```
iam:
  withOIDC: true

managedNodeGroups:
  - name: my-nodegroup
    instanceType: t3.medium
EOF
```

b. Create your cluster.

```
eksctl create cluster -f ipv6-cluster.yaml
```

Cluster creation takes several minutes. Don't proceed until you see the last line of output, which looks similar to the following output.

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

c. Confirm that default Pods are assigned `IPv6` addresses.

```
kubectl get pods -n kube-system -o wide
```

An example output is as follows.

```
NAME                        READY   STATUS    RESTARTS   AGE     IP                                     NODE                                            NOMINATED NODE   READINESS GATES
aws-node-rslts              1/1     Running   1          5m36s   2600:1f13:b66:8200:11a5:ade0:c590:6ac8   ip-192-168-34-75.region-code.compute.internal   <none>           <none>
aws-node-t74jh              1/1     Running   0          5m32s   2600:1f13:b66:8203:4516:2080:8ced:1ca9   ip-192-168-253-70.region-code.compute.internal  <none>           <none>
coredns-85d5b4454c-cw7w2    1/1     Running   0          56m     2600:1f13:b66:8203:34e5::                ip-192-168-253-70.region-code.compute.internal  <none>           <none>
coredns-85d5b4454c-tx6n8    1/1     Running   0          56m     2600:1f13:b66:8203:34e5::1               ip-192-168-253-70.region-code.compute.internal  <none>           <none>
kube-proxy-btpbk            1/1     Running   0          5m36s   2600:1f13:b66:8200:11a5:ade0:c590:6ac8   ip-192-168-34-75.region-code.compute.internal   <none>           <none>
```

```
kube-proxy-jjk2g           1/1     Running   0          5m33s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9   ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
```

d. Confirm that default services are assigned IPv6 addresses.

```
kubectl get services -n kube-system -o wide
```

An example output is as follows.

```
NAME       TYPE        CLUSTER-IP          EXTERNAL-IP   PORT(S)         AGE
 SELECTOR
kube-dns   ClusterIP   fd30:3087:b6c2::a   <none>        53/UDP,53/TCP   57m   k8s-
app=kube-dns
```

e. (Optional) Deploy a sample application or deploy the AWS Load Balancer Controller and a sample application to load balance HTTP applications with the section called "Application load balancing" or network traffic with the section called "Network load balancing" to IPv6 Pods.

f. After you've finished with the cluster and nodes that you created for this tutorial, you should clean up the resources that you created with the following command.

```
eksctl delete cluster my-cluster
```

Deploy an IPv6 cluster with AWS CLI

Important

- You must complete all steps in this procedure as the same user. To check the current user, run the following command:

```
aws sts get-caller-identity
```

- You must complete all steps in this procedure in the same shell. Several steps use variables set in previous steps. Steps that use variables won't function properly if the variable values are set in a different shell. If you use the AWS CloudShell to complete the following procedure, remember that if you don't interact with it using your keyboard or pointer for approximately 20–30 minutes, your shell session ends. Running processes do not count as interactions.

- The instructions are written for the Bash shell, and may need adjusting in other shells.

Replace all *example values* in the steps of this procedure with your own values.

a. Run the following commands to set some variables used in later steps. Replace *region-code* with the AWS Region that you want to deploy your resources in. The value can be any AWS Region that is supported by Amazon EKS. For a list of AWS Regions, see [Amazon EKS endpoints and quotas] in the AWS General Reference guide. Replace *my-cluster* with a name for your cluster. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in. Replace *my-nodegroup* with a name for your node group. The node group name can't be longer than 63 characters. It must start with letter or digit, but can also include hyphens and underscores for the remaining characters. Replace *111122223333* with your account ID.

```
export region_code=region-code
export cluster_name=my-cluster
export nodegroup_name=my-nodegroup
export account_id=111122223333
```

b. Create an Amazon VPC with public and private subnets that meets Amazon EKS and `IPv6` requirements.

   i. Run the following command to set a variable for your AWS CloudFormation stack name. You can replace *my-eks-ipv6-vpc* with any name you choose.

   ```
   export vpc_stack_name=my-eks-ipv6-vpc
   ```

   ii. Create an `IPv6` VPC using an AWS CloudFormation template.

   ```
   aws cloudformation create-stack --region $region_code --stack-name $vpc_stack_name
    \
     --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
   cloudformation/2020-10-29/amazon-eks-ipv6-vpc-public-private-subnets.yaml
   ```

   The stack takes a few minutes to create. Run the following command. Don't continue to the next step until the output of the command is `CREATE_COMPLETE`.

```
aws cloudformation describe-stacks --region $region_code --stack-name
 $vpc_stack_name --query Stacks[].StackStatus --output text
```

iii. Retrieve the IDs of the public subnets that were created.

```
aws cloudformation describe-stacks --region $region_code --stack-name
 $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SubnetsPublic`].OutputValue' --output
 text
```

An example output is as follows.

```
subnet-0a1a56c486EXAMPLE,subnet-099e6ca77aEXAMPLE
```

iv. Enable the auto-assign `IPv6` address option for the public subnets that were created.

```
aws ec2 modify-subnet-attribute --region $region_code --subnet-id
 subnet-0a1a56c486EXAMPLE --assign-ipv6-address-on-creation
aws ec2 modify-subnet-attribute --region $region_code --subnet-id
 subnet-099e6ca77aEXAMPLE --assign-ipv6-address-on-creation
```

v. Retrieve the names of the subnets and security groups created by the template from the deployed AWS CloudFormation stack and store them in variables for use in a later step.

```
security_groups=$(aws cloudformation describe-stacks --region $region_code --
stack-name $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SecurityGroups`].OutputValue' --output
 text)

public_subnets=$(aws cloudformation describe-stacks --region $region_code --stack-
name $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SubnetsPublic`].OutputValue' --output
 text)

private_subnets=$(aws cloudformation describe-stacks --region $region_code --
stack-name $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SubnetsPrivate`].OutputValue' --output
 text)

subnets=${public_subnets},${private_subnets}
```

c. Create a cluster IAM role and attach the required Amazon EKS IAM managed policy to it. Kubernetes clusters managed by Amazon EKS make calls to other AWS services on your behalf to manage the resources that you use with the service.

   i. Run the following command to create the `eks-cluster-role-trust-policy.json` file.

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

   ii. Run the following command to set a variable for your role name. You can replace *`myAmazonEKSClusterRole`* with any name you choose.

```
export cluster_role_name=myAmazonEKSClusterRole
```

   iii. Create the role.

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

   iv. Retrieve the ARN of the IAM role and store it in a variable for a later step.

```
CLUSTER_IAM_ROLE=$(aws iam get-role --role-name $cluster_role_name --
query="Role.Arn" --output text)
```

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

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

d. Create your cluster.

```
aws eks create-cluster --region $region_code --name $cluster_name --kubernetes-version 1.XX \
   --role-arn $CLUSTER_IAM_ROLE --resources-vpc-config subnetIds=
$subnets,securityGroupIds=$security_groups \
   --kubernetes-network-config ipFamily=ipv6
```

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

   The cluster takes several minutes to create. Run the following command. Don't continue to the next step until the output from the command is `ACTIVE`.

```
aws eks describe-cluster --region $region_code --name $cluster_name --query
 cluster.status
```

e. Create or update a `kubeconfig` file for your cluster so that you can communicate with your cluster.

```
aws eks update-kubeconfig --region $region_code --name $cluster_name
```

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

f. Create a node IAM role.

i. Run the following command to create the `vpc-cni-ipv6-policy.json` file.

```
cat >vpc-cni-ipv6-policy.json <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AssignIpv6Addresses",
                "ec2:DescribeInstances",
                "ec2:DescribeTags",
                "ec2:DescribeNetworkInterfaces",
```

```
                "ec2:DescribeInstanceTypes"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*"
            ]
        }
    ]
}
EOF
```

ii. Create the IAM policy.

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-document
 file://vpc-cni-ipv6-policy.json
```

iii. Run the following command to create the `node-role-trust-relationship.json` file.

```
cat >node-role-trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

iv. Run the following command to set a variable for your role name. You can replace *`AmazonEKSNodeRole`* with any name you choose.

```
export node_role_name=AmazonEKSNodeRole
```

v. Create the IAM role.

```
aws iam create-role --role-name $node_role_name --assume-role-policy-document
 file://"node-role-trust-relationship.json"
```

vi. Attach the IAM policy to the IAM role.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::$account_id:policy/
AmazonEKS_CNI_IPv6_Policy \
    --role-name $node_role_name
```

Important

For simplicity in this tutorial, the policy is attached to this IAM role. In a production cluster however, we recommend attaching the policy to a separate IAM role. For more information, see the section called "Configure for IRSA".

vii. Attach two required IAM managed policies to the IAM role.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSWorkerNodePolicy \
  --role-name $node_role_name
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEC2ContainerRegistryReadOnly \
  --role-name $node_role_name
```

viii. Retrieve the ARN of the IAM role and store it in a variable for a later step.

```
node_iam_role=$(aws iam get-role --role-name $node_role_name --query="Role.Arn" --
output text)
```

g. Create a managed node group.

i. View the IDs of the subnets that you created in a previous step.

```
echo $subnets
```

An example output is as follows.

```
subnet-0a1a56c486EXAMPLE,subnet-099e6ca77aEXAMPLE,subnet-0377963d69EXAMPLE,subnet-0c05f819
```

ii. Create the node group. Replace *0a1a56c486EXAMPLE*, *099e6ca77aEXAMPLE*, *0377963d69EXAMPLE*, and *0c05f819d5EXAMPLE* with the values returned in the output of the previous step. Be sure to remove the commas between subnet IDs from the previous output in the following command. You can replace *t3.medium* with any AWS Nitro System instance type.

```
aws eks create-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name \
    --subnets subnet-0a1a56c486EXAMPLE subnet-099e6ca77aEXAMPLE
 subnet-0377963d69EXAMPLE subnet-0c05f819d5EXAMPLE \
    --instance-types t3.medium --node-role $node_iam_role
```

The node group takes a few minutes to create. Run the following command. Don't proceed to the next step until the output returned is `ACTIVE`.

```
aws eks describe-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name \
    --query nodegroup.status --output text
```

h. Confirm that the default Pods are assigned `IPv6` addresses in the `IP` column.

```
kubectl get pods -n kube-system -o wide
```

An example output is as follows.

```
NAME                           READY   STATUS    RESTARTS   AGE     IP
                     NODE                                            NOMINATED NODE
 READINESS GATES
aws-node-rslts                 1/1     Running   1          5m36s
 2600:1f13:b66:8200:11a5:ade0:c590:6ac8   ip-192-168-34-75.region-
code.compute.internal   <none>           <none>
aws-node-t74jh                 1/1     Running   0          5m32s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9   ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
coredns-85d5b4454c-cw7w2       1/1     Running   0          56m
 2600:1f13:b66:8203:34e5::                ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
```

```
coredns-85d5b4454c-tx6n8   1/1     Running   0          56m
 2600:1f13:b66:8203:34e5::1                ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
kube-proxy-btpbk           1/1     Running   0          5m36s
 2600:1f13:b66:8200:11a5:ade0:c590:6ac8   ip-192-168-34-75.region-
code.compute.internal   <none>           <none>
kube-proxy-jjk2g           1/1     Running   0          5m33s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9   ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
```

i. Confirm that the default services are assigned `IPv6` addresses in the `IP` column.

```
kubectl get services -n kube-system -o wide
```

An example output is as follows.

```
NAME       TYPE        CLUSTER-IP          EXTERNAL-IP   PORT(S)         AGE
 SELECTOR
kube-dns   ClusterIP   fd30:3087:b6c2::a   <none>        53/UDP,53/TCP   57m   k8s-
app=kube-dns
```

j. (Optional) Deploy a sample application or deploy the AWS Load Balancer Controller and a sample application to load balance HTTP applications with the section called “Application load balancing” or network traffic with the section called “Network load balancing” to `IPv6` Pods.

k. After you’ve finished with the cluster and nodes that you created for this tutorial, you should clean up the resources that you created with the following commands. Make sure that you’re not using any of the resources outside of this tutorial before deleting them.

   i. If you’re completing this step in a different shell than you completed the previous steps in, set the values of all the variables used in previous steps, replacing the *`example values`* with the values you specified when you completed the previous steps. If you’re completing this step in the same shell that you completed the previous steps in, skip to the next step.

```
export region_code=region-code
export vpc_stack_name=my-eks-ipv6-vpc
export cluster_name=my-cluster
export nodegroup_name=my-nodegroup
export account_id=111122223333
export node_role_name=AmazonEKSNodeRole
export cluster_role_name=myAmazonEKSClusterRole
```

ii. Delete your node group.

```
aws eks delete-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name
```

Deletion takes a few minutes. Run the following command. Don't proceed to the next step if any output is returned.

```
aws eks list-nodegroups --region $region_code --cluster-name $cluster_name --query
 nodegroups --output text
```

iii. Delete the cluster.

```
aws eks delete-cluster --region $region_code --name $cluster_name
```

The cluster takes a few minutes to delete. Before continuing make sure that the cluster is deleted with the following command.

```
aws eks describe-cluster --region $region_code --name $cluster_name
```

Don't proceed to the next step until your output is similar to the following output.

```
An error occurred (ResourceNotFoundException) when calling the DescribeCluster
 operation: No cluster found for name: my-cluster.
```

iv. Delete the IAM resources that you created. Replace *AmazonEKS_CNI_IPv6_Policy* with the name you chose, if you chose a different name than the one used in previous steps.

```
aws iam detach-role-policy --role-name $cluster_role_name --policy-arn
 arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
aws iam detach-role-policy --role-name $node_role_name --policy-arn
 arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
aws iam detach-role-policy --role-name $node_role_name --policy-arn
 arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
aws iam detach-role-policy --role-name $node_role_name --policy-arn arn:aws:iam::
$account_id:policy/AmazonEKS_CNI_IPv6_Policy
aws iam delete-policy --policy-arn arn:aws:iam::$account_id:policy/
AmazonEKS_CNI_IPv6_Policy
aws iam delete-role --role-name $cluster_role_name
```

```
aws iam delete-role --role-name $node_role_name
```

v. Delete the AWS CloudFormation stack that created the VPC.

```
aws cloudformation delete-stack --region $region_code --stack-name $vpc_stack_name
```

Enable outbound internet access for Pods

Applies to: Linux `IPv4` Fargate nodes, Linux nodes with Amazon EC2 instances

If you deployed your cluster using the `IPv6` family, then the information in this topic isn't applicable to your cluster, because `IPv6` addresses are not network translated. For more information about using `IPv6` with your cluster, see the section called "IPv6".

By default, each Pod in your cluster is assigned a private `IPv4` address from a classless inter-domain routing (CIDR) block that is associated with the VPC that the Pod is deployed in. Pods in the same VPC communicate with each other using these private IP addresses as end points. When a Pod communicates to any `IPv4` address that isn't within a CIDR block that's associated to your VPC, the Amazon VPC CNI plugin (for both Linux or Windows) translates the Pod's `IPv4` address to the primary private `IPv4` address of the primary elastic network interface of the node that the Pod is running on, by default *.

> **Note**
>
> For Windows nodes, there are additional details to consider. By default, the VPC CNI plugin for Windows is defined with a networking configuration in which the traffic to a destination within the same VPC is excluded for SNAT. This means that internal VPC communication has SNAT disabled and the IP address allocated to a Pod is routable inside the VPC. But traffic to a destination outside of the VPC has the source Pod IP SNAT'ed to the instance ENI's primary IP address. This default configuration for Windows ensures that the pod can access networks outside of your VPC in the same way as the host instance.

Due to this behavior:

- Your Pods can communicate with internet resources only if the node that they're running on has a public or elastic IP address assigned to it and is in a public subnet. A public subnet's associated route table has a route to an internet gateway. We recommend deploying nodes to private subnets, whenever possible.

- For versions of the plugin earlier than `1.8.0`, resources that are in networks or VPCs that are connected to your cluster VPC using VPC peering, a transit VPC, or AWS Direct Connect can't initiate communication to your Pods behind secondary elastic network interfaces. Your Pods can initiate communication to those resources and receive responses from them, though.

If either of the following statements are true in your environment, then change the default configuration with the command that follows.

- You have resources in networks or VPCs that are connected to your cluster VPC using VPC peering, a transit VPC, or AWS Direct Connect that need to initiate communication with your Pods using an `IPv4` address and your plugin version is earlier than `1.8.0`.
- Your Pods are in a private subnet and need to communicate outbound to the internet. The subnet has a route to a NAT gateway.

```
kubectl set env daemonset -n kube-system aws-node AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

Note

The `AWS_VPC_K8S_CNI_EXTERNALSNAT` and `AWS_VPC_K8S_CNI_EXCLUDE_SNAT_CIDRS` CNI configuration variables aren't applicable to Windows nodes. Disabling SNAT isn't supported for Windows. As for excluding a list of `IPv4` CIDRs from SNAT, you can define this by specifying the `ExcludedSnatCIDRs` parameter in the Windows bootstrap script. For more information on using this parameter, see the section called "Bootstrap script configuration parameters".

Host networking

* If a Pod's spec contains `hostNetwork=true` (default is `false`), then its IP address isn't translated to a different address. This is the case for the `kube-proxy` and Amazon VPC CNI plugin for Kubernetes Pods that run on your cluster, by default. For these Pods, the IP address is the same as the node's primary IP address, so the Pod's IP address isn't translated. For more information about a Pod's `hostNetwork` setting, see PodSpec v1 core in the Kubernetes API reference.

Limit Pod traffic with Kubernetes network policies

By default, there are no restrictions in Kubernetes for IP addresses, ports, or connections between any Pods in your cluster or between your Pods and resources in any other network. You can use Kubernetes *network policy* to restrict network traffic to and from your Pods. For more information, see [Network Policies](#) in the Kubernetes documentation.

If you have version `1.13` or earlier of the Amazon VPC CNI plugin for Kubernetes on your cluster, you need to implement a third party solution to apply Kubernetes network policies to your cluster. Version `1.14` or later of the plugin can implement network policies, so you don't need to use a third party solution. In this topic, you learn how to configure your cluster to use Kubernetes network policy on your cluster without using a third party add-on.

Network policies in the Amazon VPC CNI plugin for Kubernetes are supported in the following configurations.

- Amazon EKS clusters of version `1.25` and later.
- Version 1.14 or later of the Amazon VPC CNI plugin for Kubernetes on your cluster.
- Cluster configured for `IPv4` or `IPv6` addresses.
- You can use network policies with [security groups for Pods](#). With network policies, you can control all in-cluster communication. With security groups for Pods, you can control access to AWS services from applications within a Pod.
- You can use network policies with *custom networking* and *prefix delegation*.

Considerations

Architecture

- When applying Amazon VPC CNI plugin for Kubernetes network policies to your cluster with the Amazon VPC CNI plugin for Kubernetes , you can apply the policies to Amazon EC2 Linux nodes only. You can't apply the policies to Fargate or Windows nodes.
- Network policies only apply either `IPv4` or `IPv6` addresses, but not both. In an `IPv4` cluster, the VPC CNI assigns `IPv4` address to pods and applies `IPv4` policies. In an `IPv6` cluster, the VPC CNI assigns `IPv6` address to pods and applies `IPv6` policies. Any `IPv4` network policy rules applied to an `IPv6` cluster are ignored. Any `IPv6` network policy rules applied to an `IPv4` cluster are ignored.

Network Policies

- Network Policies are only applied to Pods that are part of a Deployment. Standalone Pods that don't have a `metadata.ownerReferences` set can't have network policies applied to them.
- You can apply multiple network policies to the same Pod. When two or more policies that select the same Pod are configured, all policies are applied to the Pod.
- The maximum number of unique combinations of ports for each protocol in each `ingress:` or `egress:` selector in a network policy is 24.
- For any of your Kubernetes services, the service port must be the same as the container port. If you're using named ports, use the same name in the service spec too.

Migration

- If your cluster is currently using a third party solution to manage Kubernetes network policies, you can use those same policies with the Amazon VPC CNI plugin for Kubernetes. However you must remove your existing solution so that it isn't managing the same policies.

Installation

- The network policy feature creates and requires a `PolicyEndpoint` Custom Resource Definition (CRD) called `policyendpoints.networking.k8s.aws`. `PolicyEndpoint` objects of the Custom Resource are managed by Amazon EKS. You shouldn't modify or delete these resources.
- If you run pods that use the instance role IAM credentials or connect to the EC2 IMDS, be careful to check for network policies that would block access to the EC2 IMDS. You may need to add a network policy to allow access to EC2 IMDS. For more information, see [Instance metadata and user data] in the Amazon EC2 User Guide.

  Pods that use *IAM roles for service accounts* or *EKS Pod Identity* don't access EC2 IMDS.
- The Amazon VPC CNI plugin for Kubernetes doesn't apply network policies to additional network interfaces for each pod, only the primary interface for each pod (`eth0`). This affects the following architectures:
  - `IPv6` pods with the `ENABLE_V4_EGRESS` variable set to `true`. This variable enables the `IPv4` egress feature to connect the IPv6 pods to `IPv4` endpoints such as those outside the cluster. The `IPv4` egress feature works by creating an additional network interface with a local loopback IPv4 address.

- When using chained network plugins such as Multus. Because these plugins add network interfaces to each pod, network policies aren't applied to the chained network plugins.

Restrict Pod network traffic with Kubernetes network policies

You can use a Kubernetes network policy to restrict network traffic to and from your Pods. For more information, see Network Policies in the Kubernetes documentation.

You must configure the following in order to use this feature:

1. Set up policy enforcement at Pod startup. You do this in the `aws-node` container of the VPC CNI `DaemonSet`.
2. Enable the network policy parameter for the add-on.
3. Configure your cluster to use the Kubernetes network policy

Before you begin, review the considerations. For more information, see the section called "Considerations".

Prerequisites

The following are prerequisites for the feature:

Minimum cluster version

An existing Amazon EKS cluster. To deploy one, see *Get started*. The cluster must be Kubernetes version `1.25` or later. The cluster must be running one of the Kubernetes versions and platform versions listed in the following table. Note that any Kubernetes and platform versions later than those listed are also supported. You can check your current Kubernetes version by replacing *`my-cluster`* in the following command with the name of your cluster and then running the modified command:

```
aws eks describe-cluster
              --name my-cluster --query cluster.version --output
              text
```

Kubernetes version	Platform version
1.27.4	eks.5

Kubernetes version	Platform version
1.26.7	eks.6
1.25.12	eks.7

Minimum VPC CNI version

Version `1.14` or later of the Amazon VPC CNI plugin for Kubernetes on your cluster. You can see which version that you currently have with the following command.

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

If your version is earlier than `1.14`, see the section called "Update (EKS add-on)" to upgrade to version `1.14` or later.

Minimum Linux kernel version

Your nodes must have Linux kernel version `5.10` or later. You can check your kernel version with `uname -r`. If you're using the latest versions of the Amazon EKS optimized Amazon Linux, Amazon EKS optimized accelerated Amazon Linux AMIs, and Bottlerocket AMIs, they already have the required kernel version.

The Amazon EKS optimized accelerated Amazon Linux AMI version `v20231116` or later have kernel version `5.10`.

Step 1: Set up policy enforcement at Pod startup

The Amazon VPC CNI plugin for Kubernetes configures network policies for pods in parallel with the pod provisioning. Until all of the policies are configured for the new pod, containers in the new pod will start with a *default allow policy*. This is called *standard mode*. A default allow policy means that all ingress and egress traffic is allowed to and from the new pods. For example, the pods will not have any firewall rules enforced (all traffic is allowed) until the new pod is updated with the active policies.

With the `NETWORK_POLICY_ENFORCING_MODE` variable set to `strict`, pods that use the VPC CNI start with a *default deny policy*, then policies are configured. This is called *strict mode*. In strict mode, you must have a network policy for every endpoint that your pods need to access in your cluster. Note that this requirement applies to the CoreDNS pods. The default deny policy isn't configured for pods with Host networking.

You can change the default network policy by setting the environment variable `NETWORK_POLICY_ENFORCING_MODE` to `strict` in the `aws-node` container of the VPC CNI `DaemonSet`.

```
env:
  - name: NETWORK_POLICY_ENFORCING_MODE
    value: "strict"
```

Step 2: Enable the network policy parameter for the add-on

The network policy feature uses port 8162 on the node for metrics by default. Also, the feature used port 8163 for health probes. If you run another application on the nodes or inside pods that needs to use these ports, the app fails to run. In VPC CNI version `v1.14.1` or later, you can change these ports.

Use the following procedure to enable the network policy parameter for the add-on.

AWS Management Console

1. Open the [Amazon EKS console](#).
2. In the left navigation pane, select **Clusters**, and then select the name of the cluster that you want to configure the Amazon VPC CNI add-on for.
3. Choose the **Add-ons** tab.
4. Select the box in the top right of the add-on box and then choose **Edit**.
5. On the **Configure** ***`name of add-on`*** page:
   a. Select a `v1.14.0-eksbuild.3` or later version in the **Version** list.
   b. Expand the **Optional configuration settings**.
   c. Enter the JSON key `"enableNetworkPolicy":` and value `"true"` in **Configuration values**. The resulting text must be a valid JSON object. If this key and value are the only data in the text box, surround the key and value with curly braces `{ }`.

      The following example has network policy feature enabled and metrics and health probes are set to the default port numbers:

```
{
    "enableNetworkPolicy": "true",
    "nodeAgent": {
        "healthProbeBindAddr": "8163",
        "metricsBindAddr": "8162"
    }
}
```

Helm

If you have installed the Amazon VPC CNI plugin for Kubernetes through `helm`, you can update the configuration to change the ports.

1. Run the following command to change the ports. Set the port number in the value for either key `nodeAgent.metricsBindAddr` or key `nodeAgent.healthProbeBindAddr`, respectively.

```
helm upgrade --set nodeAgent.metricsBindAddr=8162 --set
 nodeAgent.healthProbeBindAddr=8163 aws-vpc-cni --namespace kube-system eks/aws-vpc-
cni
```

kubectl

1. Open the `aws-node` `DaemonSet` in your editor.

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

2. Replace the port numbers in the following command arguments in the `args:` in the `aws-network-policy-agent` container in the VPC CNI `aws-node` daemonset manifest.

```
    - args:
            - --metrics-bind-addr=:8162
            - --health-probe-bind-addr=:8163
```

Step 3: Mount the Berkeley Packet Filter (BPF) file system on your nodes

You must mount the Berkeley Packet Filter (BPF) file system on each of your nodes.

> **Note**
>
> If your cluster is version `1.27` or later, you can skip this step as all Amazon EKS optimized Amazon Linux and Bottlerocket AMIs for `1.27` or later have this feature already.
> For all other cluster versions, if you upgrade the Amazon EKS optimized Amazon Linux to version `v20230703` or later or you upgrade the Bottlerocket AMI to version `v1.0.2` or later, you can skip this step.

1. Mount the Berkeley Packet Filter (BPF) file system on each of your nodes.

```
sudo mount -t bpf bpffs /sys/fs/bpf
```

2. Then, add the same command to your user data in your launch template for your Amazon EC2 Auto Scaling Groups.

Step 4: Configure your cluster to use Kubernetes network policies

Configure the cluster to use Kubernetes network policies. You can set this for an Amazon EKS add-on or self-managed add-on.

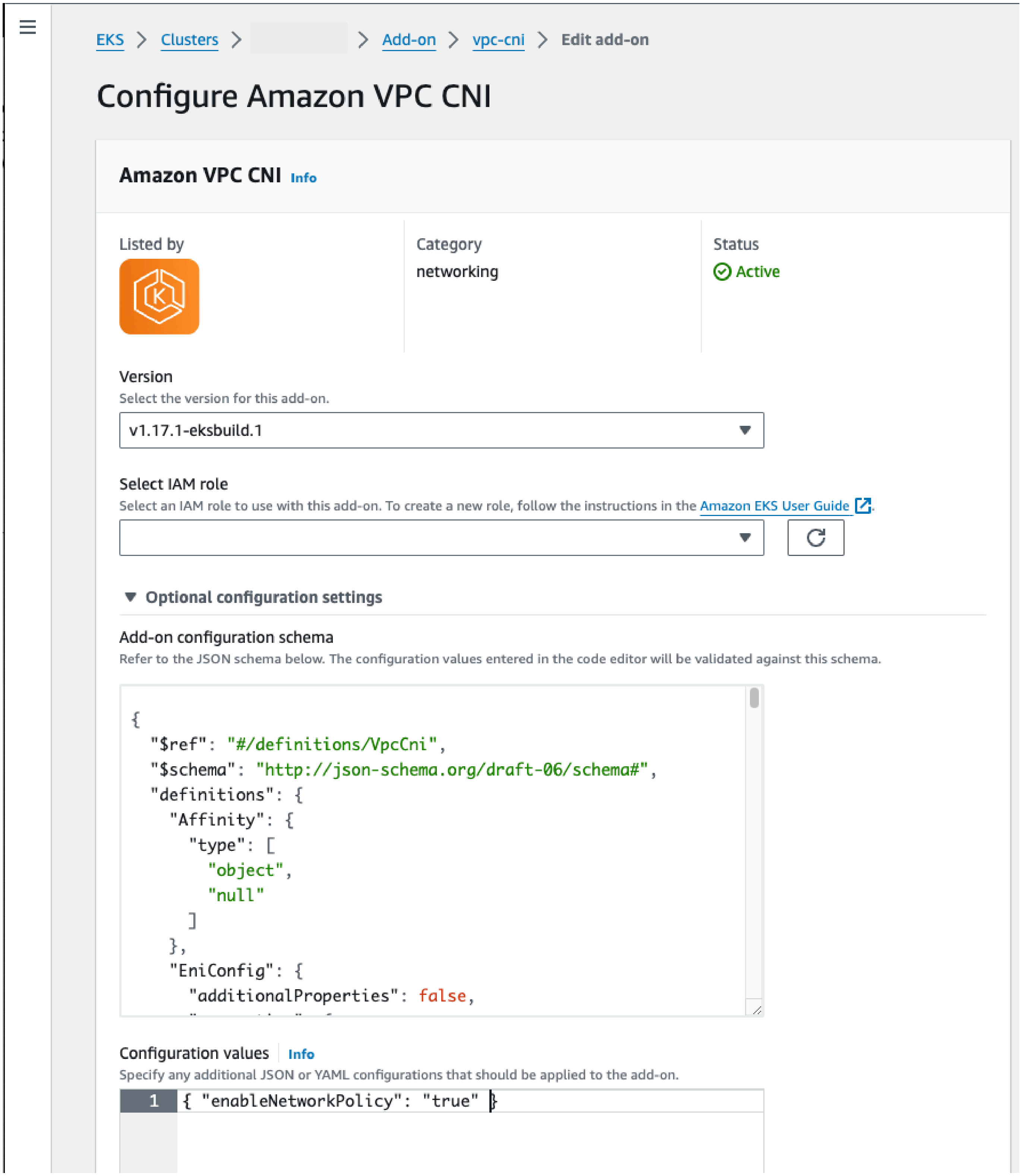
Amazon EKS add-on

AWS Management Console

1. Open the Amazon EKS console.
2. In the left navigation pane, select **Clusters**, and then select the name of the cluster that you want to configure the Amazon VPC CNI add-on for.
3. Choose the **Add-ons** tab.
4. Select the box in the top right of the add-on box and then choose **Edit**.
5. On the **Configure** ***`name of addon`*** page:
   a. Select a `v1.14.0-eksbuild.3` or later version in the **Version** list.
   b. Expand the **Optional configuration settings**.
   c. Enter the JSON key `"enableNetworkPolicy":` and value `"true"` in **Configuration values**. The resulting text must be a valid JSON object. If this key and value are the only data in the text box, surround the key and value with curly braces `{ }`. The following example shows network policy is enabled:

```
{ "enableNetworkPolicy": "true" }
```

The following screenshot shows an example of this scenario.


EKS > Clusters > > Add-on > vpc-cni > Edit add-on
Configure Amazon VPC CNI
Amazon VPC CNI Info
Listed by
Category
networking
Status
Active
Version
Select the version for this add-on.
v1.17.1-eksbuild.1
Select IAM role
Select an IAM role to use with this add-on. To create a new role, follow the instructions in the Amazon EKS User Guide.
Optional configuration settings
Add-on configuration schema
Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.
{
"$ref": "#/definitions/VpcCni",
"$schema": "http://json-schema.org/draft-06/schema#",
"definitions": {
"Affinity": {
"type": [
"object",
"null"
]
},
"EniConfig": {
"additionalProperties": false,
Configuration values Info
Specify any additional JSON or YAML configurations that should be applied to the add-on.
1 { "enableNetworkPolicy": "true" }

AWS CLI

1. Run the following AWS CLI command. Replace `my-cluster` with the name of your cluster and the IAM role ARN with the role that you are using.

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 v1.14.0-eksbuild.3 \
    --service-account-role-arn arn:aws:iam::123456789012:role/AmazonEKSVPCCNIRole \
    --resolve-conflicts PRESERVE --configuration-values '{"enableNetworkPolicy":
 "true"}'
```

Self-managed add-on

Helm

If you have installed the Amazon VPC CNI plugin for Kubernetes through `helm`, you can update the configuration to enable network policy.

1. Run the following command to enable network policy.

```
helm upgrade --set enableNetworkPolicy=true aws-vpc-cni --namespace kube-system eks/
aws-vpc-cni
```

kubectl

1. Open the `amazon-vpc-cni` `ConfigMap` in your editor.

```
kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
```

2. Add the following line to the `data` in the `ConfigMap`.

```
enable-network-policy-controller: "true"
```

Once you've added the line, your `ConfigMap` should look like the following example.

```
apiVersion: v1
 kind: ConfigMap
 metadata:
```

```
  name: amazon-vpc-cni
  namespace: kube-system
data:
  enable-network-policy-controller: "true"
```

3. Open the `aws-node` `DaemonSet` in your editor.

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

   a. Replace the `false` with `true` in the command argument `--enable-network-policy=false` in the `args:` in the `aws-network-policy-agent` container in the VPC CNI `aws-node` daemonset manifest.

```
      - args:
          - --enable-network-policy=true
```

Step 5. Next steps

After you complete the configuration, confirm that the `aws-node` pods are running on your cluster.

```
kubectl get pods -n kube-system | grep 'aws-node\|amazon'
```

An example output is as follows.

```
aws-node-gmqp7                                                2/2     Running   1 (24h ago)
 24h
aws-node-prnsh                                                2/2     Running   1 (24h ago)
 24h
```

There are 2 containers in the `aws-node` pods in versions `1.14` and later. In previous versions and if network policy is disabled, there is only a single container in the `aws-node` pods.

You can now deploy Kubernetes network policies to your cluster.

To implement Kubernetes network policies you create Kubernetes `NetworkPolicy` objects and deploy them to your cluster. `NetworkPolicy` objects are scoped to a namespace. You implement policies to allow or deny traffic between Pods based on label selectors, namespaces, and IP address ranges. For more information about creating `NetworkPolicy` objects, see Network Policies in the Kubernetes documentation.

Enforcement of Kubernetes `NetworkPolicy` objects is implemented using the Extended Berkeley Packet Filter (eBPF). Relative to `iptables` based implementations, it offers lower latency and performance characteristics, including reduced CPU utilization and avoiding sequential lookups. Additionally, eBPF probes provide access to context rich data that helps debug complex kernel level issues and improve observability. Amazon EKS supports an eBPF-based exporter that leverages the probes to log policy results on each node and export the data to external log collectors to aid in debugging. For more information, see the eBPF documentation.

Disable Kubernetes network policies for Amazon EKS Pod network traffic

Disable Kubernetes network policies to stop restricting Amazon EKS Pod network traffic

1. List all Kubernetes network policies.

```
kubectl get netpol -A
```

2. Delete each Kubernetes network policy. You must delete all network policies before disabling network policies.

```
kubectl delete netpol <policy-name>
```

3. Open the aws-node DaemonSet in your editor.

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

4. Replace the `true` with `false` in the command argument `--enable-network-policy=true` in the `args:` in the `aws-network-policy-agent` container in the VPC CNI `aws-node` daemonset manifest.

```
     - args:
        - --enable-network-policy=true
```

Troubleshooting Kubernetes network policies For Amazon EKS

You can troubleshoot and investigate network connections that use network policies by reading the Network policy logs and by running tools from the eBPF SDK.

Network policy logs

Whether connections are allowed or denied by a network policies is logged in *flow logs*. The network policy logs on each node include the flow logs for every pod that has a network policy. Network policy logs are stored at `/var/log/aws-routed-eni/network-policy-agent.log`. The following example is from a `network-policy-agent.log` file:

```
{"level":"info","timestamp":"2023-05-30T16:05:32.573Z","logger":"ebpf-
client","msg":"Flow Info: ","Src
IP":"192.168.87.155","Src Port":38971,"Dest IP":"64.6.160","Dest
Port":53,"Proto":"UDP","Verdict":"ACCEPT"}
```

Network policy logs are disabled by default. To enable the network policy logs, follow these steps:

> **Note**
>
> Network policy logs require an additional 1 vCPU for the `aws-network-policy-agent` container in the VPC CNI `aws-node` daemonset manifest.

Amazon EKS add-on

AWS Management Console

a. Open the [Amazon EKS console](https://console.aws.amazon.com/eks/home#/clusters).

b. In the left navigation pane, select **Clusters**, and then select the name of the cluster that you want to configure the Amazon VPC CNI add-on for.

c. Choose the **Add-ons** tab.

d. Select the box in the top right of the add-on box and then choose **Edit**.

e. On the **Configure** ***`name of addon`*** page:

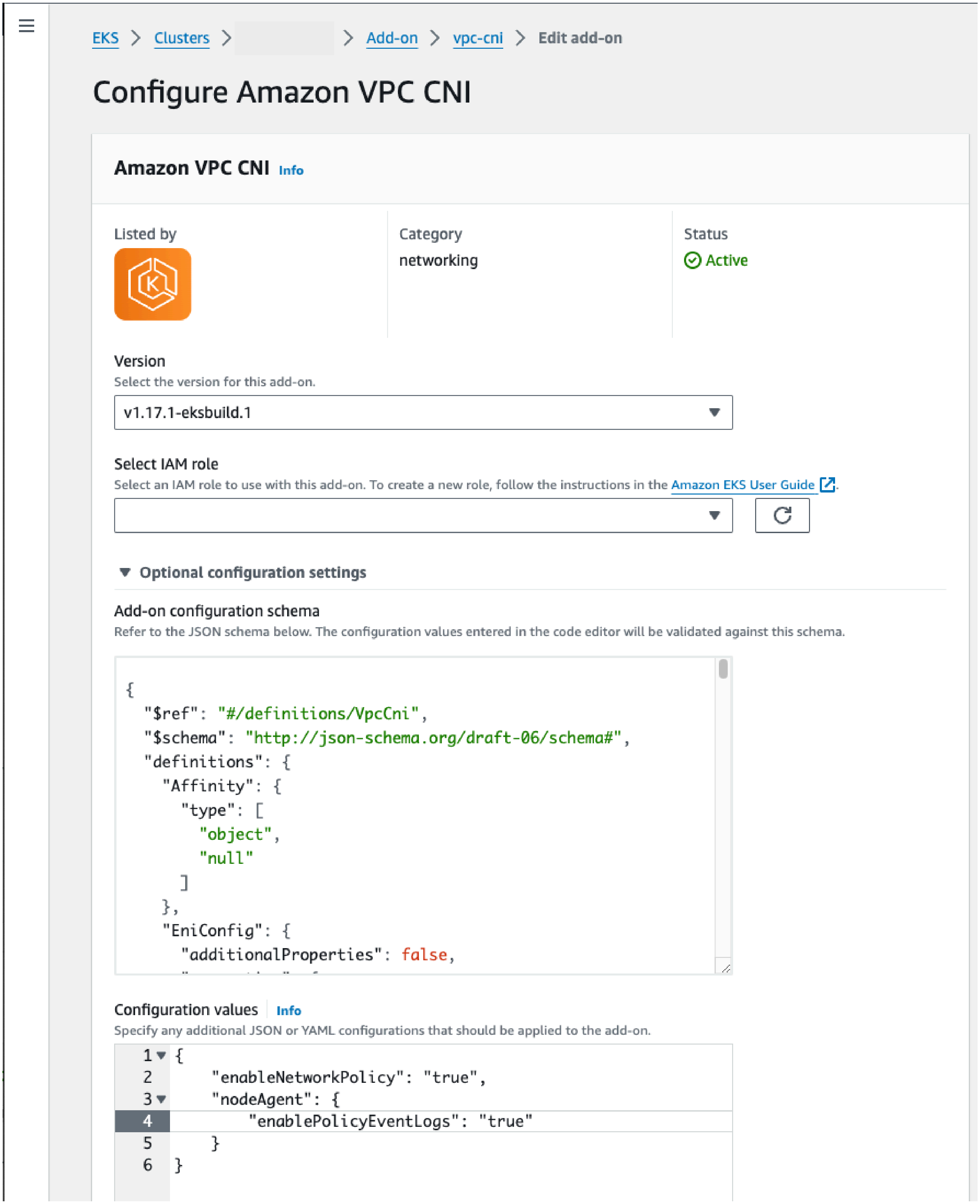
   i. Select a `v1.14.0-eksbuild.3` or later version in the **Version** dropdown list.

   ii. Expand the **Optional configuration settings**.

   iii. Enter the top-level JSON key `"nodeAgent":` and value is an object with a key `"enablePolicyEventLogs":` and value of `"true"` in **Configuration values**. The resulting text must be a valid JSON object. The following example shows network policy and the network policy logs are enabled, and the network policy logs are sent to CloudWatch Logs:

```
{
    "enableNetworkPolicy": "true",
    "nodeAgent": {
        "enablePolicyEventLogs": "true"
    }
}
```

The following screenshot shows an example of this scenario.


EKS > Clusters > > Add-on > vpc-cni > Edit add-on
Configure Amazon VPC CNI
Amazon VPC CNI Info
Listed by
Category
networking
Status
Active
Version
Select the version for this add-on.
v1.17.1-eksbuild.1
Select IAM role
Select an IAM role to use with this add-on. To create a new role, follow the instructions in the Amazon EKS User Guide.
Optional configuration settings
Add-on configuration schema
Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.
{
"$ref": "#/definitions/VpcCni",
"$schema": "http://json-schema.org/draft-06/schema#",
"definitions": {
"Affinity": {
"type": [
"object",
"null"
]
},
"EniConfig": {
"additionalProperties": false,
Configuration values Info
Specify any additional JSON or YAML configurations that should be applied to the add-on.
{
"enableNetworkPolicy": "true",
"nodeAgent": {
"enablePolicyEventLogs": "true"
}
}

AWS CLI

a. Run the following AWS CLI command. Replace `my-cluster` with the name of your cluster and replace the IAM role ARN with the role that you are using.

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version v1.14.0-eksbuild.3 \
    --service-account-role-arn arn:aws:iam::123456789012:role/AmazonEKSVPCCNIRole \
    --resolve-conflicts PRESERVE --configuration-values '{"nodeAgent": {"enablePolicyEventLogs": "true"}}'
```

Self-managed add-on

Helm

If you have installed the Amazon VPC CNI plugin for Kubernetes through `helm`, you can update the configuration to write the network policy logs.

a. Run the following command to enable network policy.

```
helm upgrade --set nodeAgent.enablePolicyEventLogs=true aws-vpc-cni --namespace kube-system eks/aws-vpc-cni
```

kubectl

If you have installed the Amazon VPC CNI plugin for Kubernetes through `kubectl`, you can update the configuration to write the network policy logs.

a. Open the `aws-node` `DaemonSet` in your editor.

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

b. Replace the `false` with `true` in the command argument `--enable-policy-event-logs=false` in the `args:` in the `aws-network-policy-agent` container in the VPC CNI `aws-node` daemonset manifest.

```
     - args:
        - --enable-policy-event-logs=true
```

Send network policy logs to Amazon CloudWatch Logs

You can monitor the network policy logs using services such as Amazon CloudWatch Logs. You can use the following methods to send the network policy logs to CloudWatch Logs.

For EKS clusters, the policy logs will be located under `/aws/eks/cluster-name/cluster/` and for self-managed K8S clusters, the logs will be placed under `/aws/k8s-cluster/cluster/`.

Send network policy logs with Amazon VPC CNI plugin for Kubernetes

If you enable network policy, a second container is add to the `aws-node` pods for a *node agent*. This node agent can send the network policy logs to CloudWatch Logs.

> **Note**
>
> Only the network policy logs are sent by the node agent. Other logs made by the VPC CNI aren't included.

Prerequisites

- Add the following permissions as a stanza or separate policy to the IAM role that you are using for the VPC CNI.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon EKS add-on

AWS Management Console

a. Open the [Amazon EKS console](https://console.aws.amazon.com/eks/home#/clusters).

b. In the left navigation pane, select **Clusters**, and then select the name of the cluster that you want to configure the Amazon VPC CNI add-on for.

c. Choose the **Add-ons** tab.

d. Select the box in the top right of the add-on box and then choose **Edit**.

e. On the **Configure** ***`name of addon`*** page:

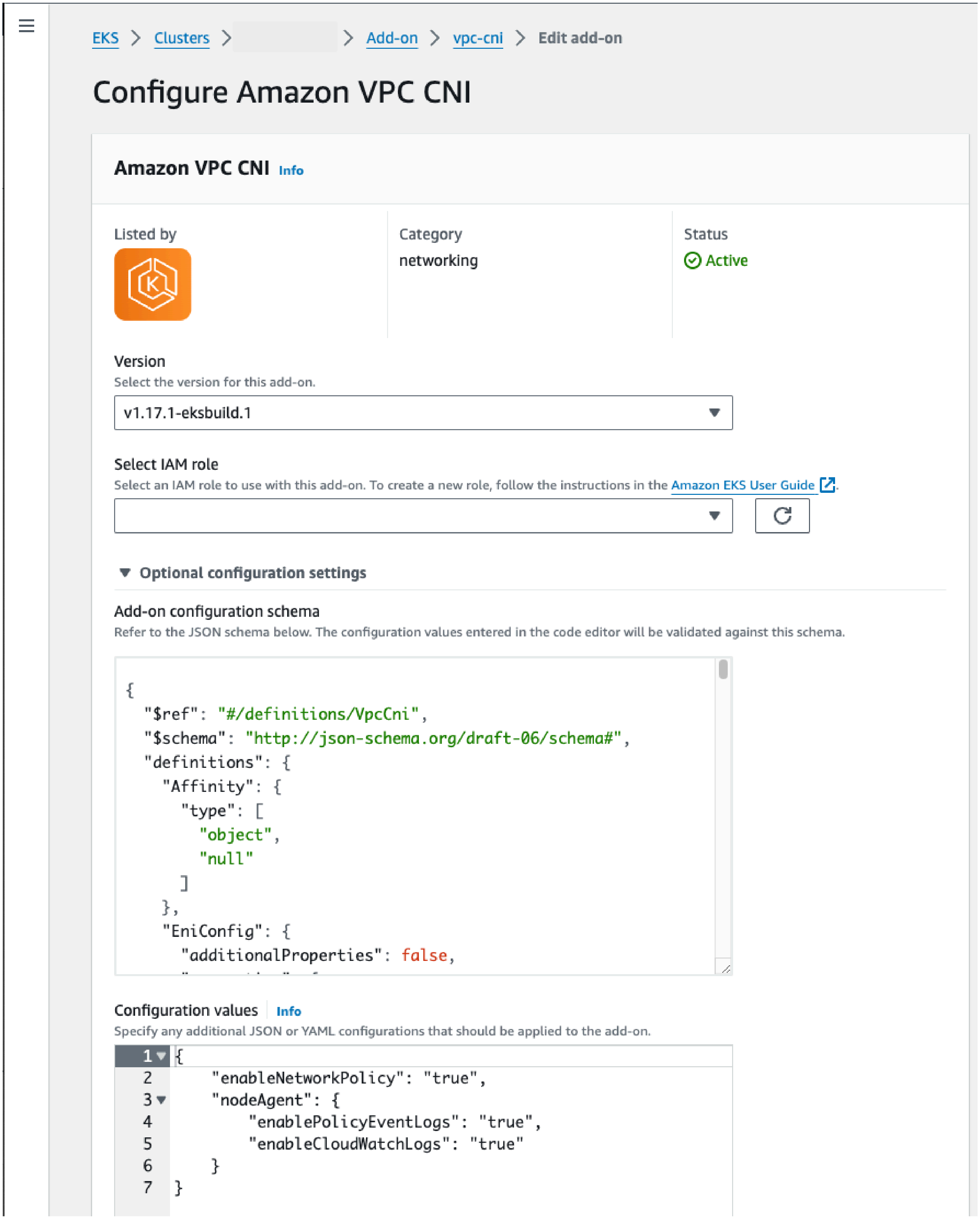
   i. Select a `v1.14.0-eksbuild.3` or later version in the **Version** dropdown list.

   ii. Expand the **Optional configuration settings**.

   iii. Enter the top-level JSON key `"nodeAgent":` and value is an object with a key `"enableCloudWatchLogs":` and value of `"true"` in **Configuration values**. The resulting text must be a valid JSON object. The following example shows network policy and the network policy logs are enabled, and the logs are sent to CloudWatch Logs:

```
{
    "enableNetworkPolicy": "true",
    "nodeAgent": {
        "enablePolicyEventLogs": "true",
        "enableCloudWatchLogs": "true",
    }
}
```

The following screenshot shows an example of this scenario.


EKS > Clusters > > Add-on > vpc-cni > Edit add-on
Configure Amazon VPC CNI
Amazon VPC CNI Info
Listed by
Category
networking
Status
Active
Version
Select the version for this add-on.
v1.17.1-eksbuild.1
Select IAM role
Select an IAM role to use with this add-on. To create a new role, follow the instructions in the Amazon EKS User Guide.
Optional configuration settings
Add-on configuration schema
Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.
{
"$ref": "#/definitions/VpcCni",
"$schema": "http://json-schema.org/draft-06/schema#",
"definitions": {
"Affinity": {
"type": [
"object",
"null"
]
},
"EniConfig": {
"additionalProperties": false,
Configuration values Info
Specify any additional JSON or YAML configurations that should be applied to the add-on.
{
"enableNetworkPolicy": "true",
"nodeAgent": {
"enablePolicyEventLogs": "true",
"enableCloudWatchLogs": "true"
}
}

AWS CLI

a. Run the following AWS CLI command. Replace `my-cluster` with the name of your cluster and replace the IAM role ARN with the role that you are using.

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-
version v1.14.0-eksbuild.3 \
    --service-account-role-arn arn:aws:iam::123456789012:role/AmazonEKSVPCCNIRole
 \
    --resolve-conflicts PRESERVE --configuration-values '{"nodeAgent":
 {"enablePolicyEventLogs": "true", "enableCloudWatchLogs": "true"}}'
```

Self-managed add-on

Helm

If you have installed the Amazon VPC CNI plugin for Kubernetes through `helm`, you can update the configuration to send network policy logs to CloudWatch Logs.

a. Run the following command to enable network policy logs and send them to CloudWatch Logs.

```
helm upgrade --set nodeAgent.enablePolicyEventLogs=true --set
 nodeAgent.enableCloudWatchLogs=true aws-vpc-cni --namespace kube-system eks/aws-
vpc-cni
```

kubectl

a. Open the `aws-node` `DaemonSet` in your editor.

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

b. Replace the `false` with `true` in two command arguments `--enable-policy-event-logs=false` and `--enable-cloudwatch-logs=false` in the `args:` in the `aws-network-policy-agent` container in the VPC CNI `aws-node` daemonset manifest.

```
     - args:
        - --enable-policy-event-logs=true
        - --enable-cloudwatch-logs=true
```

Send network policy logs with a Fluent Bit daemonset

If you are using Fluent Bit in a daemonset to send logs from your nodes, you can add configuration to include the network policy logs from network policies. You can use the following example configuration:

```
    [INPUT]
        Name              tail
        Tag               eksnp.*
        Path              /var/log/aws-routed-eni/network-policy-agent*.log
        Parser            json
        DB                /var/log/aws-routed-eni/flb_npagent.db
        Mem_Buf_Limit     5MB
        Skip_Long_Lines   On
        Refresh_Interval  10
```

Included eBPF SDK

The Amazon VPC CNI plugin for Kubernetes installs eBPF SDK collection of tools on the nodes. You can use the eBPF SDK tools to identify issues with network policies. For example, the following command lists the programs that are running on the node.

```
sudo /opt/cni/bin/aws-eks-na-cli ebpf progs
```

To run this command, you can use any method to connect to the node.

Stars demo of network policy for Amazon EKS

This demo creates a front-end, back-end, and client service on your Amazon EKS cluster. The demo also creates a management graphical user interface that shows the available ingress and egress paths between each service. We recommend that you complete the demo on a cluster that you don't run production workloads on.

Before you create any network policies, all services can communicate bidirectionally. After you apply the network policies, you can see that the client can only communicate with the front-end service, and the back-end only accepts traffic from the front-end.

1. Apply the front-end, back-end, client, and management user interface services:

```
kubectl apply -f https://eksworkshop.com/beginner/120_network-policies/calico/
stars_policy_demo/create_resources.files/namespace.yaml
```

```
kubectl apply -f https://eksworkshop.com/beginner/120_network-policies/calico/
stars_policy_demo/create_resources.files/management-ui.yaml
kubectl apply -f https://eksworkshop.com/beginner/120_network-policies/calico/
stars_policy_demo/create_resources.files/backend.yaml
kubectl apply -f https://eksworkshop.com/beginner/120_network-policies/calico/
stars_policy_demo/create_resources.files/frontend.yaml
kubectl apply -f https://eksworkshop.com/beginner/120_network-policies/calico/
stars_policy_demo/create_resources.files/client.yaml
```

2. View all Pods on the cluster.

```
kubectl get pods -A
```

An example output is as follows.

In your output, you should see pods in the namespaces shown in the following output. The *NAMES* of your pods and the number of pods in the READY column are different than those in the following output. Don't continue until you see pods with similar names and they all have Running in the STATUS column.

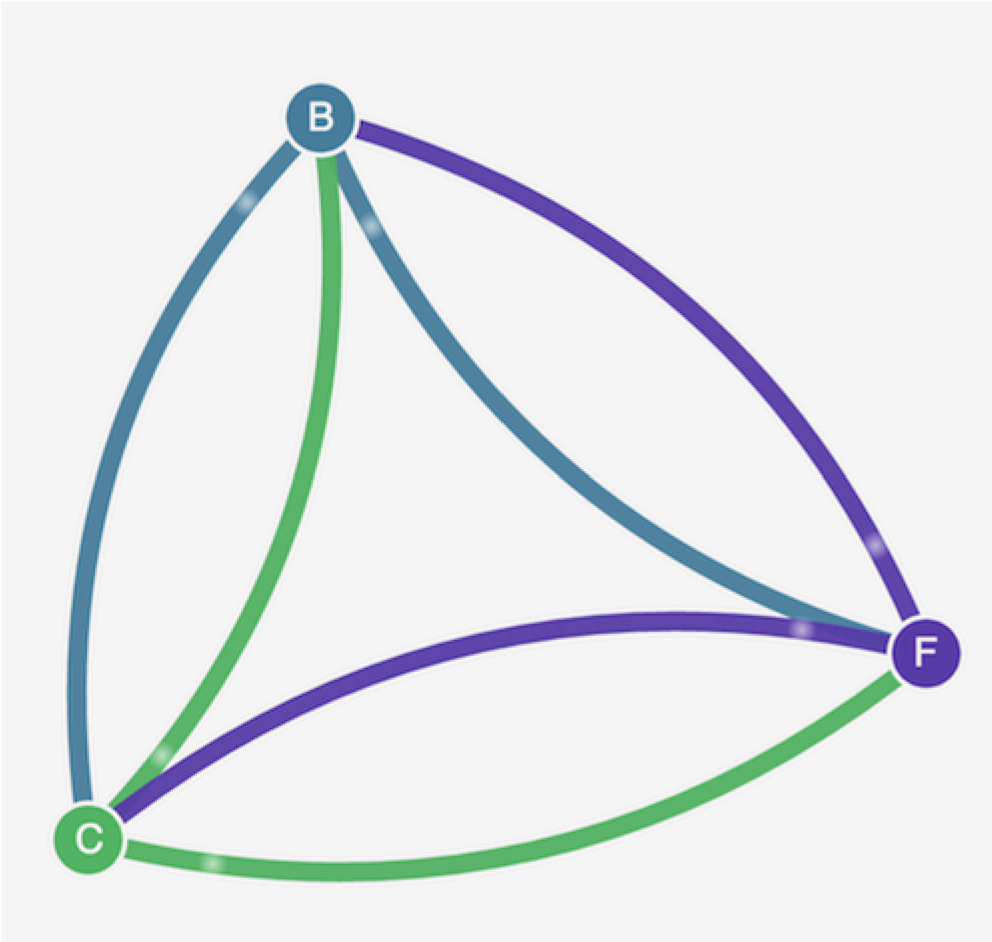
```
NAMESPACE         NAME                                       READY   STATUS
 RESTARTS   AGE
[...]
client            client-xlffc                               1/1     Running   0
      5m19s
[...]
management-ui     management-ui-qrb2g                        1/1     Running   0
      5m24s
stars             backend-sz87q                              1/1     Running   0
      5m23s
stars             frontend-cscnf                             1/1     Running   0
      5m21s
[...]
```

3. To connect to the management user interface, connect to the EXTERNAL-IP of the service running on your cluster:

```
kubectl get service/management-ui -n management-ui
```

4. Open the a browser to the location from the previous step. You should see the management user interface. The **C** node is the client service, the **F** node is the front-end service, and the **B**

node is the back-end service. Each node has full communication access to all other nodes, as indicated by the bold, colored lines.



5. Apply the following network policy in both the `stars` and `client` namespaces to isolate the services from each other:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny
spec:
  podSelector:
    matchLabels: {}
```

You can use the following commands to apply the policy to both namespaces:

```
kubectl apply -n stars -f https://eksworkshop.com/beginner/120_network-policies/
calico/stars_policy_demo/apply_network_policies.files/default-deny.yaml
```

```
kubectl apply -n client -f https://eksworkshop.com/beginner/120_network-policies/
calico/stars_policy_demo/apply_network_policies.files/default-deny.yaml
```

6. Refresh your browser. You see that the management user interface can no longer reach any of the nodes, so they don't show up in the user interface.

7. Apply the following different network policies to allow the management user interface to access the services. Apply this policy to allow the UI:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: stars
  name: allow-ui
spec:
  podSelector:
    matchLabels: {}
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              role: management-ui
```

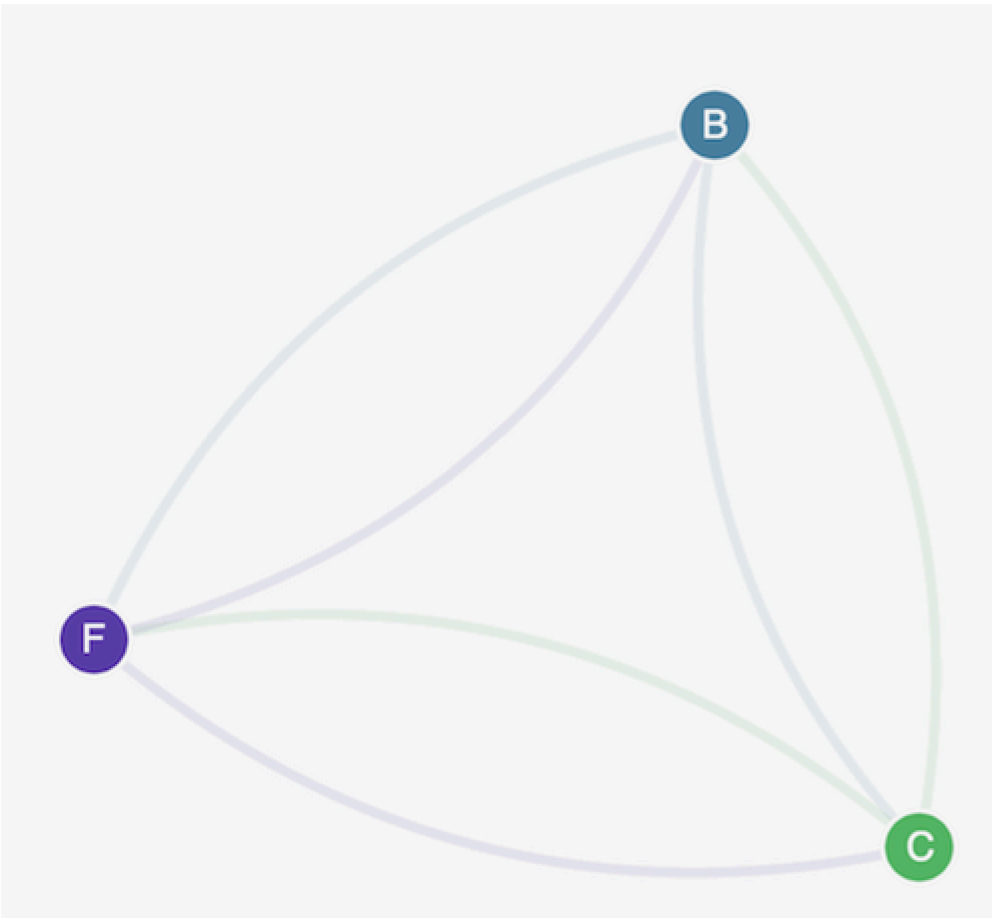
   Apply this policy to allow the client:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: client
  name: allow-ui
spec:
  podSelector:
    matchLabels: {}
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              role: management-ui
```

   You can use the following commands to apply both policies:

```
kubectl apply -f https://eksworkshop.com/beginner/120_network-policies/calico/
stars_policy_demo/apply_network_policies.files/allow-ui.yaml
kubectl apply -f https://eksworkshop.com/beginner/120_network-policies/calico/
stars_policy_demo/apply_network_policies.files/allow-ui-client.yaml
```

8. Refresh your browser. You see that the management user interface can reach the nodes again, but the nodes cannot communicate with each other.

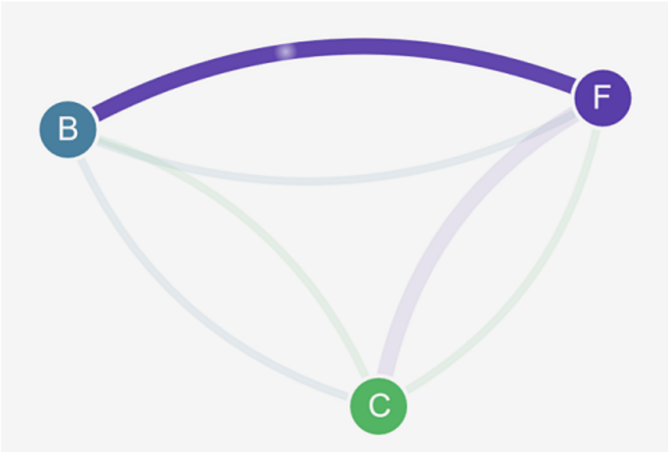


9. Apply the following network policy to allow traffic from the front-end service to the back-end service:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: stars
  name: backend-policy
spec:
  podSelector:
    matchLabels:
```

```
        role: backend
  ingress:
    - from:
        - podSelector:
            matchLabels:
              role: frontend
      ports:
        - protocol: TCP
          port: 6379
```

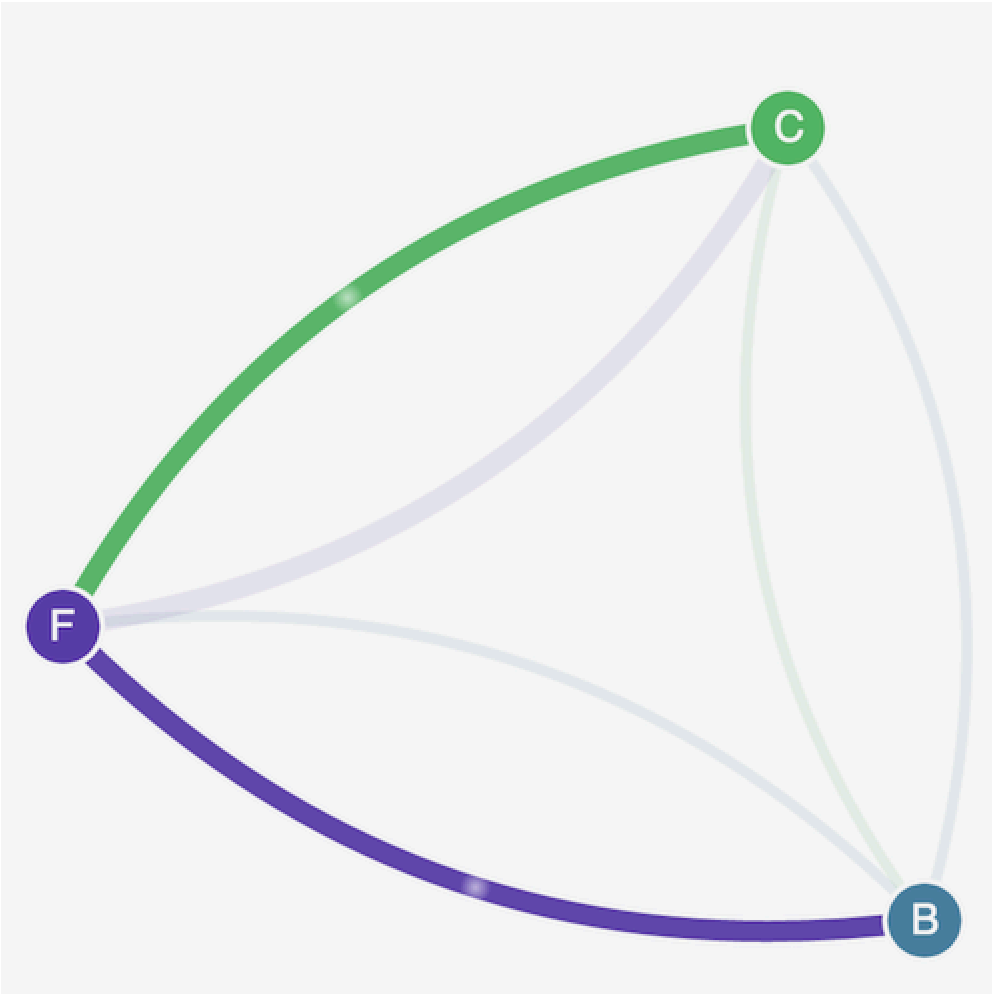
10Refresh your browser. You see that the front-end can communicate with the back-end.



11Apply the following network policy to allow traffic from the client to the front-end service:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: stars
  name: frontend-policy
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              role: client
      ports:
        - protocol: TCP
          port: 80
```

12. Refresh your browser. You see that the client can communicate to the front-end service. The front-end service can still communicate to the back-end service.



13. (Optional) When you are done with the demo, you can delete its resources.

```
kubectl delete -f https://eksworkshop.com/beginner/120_network-policies/calico/
stars_policy_demo/create_resources.files/client.yaml
kubectl delete -f https://eksworkshop.com/beginner/120_network-policies/calico/
stars_policy_demo/create_resources.files/frontend.yaml
kubectl delete -f https://eksworkshop.com/beginner/120_network-policies/calico/
stars_policy_demo/create_resources.files/backend.yaml
kubectl delete -f https://eksworkshop.com/beginner/120_network-policies/calico/
stars_policy_demo/create_resources.files/management-ui.yaml
kubectl delete -f https://eksworkshop.com/beginner/120_network-policies/calico/
stars_policy_demo/create_resources.files/namespace.yaml
```

Even after deleting the resources, there can still be network policy endpoints on the nodes that might interfere in unexpected ways with networking in your cluster. The only sure way to remove these rules is to reboot the nodes or terminate all of the nodes and recycle them. To terminate all nodes, either set the Auto Scaling Group desired count to 0, then back up to the desired number, or just terminate the nodes.

Deploy Pods in alternate subnets with custom networking

Applies to: Linux `IPv4` Fargate nodes, Linux nodes with Amazon EC2 instances

By default, when the Amazon VPC CNI plugin for Kubernetes creates secondary elastic network interfaces (network interfaces) for your Amazon EC2 node, it creates them in the same subnet as the node's primary network interface. It also associates the same security groups to the secondary network interface that are associated to the primary network interface. For one or more of the following reasons, you might want the plugin to create secondary network interfaces in a different subnet or want to associate different security groups to the secondary network interfaces, or both:

- There's a limited number of `IPv4` addresses that are available in the subnet that the primary network interface is in. This might limit the number of Pods that you can create in the subnet. By using a different subnet for secondary network interfaces, you can increase the number of available `IPv4` addresses available for Pods.
- For security reasons, your Pods might need to use a different subnet or security groups than the node's primary network interface.
- The nodes are configured in public subnets, and you want to place the Pods in private subnets. The route table associated to a public subnet includes a route to an internet gateway. The route table associated to a private subnet doesn't include a route to an internet gateway.

Considerations

The following are considerations for using the feature.

- With custom networking enabled, no IP addresses assigned to the primary network interface are assigned to Pods. Only IP addresses from secondary network interfaces are assigned to Pods.
- If your cluster uses the `IPv6` family, you can't use custom networking.

- If you plan to use custom networking only to help alleviate `IPv4` address exhaustion, you can create a cluster using the `IPv6` family instead. For more information, see the section called "IPv6".
- Even though Pods deployed to subnets specified for secondary network interfaces can use different subnet and security groups than the node's primary network interface, the subnets and security groups must be in the same VPC as the node.
- For Fargate, subnets are controlled through the Fargate profile. For more information, see the section called "Define profiles".

Customize the secondary network interface in Amazon EKS nodes

Complete the following before you start the tutorial:

- Review the considerations
- Familiarity with how the Amazon VPC CNI plugin for Kubernetes creates secondary network interfaces and assigns IP addresses to Pods. For more information, see ENI Allocation on GitHub.
- Version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version, use `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*. The AWS CLI version that is installed in AWS CloudShell might also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the *AWS CloudShell User Guide*.
- The `kubectl` command line tool is installed on your device or AWS CloudShell. To install or upgrade `kubectl`, see the section called "Set up kubectl and eksctl".
- We recommend that you complete the steps in this topic in a Bash shell. If you aren't using a Bash shell, some script commands such as line continuation characters and the way variables are set and used require adjustment for your shell. Additionally, the quoting and escaping rules for your shell might be different. For more information, see Using quotation marks with strings in the AWS CLI in the AWS Command Line Interface User Guide.

For this tutorial, we recommend using the example values, except where it's noted to replace them. You can replace any example value when completing the steps for a production cluster. We recommend completing all steps in the same terminal. This is because variables are set and used throughout the steps and won't exist in different terminals.

The commands in this topic are formatted using the conventions listed in Using the AWS CLI examples. If you're running commands from the command line against resources that are in a different AWS Region than the default AWS Region defined in the AWS CLI profile that you're using, then you need to add `--region us-west-2` to the commands, replacing `us-west-2` with your AWS region.

When you want to deploy custom networking to your production cluster, skip to the section called "Step 2: Configure your VPC".

Step 1: Create a test VPC and cluster

The following procedures help you create a test VPC and cluster and configure custom networking for that cluster. We don't recommend using the test cluster for production workloads because several unrelated features that you might use on your production cluster aren't covered in this topic. For more information, see the section called "Create a cluster".

1. Run the following command to define the `account_id` variable.

```
account_id=$(aws sts get-caller-identity --query Account --output text)
```

2. Create a VPC.

   a. If you are deploying to a test system, create a VPC using an Amazon EKS AWS CloudFormation template.

```
aws cloudformation create-stack --stack-name my-eks-custom-networking-vpc \
  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml \
  --parameters ParameterKey=VpcBlock,ParameterValue=192.168.0.0/24 \
  ParameterKey=PrivateSubnet01Block,ParameterValue=192.168.0.64/27 \
  ParameterKey=PrivateSubnet02Block,ParameterValue=192.168.0.96/27 \
  ParameterKey=PublicSubnet01Block,ParameterValue=192.168.0.0/27 \
  ParameterKey=PublicSubnet02Block,ParameterValue=192.168.0.32/27
```

   b. The AWS CloudFormation stack takes a few minutes to create. To check on the stack's deployment status, run the following command.

```
aws cloudformation describe-stacks --stack-name my-eks-custom-networking-vpc --
query Stacks\[\].StackStatus  --output text
```

Don't continue to the next step until the output of the command is `CREATE_COMPLETE`.

c. Define variables with the values of the private subnet IDs created by the template.

```
subnet_id_1=$(aws cloudformation describe-stack-resources --stack-name my-eks-
custom-networking-vpc \
    --query "StackResources[?
LogicalResourceId=='PrivateSubnet01'].PhysicalResourceId" --output text)
subnet_id_2=$(aws cloudformation describe-stack-resources --stack-name my-eks-
custom-networking-vpc \
    --query "StackResources[?
LogicalResourceId=='PrivateSubnet02'].PhysicalResourceId" --output text)
```

d. Define variables with the Availability Zones of the subnets retrieved in the previous step.

```
az_1=$(aws ec2 describe-subnets --subnet-ids $subnet_id_1 --query
 'Subnets[*].AvailabilityZone' --output text)
az_2=$(aws ec2 describe-subnets --subnet-ids $subnet_id_2 --query
 'Subnets[*].AvailabilityZone' --output text)
```

3. Create a cluster IAM role.

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

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

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

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

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

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

4. Create an Amazon EKS cluster and configure your device to communicate with it.

a. Create a cluster.

```
aws eks create-cluster --name my-custom-networking-cluster \
   --role-arn arn:aws:iam::$account_id:role/myCustomNetworkingAmazonEKSClusterRole \
   --resources-vpc-config subnetIds=$subnet_id_1","$subnet_id_2
```

Note

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

b. The cluster takes several minutes to create. To check on the cluster's deployment status, run the following command.

```
aws eks describe-cluster --name my-custom-networking-cluster --query
 cluster.status
```

Don't continue to the next step until the output of the command is `"ACTIVE"`.

c. Configure `kubectl` to communicate with your cluster.

```
aws eks update-kubeconfig --name my-custom-networking-cluster
```

Step 2: Configure your VPC

This tutorial requires the VPC created in the section called "Step 1: Create a test VPC and cluster". For a production cluster, adjust the steps accordingly for your VPC by replacing all of the example values with your own.

1. Confirm that your currently-installed Amazon VPC CNI plugin for Kubernetes is the latest version. To determine the latest version for the Amazon EKS add-on type and update your version to it, see the section called "Update an add-on". To determine the latest version for the self-managed add-on type and update your version to it, see the section called "Amazon VPC CNI".
2. Retrieve the ID of your cluster VPC and store it in a variable for use in later steps.

```
vpc_id=$(aws eks describe-cluster --name my-custom-networking-cluster --query
 "cluster.resourcesVpcConfig.vpcId" --output text)
```

3. Associate an additional Classless Inter-Domain Routing (CIDR) block with your cluster's VPC. The CIDR block can't overlap with any existing associated CIDR blocks.

a. View the current CIDR blocks associated to your VPC.

```
aws ec2 describe-vpcs --vpc-ids $vpc_id \
    --query 'Vpcs[*].CidrBlockAssociationSet[*].{CIDRBlock: CidrBlock, State:
 CidrBlockState.State}' --out table
```

An example output is as follows.

```
----------------------------------
|          DescribeVpcs          |
+-----------------+--------------+
```

```
|    CIDRBlock     |    State     |
+-----------------+--------------+
|  192.168.0.0/24 |  associated  |
+-----------------+--------------+
```

b. Associate an additional CIDR block to your VPC. Replace the CIDR block value in the following command. For more information, see Associate additional IPv4 CIDR blocks with your VPC in the Amazon VPC User Guide.

```
aws ec2 associate-vpc-cidr-block --vpc-id $vpc_id --cidr-block 192.168.1.0/24
```

c. Confirm that the new block is associated.

```
aws ec2 describe-vpcs --vpc-ids $vpc_id --query
 'Vpcs[*].CidrBlockAssociationSet[*].{CIDRBlock: CidrBlock, State:
 CidrBlockState.State}' --out table
```

An example output is as follows.

```
----------------------------------
|          DescribeVpcs          |
+-----------------+--------------+
|    CIDRBlock     |    State     |
+-----------------+--------------+
|  192.168.0.0/24 |  associated  |
|  192.168.1.0/24 |  associated  |
+-----------------+--------------+
```

Don’t proceed to the next step until your new CIDR block’s `State` is `associated`.

4. Create as many subnets as you want to use in each Availability Zone that your existing subnets are in. Specify a CIDR block that’s within the CIDR block that you associated with your VPC in a previous step.

   a. Create new subnets. Replace the CIDR block values in the following command. The subnets must be created in a different VPC CIDR block than your existing subnets are in, but in the same Availability Zones as your existing subnets. In this example, one subnet is created in the new CIDR block in each Availability Zone that the current private subnets exist in. The IDs of the subnets created are stored in variables for use in later steps. The `Name` values match the values assigned to the subnets created using the Amazon EKS VPC template in a previous step. Names aren’t required. You can use different names.

```
new_subnet_id_1=$(aws ec2 create-subnet --vpc-id $vpc_id --availability-zone $az_1
 --cidr-block 192.168.1.0/27 \
    --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=my-eks-custom-
networking-vpc-PrivateSubnet01},{Key=kubernetes.io/role/internal-elb,Value=1}]' \
    --query Subnet.SubnetId --output text)
new_subnet_id_2=$(aws ec2 create-subnet --vpc-id $vpc_id --availability-zone $az_2
 --cidr-block 192.168.1.32/27 \
    --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=my-eks-custom-
networking-vpc-PrivateSubnet02},{Key=kubernetes.io/role/internal-elb,Value=1}]' \
    --query Subnet.SubnetId --output text)
```

Important

By default, your new subnets are implicitly associated with your VPC's main route table. This route table allows communication between all the resources that are deployed in the VPC. However, it doesn't allow communication with resources that have IP addresses that are outside the CIDR blocks that are associated with your VPC. You can associate your own route table to your subnets to change this behavior. For more information, see Subnet route tables in the Amazon VPC User Guide.

b. View the current subnets in your VPC.

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$vpc_id" \
    --query 'Subnets[*].{SubnetId: SubnetId,AvailabilityZone:
 AvailabilityZone,CidrBlock: CidrBlock}' \
    --output table
```

An example output is as follows.

```
----------------------------------------------------------------------
|                           DescribeSubnets                          |
+------------------+--------------------+----------------------------+
| AvailabilityZone |     CidrBlock      |         SubnetId           |
+------------------+--------------------+----------------------------+
|  us-west-2d      |  192.168.0.0/27    |     subnet-example1        |
|  us-west-2a      |  192.168.0.32/27   |     subnet-example2        |
|  us-west-2a      |  192.168.0.64/27   |     subnet-example3        |
|  us-west-2d      |  192.168.0.96/27   |     subnet-example4        |
|  us-west-2a      |  192.168.1.0/27    |     subnet-example5        |
```

```
|  us-west-2d       |  192.168.1.32/27    |     subnet-example6         |
+-------------------+---------------------+-----------------------------+
```

You can see the subnets in the `192.168.1.0` CIDR block that you created are in the same Availability Zones as the subnets in the `192.168.0.0` CIDR block.

Step 3: Configure Kubernetes resources

1. Set the `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG` environment variable to `true` in the `aws-node` DaemonSet.

```
kubectl set env daemonset aws-node -n kube-system
 AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
```

2. Retrieve the ID of your cluster security group and store it in a variable for use in the next step. Amazon EKS automatically creates this security group when you create your cluster.

```
cluster_security_group_id=$(aws eks describe-cluster --name my-custom-networking-
cluster --query cluster.resourcesVpcConfig.clusterSecurityGroupId --output text)
```

3. Create an `ENIConfig` custom resource for each subnet that you want to deploy Pods in.

   a. Create a unique file for each network interface configuration.

      The following commands create separate `ENIConfig` files for the two subnets that were created in a previous step. The value for `name` must be unique. The name is the same as the Availability Zone that the subnet is in. The cluster security group is assigned to the `ENIConfig`.

```
cat >$az_1.yaml <<EOF
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
  name: $az_1
spec:
  securityGroups:
    - $cluster_security_group_id
  subnet: $new_subnet_id_1
EOF
```

```
cat >$az_2.yaml <<EOF
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
  name: $az_2
spec:
  securityGroups:
    - $cluster_security_group_id
  subnet: $new_subnet_id_2
EOF
```

For a production cluster, you can make the following changes to the previous commands:

- Replace $cluster_security_group_id with the ID of an existing security group that you want to use for each `ENIConfig`.
- We recommend naming your `ENIConfigs` the same as the Availability Zone that you'll use the `ENIConfig` for, whenever possible. You might need to use different names for your `ENIConfigs` than the names of the Availability Zones for a variety of reasons. For example, if you have more than two subnets in the same Availability Zone and want to use them both with custom networking, then you need multiple `ENIConfigs` for the same Availability Zone. Since each `ENIConfig` requires a unique name, you can't name more than one of your `ENIConfigs` using the Availability Zone name.

  If your `ENIConfig` names aren't all the same as Availability Zone names, then replace $az_1 and $az_2 with your own names in the previous commands and annotate your nodes with the ENIConfig later in this tutorial.

  > **Note**
  >
  > If you don't specify a valid security group for use with a production cluster and you're using:

- version `1.8.0` or later of the Amazon VPC CNI plugin for Kubernetes, then the security groups associated with the node's primary elastic network interface are used.
- a version of the Amazon VPC CNI plugin for Kubernetes that's earlier than `1.8.0`, then the default security group for the VPC is assigned to secondary network interfaces.

> **Important**
>
> - `AWS_VPC_K8S_CNI_EXTERNALSNAT=false` is a default setting in the configuration for the Amazon VPC CNI plugin for Kubernetes. If you're using the default setting, then traffic that is destined for IP addresses that aren't within one of the CIDR blocks associated with your VPC use the security groups and subnets of your node's primary network interface. The subnets and security groups defined in your `ENIConfigs` that are used to create secondary network interfaces aren't used for this traffic. For more information about this setting, see the section called "Outbound traffic".
> - If you also use security groups for Pods, the security group that's specified in a `SecurityGroupPolicy` is used instead of the security group that's specified in the `ENIConfigs`. For more information, see the section called "Security groups for Pods".

b. Apply each custom resource file that you created to your cluster with the following commands.

```
kubectl apply -f $az_1.yaml
kubectl apply -f $az_2.yaml
```

4. Confirm that your `ENIConfigs` were created.

```
kubectl get ENIConfigs
```

An example output is as follows.

```
NAME          AGE
us-west-2a    117s
us-west-2d    105s
```

5. If you're enabling custom networking on a production cluster and named your `ENIConfigs` something other than the Availability Zone that you're using them for, then skip to the next step to deploy Amazon EC2 nodes.

Enable Kubernetes to automatically apply the `ENIConfig` for an Availability Zone to any new Amazon EC2 nodes created in your cluster.

a. For the test cluster in this tutorial, skip to the next step.

   For a production cluster, check to see if an annotation with the key `k8s.amazonaws.com/eniConfig` for the `ENI_CONFIG_ANNOTATION_DEF` environment variable exists in the container spec for the `aws-node` DaemonSet.

```
kubectl describe daemonset aws-node -n kube-system | grep
 ENI_CONFIG_ANNOTATION_DEF
```

   If output is returned, the annotation exists. If no output is returned, then the variable is not set. For a production cluster, you can use either this setting or the setting in the following step. If you use this setting, it overrides the setting in the following step. In this tutorial, the setting in the next step is used.

b. Update your `aws-node` DaemonSet to automatically apply the `ENIConfig` for an Availability Zone to any new Amazon EC2 nodes created in your cluster.

```
kubectl set env daemonset aws-node -n kube-system
 ENI_CONFIG_LABEL_DEF=topology.kubernetes.io/zone
```

Step 4: Deploy Amazon EC2 nodes

1. Create a node IAM role.

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

```
cat >node-role-trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

b. Create an IAM role and store its returned Amazon Resource Name (ARN) in a variable for use in a later step.

```
node_role_arn=$(aws iam create-role --role-name myCustomNetworkingNodeRole --
assume-role-policy-document file://"node-role-trust-relationship.json" \
    --query Role.Arn --output text)
```

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

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

Important

For simplicity in this tutorial, the AmazonEKS_CNI_Policy policy is attached to the node IAM role. In a production cluster however, we recommend attaching the policy to a separate IAM role that is used only with the Amazon VPC CNI plugin for Kubernetes. For more information, see the section called "Configure for IRSA".

2. Create one of the following types of node groups. To determine the instance type that you want to deploy, see the section called "Amazon EC2 instance types". For this tutorial, complete the **Managed**, **Without a launch template or with a launch template without an AMI ID specified** option. If you're going to use the node group for production workloads, then we recommend that you familiarize yourself with all of the managed node group the section called "Create" and self-managed node group the section called "Self-managed nodes" options before deploying the node group.

   - **Managed –** Deploy your node group using one of the following options:
     - **Without a launch template or with a launch template without an AMI ID specified –** Run the following command. For this tutorial, use the example values. For a production node group, replace all example values with your own. The node group name can't be longer

than 63 characters. It must start with letter or digit, but can also include hyphens and underscores for the remaining characters.

```
aws eks create-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name my-nodegroup \
    --subnets $subnet_id_1 $subnet_id_2 --instance-types t3.medium --node-role $node_role_arn
```

- **With a launch template with a specified AMI ID**

  A. Determine the Amazon EKS recommended number of maximum Pods for your nodes. Follow the instructions in Amazon EKS recommended maximum Pods for each Amazon EC2 instance type, adding `--cni-custom-networking-enabled` to step 3 in that topic. Note the output for use in the next step.

  B. In your launch template, specify an Amazon EKS optimized AMI ID, or a custom AMI built off the Amazon EKS optimized AMI, then deploy the node group using a launch template and provide the following user data in the launch template. This user data passes arguments into the `bootstrap.sh` file. For more information about the bootstrap file, see bootstrap.sh on GitHub. You can replace `20` with either the value from the previous step (recommended) or your own value.

  ```
  /etc/eks/bootstrap.sh my-custom-networking-cluster --use-max-pods false --kubelet-extra-args '--max-pods=20'
  ```

  If you've created a custom AMI that is not built off the Amazon EKS optimized AMI, then you need to custom create the configuration yourself.

- **Self-managed**

  i. Determine the Amazon EKS recommended number of maximum Pods for your nodes. Follow the instructions in the section called "Amazon EKS recommended maximum Pods for each Amazon EC2 instance type", adding `--cni-custom-networking-enabled` to step 3 in that topic. Note the output for use in the next step.

  ii. Deploy the node group using the instructions in the section called "Amazon Linux". Specify the following text for the **BootstrapArguments** parameter. You can replace `20` with either the value from the previous step (recommended) or your own value.

  ```
  --use-max-pods false --kubelet-extra-args '--max-pods=20'
  ```

> **Note**
>
> If you want nodes in a production cluster to support a significantly higher number of Pods, run the script in the section called “Amazon EKS recommended maximum Pods for each Amazon EC2 instance type” again. Also, add the `--cni-prefix-delegation-enabled` option to the command. For example, `110` is returned for an `m5.large` instance type. For instructions on how to enable this capability, see the section called “Increase IP addresses”. You can use this capability with custom networking.

3. Node group creation takes several minutes. You can check the status of the creation of a managed node group with the following command.

```
aws eks describe-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name my-nodegroup --query nodegroup.status --output text
```

   Don’t continue to the next step until the output returned is `ACTIVE`.

4. For the tutorial, you can skip this step.

   For a production cluster, if you didn’t name your `ENIConfigs` the same as the Availability Zone that you’re using them for, then you must annotate your nodes with the `ENIConfig` name that should be used with the node. This step isn’t necessary if you only have one subnet in each Availability Zone and you named your `ENIConfigs` with the same names as your Availability Zones. This is because the Amazon VPC CNI plugin for Kubernetes automatically associates the correct `ENIConfig` with the node for you when you enabled it to do so in a previous step.

   a. Get the list of nodes in your cluster.

```
kubectl get nodes
```

   An example output is as follows.

```
NAME                                          STATUS   ROLES    AGE     VERSION
ip-192-168-0-126.us-west-2.compute.internal   Ready    <none>   8m49s   v1.22.9-eks-810597c
ip-192-168-0-92.us-west-2.compute.internal    Ready    <none>   8m34s   v1.22.9-eks-810597c
```

b. Determine which Availability Zone each node is in. Run the following command for each node that was returned in the previous step, replacing the IP addresses based on the previous output.

```
aws ec2 describe-instances --filters Name=network-interface.private-dns-
name,Values=ip-192-168-0-126.us-west-2.compute.internal \
--query 'Reservations[].Instances[].{AvailabilityZone: Placement.AvailabilityZone,
 SubnetId: SubnetId}'
```

An example output is as follows.

```
[
    {
        "AvailabilityZone": "us-west-2d",
        "SubnetId": "subnet-Example5"
    }
]
```

c. Annotate each node with the `ENIConfig` that you created for the subnet ID and Availability Zone. You can only annotate a node with one `ENIConfig`, though multiple nodes can be annotated with the same `ENIConfig`. Replace the example values with your own.

```
kubectl annotate node ip-192-168-0-126.us-west-2.compute.internal
 k8s.amazonaws.com/eniConfig=EniConfigName1
kubectl annotate node ip-192-168-0-92.us-west-2.compute.internal
 k8s.amazonaws.com/eniConfig=EniConfigName2
```

5. If you had nodes in a production cluster with running Pods before you switched to using the custom networking feature, complete the following tasks:

   a. Make sure that you have available nodes that are using the custom networking feature.

   b. Cordon and drain the nodes to gracefully shut down the Pods. For more information, see Safely Drain a Node in the Kubernetes documentation.

   c. Terminate the nodes. If the nodes are in an existing managed node group, you can delete the node group. Run the following command.

```
aws eks delete-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-
name my-nodegroup
```

Only new nodes that are registered with the `k8s.amazonaws.com/eniConfig` label use the custom networking feature.

6. Confirm that Pods are assigned an IP address from a CIDR block that's associated to one of the subnets that you created in a previous step.

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

An example output is as follows.

```
NAMESPACE     NAME                        READY   STATUS    RESTARTS   AGE     IP
          NODE                                        NOMINATED NODE   READINESS
 GATES
kube-system   aws-node-2rkn4              1/1     Running   0          7m19s
 192.168.0.92    ip-192-168-0-92.us-west-2.compute.internal    <none>
 <none>
kube-system   aws-node-k96wp              1/1     Running   0          7m15s
 192.168.0.126   ip-192-168-0-126.us-west-2.compute.internal   <none>
 <none>
kube-system   coredns-657694c6f4-smcgr    1/1     Running   0          56m
 192.168.1.23    ip-192-168-0-92.us-west-2.compute.internal    <none>
 <none>
kube-system   coredns-657694c6f4-stwv9    1/1     Running   0          56m
 192.168.1.28    ip-192-168-0-92.us-west-2.compute.internal    <none>
 <none>
kube-system   kube-proxy-jgshq            1/1     Running   0          7m19s
 192.168.0.92    ip-192-168-0-92.us-west-2.compute.internal    <none>
 <none>
kube-system   kube-proxy-wx9vk            1/1     Running   0          7m15s
 192.168.0.126   ip-192-168-0-126.us-west-2.compute.internal   <none>
 <none>
```

You can see that the coredns Pods are assigned IP addresses from the `192.168.1.0` CIDR block that you added to your VPC. Without custom networking, they would have been assigned addresses from the `192.168.0.0` CIDR block, because it was the only CIDR block originally associated with the VPC.

If a Pod's `spec` contains `hostNetwork=true`, it's assigned the primary IP address of the node. It isn't assigned an address from the subnets that you added. By default, this value is set to `false`. This value is set to `true` for the `kube-proxy` and Amazon VPC CNI plugin for

Kubernetes (`aws-node`) Pods that run on your cluster. This is why the `kube-proxy` and the plugin's `aws-node` Pods aren't assigned 192.168.1.x addresses in the previous output. For more information about a Pod's `hostNetwork` setting, see [PodSpec v1 core](#) in the Kubernetes API reference.

Step 5: Delete tutorial resources

After you complete the tutorial, we recommend that you delete the resources that you created. You can then adjust the steps to enable custom networking for a production cluster.

1. If the node group that you created was just for testing, then delete it.

```
aws eks delete-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name
 my-nodegroup
```

2. Even after the AWS CLI output says that the cluster is deleted, the delete process might not actually be complete. The delete process takes a few minutes. Confirm that it's complete by running the following command.

```
aws eks describe-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-
name my-nodegroup --query nodegroup.status --output text
```

   Don't continue until the returned output is similar to the following output.

```
An error occurred (ResourceNotFoundException) when calling the DescribeNodegroup
 operation: No node group found for name: my-nodegroup.
```

3. If the node group that you created was just for testing, then delete the node IAM role.

   a. Detach the policies from the role.

```
aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn
 arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn
 arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn
 arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
```

   b. Delete the role.

```
aws iam delete-role --role-name myCustomNetworkingNodeRole
```

4. Delete the cluster.

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

Confirm the cluster is deleted with the following command.

```
aws eks describe-cluster --name my-custom-networking-cluster --query cluster.status
 --output text
```

When output similar to the following is returned, the cluster is successfully deleted.

```
An error occurred (ResourceNotFoundException) when calling the DescribeCluster
 operation: No cluster found for name: my-custom-networking-cluster.
```

5. Delete the cluster IAM role.

   a. Detach the policies from the role.

   ```
   aws iam detach-role-policy --role-name myCustomNetworkingAmazonEKSClusterRole --
   policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
   ```

   b. Delete the role.

   ```
   aws iam delete-role --role-name myCustomNetworkingAmazonEKSClusterRole
   ```

6. Delete the subnets that you created in a previous step.

```
aws ec2 delete-subnet --subnet-id $new_subnet_id_1
aws ec2 delete-subnet --subnet-id $new_subnet_id_2
```

7. Delete the VPC that you created.

```
aws cloudformation delete-stack --stack-name my-eks-custom-networking-vpc
```

Assign more IP addresses to Amazon EKS nodes with prefixes

Applies to: Linux and Windows nodes with Amazon EC2 instances

Applies to: Public and private subnets

Each Amazon EC2 instance supports a maximum number of elastic network interfaces and a maximum number of IP addresses that can be assigned to each network interface. Each node requires one IP address for each network interface. All other available IP addresses can be assigned to Pods. Each Pod requires its own IP address. As a result, you might have nodes that have available compute and memory resources, but can't accommodate additional Pods because the node has run out of IP addresses to assign to Pods.

You can increase the number of IP addresses that nodes can assign to Pods by assigning IP prefixes, rather than assigning individual secondary IP addresses to your nodes. Each prefix includes several IP addresses. If you don't configure your cluster for IP prefix assignment, your cluster must make more Amazon EC2 application programming interface (API) calls to configure network interfaces and IP addresses necessary for Pod connectivity. As clusters grow to larger sizes, the frequency of these API calls can lead to longer Pod and instance launch times. This results in scaling delays to meet the demand of large and spiky workloads, and adds cost and management overhead because you need to provision additional clusters and VPCs to meet scaling requirements. For more information, see Kubernetes Scalability thresholds on GitHub.

Compatibility with Amazon VPC CNI plugin for Kubernetes features

You can use IP prefixes with the following features:

- IPv4 Source Network Address Translation - For more information, see the section called "Outbound traffic".
- IPv6 addresses to clusters, Pods, and services - For more information, see the section called "IPv6".
- Restricting traffic using Kubernetes network policies - For more information, see the section called "Kubernetes policies".

The following list provides information about the Amazon VPC CNI plugin settings that apply. For more information about each setting, see amazon-vpc-cni-k8s on GitHub.

- `WARM_IP_TARGET`
- `MINIMUM_IP_TARGET`
- `WARM_PREFIX_TARGET`

Considerations

Consider the following when you use this feature:

- Each Amazon EC2 instance type supports a maximum number of Pods. If your managed node group consists of multiple instance types, the smallest number of maximum Pods for an instance in the cluster is applied to all nodes in the cluster.
- By default, the maximum number of `Pods` that you can run on a node is 110, but you can change that number. If you change the number and have an existing managed node group, the next AMI or launch template update of your node group results in new nodes coming up with the changed value.
- When transitioning from assigning IP addresses to assigning IP prefixes, we recommend that you create new node groups to increase the number of available IP addresses, rather than doing a rolling replacement of existing nodes. Running Pods on a node that has both IP addresses and prefixes assigned can lead to inconsistency in the advertised IP address capacity, impacting the future workloads on the node. For the recommended way of performing the transition, see [Replace all nodes during migration from Secondary IP mode to Prefix Delegation mode or vice versa](#) in the Amazon EKS best practices guide.
- The security group scope is at the node-level - For more information, see [Security group](#).
- IP prefixes assigned to a network interface support high Pod density per node and have the best launch time.
- IP prefixes and IP addresses are associated with standard Amazon EC2 elastic network interfaces. Pods requiring specific security groups are assigned the primary IP address of a branch network interface. You can mix Pods getting IP addresses, or IP addresses from IP prefixes with Pods getting branch network interfaces on the same node.
- For clusters with Linux nodes only.
  - After you configure the add-on to assign prefixes to network interfaces, you can't downgrade your Amazon VPC CNI plugin for Kubernetes add-on to a version lower than `1.9.0` (or `1.10.1`) without removing all nodes in all node groups in your cluster.
  - If you're also using security groups for Pods, with `POD_SECURITY_GROUP_ENFORCING_MODE=standard` and `AWS_VPC_K8S_CNI_EXTERNALSNAT=false`, when your Pods communicate with endpoints outside of your VPC, the node's security groups are used, rather than any security groups you've assigned to your Pods.

If you're also using security groups for Pods, with `POD_SECURITY_GROUP_ENFORCING_MODE=strict`, when your `Pods` communicate with endpoints outside of your VPC, the `Pod's` security groups are used.

Increase the available IP addresses for your Amazon EKS node

You can increase the number of IP addresses that nodes can assign to Pods by assigning IP prefixes, rather than assigning individual secondary IP addresses to your nodes.

Prerequisites

- You need an existing cluster. To deploy one, see the section called "Create a cluster".
- The subnets that your Amazon EKS nodes are in must have sufficient contiguous `/28` (for `IPv4` clusters) or `/80` (for `IPv6` clusters) Classless Inter-Domain Routing (CIDR) blocks. You can only have Linux nodes in an `IPv6` cluster. Using IP prefixes can fail if IP addresses are scattered throughout the subnet CIDR. We recommend the following:
  - Using a subnet CIDR reservation so that even if any IP addresses within the reserved range are still in use, upon their release, the IP addresses aren't reassigned. This ensures that prefixes are available for allocation without segmentation.
  - Use new subnets that are specifically used for running the workloads that IP prefixes are assigned to. Both Windows and Linux workloads can run in the same subnet when assigning IP prefixes.
- To assign IP prefixes to your nodes, your nodes must be AWS Nitro-based. Instances that aren't Nitro-based continue to allocate individual secondary IP addresses, but have a significantly lower number of IP addresses to assign to Pods than Nitro-based instances do.
- **For clusters with Linux nodes only** – If your cluster is configured for the `IPv4` family, you must have version `1.9.0` or later of the Amazon VPC CNI plugin for Kubernetes add-on installed. You can check your current version with the following command.

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

  If your cluster is configured for the `IPv6` family, you must have version `1.10.1` of the add-on installed. If your plugin version is earlier than the required versions, you must update it. For more information, see the updating sections of Assign IPs to Pods with the Amazon VPC CNI.
- **For clusters with Windows nodes only**

- Your cluster and its platform version must be at, or later than the versions in the following table. To upgrade your cluster version, see the section called "Update Kubernetes version". If your cluster isn't at the minimum platform version, then you can't assign IP prefixes to your nodes until Amazon EKS has updated your platform version.

Kubernetes version	Platform version
1.27	eks.3
1.26	eks.4
1.25	eks.5

  You can check your current Kubernetes and platform version by replacing *my-cluster* in the following command with the name of your cluster and then running the modified command: `aws eks describe-cluster --name my-cluster --query 'cluster.{"Kubernetes Version": version, "Platform Version": platformVersion}'`.
- You must have Windows support enabled for your cluster. For more information, see the section called "Enable Windows support".

Assign IP address prefixes to nodes

Configure your cluster to assign IP address prefixes to nodes. Complete the procedure that matches your node's operating system.

Linux

1. Enable the parameter to assign prefixes to network interfaces for the Amazon VPC CNI DaemonSet. When you deploy a `1.21` or later cluster, version `1.10.1` or later of the Amazon VPC CNI plugin for Kubernetes add-on is deployed with it. If you created the cluster with the `IPv6` family, this setting was set to `true` by default. If you created the cluster with the `IPv4` family, this setting was set to `false` by default.

   ```
   kubectl set env daemonset aws-node -n kube-system ENABLE_PREFIX_DELEGATION=true
   ```

> ⚠ **Important**
>
> Even if your subnet has available IP addresses, if the subnet does not have any contiguous /28 blocks available, you will see the following error in the Amazon VPC CNI plugin for Kubernetes logs.
>
> ```
> InsufficientCidrBlocks: The specified subnet does not have enough free cidr
>  blocks to satisfy the request
> ```
>
> This can happen due to fragmentation of existing secondary IP addresses spread out across a subnet. To resolve this error, either create a new subnet and launch Pods there, or use an Amazon EC2 subnet CIDR reservation to reserve space within a subnet for use with prefix assignment. For more information, see Subnet CIDR reservations in the Amazon VPC User Guide.

2. If you plan to deploy a managed node group without a launch template, or with a launch template that you haven't specified an AMI ID in, and you're using a version of the Amazon VPC CNI plugin for Kubernetes at or later than the versions listed in the prerequisites, then skip to the next step. Managed node groups automatically calculates the maximum number of Pods for you.

   If you're deploying a self-managed node group or a managed node group with a launch template that you have specified an AMI ID in, then you must determine the Amazon EKS recommend number of maximum Pods for your nodes. Follow the instructions in Amazon EKS recommended maximum Pods for each Amazon EC2 instance type, adding `--cni-prefix-delegation-enabled` to step 3. Note the output for use in a later step.

   > ⚠ **Important**
   >
   > Managed node groups enforces a maximum number on the value of `maxPods`. For instances with less than 30 vCPUs the maximum number is 110 and for all other instances the maximum number is 250. This maximum number is applied whether prefix delegation is enabled or not.

3. If you're using a `1.21` or later cluster configured for `IPv6`, skip to the next step.

Specify the parameters in one of the following options. To determine which option is right for you and what value to provide for it, see WARM_PREFIX_TARGET, WARM_IP_TARGET, and MINIMUM_IP_TARGET on GitHub.

You can replace the *example values* with a value greater than zero.

- `WARM_PREFIX_TARGET`

```
kubectl set env ds aws-node -n kube-system WARM_PREFIX_TARGET=1
```

- `WARM_IP_TARGET` or `MINIMUM_IP_TARGET` – If either value is set, it overrides any value set for `WARM_PREFIX_TARGET`.

```
kubectl set env ds aws-node -n kube-system WARM_IP_TARGET=5
```

```
kubectl set env ds aws-node -n kube-system MINIMUM_IP_TARGET=2
```

4. Create one of the following types of node groups with at least one Amazon EC2 Nitro Amazon Linux 2 instance type. For a list of Nitro instance types, see Instances built on the Nitro System in the Amazon EC2 User Guide. This capability is not supported on Windows. For the options that include *110*, replace it with either the value from step 3 (recommended), or your own value.

- **Self-managed** – Deploy the node group using the instructions in Create self-managed Amazon Linux nodes. Specify the following text for the **BootstrapArguments** parameter.

```
--use-max-pods false --kubelet-extra-args '--max-pods=110'
```

If you're using `eksctl` to create the node group, you can use the following command.

```
eksctl create nodegroup --cluster my-cluster --managed=false --max-pods-per-node
 110
```

- **Managed** – Deploy your node group using one of the following options:
  - **Without a launch template or with a launch template without an AMI ID specified** – Complete the procedure in Create a managed node group for your cluster. Managed node groups automatically calculates the Amazon EKS recommended `max-pods` value for you.
  - **With a launch template with a specified AMI ID** – In your launch template, specify an Amazon EKS optimized AMI ID, or a custom AMI built off the Amazon EKS optimized AMI, then deploy the node group using a launch template and provide the following user data

in the launch template. This user data passes arguments into the `bootstrap.sh` file. For more information about the bootstrap file, see bootstrap.sh on GitHub.

```
/etc/eks/bootstrap.sh my-cluster \
  --use-max-pods false \
  --kubelet-extra-args '--max-pods=110'
```

If you're using `eksctl` to create the node group, you can use the following command.

```
eksctl create nodegroup --cluster my-cluster --max-pods-per-node 110
```

If you've created a custom AMI that is not built off the Amazon EKS optimized AMI, then you need to custom create the configuration yourself.

> **Note**
>
> If you also want to assign IP addresses to Pods from a different subnet than the instance's, then you need to enable the capability in this step. For more information, see the section called "Custom networking".

Windows

1. Enable assignment of IP prefixes.

   a. Open the `amazon-vpc-cni` `ConfigMap` for editing.

   ```
   kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
   ```

   b. Add the following line to the `data` section.

   ```
     enable-windows-prefix-delegation: "true"
   ```

   c. Save the file and close the editor.

   d. Confirm that the line was added to the `ConfigMap`.

   ```
   kubectl get configmap -n kube-system amazon-vpc-cni -o "jsonpath={.data.enable-
   windows-prefix-delegation}"
   ```

If the returned output isn't `true`, then there might have been an error. Try completing the step again.

> **Important**
>
> Even if your subnet has available IP addresses, if the subnet does not have any contiguous /28 blocks available, you will see the following error in the Amazon VPC CNI plugin for Kubernetes logs.
>
> ```
> InsufficientCidrBlocks: The specified subnet does not have enough free cidr
>  blocks to satisfy the request
> ```
>
> This can happen due to fragmentation of existing secondary IP addresses spread out across a subnet. To resolve this error, either create a new subnet and launch Pods there, or use an Amazon EC2 subnet CIDR reservation to reserve space within a subnet for use with prefix assignment. For more information, see [Subnet CIDR reservations] in the Amazon VPC User Guide.

2. (Optional) Specify additional configuration for controlling the pre-scaling and dynamic scaling behavior for your cluster. For more information, see [Configuration options with Prefix Delegation mode on Windows] on GitHub.

   a. Open the `amazon-vpc-cni` `ConfigMap` for editing.

   ```
   kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
   ```

   b. Replace the *`example values`* with a value greater than zero and add the entries that you require to the `data` section of the `ConfigMap`. If you set a value for either `warm-ip-target` or `minimum-ip-target`, the value overrides any value set for `warm-prefix-target`.

   ```
     warm-prefix-target: "1"
     warm-ip-target: "5"
     minimum-ip-target: "2"
   ```

   c. Save the file and close the editor.

3. Create Windows node groups with at least one Amazon EC2 Nitro instance type. For a list of Nitro instance types, see [Instances built on the Nitro System] in the Amazon EC2 User Guide. By default, the maximum number of Pods that you can deploy to a node is 110. If you want

to increase or decrease that number, specify the following in the user data for the bootstrap configuration. Replace *max-pods-quantity* with your max pods value.

```
-KubeletExtraArgs '--max-pods=max-pods-quantity'
```

If you're deploying managed node groups, this configuration needs to be added in the launch template. For more information, see the section called "Launch templates". For more information about the configuration parameters for Windows bootstrap script, see the section called "Bootstrap script configuration parameters".

Determine max Pods and available IP addresses

1. Once your nodes are deployed, view the nodes in your cluster.

```
kubectl get nodes
```

An example output is as follows.

```
NAME                                          STATUS     ROLES    AGE   VERSION
ip-192-168-22-103.region-code.compute.internal   Ready      <none>   19m   v1.XX.X-
eks-6b7464
ip-192-168-97-94.region-code.compute.internal    Ready      <none>   19m   v1.XX.X-
eks-6b7464
```

2. Describe one of the nodes to determine the value of `max-pods` for the node and the number of available IP addresses. Replace *192.168.30.193* with the `IPv4` address in the name of one of your nodes returned in the previous output.

```
kubectl describe node ip-192-168-30-193.region-code.compute.internal | grep 'pods\|
PrivateIPv4Address'
```

An example output is as follows.

```
pods:                                   110
vpc.amazonaws.com/PrivateIPv4Address:   144
```

In the previous output, `110` is the maximum number of Pods that Kubernetes will deploy to the node, even though *144* IP addresses are available.

Assign security groups to individual Pods

Applies to: Linux nodes with Amazon EC2 instances

Applies to: Private subnets

Security groups for Pods integrate Amazon EC2 security groups with Kubernetes Pods. You can use Amazon EC2 security groups to define rules that allow inbound and outbound network traffic to and from Pods that you deploy to nodes running on many Amazon EC2 instance types and Fargate. For a detailed explanation of this capability, see the Introducing security groups for Pods blog post.

Compatibility with Amazon VPC CNI plugin for Kubernetes features

You can use security groups for Pods with the following features:

- IPv4 Source Network Address Translation - For more information, see the section called “Outbound traffic”.
- IPv6 addresses to clusters, Pods, and services - For more information, see the section called “IPv6”.
- Restricting traffic using Kubernetes network policies - For more information, see the section called “Kubernetes policies”.

Considerations

Before deploying security groups for Pods, consider the following limitations and conditions:

- Security groups for Pods can't be used with Windows nodes.
- Security groups for Pods can be used with clusters configured for the `IPv6` family that contain Amazon EC2 nodes by using version 1.16.0 or later of the Amazon VPC CNI plugin. You can use security groups for Pods with clusters configure `IPv6` family that contain only Fargate nodes by using version 1.7.7 or later of the Amazon VPC CNI plugin. For more information, see the section called “IPv6”
- Security groups for Pods are supported by most Nitro-based Amazon EC2 instance families, though not by all generations of a family. For example, the `m5`, `c5`, `r5`, `m6g`, `c6g`, and `r6g` instance family and generations are supported. No instance types in the `t` family are supported. For a complete list of supported instance types, see the limits.go file on GitHub. Your nodes must be one of the listed instance types that have `IsTrunkingCompatible: true` in that file.

- If you're also using Pod security policies to restrict access to Pod mutation, then the `eks:vpc-resource-controller` Kubernetes user must be specified in the Kubernetes `ClusterRoleBinding` for the `role` that your `psp` is assigned to. If you're using the default Amazon EKS `psp`, `role`, and `ClusterRoleBinding`, this is the `eks:podsecuritypolicy:authenticated` `ClusterRoleBinding`. For example, you add the user to the `subjects:` section, as shown in the following example:

```
[...]
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:authenticated
  - apiGroup: rbac.authorization.k8s.io
    kind: User
    name: eks:vpc-resource-controller
  - kind: ServiceAccount
    name: eks-vpc-resource-controller
```

- If you're using custom networking and security groups for Pods together, the security group specified by security groups for Pods is used instead of the security group specified in the `ENIConfig`.
- If you're using version `1.10.2` or earlier of the Amazon VPC CNI plugin and you include the `terminationGracePeriodSeconds` setting in your Pod spec, the value for the setting can't be zero.
- If you're using version `1.10` or earlier of the Amazon VPC CNI plugin, or version `1.11` with `POD_SECURITY_GROUP_ENFORCING_MODE=strict`, which is the default setting, then Kubernetes services of type `NodePort` and `LoadBalancer` using instance targets with an `externalTrafficPolicy` set to `Local` aren't supported with Pods that you assign security groups to. For more information about using a load balancer with instance targets, see the section called "Network load balancing".
- If you're using version `1.10` or earlier of the Amazon VPC CNI plugin or version `1.11` with `POD_SECURITY_GROUP_ENFORCING_MODE=strict`, which is the default setting, source NAT is disabled for outbound traffic from Pods with assigned security groups so that outbound security group rules are applied. To access the internet, Pods with assigned security groups must be launched on nodes that are deployed in a private subnet configured with a NAT gateway or instance. Pods with assigned security groups deployed to public subnets are not able to access the internet.

If you're using version `1.11` or later of the plugin with `POD_SECURITY_GROUP_ENFORCING_MODE=standard`, then Pod traffic destined for outside of the VPC is translated to the IP address of the instance's primary network interface. For this traffic, the rules in the security groups for the primary network interface are used, rather than the rules in the Pod's security groups.

- To use Calico network policy with Pods that have associated security groups, you must use version `1.11.0` or later of the Amazon VPC CNI plugin and set `POD_SECURITY_GROUP_ENFORCING_MODE=standard`. Otherwise, traffic flow to and from Pods with associated security groups are not subjected to Calico network policy enforcement and are limited to Amazon EC2 security group enforcement only. To update your Amazon VPC CNI version, see the section called "Amazon VPC CNI"
- Pods running on Amazon EC2 nodes that use security groups in clusters that use NodeLocal DNSCache are only supported with version `1.11.0` or later of the Amazon VPC CNI plugin and with `POD_SECURITY_GROUP_ENFORCING_MODE=standard`. To update your Amazon VPC CNI plugin version, see the section called "Amazon VPC CNI"
- Security groups for Pods might lead to higher Pod startup latency for Pods with high churn. This is due to rate limiting in the resource controller.
- The EC2 security group scope is at the Pod-level - For more information, see Security group.

  If you set `POD_SECURITY_GROUP_ENFORCING_MODE=standard` and `AWS_VPC_K8S_CNI_EXTERNALSNAT=false`, traffic destined for endpoints outside the VPC use the node's security groups, not the Pod's security groups.

Configure the Amazon VPC CNI plugin for Kubernetes for security groups for Amazon EKS Pods

If you use Pods with Amazon EC2 instances, you need to configure the Amazon VPC CNI plugin for Kubernetes for security groups

If you use Fargate Pods only, and don't have any Amazon EC2 nodes in your cluster, see the section called "SecurityGroupPolicy".

1. Check your current Amazon VPC CNI plugin for Kubernetes version with the following command:

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

An example output is as follows.

```
v1.7.6
```

If your Amazon VPC CNI plugin for Kubernetes version is earlier than `1.7.7`, then update the plugin to version `1.7.7` or later. For more information, see the section called "Amazon VPC CNI"

2. Add the AmazonEKSVPCResourceController managed IAM policy to the cluster role that is associated with your Amazon EKS cluster. The policy allows the role to manage network interfaces, their private IP addresses, and their attachment and detachment to and from network instances.

   a. Retrieve the name of your cluster IAM role and store it in a variable. Replace *`my-cluster`* with the name of your cluster.

   ```
   cluster_role=$(aws eks describe-cluster --name my-cluster --query cluster.roleArn
    --output text | cut -d / -f 2)
   ```

   b. Attach the policy to the role.

   ```
   aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
   AmazonEKSVPCResourceController --role-name $cluster_role
   ```

3. Enable the Amazon VPC CNI add-on to manage network interfaces for Pods by setting the `ENABLE_POD_ENI` variable to `true` in the `aws-node` DaemonSet. Once this setting is set to `true`, for each node in the cluster the add-on creates a `cninode` custom resource. The VPC resource controller creates and attaches one special network interface called a *trunk network interface* with the description `aws-k8s-trunk-eni`.

```
kubectl set env daemonset aws-node -n kube-system ENABLE_POD_ENI=true
```

Note

The trunk network interface is included in the maximum number of network interfaces supported by the instance type. For a list of the maximum number of network interfaces supported by each instance type, see IP addresses per network interface per instance type in the *Amazon EC2 User Guide*. If your node already has the maximum number of standard network interfaces attached to it then the VPC resource controller will reserve a space. You will have to scale down your running Pods enough for the controller to detach and delete a standard network interface, create the trunk network interface, and attach it to the instance.

4. You can see which of your nodes have a `CNINode` custom resource with the following command. If `No resources found` is returned, then wait several seconds and try again. The previous step requires restarting the Amazon VPC CNI plugin for Kubernetes Pods, which takes several seconds.

```
kubectl get cninode -A
     NAME FEATURES
     ip-192-168-64-141.us-west-2.compute.internal [{"name":"SecurityGroupsForPods"}]
     ip-192-168-7-203.us-west-2.compute.internal [{"name":"SecurityGroupsForPods"}]
```

If you are using VPC CNI versions older than `1.15`, node labels were used instead of the `CNINode` custom resource. You can see which of your nodes have the node label `aws-k8s-trunk-eni` set to `true` with the following command. If `No resources found` is returned, then wait several seconds and try again. The previous step requires restarting the Amazon VPC CNI plugin for Kubernetes Pods, which takes several seconds.

```
kubectl get nodes -o wide -l vpc.amazonaws.com/has-trunk-attached=true
-
```

Once the trunk network interface is created, Pods are assigned secondary IP addresses from the trunk or standard network interfaces. The trunk interface is automatically deleted if the node is deleted.

When you deploy a security group for a Pod in a later step, the VPC resource controller creates a special network interface called a *branch network interface* with a description of `aws-k8s-branch-eni` and associates the security groups to it. Branch network interfaces are created in addition to the standard and trunk network interfaces attached to the node.

If you are using liveness or readiness probes, then you also need to disable *TCP early demux*, so that the `kubelet` can connect to Pods on branch network interfaces using TCP. To disable *TCP early demux*, run the following command:

```
kubectl patch daemonset aws-node -n kube-system \
  -p '{"spec": {"template": {"spec": {"initContainers": [{"env":
[{"name":"DISABLE_TCP_EARLY_DEMUX","value":"true"}],"name":"aws-vpc-cni-init"}]}}}}'
```

> **Note**
>
> If you're using `1.11.0` or later of the Amazon VPC CNI plugin for Kubernetes add-on and set `POD_SECURITY_GROUP_ENFORCING_MODE=standard`, as described in the next step, then you don't need to run the previous command.

5. If your cluster uses `NodeLocal DNSCache`, or you want to use Calico network policy with your Pods that have their own security groups, or you have Kubernetes services of type `NodePort` and `LoadBalancer` using instance targets with an `externalTrafficPolicy` set to `Local` for Pods that you want to assign security groups to, then you must be using version `1.11.0` or later of the Amazon VPC CNI plugin for Kubernetes add-on, and you must enable the following setting:

   ```
   kubectl set env daemonset aws-node -n kube-system
    POD_SECURITY_GROUP_ENFORCING_MODE=standard
   ```

   IMPORTANT: **Pod security group rules aren't applied to traffic between Pods or between Pods and services, such as `kubelet` or `nodeLocalDNS`, that are on the same node. Pods using different security groups on the same node can't communicate because they are configured in different subnets, and routing is disabled between these subnets.** Outbound traffic from Pods to addresses outside of the VPC is network address translated to the IP address of the instance's primary network interface (unless you've also set `AWS_VPC_K8S_CNI_EXTERNALSNAT=true`). For this traffic, the rules in the security groups for the primary network interface are used, rather than the rules in the Pod's security groups. ** For this setting to apply to existing Pods, you must restart the Pods or the nodes that the Pods are running on.

6. To see how to use a security group policy for your Pod, see the section called "SecurityGroupPolicy".

Use a security group policy for an Amazon EKS Pod

To use security groups for Pods, you must have an existing security group. The following steps show you how to use the security group policy for a Pod. Unless otherwise noted, complete all steps from the same terminal because variables are used in the following steps that don't persist across terminals.

If you have a Pod with Amazon EC2 instances, you must configure the plugin before you use this procedure. For more information, see the section called “Configure”.

1. Create a Kubernetes namespace to deploy resources to. You can replace *my-namespace* with the name of a namespace that you want to use.

```
kubectl create namespace my-namespace
```

2. Deploy an Amazon EKS `SecurityGroupPolicy` to your cluster.

   a. Copy the following contents to your device. You can replace *podSelector* with `serviceAccountSelector` if you'd rather select Pods based on service account labels. You must specify one selector or the other. An empty `podSelector` (example: `podSelector: {}`) selects all Pods in the namespace. You can change *my-role* to the name of your role. An empty `serviceAccountSelector` selects all service accounts in the namespace. You can replace *my-security-group-policy* with a name for your `SecurityGroupPolicy` and *my-namespace* with the namespace that you want to create the `SecurityGroupPolicy` in.

   You must replace *my_pod_security_group_id* with the ID of an existing security group. If you don't have an existing security group, then you must create one. For more information, see Amazon EC2 security groups for Linux instances in the Amazon EC2 User Guide. You can specify 1-5 security group IDs. If you specify more than one ID, then the combination of all the rules in all the security groups are effective for the selected Pods.

```
cat >my-security-group-policy.yaml <<EOF
apiVersion: vpcresources.k8s.aws/v1beta1
kind: SecurityGroupPolicy
metadata:
  name: my-security-group-policy
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      role: my-role
  securityGroups:
    groupIds:
      - my_pod_security_group_id
EOF
```

> **Important**
>
> The security group or groups that you specify for your Pods must meet the following criteria:
>
> - They must exist. If they don't exist, then, when you deploy a Pod that matches the selector, your Pod remains stuck in the creation process. If you describe the Pod, you'll see an error message similar to the following one: `An error occurred (InvalidSecurityGroupID.NotFound) when calling the CreateNetworkInterface operation: The securityGroup ID 'sg-05b1d815d1EXAMPLE' does not exist.`
> - They must allow inbound communication from the security group applied to your nodes (for `kubelet`) over any ports that you've configured probes for.
> - They must allow outbound communication over TCP and UDP ports 53 to a security group assigned to the Pods (or nodes that the Pods run on) running CoreDNS. The security group for your CoreDNS Pods must allow inbound TCP and UDP port 53 traffic from the security group that you specify.
> - They must have necessary inbound and outbound rules to communicate with other Pods that they need to communicate with.
> - They must have rules that allow the Pods to communicate with the Kubernetes control plane if you're using the security group with Fargate. The easiest way to do this is to specify the cluster security group as one of the security groups.
>
> Security group policies only apply to newly scheduled Pods. They do not affect running Pods.

b. Deploy the policy.

```
kubectl apply -f my-security-group-policy.yaml
```

3. Deploy a sample application with a label that matches the *`my-role`* value for *`podSelector`* that you specified in a previous step.

   a. Copy the following contents to your device. Replace the *`example values`* with your own and then run the modified command. If you replace *`my-role`*, make sure that it's the same as the value you specified for the selector in a previous step.

```
cat >sample-application.yaml <<EOF
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
  namespace: my-namespace
  labels:
    app: my-app
spec:
  replicas: 4
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
        role: my-role
    spec:
      terminationGracePeriodSeconds: 120
      containers:
      - name: nginx
        image: public.ecr.aws/nginx/nginx:1.23
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: my-app
  namespace: my-namespace
  labels:
    app: my-app
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
EOF
```

b. Deploy the application with the following command. When you deploy the application, the Amazon VPC CNI plugin for Kubernetes matches the `role` label and the security groups that you specified in the previous step are applied to the Pod.

```
kubectl apply -f sample-application.yaml
```

4. View the Pods deployed with the sample application. For the remainder of this topic, this terminal is referred to as `TerminalA`.

```
kubectl get pods -n my-namespace -o wide
```

An example output is as follows.

```
NAME                                 READY   STATUS    RESTARTS   AGE     IP
  NODE                                            NOMINATED NODE   READINESS GATES
my-deployment-5df6f7687b-4fbjm   1/1     Running   0          7m51s   192.168.53.48
  ip-192-168-33-28.region-code.compute.internal   <none>           <none>
my-deployment-5df6f7687b-j9fl4   1/1     Running   0          7m51s   192.168.70.145
  ip-192-168-92-33.region-code.compute.internal   <none>           <none>
my-deployment-5df6f7687b-rjxcz   1/1     Running   0          7m51s   192.168.73.207
  ip-192-168-92-33.region-code.compute.internal   <none>           <none>
my-deployment-5df6f7687b-zmb42   1/1     Running   0          7m51s   192.168.63.27
  ip-192-168-33-28.region-code.compute.internal   <none>           <none>
```

Note

Try these tips if any Pods are stuck.

- If any Pods are stuck in the `Waiting` state, then run `kubectl describe pod my-deployment-xxxxxxxxxx-xxxxx -n my-namespace`. If you see `Insufficient permissions: Unable to create Elastic Network Interface.`, confirm that you added the IAM policy to the IAM cluster role in a previous step.
- If any Pods are stuck in the `Pending` state, confirm that your node instance type is listed in limits.go and that the product of the maximum number of branch network interfaces supported by the instance type multiplied times the number of nodes in your node group hasn't already been met. For example, an `m5.large` instance supports nine branch network interfaces. If your node group has five nodes, then a maximum of 45 branch network interfaces can be created for the node group. The

46th Pod that you attempt to deploy will sit in `Pending` state until another Pod that has associated security groups is deleted.

If you run `kubectl describe pod my-deployment-xxxxxxxxxx-xxxxx -n my-namespace` and see a message similar to the following message, then it can be safely ignored. This message might appear when the Amazon VPC CNI plugin for Kubernetes tries to set up host networking and fails while the network interface is being created. The plugin logs this event until the network interface is created.

```
Failed to create Pod sandbox: rpc error: code = Unknown desc = failed to set up
 sandbox container "e24268322e55c8185721f52df6493684f6c2c3bf4fd59c9c121fd4cdc894579f"
 network for Pod "my-deployment-5df6f7687b-4fbjm": networkPlugin
cni failed to set up Pod "my-deployment-5df6f7687b-4fbjm-c89wx_my-namespace" network:
 add cmd: failed to assign an IP address to container
```

You can't exceed the maximum number of Pods that can be run on the instance type. For a list of the maximum number of Pods that you can run on each instance type, see eni-max-pods.txt on GitHub. When you delete a Pod that has associated security groups, or delete the node that the Pod is running on, the VPC resource controller deletes the branch network interface. If you delete a cluster with Pods using Pods for security groups, then the controller doesn't delete the branch network interfaces, so you'll need to delete them yourself. For information about how to delete network interfaces, see Delete a network interface in the Amazon EC2 User Guide.

5. In a separate terminal, shell into one of the Pods. For the remainder of this topic, this terminal is referred to as `TerminalB`. Replace *5df6f7687b-4fbjm* with the ID of one of the Pods returned in your output from the previous step.

```
kubectl exec -it -n my-namespace my-deployment-5df6f7687b-4fbjm -- /bin/bash
```

6. From the shell in `TerminalB`, confirm that the sample application works.

```
curl my-app
```

An example output is as follows.

```
<!DOCTYPE html>
<html>
<head>
```

```
<title>Welcome to nginx!</title>
[...]
```

You received the output because all Pods running the application are associated with the security group that you created. That group contains a rule that allows all traffic between all Pods that the security group is associated to. DNS traffic is allowed outbound from that security group to the cluster security group, which is associated with your nodes. The nodes are running the CoreDNS Pods, which your Pods did a name lookup to.

7. From `TerminalA`, remove the security group rules that allow DNS communication to the cluster security group from your security group. If you didn't add the DNS rules to the cluster security group in a previous step, then replace *$my_cluster_security_group_id* with the ID of the security group that you created the rules in.

```
aws ec2 revoke-security-group-ingress --group-id $my_cluster_security_group_id --
security-group-rule-ids $my_tcp_rule_id
aws ec2 revoke-security-group-ingress --group-id $my_cluster_security_group_id --
security-group-rule-ids $my_udp_rule_id
```

8. From `TerminalB`, attempt to access the application again.

```
curl my-app
```

An example output is as follows.

```
curl: (6) Could not resolve host: my-app
```

The attempt fails because the Pod is no longer able to access the CoreDNS Pods, which have the cluster security group associated to them. The cluster security group no longer has the security group rules that allow DNS communication from the security group associated to your Pod.

If you attempt to access the application using the IP addresses returned for one of the Pods in a previous step, you still receive a response because all ports are allowed between Pods that have the security group associated to them and a name lookup isn't required.

9. Once you've finished experimenting, you can remove the sample security group policy, application, and security group that you created. Run the following commands from `TerminalA`.

```
kubectl delete namespace my-namespace
```

```
aws ec2 revoke-security-group-ingress --group-id $my_pod_security_group_id --
security-group-rule-ids $my_inbound_self_rule_id
wait
sleep 45s
aws ec2 delete-security-group --group-id $my_pod_security_group_id
```

Attach multiple network interfaces to Pods with Multus

Multus CNI is a container network interface (CNI) plugin for Amazon EKS that enables attaching multiple network interfaces to a Pod. For more information, see the Multus-CNI documentation on GitHub.

In Amazon EKS, each Pod has one network interface assigned by the Amazon VPC CNI plugin. With Multus, you can create a multi-homed Pod that has multiple interfaces. This is accomplished by Multus acting as a "meta-plugin"; a CNI plugin that can call multiple other CNI plugins. AWS support for Multus comes configured with the Amazon VPC CNI plugin as the default delegate plugin.

- Amazon EKS won't be building and publishing single root I/O virtualization (SR-IOV) and Data Plane Development Kit (DPDK) CNI plugins. However, you can achieve packet acceleration by connecting directly to Amazon EC2 Elastic Network Adapters (ENA) through Multus managed host-device and `ipvlan` plugins.
- Amazon EKS is supporting Multus, which provides a generic process that enables simple chaining of additional CNI plugins. Multus and the process of chaining is supported, but AWS won't provide support for all compatible CNI plugins that can be chained, or issues that may arise in those CNI plugins that are unrelated to the chaining configuration.
- Amazon EKS is providing support and life cycle management for the Multus plugin, but isn't responsible for any IP address or additional management associated with the additional network interfaces. The IP address and management of the default network interface utilizing the Amazon VPC CNI plugin remains unchanged.
- Only the Amazon VPC CNI plugin is officially supported as the default delegate plugin. You need to modify the published Multus installation manifest to reconfigure the default delegate plugin to an alternate CNI if you choose not to use the Amazon VPC CNI plugin for primary networking.
- Multus is only supported when using the Amazon VPC CNI as the primary CNI. We do not support the Amazon VPC CNI when used for higher order interfaces, secondary or otherwise.
- To prevent the Amazon VPC CNI plugin from trying to manage additional network interfaces assigned to Pods, add the following tag to the network interface:

key

: `node.k8s.amazonaws.com/no_manage`

value

: `true`

- Multus is compatible with network policies, but the policy has to be enriched to include ports and IP addresses that may be part of additional network interfaces attached to Pods.

For an implementation walk through, see the Multus Setup Guide on GitHub.

Alternate CNI plugins for Amazon EKS clusters

The Amazon VPC CNI plugin for Kubernetes is the only CNI plugin supported by Amazon EKS with Amazon EC2 nodes. Amazon EKS supports the core capabilities of Cilium and Calico for Amazon EKS Hybrid Nodes. Amazon EKS runs upstream Kubernetes, so you can install alternate compatible CNI plugins to Amazon EC2 nodes in your cluster. If you have Fargate nodes in your cluster, the Amazon VPC CNI plugin for Kubernetes is already on your Fargate nodes. It's the only CNI plugin you can use with Fargate nodes. An attempt to install an alternate CNI plugin on Fargate nodes fails.

If you plan to use an alternate CNI plugin on Amazon EC2 nodes, we recommend that you obtain commercial support for the plugin or have the in-house expertise to troubleshoot and contribute fixes to the CNI plugin project.

Amazon EKS maintains relationships with a network of partners that offer support for alternate compatible CNI plugins. For details about the versions, qualifications, and testing performed, see the following partner documentation.

Partner	Product	Documentation
Tigera	Calico	Installation instructions
Isovalent	Cilium	Installation instructions
Juniper	Cloud-Native Contrail Networking (CN2)	Installation instructions

Partner	Product	Documentation
VMware	Antrea	Installation instructions

Amazon EKS aims to give you a wide selection of options to cover all use cases.

Alternate compatible network policy plugins

Calico is a widely adopted solution for container networking and security. Using Calico on EKS provides a fully compliant network policy enforcement for your EKS clusters. Additionally, you can opt to use Calico's networking, which conserve IP addresses from your underlying VPC. Calico Cloud enhances the features of Calico Open Source, providing advanced security and observability capabilities.

Traffic flow to and from Pods with associated security groups are not subjected to Calico network policy enforcement and are limited to Amazon VPC security group enforcement only.

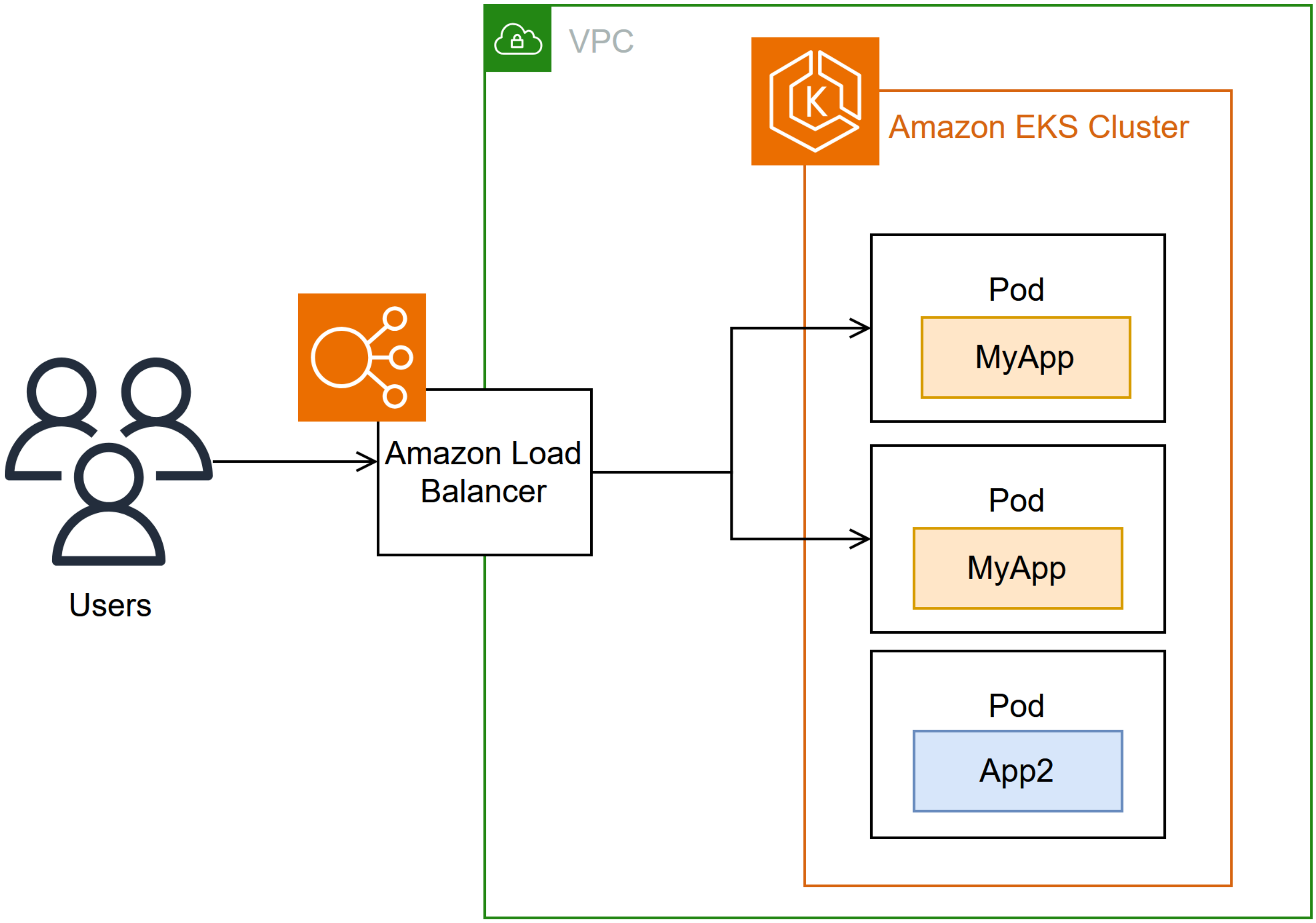
If you use Calico network policy enforcement, we recommend that you set the environment variable `ANNOTATE_POD_IP` to `true` to avoid a known issue with Kubernetes. To use this feature, you must add `patch` permission for pods to the `aws-node` ClusterRole. Note that adding patch permissions to the `aws-node` DaemonSet increases the security scope for the plugin. For more information, see ANNOTATE_POD_IP in the VPC CNI repo on GitHub.

Considerations for Amazon EKS Auto Mode

Amazon EKS Auto Mode does not support alternate CNI plugins or network policy plugins. For more information, see *EKS Auto Mode*.

Route internet traffic with AWS Load Balancer Controller

The AWS Load Balancer Controller manages AWS Elastic Load Balancers for a Kubernetes cluster. You can use the controller to expose your cluster apps to the internet. The controller provisions AWS load balancers that point to cluster Service or Ingress resources. In other words, the controller creates a single IP address or DNS name that points to multiple pods in your cluster.



The controller watches for Kubernetes Ingress or Service resources. In response, it creates the appropriate AWS Elastic Load Balancing resources. You can configure the specific behavior of the load balancers by applying annotations to the Kubernetes resources. For example, you can attach AWS security groups to load balancers using annotations.

The controller provisions the following resources:

Kubernetes `Ingress`

The LBC creates an [AWS Application Load Balancer (ALB)](#) when you create a Kubernetes `Ingress`. [Review the annotations you can apply to an Ingress resource.](#)

Kubernetes service of the `LoadBalancer` type

The LBC creates an [AWS Network Load Balancer (NLB)](#)when you create a Kubernetes service of type `LoadBalancer`. [Review the annotations you can apply to a Service resource.](#)

In the past, the Kubernetes network load balancer was used for *instance* targets, but the LBC was used for *IP* targets. With the AWS Load Balancer Controller version `2.3.0` or later, you can create NLBs using either target type. For more information about NLB target types, see Target type in the User Guide for Network Load Balancers.

The controller is an open-source project managed on GitHub.

Before deploying the controller, we recommend that you review the prerequisites and considerations in Route application and HTTP traffic with Application Load Balancers and the section called “Network load balancing”. In those topics, you will deploy a sample app that includes an AWS load balancer.

Install the controller

You can use one of the following procedures to install the AWS Load Balancer Controller:

- If you are new to Amazon EKS we recommend that you use Helm for the installation because it simplifies the AWS Load Balancer Controller installation. For more information, see the section called “Install with Helm”.
- For advanced configurations, such as clusters with restricted network access to public container registries, use Kubernetes Manifests. For more information, see the section called “Install with manifests”.

Migrate from deprecated controller versions

- If you have deprecated versions of the AWS Load Balancer Controller installed, see the section called “Migrate from deprecated”.
- Deprecated versions cannot be upgraded. They must be removed and a current version of the AWS Load Balancer Controller installed.
- Deprecated versions include:
  - AWS ALB Ingress Controller for Kubernetes ("Ingress Controller"), a predecessor to the AWS Load Balancer Controller.
  - Any `0.1.x` version of the AWS Load Balancer Controller

Legacy cloud provider

Kubernetes includes a legacy cloud provider for AWS. The legacy cloud provider is capable of provisioning AWS load balancers, similar to the AWS Load Balancer Controller. The legacy cloud provider creates Classic Load Balancers. If you do not install the AWS Load Balancer Controller, Kubernetes will default to using the legacy cloud provider. You should install the AWS Load Balancer Controller and avoid using the legacy cloud provider.

> **Important**
>
> In versions 2.5 and newer, the AWS Load Balancer Controller becomes the default controller for Kubernetes *service* resources with the `type: LoadBalancer` and makes an AWS Network Load Balancer (NLB) for each service. It does this by making a mutating webhook for services, which sets the `spec.loadBalancerClass` field to `service.k8s.aws/nlb` for new services of `type: LoadBalancer`. You can turn off this feature and revert to using the legacy Cloud Provider as the default controller, by setting the helm chart value `enableServiceMutatorWebhook` to `false`. The cluster won't provision new Classic Load Balancers for your services unless you turn off this feature. Existing Classic Load Balancers will continue to work.

Install AWS Load Balancer Controller with Helm

> **Tip**
>
> With Amazon EKS Auto Mode, you don't need to install or upgrade networking add-ons. Auto Mode includes pod networking and load balancing capabilities.
> For more information, see *EKS Auto Mode*.

This topic describes how to install the AWS Load Balancer Controller using Helm, a package manager for Kubernetes, and `eksctl`. The controller is installed with default options. For more information about the controller, including details on configuring it with annotations, see the AWS Load Balancer Controller Documentation on GitHub.

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

Prerequisites

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

- An existing Amazon EKS cluster. To deploy one, see *Get started*.
- An existing AWS Identity and Access Management (IAM) OpenID Connect (OIDC) provider for your cluster. To determine whether you already have one, or to create one, see the section called "IAM OIDC provider".
- Make sure that your Amazon VPC CNI plugin for Kubernetes, `kube-proxy`, and CoreDNS add-ons are at the minimum versions listed in Service account tokens.
- Familiarity with AWS Elastic Load Balancing. For more information, see the Elastic Load Balancing User Guide.
- Familiarity with Kubernetes service and ingress resources.
- Helm installed locally.

Step 1: Create IAM Role using `eksctl`

> **Note**
>
> You only need to create an IAM Role for the AWS Load Balancer Controller once per AWS account. Check if `AmazonEKSLoadBalancerControllerRole` exists in the IAM Console. If this role exists, skip to Step 2: Install AWS Load Balancer Controller.

> **Note**
>
> Below example is referring to the AWS Load Balancer Controller **v2.11.0** release version. For more information about all releases, see the AWS Load Balancer Controller Release Page on GitHub.

1. Download an IAM policy for the AWS Load Balancer Controller that allows it to make calls to AWS APIs on your behalf.

Example

AWS

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/v2.11.0/docs/install/iam_policy.json
```

AWS GovCloud (US)

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/v2.11.0/docs/install/iam_policy_us-gov.json
```

```
mv iam_policy_us-gov.json iam_policy.json
```

2. Create an IAM policy using the policy downloaded in the previous step.

```
aws iam create-policy \
    --policy-name AWSLoadBalancerControllerIAMPolicy \
    --policy-document file://iam_policy.json
```

> **Note**
>
> If you view the policy in the AWS Management Console, the console shows warnings for the **ELB** service, but not for the **ELB v2** service. This happens because some of the actions in the policy exist for **ELB v2**, but not for **ELB**. You can ignore the warnings for **ELB**.

3. Replace *my-cluster* with the name of your cluster, *111122223333* with your account ID, and then run the command.

```
eksctl create iamserviceaccount \
  --cluster=my-cluster \
  --namespace=kube-system \
  --name=aws-load-balancer-controller \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --attach-policy-arn=arn:aws:iam::111122223333:policy/
AWSLoadBalancerControllerIAMPolicy \
  --approve
```

Step 2: Install AWS Load Balancer Controller

1. Add the `eks-charts` Helm chart repository. AWS maintains this repository on GitHub.

```
helm repo add eks https://aws.github.io/eks-charts
```

2. Update your local repo to make sure that you have the most recent charts.

```
helm repo update eks
```

3. Install the AWS Load Balancer Controller.

   If you're deploying the controller to Amazon EC2 nodes that have restricted access to the Amazon EC2 instance metadata service (IMDS), or if you're deploying to Fargate or Amazon EKS Hybrid Nodes, then add the following flags to the `helm` command that follows:

   - `--set region=`*`region-code`*
   - `--set vpcId=`*`vpc-xxxxxxxx`*

     Replace *`my-cluster`* with the name of your cluster. In the following command, `aws-load-balancer-controller` is the Kubernetes service account that you created in a previous step.

     For more information about configuring the helm chart, see values.yaml on GitHub.

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  -n kube-system \
  --set clusterName=my-cluster \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller
```

> **Important**
>
> The deployed chart doesn't receive security updates automatically. You need to manually upgrade to a newer chart when it becomes available. When upgrading, change *`install`* to `upgrade` in the previous command.

The `helm install` command automatically installs the custom resource definitions (CRDs) for the controller. The `helm upgrade` command does not. If you use `helm upgrade,` you must manually install the CRDs. Run the following command to install the CRDs:

```
wget https://raw.githubusercontent.com/aws/eks-charts/master/stable/aws-load-balancer-controller/crds/crds.yaml
kubectl apply -f crds.yaml
```

Step 3: Verify that the controller is installed

1. Verify that the controller is installed.

   ```
   kubectl get deployment -n kube-system aws-load-balancer-controller
   ```

   An example output is as follows.

   ```
   NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
   aws-load-balancer-controller   2/2     2            2           84s
   ```

   You receive the previous output if you deployed using Helm. If you deployed using the Kubernetes manifest, you only have one replica.

2. Before using the controller to provision AWS resources, your cluster must meet specific requirements. For more information, see the section called “Application load balancing” and the section called “Network load balancing”.

Install AWS Load Balancer Controller with manifests

> **Tip**
>
> With Amazon EKS Auto Mode, you don’t need to install or upgrade networking add-ons. Auto Mode includes pod networking and load balancing capabilities.
> For more information, see *EKS Auto Mode*.

This topic describes how to install the controller by downloading and applying Kubernetes manifests. You can view the full documentation for the controller on GitHub.

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

Prerequisites

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

- An existing Amazon EKS cluster. To deploy one, see *Get started*.
- An existing AWS Identity and Access Management (IAM) OpenID Connect (OIDC) provider for your cluster. To determine whether you already have one, or to create one, see the section called "IAM OIDC provider".
- Make sure that your Amazon VPC CNI plugin for Kubernetes, `kube-proxy`, and CoreDNS add-ons are at the minimum versions listed in Service account tokens.
- Familiarity with AWS Elastic Load Balancing. For more information, see the Elastic Load Balancing User Guide.
- Familiarity with Kubernetes service and ingress resources.

Step 1: Configure IAM

> **Note**
>
> You only need to create a role for the AWS Load Balancer Controller one per AWS account. Check if `AmazonEKSLoadBalancerControllerRole` exists in the IAM Console. If this role exists, skip to Step 2: Install cert-manager.

> **Note**
>
> Below example is referring to the AWS Load Balancer Controller **v2.11.0** release version. For more inforamtion about all releases, see the AWS Load Balancer Controller Release Page on GitHub.

1. Download an IAM policy for the AWS Load Balancer Controller that allows it to make calls to AWS APIs on your behalf.

Example

AWS

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/v2.11.0/docs/install/iam_policy.json
```

AWS GovCloud (US)

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/v2.11.0/docs/install/iam_policy_us-gov.json
```

```
mv iam_policy_us-gov.json iam_policy.json
```

2. Create an IAM policy using the policy downloaded in the previous step.

```
aws iam create-policy \
    --policy-name AWSLoadBalancerControllerIAMPolicy \
    --policy-document file://iam_policy.json
```

> **Note**
>
> If you view the policy in the AWS Management Console, the console shows warnings for the **ELB** service, but not for the **ELB v2** service. This happens because some of the actions in the policy exist for **ELB v2**, but not for **ELB**. You can ignore the warnings for **ELB**.

Example

eksctl

a. Replace *my-cluster* with the name of your cluster, *111122223333* with your account ID, and then run the command.

```
eksctl create iamserviceaccount \
  --cluster=my-cluster \
  --namespace=kube-system \
  --name=aws-load-balancer-controller \
```

```
  --role-name AmazonEKSLoadBalancerControllerRole \
  --attach-policy-arn=arn:aws:iam::111122223333:policy/
AWSLoadBalancerControllerIAMPolicy \
  --approve
```

AWS CLI and kubectl

a. Retrieve your cluster's OIDC provider ID and store it in a variable.

```
oidc_id=$(aws eks describe-cluster --name my-cluster --query
 "cluster.identity.oidc.issuer" --output text | cut -d '/' -f 5)
```

b. Determine whether an IAM OIDC provider with your cluster's ID is already in your account. You need OIDC configured for both the cluster and IAM.

```
aws iam list-open-id-connect-providers | grep $oidc_id | cut -d "/" -f4
```

If output is returned, then you already have an IAM OIDC provider for your cluster. If no output is returned, then you must create an IAM OIDC provider for your cluster. For more information, see the section called "IAM OIDC provider".

c. Copy the following contents to your device. Replace *111122223333* with your account ID. Replace *region-code* with the AWS Region that your cluster is in. Replace *EXAMPLED539D4633E53DE1B71EXAMPLE* with the output returned in the previous step.

```
cat >load-balancer-role-trust-policy.json <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
```

```
                    "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:aws-
load-balancer-controller"
                }
            }
        }
    ]
}
EOF
```

d. Create the IAM role.

```
aws iam create-role \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --assume-role-policy-document file://"load-balancer-role-trust-policy.json"
```

e. Attach the required Amazon EKS managed IAM policy to the IAM role. Replace *111122223333* with your account ID.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::111122223333:policy/AWSLoadBalancerControllerIAMPolicy
 \
  --role-name AmazonEKSLoadBalancerControllerRole
```

f. Copy the following contents to your device. Replace *111122223333* with your account ID. After replacing the text, run the modified command to create the `aws-load-balancer-controller-service-account.yaml` file.

```
cat >aws-load-balancer-controller-service-account.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/name: aws-load-balancer-controller
  name: aws-load-balancer-controller
  namespace: kube-system
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/
AmazonEKSLoadBalancerControllerRole
EOF
```

g. Create the Kubernetes service account on your cluster. The Kubernetes service account named `aws-load-balancer-controller` is annotated with the IAM role that you created named *AmazonEKSLoadBalancerControllerRole*.

```
kubectl apply -f aws-load-balancer-controller-service-account.yaml
```

Step 2: Install `cert-manager`

Install `cert-manager` using one of the following methods to inject certificate configuration into the webhooks. For more information, see Getting Started in the *cert-manager Documentation*.

We recommend using the `quay.io` container registry to install `cert-manager`. If your nodes do not have access to the `quay.io` container registry, Install `cert-manager` using Amazon ECR (see below).

Example

Quay.io

a. If your nodes have access to the `quay.io` container registry, install `cert-manager` to inject certificate configuration into the webhooks.

```
kubectl apply \
    --validate=false \
    -f https://github.com/jetstack/cert-manager/releases/download/v1.13.5/cert-
manager.yaml
```

Amazon ECR

a. Install `cert-manager` using one of the following methods to inject certificate configuration into the webhooks. For more information, see Getting Started in the *cert-manager Documentation*.

b. Download the manifest.

```
curl -Lo cert-manager.yaml https://github.com/jetstack/cert-manager/releases/
download/v1.13.5/cert-manager.yaml
```

c. Pull the following images and push them to a repository that your nodes have access to. For more information on how to pull, tag, and push the images to your own repository, see the section called "Copy an image to a repository".

```
quay.io/jetstack/cert-manager-cainjector:v1.13.5
quay.io/jetstack/cert-manager-controller:v1.13.5
quay.io/jetstack/cert-manager-webhook:v1.13.5
```

d. Replace `quay.io` in the manifest for the three images with your own registry name. The following command assumes that your private repository's name is the same as the source repository. Replace *`111122223333.dkr.ecr.region-code.amazonaws.com`* with your private registry.

```
sed -i.bak -e 's|quay.io|111122223333.dkr.ecr.region-code.amazonaws.com|' ./cert-manager.yaml
```

e. Apply the manifest.

```
kubectl apply \
    --validate=false \
    -f ./cert-manager.yaml
```

Step 3: Install AWS Load Balancer Controller

1. Download the controller specification. For more information about the controller, see the documentation on GitHub.

```
curl -Lo v2_11_0_full.yaml https://github.com/kubernetes-sigs/aws-load-balancer-controller/releases/download/v2.11.0/v2_11_0_full.yaml
```

2. Make the following edits to the file.

   a. If you downloaded the `v2_11_0_full.yaml` file, run the following command to remove the `ServiceAccount` section in the manifest. If you don't remove this section, the required annotation that you made to the service account in a previous step is overwritten. Removing this section also preserves the service account that you created in a previous step if you delete the controller.

```
sed -i.bak -e '690,698d' ./v2_11_0_full.yaml
```

If you downloaded a different file version, then open the file in an editor and remove the following lines.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/name: aws-load-balancer-controller
  name: aws-load-balancer-controller
  namespace: kube-system
---
```

b. Replace `your-cluster-name` in the `Deployment` `spec` section of the file with the name of your cluster by replacing *`my-cluster`* with the name of your cluster.

```
sed -i.bak -e 's|your-cluster-name|my-cluster|' ./v2_11_0_full.yaml
```

c. If your nodes don't have access to the Amazon EKS Amazon ECR image repositories, then you need to pull the following image and push it to a repository that your nodes have access to. For more information on how to pull, tag, and push an image to your own repository, see the section called "Copy an image to a repository".

```
public.ecr.aws/eks/aws-load-balancer-controller:v2.11.0
```

Add your registry's name to the manifest. The following command assumes that your private repository's name is the same as the source repository and adds your private registry's name to the file. Replace *`111122223333.dkr.ecr.region-code.amazonaws.com`* with your registry. This line assumes that you named your private repository the same as the source repository. If not, change the `eks/aws-load-balancer-controller` text after your private registry name to your repository name.

```
sed -i.bak -e 's|public.ecr.aws/eks/aws-load-balancer-controller|
111122223333.dkr.ecr.region-code.amazonaws.com/eks/aws-load-balancer-
controller|' ./v2_11_0_full.yaml
```

d. (Required only for Fargate or Restricted IMDS)

If you're deploying the controller to Amazon EC2 nodes that have restricted access to the Amazon EC2 instance metadata service (IMDS), or if you're deploying to Fargate or Amazon EKS Hybrid Nodes, then add the `following parameters` under `- args:`.

```
[...]
spec:
      containers:
        - args:
            - --cluster-name=your-cluster-name
            - --ingress-class=alb
            - --aws-vpc-id=vpc-xxxxxxxx
            - --aws-region=region-code


[...]
```

3. Apply the file.

```
kubectl apply -f v2_11_0_full.yaml
```

4. Download the `IngressClass` and `IngressClassParams` manifest to your cluster.

```
curl -Lo v2_11_0_ingclass.yaml https://github.com/kubernetes-sigs/aws-load-balancer-controller/releases/download/v2.11.0/v2_11_0_ingclass.yaml
```

5. Apply the manifest to your cluster.

```
kubectl apply -f v2_11_0_ingclass.yaml
```

Step 4: Verify that the controller is installed

1. Verify that the controller is installed.

```
kubectl get deployment -n kube-system aws-load-balancer-controller
```

An example output is as follows.

```
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-controller   2/2     2            2           84s
```

You receive the previous output if you deployed using Helm. If you deployed using the Kubernetes manifest, you only have one replica.

2. Before using the controller to provision AWS resources, your cluster must meet specific requirements. For more information, see the section called "Application load balancing" and the section called "Network load balancing".

Migrate apps from deprecated ALB Ingress Controller

This topic describes how to migrate from deprecated controller versions. More specifically, it describes how to remove deprecated versions of the AWS Load Balancer Controller.

- Deprecated versions cannot be upgraded. You must remove them first, and then install a current version.
- Deprecated versions include:
  - AWS ALB Ingress Controller for Kubernetes ("Ingress Controller"), a predecessor to the AWS Load Balancer Controller.
  - Any `0.1.x` version of the AWS Load Balancer Controller

Remove the deprecated controller version

> **Note**
>
> You may have installed the deprecated version using Helm or manually with Kubernetes manifests. Complete the procedure using the tool that you originally installed it with.

1. If you installed the `incubator/aws-alb-ingress-controller` Helm chart, uninstall it.

```
helm delete aws-alb-ingress-controller -n kube-system
```

2. If you have version `0.1.x` of the `eks-charts/aws-load-balancer-controller` chart installed, uninstall it. The upgrade from `0.1.x` to version `1.0.0` doesn't work due to incompatibility with the webhook API version.

```
helm delete aws-load-balancer-controller -n kube-system
```

3. Check to see if the controller is currently installed.

```
kubectl get deployment -n kube-system alb-ingress-controller
```

This is the output if the controller isn't installed.

```
Error from server (NotFound): deployments.apps "alb-ingress-controller" not found
```

This is the output if the controller is installed.

```
NAME                   READY UP-TO-DATE AVAILABLE AGE
alb-ingress-controller 1/1   1          1         122d
```

4. Enter the following commands to remove the controller.

```
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.8/docs/examples/alb-ingress-controller.yaml
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.8/docs/examples/rbac-role.yaml
```

Migrate to AWS Load Balancer Controller

To migrate from the ALB Ingress Controller for Kubernetes to the AWS Load Balancer Controller, you need to:

1. Remove the ALB Ingress Controller (see above).
2. Install the AWS Load Balancer Controller.
3. Add an additional policy to the IAM Role used by the AWS Load Balancer Controller. This policy permits the LBC to manage resources created by the ALB Ingress Controller for Kubernetes.
4. Download the IAM policy. This policy permits the AWS Load Balancer Controller to manage resources created by the ALB Ingress Controller for Kubernetes. You can also view the policy.

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/v2.11.0/docs/install/iam_policy_v1_to_v2_additional.json
```

5. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-West) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:`.

```
sed -i.bak -e 's|arn:aws:|arn:aws-us-gov:|' iam_policy_v1_to_v2_additional.json
```

6. Create the IAM policy and note the ARN that is returned.

```
aws iam create-policy \
  --policy-name AWSLoadBalancerControllerAdditionalIAMPolicy \
  --policy-document file://iam_policy_v1_to_v2_additional.json
```

7. Attach the IAM policy to the IAM role used by the AWS Load Balancer Controller. Replace *your-role-name* with the name of the role, such as `AmazonEKSLoadBalancerControllerRole`.

   If you created the role using `eksctl`, then to find the role name that was created, open the AWS CloudFormation console and select the **eksctl-*my-cluster*-addon-iamserviceaccount-kube-system-aws-load-balancer-controller** stack. Select the **Resources** tab. The role name is in the **Physical ID** column.

```
aws iam attach-role-policy \
  --role-name your-role-name \
  --policy-arn arn:aws:iam::111122223333:policy/
AWSLoadBalancerControllerAdditionalIAMPolicy
```

Manage CoreDNS for DNS in Amazon EKS clusters

> **Tip**
>
> With Amazon EKS Auto Mode, you don't need to install or upgrade networking add-ons. Auto Mode includes pod networking and load balancing capabilities.
> For more information, see *EKS Auto Mode*.

CoreDNS is a flexible, extensible DNS server that can serve as the Kubernetes cluster DNS. When you launch an Amazon EKS cluster with at least one node, two replicas of the CoreDNS image are deployed by default, regardless of the number of nodes deployed in your cluster. The CoreDNS Pods provide name resolution for all Pods in the cluster. The CoreDNS Pods can be deployed to Fargate nodes if your cluster includes a Fargate Profile with a namespace that matches the namespace for the CoreDNS `deployment`. For more information on Fargate Profiles, see the section called "Define profiles". For more information about CoreDNS, see Using CoreDNS for Service Discovery in the Kubernetes documentation.

CoreDNS versions

The following table lists the latest version of the Amazon EKS add-on type for each Kubernetes version.

Kubernetes version	CoreDNS version
1.32	v1.11.4-eksbuild.2
1.31	v1.11.4-eksbuild.2
1.30	v1.11.4-eksbuild.2
1.29	v1.11.4-eksbuild.2
1.28	v1.10.1-eksbuild.18
1.27	v1.10.1-eksbuild.18
1.26	v1.9.3-eksbuild.22
1.25	v1.9.3-eksbuild.22
1.24	v1.9.3-eksbuild.22

> **Important**
>
> If you're self-managing this add-on, the versions in the table might not be the same as the available self-managed versions. For more information about updating the self-managed type of this add-on, see the section called "Update (self-managed)".

Important CoreDNS upgrade considerations

- To improve the stability and availability of the CoreDNS Deployment, versions `v1.9.3-eksbuild.6` and later and `v1.10.1-eksbuild.3` are deployed with a `PodDisruptionBudget`. If you've deployed an existing `PodDisruptionBudget`, your upgrade to these versions might fail. If the upgrade fails, completing one of the following tasks should resolve the issue:

  - When doing the upgrade of the Amazon EKS add-on, choose to override the existing settings as your conflict resolution option. If you've made other custom settings to the Deployment, make sure to back up your settings before upgrading so that you can reapply your other custom settings after the upgrade.
  - Remove your existing `PodDisruptionBudget` and try the upgrade again.
- In EKS add-on versions `v1.9.3-eksbuild.3` and later and `v1.10.1-eksbuild.6` and later, the CoreDNS Deployment sets the `readinessProbe` to use the `/ready` endpoint. This endpoint is enabled in the `Corefile` configuration file for CoreDNS.

  If you use a custom `Corefile`, you must add the `ready` plugin to the config, so that the `/ready` endpoint is active in CoreDNS for the probe to use.
- In EKS add-on versions `v1.9.3-eksbuild.7` and later and `v1.10.1-eksbuild.4` and later, you can change the `PodDisruptionBudget`. You can edit the add-on and change these settings in the **Optional configuration settings** using the fields in the following example. This example shows the default `PodDisruptionBudget`.

```
{
    "podDisruptionBudget": {
        "enabled": true,
        "maxUnavailable": 1
        }
}
```

  You can set `maxUnavailable` or `minAvailable`, but you can't set both in a single `PodDisruptionBudget`. For more information about `PodDisruptionBudgets`, see [Specifying a PodDisruptionBudget] in the *Kubernetes documentation*.

  Note that if you set `enabled` to `false`, the `PodDisruptionBudget` isn't removed. After you set this field to `false`, you must delete the `PodDisruptionBudget` object. Similarly, if you edit the add-on to use an older version of the add-on (downgrade the add-on) after upgrading to a version with a `PodDisruptionBudget`, the `PodDisruptionBudget` isn't removed. To delete the `PodDisruptionBudget`, you can run the following command:

```
kubectl delete poddisruptionbudget coredns -n kube-system
```

- In EKS add-on versions `v1.10.1-eksbuild.5` and later, change the default toleration from `node-role.kubernetes.io/master:NoSchedule` to `node-role.kubernetes.io/`

`control-plane:NoSchedule` to comply with KEP 2067. For more information about KEP 2067, see KEP-2067: Rename the kubeadm "master" label and taint in the *Kubernetes Enhancement Proposals (KEPs)* on GitHub.

In EKS add-on versions `v1.8.7-eksbuild.8` and later and `v1.9.3-eksbuild.9` and later, both tolerations are set to be compatible with every Kubernetes version.

- In EKS add-on versions `v1.9.3-eksbuild.11` and `v1.10.1-eksbuild.7` and later, the CoreDNS Deployment sets a default value for `topologySpreadConstraints`. The default value ensures that the CoreDNS Pods are spread across the Availability Zones if there are nodes in multiple Availability Zones available. You can set a custom value that will be used instead of the default value. The default value follows:

```
topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: ScheduleAnyway
    labelSelector:
      matchLabels:
        k8s-app: kube-dns
```

CoreDNS `v1.11` upgrade considerations

- In EKS add-on versions `v1.11.1-eksbuild.4` and later, the container image is based on a minimal base image maintained by Amazon EKS Distro, which contains minimal packages and doesn't have shells. For more information, see Amazon EKS Distro. The usage and troubleshooting of the CoreDNS image remains the same.

Create the CoreDNS Amazon EKS add-on

Create the CoreDNS Amazon EKS add-on. You must have a cluster before you create the add-on. For more information, see the section called "Create a cluster".

1. See which version of the add-on is installed on your cluster.

```
kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -
d : -f 3
```

An example output is as follows.

```
v1.10.1-eksbuild.13
```

2. See which type of the add-on is installed on your cluster. Depending on the tool that you created your cluster with, you might not currently have the Amazon EKS add-on type installed on your cluster. Replace *my-cluster* with the name of your cluster.

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
 addon.addonVersion --output text
```

   If a version number is returned, you have the Amazon EKS type of the add-on installed on your cluster and don't need to complete the remaining steps in this procedure. If an error is returned, you don't have the Amazon EKS type of the add-on installed on your cluster. Complete the remaining steps of this procedure to install it.

3. Save the configuration of your currently installed add-on.

```
kubectl get deployment coredns -n kube-system -o yaml > aws-k8s-coredns-old.yaml
```

4. Create the add-on using the AWS CLI. If you want to use the AWS Management Console or `eksctl` to create the add-on, see the section called "Create an add-on" and specify `coredns` for the add-on name. Copy the command that follows to your device. Make the following modifications to the command, as needed, and then run the modified command.

   - Replace *my-cluster* with the name of your cluster.
   - Replace *v1.11.3-eksbuild.1* with the latest version listed in the latest version table for your cluster version.

```
aws eks create-addon --cluster-name my-cluster --addon-name coredns --addon-version
 v1.11.3-eksbuild.1
```

   If you've applied custom settings to your current add-on that conflict with the default settings of the Amazon EKS add-on, creation might fail. If creation fails, you receive an error that can help you resolve the issue. Alternatively, you can add `--resolve-conflicts OVERWRITE` to the previous command. This allows the add-on to overwrite any existing custom settings. Once you've created the add-on, you can update it with your custom settings.

5. Confirm that the latest version of the add-on for your cluster's Kubernetes version was added to your cluster. Replace *my-cluster* with the name of your cluster.

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
 addon.addonVersion --output text
```

It might take several seconds for add-on creation to complete.

An example output is as follows.

```
v1.11.3-eksbuild.1
```

6. If you made custom settings to your original add-on, before you created the Amazon EKS add-on, use the configuration that you saved in a previous step to update the Amazon EKS add-on with your custom settings. For instructions to update the add-on, see the section called "Update (EKS add-on)".

Update the CoreDNS Amazon EKS add-on

Update the Amazon EKS type of the add-on. If you haven't added the Amazon EKS add-on to your cluster, either add it or see the section called "Update (self-managed)".

Before you begin, review the upgrade considerations. For more information, see the section called "Important CoreDNS upgrade considerations".

1. See which version of the add-on is installed on your cluster. Replace *my-cluster* with your cluster name.

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
 "addon.addonVersion" --output text
```

An example output is as follows.

```
v1.10.1-eksbuild.13
```

If the version returned is the same as the version for your cluster's Kubernetes version in the latest version table, then you already have the latest version installed on your cluster and don't need to complete the rest of this procedure. If you receive an error, instead of a version number in your output, then you don't have the Amazon EKS type of the add-on installed on your cluster. You need to create the add-on before you can update it with this procedure.

2. Save the configuration of your currently installed add-on.

```
kubectl get deployment coredns -n kube-system -o yaml > aws-k8s-coredns-old.yaml
```

3. Update your add-on using the AWS CLI. If you want to use the AWS Management Console or `eksctl` to update the add-on, see the section called "Update an add-on". Copy the command that follows to your device. Make the following modifications to the command, as needed, and then run the modified command.

   - Replace *my-cluster* with the name of your cluster.
   - Replace *v1.11.3-eksbuild.1* with the latest version listed in the latest version table for your cluster version.
   - The `--resolve-conflicts`*PRESERVE* option preserves existing configuration values for the add-on. If you've set custom values for add-on settings, and you don't use this option, Amazon EKS overwrites your values with its default values. If you use this option, then we recommend testing any field and value changes on a non-production cluster before updating the add-on on your production cluster. If you change this value to `OVERWRITE`, all settings are changed to Amazon EKS default values. If you've set custom values for any settings, they might be overwritten with Amazon EKS default values. If you change this value to `none`, Amazon EKS doesn't change the value of any settings, but the update might fail. If the update fails, you receive an error message to help you resolve the conflict.
   - If you're not updating a configuration setting, remove `--configuration-values '{`*"replicaCount":3*`}'` from the command. If you're updating a configuration setting, replace *"replicaCount":3* with the setting that you want to set. In this example, the number of replicas of CoreDNS is set to 3. The value that you specify must be valid for the configuration schema. If you don't know the configuration schema, run `aws eks describe-addon-configuration --addon-name coredns --addon-version` *v1.11.3-eksbuild.1* , replacing *v1.11.3-eksbuild.1* with the version number of the add-on that you want to see the configuration for. The schema is returned in the output. If you have any existing custom configuration, want to remove it all, and set the values for all settings back to Amazon EKS defaults, remove *"replicaCount":3* from the command, so that you have empty `{}`. For more information about CoreDNS settings, see Customizing DNS Service in the Kubernetes documentation.

   ```
   aws eks update-addon --cluster-name my-cluster --addon-name coredns --addon-version
    v1.11.3-eksbuild.1 \
       --resolve-conflicts PRESERVE --configuration-values '{"replicaCount":3}'
   ```

It might take several seconds for the update to complete.

4. Confirm that the add-on version was updated. Replace *my-cluster* with the name of your cluster.

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

It might take several seconds for the update to complete.

An example output is as follows.

```
{
    "addon": {
        "addonName": "coredns",
        "clusterName": "my-cluster",
        "status": "ACTIVE",
        "addonVersion": "v1.11.3-eksbuild.1",
        "health": {
            "issues": []
        },
        "addonArn": "arn:aws:eks:region:111122223333:addon/my-cluster/coredns/
d2c34f06-1111-2222-1eb0-24f64ce37fa4",
        "createdAt": "2023-03-01T16:41:32.442000+00:00",
        "modifiedAt": "2023-03-01T18:16:54.332000+00:00",
        "tags": {},
        "configurationValues": "{\"replicaCount\":3}"
    }
}
```

Update the CoreDNS Amazon EKS self-managed add-on

Important

We recommend adding the Amazon EKS type of the add-on to your cluster instead of using the self-managed type of the add-on. If you're not familiar with the difference between the types, see the section called "Amazon EKS add-ons". For more information about adding an Amazon EKS add-on to your cluster, see the section called "Create an add-on". If you're unable to use the Amazon EKS add-on, we encourage you to submit an issue about why you can't to the Containers roadmap GitHub repository.

Before you begin, review the upgrade considerations. For more information, see the section called "Important CoreDNS upgrade considerations".

1. Confirm that you have the self-managed type of the add-on installed on your cluster. Replace *my-cluster* with the name of your cluster.

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
    addon.addonVersion --output text
   ```

   If an error message is returned, you have the self-managed type of the add-on installed on your cluster. Complete the remaining steps in this procedure. If a version number is returned, you have the Amazon EKS type of the add-on installed on your cluster. To update the Amazon EKS type of the add-on, use the procedure in Update the CoreDNS Amazon EKS add-on, rather than using this procedure. If you're not familiar with the differences between the add-on types, see the section called "Amazon EKS add-ons".

2. See which version of the container image is currently installed on your cluster.

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

   An example output is as follows.

   ```
   v1.8.7-eksbuild.2
   ```

3. If your current CoreDNS version is `v1.5.0` or later, but earlier than the version listed in the CoreDNS versions table, then skip this step. If your current version is earlier than `1.5.0`, then you need to modify the `ConfigMap` for CoreDNS to use the forward add-on, rather than the proxy add-on.

   a. Open the `ConfigMap` with the following command.

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

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

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

4. If you originally deployed your cluster on Kubernetes `1.17` or earlier, then you may need to remove a discontinued line from your CoreDNS manifest.

> **Important**
>
> You must complete this step before updating to CoreDNS version `1.7.0`, but it's recommended that you complete this step even if you're updating to an earlier version.

a. Check to see if your CoreDNS manifest has the line.

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

If no output is returned, your manifest doesn't have the line and you can skip to the next step to update CoreDNS. If output is returned, then you need to remove the line.

b. Edit the `ConfigMap` with the following command, removing the line in the file that has the word `upstream` in it. Do not change anything else in the file. Once the line is removed, save the changes.

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

5. Retrieve your current CoreDNS image version:

```
kubectl describe deployment coredns -n kube-system | grep Image
```

An example output is as follows.

```
602401143452.dkr.ecr.region-code.amazonaws.com/eks/coredns:v1.8.7-eksbuild.2
```

6. If you're updating to CoreDNS `1.8.3` or later, then you need to add the `endpointslices` permission to the `system:coredns` Kubernetes `clusterrole`.

```
kubectl edit clusterrole system:coredns -n kube-system
```

Add the following lines under the existing permissions lines in the `rules` section of the file.

```
[...]
- apiGroups:
  - discovery.k8s.io
```

```
  resources:
  - endpointslices
  verbs:
  - list
  - watch
[...]
```

7. Update the CoreDNS add-on by replacing *602401143452* and *region-code* with the values from the output returned in a previous step. Replace *v1.11.3-eksbuild.1* with the CoreDNS version listed in the latest versions table for your Kubernetes version.

```
kubectl set image deployment.apps/coredns -n kube-system
 coredns=602401143452.dkr.ecr.region-code.amazonaws.com/eks/coredns:v1.11.3-
eksbuild.1
```

   An example output is as follows.

```
deployment.apps/coredns image updated
```

8. Check the container image version again to confirm that it was updated to the version that you specified in the previous step.

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

   An example output is as follows.

```
v1.11.3-eksbuild.1
```

Scale CoreDNS Pods for high DNS traffic

When you launch an Amazon EKS cluster with at least one node, a Deployment of two replicas of the CoreDNS image are deployed by default, regardless of the number of nodes deployed in your cluster. The CoreDNS Pods provide name resolution for all Pods in the cluster. Applications use name resolution to connect to pods and services in the cluster as well as connecting to services outside the cluster. As the number of requests for name resolution (queries) from pods increase, the CoreDNS pods can get overwhelmed and slow down, and reject requests that the pods can't handle.

To handle the increased load on the CoreDNS pods, consider an autoscaling system for CoreDNS. Amazon EKS can manage the autoscaling of the CoreDNS Deployment in the EKS Add-on version of CoreDNS. This CoreDNS autoscaler continuously monitors the cluster state, including the number of nodes and CPU cores. Based on that information, the controller will dynamically adapt the number of replicas of the CoreDNS deployment in an EKS cluster. This feature works for CoreDNS `v1.9` and EKS release version `1.25` and later. For more information about which versions are compatible with CoreDNS Autoscaling, see the following section.

We recommend using this feature in conjunction with other EKS Cluster Autoscaling best practices to improve overall application availability and cluster scalability.

Prerequisites

For Amazon EKS to scale your CoreDNS deployment, there are three prerequisites:

- You must be using the *EKS Add-on* version of CoreDNS.
- Your cluster must be running at least the minimum cluster versions and platform versions.
- Your cluster must be running at least the minimum version of the EKS Add-on of CoreDNS.

Minimum cluster version

Autoscaling of CoreDNS is done by a new component in the cluster control plane, managed by Amazon EKS. Because of this, you must upgrade your cluster to an EKS release that supports the minimum platform version that has the new component.

A new Amazon EKS cluster. To deploy one, see *Get started*. The cluster must be Kubernetes version `1.25` or later. The cluster must be running one of the Kubernetes versions and platform versions listed in the following table or a later version. Note that any Kubernetes and platform versions later than those listed are also supported. You can check your current Kubernetes version by replacing *`my-cluster`* in the following command with the name of your cluster and then running the modified command:

```
aws eks describe-cluster
              --name my-cluster --query cluster.version --output
              text
```

Kubernetes version	Platform version
1.29.3	eks.7
1.28.8	eks.13
1.27.12	eks.17
1.26.15	eks.18
1.25.16	eks.19

> **Note**
>
> Every platform version of later Kubernetes versions are also supported, for example Kubernetes version `1.30` from `eks.1` and on.

Minimum EKS Add-on version

Kubernetes version	1.29	1.28	1.27	1.26	1.25
	v1.11.1-eksbuild.9	v1.10.1-eksbuild.11	v1.10.1-eksbuild.11	v1.9.3-eksbuild.15	v1.9.3-eksbuild.15

Configuring CoreDNS autoscaling in the AWS Management Console

1. Ensure that your cluster is at or above the minimum cluster version.

   Amazon EKS upgrades clusters between platform versions of the same Kubernetes version automatically, and you can't start this process yourself. Instead, you can upgrade your cluster to the next Kubernetes version, and the cluster will be upgraded to that K8s version and the latest platform version. For example, if you upgrade from `1.25` to `1.26`, the cluster will upgrade to `1.26.15 eks.18`.

New Kubernetes versions sometimes introduce significant changes. Therefore, we recommend that you test the behavior of your applications by using a separate cluster of the new Kubernetes version before you update your production clusters.

To upgrade a cluster to a new Kubernetes version, follow the procedure in Update existing cluster to new Kubernetes version.

2. Ensure that you have the EKS Add-on for CoreDNS, not the self-managed CoreDNS Deployment.

   Depending on the tool that you created your cluster with, you might not currently have the Amazon EKS add-on type installed on your cluster. To see which type of the add-on is installed on your cluster, you can run the following command. Replace `my-cluster` with the name of your cluster.

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
 addon.addonVersion --output text
```

   If a version number is returned, you have the Amazon EKS type of the add-on installed on your cluster and you can continue with the next step. If an error is returned, you don't have the Amazon EKS type of the add-on installed on your cluster. Complete the remaining steps of the procedure Create the CoreDNS Amazon EKS add-on to replace the self-managed version with the Amazon EKS add-on.

3. Ensure that your EKS Add-on for CoreDNS is at a version the same or higher than the minimum EKS Add-on version.

   See which version of the add-on is installed on your cluster. You can check in the AWS Management Console or run the following command:

```
kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -
d : -f 3
```

   An example output is as follows.

```
v1.10.1-eksbuild.13
```

   Compare this version with the minimum EKS Add-on version in the previous section. If needed, upgrade the EKS Add-on to a higher version by following the procedure Update the CoreDNS Amazon EKS add-on.

4. Add the autoscaling configuration to the **Optional configuration settings** of the EKS Add-on.

   a. Open the [Amazon EKS console](https://console.aws.amazon.com/eks/home#/clusters).

   b. In the left navigation pane, select **Clusters**, and then select the name of the cluster that you want to configure the add-on for.

   c. Choose the **Add-ons** tab.

   d. Select the box in the top right of the CoreDNS add-on box and then choose **Edit**.

   e. On the **Configure CoreDNS** page:

      i. Select the **Version** that you'd like to use. We recommend that you keep the same version as the previous step, and update the version and configuration in separate actions.

      ii. Expand the **Optional configuration settings**.

      iii. Enter the JSON key `"autoscaling":` and value of a nested JSON object with a key `"enabled":` and value `true` in **Configuration values**. The resulting text must be a valid JSON object. If this key and value are the only data in the text box, surround the key and value with curly braces `{ }`. The following example shows autoscaling is enabled:

```
{
  "autoScaling": {
    "enabled": true
  }
}
```

      iv. (Optional) You can provide minimum and maximum values that autoscaling can scale the number of CoreDNS pods to.

      The following example shows autoscaling is enabled and all of the optional keys have values. We recommend that the minimum number of CoreDNS pods is always greater than 2 to provide resilience for the DNS service in the cluster.

```
{
  "autoScaling": {
    "enabled": true,
    "minReplicas": 2,
    "maxReplicas": 10
  }
}
```

   f. To apply the new configuration by replacing the CoreDNS pods, choose **Save changes**.

Amazon EKS applies changes to the EKS Add-ons by using a *rollout* of the Kubernetes Deployment for CoreDNS. You can track the status of the rollout in the **Update history** of the add-on in the AWS Management Console and with `kubectl rollout status deployment/coredns --namespace kube-system`.

`kubectl rollout` has the following commands:

```
kubectl rollout

history  -- View rollout history
pause    -- Mark the provided resource as paused
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
undo     -- Undo a previous rollout
```

If the rollout takes too long, Amazon EKS will undo the rollout, and a message with the type of **Addon Update** and a status of **Failed** will be added to the **Update history** of the add-on. To investigate any issues, start from the history of the rollout, and run `kubectl logs` on a CoreDNS pod to see the logs of CoreDNS.

5. If the new entry in the **Update history** has a status of **Successful**, then the rollout has completed and the add-on is using the new configuration in all of the CoreDNS pods. As you change the number of nodes and CPU cores of nodes in the cluster, Amazon EKS scales the number of replicas of the CoreDNS deployment.

Configuring CoreDNS autoscaling in the AWS Command Line Interface

1. Ensure that your cluster is at or above the minimum cluster version.

   Amazon EKS upgrades clusters between platform versions of the same Kubernetes version automatically, and you can’t start this process yourself. Instead, you can upgrade your cluster to the next Kubernetes version, and the cluster will be upgraded to that K8s version and the latest platform version. For example, if you upgrade from `1.25` to `1.26`, the cluster will upgrade to `1.26.15 eks.18`.

   New Kubernetes versions sometimes introduce significant changes. Therefore, we recommend that you test the behavior of your applications by using a separate cluster of the new Kubernetes version before you update your production clusters.

To upgrade a cluster to a new Kubernetes version, follow the procedure in Update existing cluster to new Kubernetes version.

2. Ensure that you have the EKS Add-on for CoreDNS, not the self-managed CoreDNS Deployment.

   Depending on the tool that you created your cluster with, you might not currently have the Amazon EKS add-on type installed on your cluster. To see which type of the add-on is installed on your cluster, you can run the following command. Replace `my-cluster` with the name of your cluster.

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
    addon.addonVersion --output text
   ```

   If a version number is returned, you have the Amazon EKS type of the add-on installed on your cluster. If an error is returned, you don't have the Amazon EKS type of the add-on installed on your cluster. Complete the remaining steps of the procedure Create the CoreDNS Amazon EKS add-on to replace the self-managed version with the Amazon EKS add-on.

3. Ensure that your EKS Add-on for CoreDNS is at a version the same or higher than the minimum EKS Add-on version.

   See which version of the add-on is installed on your cluster. You can check in the AWS Management Console or run the following command:

   ```
   kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -
   d : -f 3
   ```

   An example output is as follows.

   ```
   v1.10.1-eksbuild.13
   ```

   Compare this version with the minimum EKS Add-on version in the previous section. If needed, upgrade the EKS Add-on to a higher version by following the procedure Update the CoreDNS Amazon EKS add-on.

4. Add the autoscaling configuration to the **Optional configuration settings** of the EKS Add-on.

   Run the following AWS CLI command. Replace `my-cluster` with the name of your cluster and the IAM role ARN with the role that you are using.

```
aws eks update-addon --cluster-name my-cluster --addon-name coredns \
    --resolve-conflicts PRESERVE --configuration-values '{"autoScaling":
{"enabled":true}}'
```

Amazon EKS applies changes to the EKS Add-ons by using a *rollout* of the Kubernetes Deployment for CoreDNS. You can track the status of the rollout in the **Update history** of the add-on in the AWS Management Console and with `kubectl rollout status deployment/coredns --namespace kube-system`.

`kubectl rollout` has the following commands:

```
kubectl rollout

history  -- View rollout history
pause    -- Mark the provided resource as paused
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
undo     -- Undo a previous rollout
```

If the rollout takes too long, Amazon EKS will undo the rollout, and a message with the type of **Addon Update** and a status of **Failed** will be added to the **Update history** of the add-on. To investigate any issues, start from the history of the rollout, and run `kubectl logs` on a CoreDNS pod to see the logs of CoreDNS.

5. (Optional) You can provide minimum and maximum values that autoscaling can scale the number of CoreDNS pods to.

   The following example shows autoscaling is enabled and all of the optional keys have values. We recommend that the minimum number of CoreDNS pods is always greater than 2 to provide resilience for the DNS service in the cluster.

```
aws eks update-addon --cluster-name my-cluster --addon-name coredns \
    --resolve-conflicts PRESERVE --configuration-values '{"autoScaling":
{"enabled":true,"minReplicas":2,"maxReplicas":10}}'
```

6. Check the status of the update to the add-on by running the following command:

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns \
```

If you see this line: `"status": "ACTIVE"`, then the rollout has completed and the add-on is using the new configuration in all of the CoreDNS pods. As you change the number of nodes and CPU cores of nodes in the cluster, Amazon EKS scales the number of replicas of the CoreDNS deployment.

Monitor Kubernetes DNS resolution with CoreDNS metrics

CoreDNS as an EKS add-on exposes the metrics from CoreDNS on port `9153` in the Prometheus format in the `kube-dns` service. You can use Prometheus, the Amazon CloudWatch agent, or any other compatible system to scrape (collect) these metrics.

For an example *scrape configuration* that is compatible with both Prometheus and the CloudWatch agent, see CloudWatch agent configuration for Prometheus in the *Amazon CloudWatch User Guide*.

Manage `kube-proxy` in Amazon EKS clusters

Tip

With Amazon EKS Auto Mode, you don't need to install or upgrade networking add-ons. Auto Mode includes pod networking and load balancing capabilities.
For more information, see *EKS Auto Mode*.

We recommend adding the Amazon EKS type of the add-on to your cluster instead of using the self-managed type of the add-on. If you're not familiar with the difference between the types, see the section called "Amazon EKS add-ons". For more information about adding an Amazon EKS add-on to your cluster, see the section called "Create an add-on". If you're unable to use the Amazon EKS add-on, we encourage you to submit an issue about why you can't to the Containers roadmap GitHub repository.

The `kube-proxy` add-on is deployed on each Amazon EC2 node in your Amazon EKS cluster. It maintains network rules on your nodes and enables network communication to your Pods. The add-on isn't deployed to Fargate nodes in your cluster. For more information, see kube-proxy in the Kubernetes documentation.

Install as Amazon EKS Add-on

`kube-proxy` versions

The following table lists the latest version of the Amazon EKS add-on type for each Kubernetes version.

Kubernetes version	`kube-proxy` version
1.32	v1.32.0-eksbuild.2
1.31	v1.31.3-eksbuild.2
1.30	v1.30.9-eksbuild.3
1.29	v1.29.13-eksbuild.3
1.28	v1.28.15-eksbuild.9
1.27	v1.27.16-eksbuild.19
1.26	v1.26.15-eksbuild.24
1.25	v1.25.16-eksbuild.27
1.24	v1.24.17-eksbuild.19

> **Note**
>
> An earlier version of the documentation was incorrect. `kube-proxy` versions `v1.28.5`, `v1.27.9`, and `v1.26.12` aren't available.
> If you're self-managing this add-on, the versions in the table might not be the same as the available self-managed versions.

`kube-proxy` container image migration

There are two types of the `kube-proxy` container image available for each Amazon EKS cluster version:

- **Default –** This image type is based on a Debian-based Docker image that is maintained by the Kubernetes upstream community.
- **Minimal –** This image type is based on a minimal base image maintained by Amazon EKS Distro, which contains minimal packages and doesn't have shells. For more information, see Amazon EKS Distro.

The following table lists the latest available self-managed `kube-proxy` container image version for each Amazon EKS cluster version.

Version	kube-proxy (default type)	kube-proxy (minimal type)
1.32	Only minimal type is available	v1.32.0-minimal-eksbuild.2
1.31	Only minimal type is available	v1.31.3-minimal-eksbuild.2
1.30	Only minimal type is available	v1.30.9-minimal-eksbuild.3
1.29	Only minimal type is available	v1.29.13-minimal-eksbuild.3
1.28	Only minimal type is available	v1.28.15-minimal-eksbuild.9
1.27	Only minimal type is available	v1.27.16-minimal-eksbuild.19
1.26	Only minimal type is available	v1.26.15-minimal-eksbuild.24
1.25	Only minimal type is available	v1.25.16-minimal-eksbuild.27
1.24	v1.24.10-eksbuild.2	v1.24.17-minimal-eksbuild.19

- The default image type isn't available for Kubernetes version `1.25` and later. You must use the minimal image type.
- When you update an Amazon EKS add-on type, you specify a valid Amazon EKS add-on version, which might not be a version listed in this table. This is because Amazon EKS add-on versions don't always match container image versions specified when updating the self-managed type of this add-on. When you update the self-managed type of this add-on, you specify a valid container image version listed in this table.

Update the Kubernetes kube-proxy self-managed add-on

> **Important**
>
> We recommend adding the Amazon EKS type of the add-on to your cluster instead of using the self-managed type of the add-on. If you're not familiar with the difference between the types, see the section called "Amazon EKS add-ons". For more information about adding an Amazon EKS add-on to your cluster, see the section called "Create an add-on". If you're unable to use the Amazon EKS add-on, we encourage you to submit an issue about why you can't to the Containers roadmap GitHub repository.

Prerequisites

- An existing Amazon EKS cluster. To deploy one, see *Get started*.

Considerations

- `Kube-proxy` on an Amazon EKS cluster has the same compatibility and skew policy as Kubernetes. Learn how to Verifying Amazon EKS add-on version compatibility with a cluster.

1. Confirm that you have the self-managed type of the add-on installed on your cluster. Replace *my-cluster* with the name of your cluster.

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name kube-proxy --query
    addon.addonVersion --output text
   ```

   If an error message is returned, you have the self-managed type of the add-on installed on your cluster. The remaining steps in this topic are for updating the self-managed type of the add-on. If a version number is returned, you have the Amazon EKS type of the add-on installed on your cluster. To update it, use the procedure in Updating an Amazon EKS add-on, rather than using the procedure in this topic. If you're not familiar with the differences between the add-on types, see the section called "Amazon EKS add-ons".

2. See which version of the container image is currently installed on your cluster.

   ```
   kubectl describe daemonset kube-proxy -n kube-system | grep Image
   ```

   An example output is as follows.

```
Image:    602401143452.dkr.ecr.region-code.amazonaws.com/eks/kube-proxy:v1.29.1-
eksbuild.2
```

In the example output, *v1.29.1-eksbuild.2* is the version installed on the cluster.

3. Update the `kube-proxy` add-on by replacing *602401143452* and *region-code* with the values from your output in the previous step. Replace *v1.30.6-eksbuild.3* with the `kube-proxy` version listed in the [Latest available self-managed kube-proxy container image version for each Amazon EKS cluster version](#) table.

   > **Important**
   >
   > The manifests for each image type are different and not compatible between the *default* or *minimal* image types. You must use the same image type as the previous image, so that the entrypoint and arguments match.

   ```
   kubectl set image daemonset.apps/kube-proxy -n kube-system kube-
   proxy=602401143452.dkr.ecr.region-code.amazonaws.com/eks/kube-proxy:v1.30.6-
   eksbuild.3
   ```

   An example output is as follows.

   ```
   daemonset.apps/kube-proxy image updated
   ```

4. Confirm that the new version is now installed on your cluster.

   ```
   kubectl describe daemonset kube-proxy -n kube-system | grep Image | cut -d ":" -f 3
   ```

   An example output is as follows.

   ```
   v1.30.0-eksbuild.3
   ```

5. If you're using `x86` and `Arm` nodes in the same cluster and your cluster was deployed before August 17, 2020. Then, edit your `kube-proxy` manifest to include a node selector for multiple hardware architectures with the following command. This is a one-time operation. After you've added the selector to your manifest, you don't need to add it each time you

update the add-on. If your cluster was deployed on or after August 17, 2020, then `kube-proxy` is already multi-architecture capable.

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

Add the following node selector to the file in the editor and then save the file. For an example of where to include this text in the editor, see the [CNI manifest] file on GitHub. This enables Kubernetes to pull the correct hardware image based on the node's hardware architecture.

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

6. If your cluster was originally created with Kubernetes version `1.14` or later, then you can skip this step because `kube-proxy` already includes this `Affinity Rule`. If you originally created an Amazon EKS cluster with Kubernetes version `1.13` or earlier and intend to use Fargate nodes in your cluster, then edit your `kube-proxy` manifest to include a `NodeAffinity` rule to prevent `kube-proxy` Pods from scheduling on Fargate nodes. This is a one-time edit. Once you've added the `Affinity Rule` to your manifest, you don't need to add it each time that you update the add-on. Edit your `kube-proxy` DaemonSet.

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

Add the following `Affinity Rule` to the DaemonSet `spec` section of the file in the editor and then save the file. For an example of where to include this text in the editor, see the [CNI manifest] file on GitHub.

```
- key: eks.amazonaws.com/compute-type
  operator: NotIn
  values:
  - fargate
```

Learn how to deploy workloads and add-ons to Amazon EKS

Your workloads are deployed in containers, which are deployed in Pods in Kubernetes. A Pod includes one or more containers. Typically, one or more Pods that provide the same service are deployed in a Kubernetes service. Once you've deployed multiple Pods that provide the same service, you can:

- View information about the workloads running on each of your clusters using the AWS Management Console.
- Vertically scale Pods up or down with the Kubernetes Vertical Pod Autoscaler.
- Horizontally scale the number of Pods needed to meet demand up or down with the Kubernetes Horizontal Pod Autoscaler.
- Create an external (for internet-accessible Pods) or an internal (for private Pods) network load balancer to balance network traffic across Pods. The load balancer routes traffic at Layer 4 of the OSI model.
- Create an Application Load Balancer to balance application traffic across Pods. The application load balancer routes traffic at Layer 7 of the OSI model.
- If you're new to Kubernetes, this topic helps you Deploy a sample application.
- You can restrict IP addresses that can be assigned to a service with `externalIPs`.

Deploy a sample application on Linux

In this topic, you deploy a sample application to your cluster on linux nodes.

Prerequisites

- An existing Kubernetes cluster with at least one node. If you don't have an existing Amazon EKS cluster, you can deploy one using one of the guides in *Get started*.
- `Kubectl` installed on your computer. For more information, see the section called "Set up kubectl and eksctl".
- `Kubectl` configured to communicate with your cluster. For more information, see the section called "Access cluster with kubectl".

- If you plan to deploy your sample workload to Fargate, then you must have an existing Fargate profile that includes the same namespace created in this tutorial, which is `eks-sample-app`, unless you change the name. If you created a cluster with one of the gudes in *Get started*, then you'll have to create a new profile, or add the namespace to your existing profile, because the profile created in the getting started guides doesn't specify the namespace used in this tutorial. Your VPC must also have at least one private subnet.

Though many variables are changeable in the following steps, we recommend only changing variable values where specified. Once you have a better understanding of Kubernetes Pods, deployments, and services, you can experiment with changing other values.

Create a namespace

A namespace allows you to group resources in Kubernetes. For more information, see Namespaces in the Kubernetes documentation. If you plan to deploy your sample application to Simplify compute management with AWS Fargate, make sure that the value for `namespace` in your Define which Pods use AWS Fargate when launched is `eks-sample-app`.

```
kubectl create namespace eks-sample-app
```

Create a Kubernetes deployment

Create a Kubernetes deployment. This sample deployment pulls a container image from a public repository and deploys three replicas (individual Pods) of it to your cluster. To learn more, see Deployments in the Kubernetes documentation.

1. Save the following contents to a file named `eks-sample-deployment.yaml`. The containers in the sample application don't use network storage, but you might have applications that need to. For more information, see *Store app data*.

   - The `amd64` or `arm64 values` under the `kubernetes.io/arch` key mean that the application can be deployed to either hardware architecture (if you have both in your cluster). This is possible because this image is a multi-architecture image, but not all are. You can determine the hardware architecture that the image is supported on by viewing the image details in the repository that you're pulling it from. When deploying images that don't support a hardware architecture type, or that you don't want the image deployed to, remove that type from the manifest. For more information, see Well-Known Labels, Annotations and Taints in the Kubernetes documentation.

- The `kubernetes.io/os: linux` `nodeSelector` means that if you had Linux and Windows nodes (for example) in your cluster, the image would only be deployed to Linux nodes. For more information, see Well-Known Labels, Annotations and Taints in the Kubernetes documentation.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: eks-sample-linux-deployment
  namespace: eks-sample-app
  labels:
    app: eks-sample-linux-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: eks-sample-linux-app
  template:
    metadata:
      labels:
        app: eks-sample-linux-app
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/arch
                operator: In
                values:
                - amd64
                - arm64
      containers:
      - name: nginx
        image: public.ecr.aws/nginx/nginx:1.23
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
      nodeSelector:
        kubernetes.io/os: linux
```

2. Apply the deployment manifest to your cluster.

```
kubectl apply -f eks-sample-deployment.yaml
```

Create a service

A service allows you to access all replicas through a single IP address or name. For more information, see Service in the Kubernetes documentation. Though not implemented in the sample application, if you have applications that need to interact with other AWS services, we recommend that you create Kubernetes service accounts for your Pods, and associate them to AWS IAM accounts. By specifying service accounts, your Pods have only the minimum permissions that you specify for them to interact with other services. For more information, see the section called "Credentials with IRSA".

1. Save the following contents to a file named `eks-sample-service.yaml`. Kubernetes assigns the service its own IP address that is accessible only from within the cluster. To access the service from outside of your cluster, deploy the AWS Load Balancer Controller to load balance application or network traffic to the service.

```
apiVersion: v1
kind: Service
metadata:
  name: eks-sample-linux-service
  namespace: eks-sample-app
  labels:
    app: eks-sample-linux-app
spec:
  selector:
    app: eks-sample-linux-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

2. Apply the service manifest to your cluster.

```
kubectl apply -f eks-sample-service.yaml
```

Review resources created

1. View all resources that exist in the `eks-sample-app` namespace.

```
kubectl get all -n eks-sample-app
```

An example output is as follows.

```
NAME                                               READY   STATUS    RESTARTS   AGE
pod/eks-sample-linux-deployment-65b7669776-m6qxz   1/1     Running   0          27m
pod/eks-sample-linux-deployment-65b7669776-mmxvd   1/1     Running   0          27m
pod/eks-sample-linux-deployment-65b7669776-qzn22   1/1     Running   0          27m

NAME                                  TYPE         CLUSTER-IP      EXTERNAL-IP   PORT(S)
   AGE
service/eks-sample-linux-service   ClusterIP    10.100.74.8     <none>        80/TCP
   32m

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/eks-sample-linux-deployment 3/3     3            3           27m

NAME                                                      DESIRED   CURRENT   READY
 AGE
replicaset.apps/eks-sample-linux-deployment-776d8f8fd8    3         3         3
 27m
```

In the output, you see the service and deployment that were specified in the sample manifests deployed in previous steps. You also see three Pods. This is because `3 replicas` were specified in the sample manifest. For more information about Pods, see Pods in the Kubernetes documentation. Kubernetes automatically creates the `replicaset` resource, even though it isn't specified in the sample manifests. For more information about `ReplicaSets`, see ReplicaSet in the Kubernetes documentation.

> **Note**
>
> Kubernetes maintains the number of replicas that are specified in the manifest. If this were a production deployment and you wanted Kubernetes to horizontally scale the number of replicas or vertically scale the compute resources for the Pods, use the Scale

pod deployments with Horizontal Pod Autoscaler and the Adjust pod resources with Vertical Pod Autoscaler to do so.

2. View the details of the deployed service.

```
kubectl -n eks-sample-app describe service eks-sample-linux-service
```

An example output is as follows.

```
Name:                eks-sample-linux-service
Namespace:           eks-sample-app
Labels:              app=eks-sample-linux-app
Annotations:         <none>
Selector:            app=eks-sample-linux-app
Type:                ClusterIP
IP Families:         <none>
IP:                  10.100.74.8
IPs:                 10.100.74.8
Port:                <unset>  80/TCP
TargetPort:          80/TCP
Endpoints:           192.168.24.212:80,192.168.50.185:80,192.168.63.93:80
Session Affinity:    None
Events:              <none>
```

In the previous output, the value for `IP:` is a unique IP address that can be reached from any node or Pod within the cluster, but it can't be reached from outside of the cluster. The values for `Endpoints` are IP addresses assigned from within your VPC to the Pods that are part of the service.

3. View the details of one of the Pods listed in the output when you viewed the namespace in a previous step. Replace *776d8f8fd8-78w66* with the value returned for one of your Pods.

```
kubectl -n eks-sample-app describe pod eks-sample-linux-deployment-65b7669776-m6qxz
```

Abbreviated example output

```
Name:           eks-sample-linux-deployment-65b7669776-m6qxz
Namespace:      eks-sample-app
Priority:       0
Node:           ip-192-168-45-132.us-west-2.compute.internal/192.168.45.132
```

```
[...]
IP:           192.168.63.93
IPs:
  IP:           192.168.63.93
Controlled By:  ReplicaSet/eks-sample-linux-deployment-65b7669776
[...]
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
[...]
Events:
  Type    Reason     Age    From
 Message
  ----    ------     ----   ----
 -------
  Normal  Scheduled  3m20s  default-scheduler
 Successfully assigned eks-sample-app/eks-sample-linux-deployment-65b7669776-m6qxz to
 ip-192-168-45-132.us-west-2.compute.internal
[...]
```

In the previous output, the value for `IP:` is a unique IP that's assigned to the Pod from the CIDR block assigned to the subnet that the node is in. If you prefer to assign Pods IP addresses from different CIDR blocks, you can change the default behavior. For more information, see the section called "Custom networking". You can also see that the Kubernetes scheduler scheduled the Pod on the Node with the IP address *192.168.45.132*.

> **Tip**
>
> Rather than using the command line, you can view many details about Pods, services, deployments, and other Kubernetes resources in the AWS Management Console. For more information, see the section called "Access cluster resources".

Run a shell on a Pod

1. Run a shell on the Pod that you described in the previous step, replacing *65b7669776-m6qxz* with the ID of one of your Pods.

```
kubectl exec -it eks-sample-linux-deployment-65b7669776-m6qxz -n eks-sample-app -- /
bin/bash
```

2. From the Pod shell, view the output from the web server that was installed with your deployment in a previous step. You only need to specify the service name. It is resolved to the service's IP address by CoreDNS, which is deployed with an Amazon EKS cluster, by default.

```
curl eks-sample-linux-service
```

An example output is as follows.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

3. From the Pod shell, view the DNS server for the Pod.

```
cat /etc/resolv.conf
```

An example output is as follows.

```
nameserver 10.100.0.10
search eks-sample-app.svc.cluster.local svc.cluster.local cluster.local us-
west-2.compute.internal
options ndots:5
```

In the previous output, `10.100.0.10` is automatically assigned as the `nameserver` for all Pods deployed to the cluster.

4. Disconnect from the Pod by typing `exit`.

5. Once you're finished with the sample application, you can remove the sample namespace, service, and deployment with the following command.

```
kubectl delete namespace eks-sample-app
```

Next Steps

After you deploy the sample application, you might want to try some of the following exercises:

- Route application and HTTP traffic with Application Load Balancers
- Route TCP and UDP traffic with Network Load Balancers

Deploy a sample application on Windows

In this topic, you deploy a sample application to your cluster on Windows nodes.

Prerequisites

- An existing Kubernetes cluster with at least one node. If you don't have an existing Amazon EKS cluster, you can deploy one using one of the guides in *Get started*. You must have Windows support enabled for your cluster and at least one Amazon EC2 Windows node.
- `Kubectl` installed on your computer. For more information, see the section called "Set up kubectl and eksctl".
- `Kubectl` configured to communicate with your cluster. For more information, see the section called "Access cluster with kubectl".
- If you plan to deploy your sample workload to Fargate, then you must have an existing Fargate profile that includes the same namespace created in this tutorial, which is `eks-sample-app`, unless you change the name. If you created a cluster with one of the gudes in *Get started*, then you'll have to create a new profile, or add the namespace to your existing profile, because the profile created in the getting started guides doesn't specify the namespace used in this tutorial. Your VPC must also have at least one private subnet.

Though many variables are changeable in the following steps, we recommend only changing variable values where specified. Once you have a better understanding of Kubernetes Pods, deployments, and services, you can experiment with changing other values.

Create a namespace

A namespace allows you to group resources in Kubernetes. For more information, see Namespaces in the Kubernetes documentation. If you plan to deploy your sample application to Simplify compute management with AWS Fargate, make sure that the value for `namespace` in your Define which Pods use AWS Fargate when launched is `eks-sample-app`.

```
kubectl create namespace eks-sample-app
```

Create a Kubernetes deployment

This sample deployment pulls a container image from a public repository and deploys three replicas (individual Pods) of it to your cluster. To learn more, see Deployments in the Kubernetes documentation.

1. Save the following contents to a file named `eks-sample-deployment.yaml`. The containers in the sample application don't use network storage, but you might have applications that need to. For more information, see *Store app data*.
   - The `kubernetes.io/os: windows` `nodeSelector` means that if you had Windows and Linux nodes (for example) in your cluster, the image would only be deployed to Windows nodes. For more information, see Well-Known Labels, Annotations and Taints in the Kubernetes documentation.

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: eks-sample-windows-deployment
     namespace: eks-sample-app
     labels:
       app: eks-sample-windows-app
   spec:
     replicas: 3
     selector:
       matchLabels:
         app: eks-sample-windows-app
     template:
       metadata:
         labels:
           app: eks-sample-windows-app
       spec:
         affinity:
           nodeAffinity:
             requiredDuringSchedulingIgnoredDuringExecution:
               nodeSelectorTerms:
               - matchExpressions:
   ```

```
                - key: beta.kubernetes.io/arch
                  operator: In
                  values:
                  - amd64
      containers:
      - name: windows-server-iis
        image: mcr.microsoft.com/windows/servercore:ltsc2019
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
        command:
        - powershell.exe
        - -command
        - "Add-WindowsFeature Web-Server; Invoke-WebRequest -UseBasicParsing
 -Uri 'https://dotnetbinaries.blob.core.windows.net/servicemonitor/2.0.1.6/
ServiceMonitor.exe' -OutFile 'C:\\ServiceMonitor.exe'; echo '<html><body><br/
><br/><marquee><H1>Hello EKS!!!<H1><marquee></body><html>' > C:\\inetpub\\wwwroot\
\default.html; C:\\ServiceMonitor.exe 'w3svc'; "
      nodeSelector:
        kubernetes.io/os: windows
```

2. Apply the deployment manifest to your cluster.

```
kubectl apply -f eks-sample-deployment.yaml
```

Create a service

A service allows you to access all replicas through a single IP address or name. For more information, see Service in the Kubernetes documentation. Though not implemented in the sample application, if you have applications that need to interact with other AWS services, we recommend that you create Kubernetes service accounts for your Pods, and associate them to AWS IAM accounts. By specifying service accounts, your Pods have only the minimum permissions that you specify for them to interact with other services. For more information, see the section called "Credentials with IRSA".

1. Save the following contents to a file named `eks-sample-service.yaml`. Kubernetes assigns the service its own IP address that is accessible only from within the cluster. To access the service from outside of your cluster, deploy the AWS Load Balancer Controller to load balance application or network traffic to the service.

```
apiVersion: v1
kind: Service
metadata:
  name: eks-sample-windows-service
  namespace: eks-sample-app
  labels:
    app: eks-sample-windows-app
spec:
  selector:
    app: eks-sample-windows-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

2. Apply the service manifest to your cluster.

```
kubectl apply -f eks-sample-service.yaml
```

Review resources created

1. View all resources that exist in the `eks-sample-app` namespace.

```
kubectl get all -n eks-sample-app
```

An example output is as follows.

```
NAME                                                 READY   STATUS    RESTARTS   AGE
pod/eks-sample-windows-deployment-65b7669776-m6qxz   1/1     Running   0          27m
pod/eks-sample-windows-deployment-65b7669776-mmxvd   1/1     Running   0          27m
pod/eks-sample-windows-deployment-65b7669776-qzn22   1/1     Running   0          27m

NAME                               TYPE         CLUSTER-IP      EXTERNAL-IP   PORT(S)
   AGE
service/eks-sample-windows-service   ClusterIP    10.100.74.8     <none>        80/
TCP    32m

NAME                                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/eks-sample-windows-deployment 3/3     3            3           27m
```

```
NAME                                                    DESIRED   CURRENT   READY
 AGE
replicaset.apps/eks-sample-windows-deployment-776d8f8fd8    3         3         3
   27m
```

In the output, you see the service and deployment that were specified in the sample manifests deployed in previous steps. You also see three Pods. This is because `3 replicas` were specified in the sample manifest. For more information about Pods, see Pods in the Kubernetes documentation. Kubernetes automatically creates the `replicaset` resource, even though it isn't specified in the sample manifests. For more information about `ReplicaSets`, see ReplicaSet in the Kubernetes documentation.

> **Note**
>
> Kubernetes maintains the number of replicas that are specified in the manifest. If this were a production deployment and you wanted Kubernetes to horizontally scale the number of replicas or vertically scale the compute resources for the Pods, use the Scale pod deployments with Horizontal Pod Autoscaler and the Adjust pod resources with Vertical Pod Autoscaler to do so.

2. View the details of the deployed service.

```
kubectl -n eks-sample-app describe service eks-sample-windows-service
```

An example output is as follows.

```
Name:              eks-sample-windows-service
Namespace:         eks-sample-app
Labels:            app=eks-sample-windows-app
Annotations:       <none>
Selector:          app=eks-sample-windows-app
Type:              ClusterIP
IP Families:       <none>
IP:                10.100.74.8
IPs:               10.100.74.8
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         192.168.24.212:80,192.168.50.185:80,192.168.63.93:80
Session Affinity:  None
```

```
Events:            <none>
```

In the previous output, the value for `IP:` is a unique IP address that can be reached from any node or Pod within the cluster, but it can't be reached from outside of the cluster. The values for `Endpoints` are IP addresses assigned from within your VPC to the Pods that are part of the service.

3. View the details of one of the Pods listed in the output when you viewed the namespace in a previous step. Replace *776d8f8fd8-78w66* with the value returned for one of your Pods.

```
kubectl -n eks-sample-app describe pod eks-sample-windows-deployment-65b7669776-m6qxz
```

Abbreviated example output

```
Name:          eks-sample-windows-deployment-65b7669776-m6qxz
Namespace:     eks-sample-app
Priority:      0
Node:          ip-192-168-45-132.us-west-2.compute.internal/192.168.45.132
[...]
IP:            192.168.63.93
IPs:
  IP:            192.168.63.93
Controlled By:  ReplicaSet/eks-sample-windows-deployment-65b7669776
[...]
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
[...]
Events:
  Type    Reason     Age    From
 Message
  ----    ------     ----   ----
 -------
  Normal  Scheduled  3m20s  default-scheduler
 Successfully assigned eks-sample-app/eks-sample-windows-deployment-65b7669776-m6qxz
 to ip-192-168-45-132.us-west-2.compute.internal
[...]
```

In the previous output, the value for `IP:` is a unique IP that's assigned to the Pod from the CIDR block assigned to the subnet that the node is in. If you prefer to assign Pods IP addresses from different CIDR blocks, you can change the default behavior. For more information, see the section called "Custom networking". You can also see that the Kubernetes scheduler scheduled the Pod on the `Node` with the IP address *192.168.45.132*.

> **Tip**
>
> Rather than using the command line, you can view many details about Pods, services, deployments, and other Kubernetes resources in the AWS Management Console. For more information, see the section called "Access cluster resources".

Run a shell on a Pod

1. Run a shell on the Pod that you described in the previous step, replacing *65b7669776-m6qxz* with the ID of one of your Pods.

```
kubectl exec -it eks-sample-windows-deployment-65b7669776-m6qxz -n eks-sample-app --
 powershell.exe
```

2. From the Pod shell, view the output from the web server that was installed with your deployment in a previous step. You only need to specify the service name. It is resolved to the service's IP address by CoreDNS, which is deployed with an Amazon EKS cluster, by default.

```
Invoke-WebRequest -uri eks-sample-windows-service/default.html -UseBasicParsing
```

An example output is as follows.

```
StatusCode        : 200
StatusDescription : OK
Content           : < h t m l > < b o d y > < b r / > < b r / > < m a r q u e e > < H
 1 > H e l l o
                      E K S ! ! ! < H 1 > < m a r q u e e > < / b o d y > < h t m l >
```

3. From the Pod shell, view the DNS server for the Pod.

```
Get-NetIPConfiguration
```

Abbreviated output

```
InterfaceAlias          : vEthernet
[...]
IPv4Address             : 192.168.63.14
[...]
DNSServer               : 10.100.0.10
```

In the previous output, `10.100.0.10` is automatically assigned as the DNS server for all Pods deployed to the cluster.

4. Disconnect from the Pod by typing `exit`.
5. Once you're finished with the sample application, you can remove the sample namespace, service, and deployment with the following command.

```
kubectl delete namespace eks-sample-app
```

Next Steps

After you deploy the sample application, you might want to try some of the following exercises:

- Route application and HTTP traffic with Application Load Balancers
- Route TCP and UDP traffic with Network Load Balancers

Adjust pod resources with Vertical Pod Autoscaler

The Kubernetes Vertical Pod Autoscaler automatically adjusts the CPU and memory reservations for your Pods to help "right size" your applications. This adjustment can improve cluster resource utilization and free up CPU and memory for other Pods. This topic helps you to deploy the Vertical Pod Autoscaler to your cluster and verify that it is working.

- You have an existing Amazon EKS cluster. If you don't, see *Get started*.
- You have the Kubernetes Metrics Server installed. For more information, see the section called "Metrics server".
- You are using a `kubectl` client that is configured to communicate with your Amazon EKS cluster.
- OpenSSL `1.1.1` or later installed on your device.

Deploy the Vertical Pod Autoscaler

In this section, you deploy the Vertical Pod Autoscaler to your cluster.

1. Open a terminal window and navigate to a directory where you would like to download the Vertical Pod Autoscaler source code.
2. Clone the kubernetes/autoscalerGitHub repository.

```
git clone https://github.com/kubernetes/autoscaler.git
```

3. Change to the `vertical-pod-autoscaler` directory.

```
cd autoscaler/vertical-pod-autoscaler/
```

4. (Optional) If you have already deployed another version of the Vertical Pod Autoscaler, remove it with the following command.

```
./hack/vpa-down.sh
```

5. If your nodes don't have internet access to the `registry.k8s.io` container registry, then you need to pull the following images and push them to your own private repository. For more information about how to pull the images and push them to your own private repository, see the section called "Copy an image to a repository".

```
registry.k8s.io/autoscaling/vpa-admission-controller:0.10.0
registry.k8s.io/autoscaling/vpa-recommender:0.10.0
registry.k8s.io/autoscaling/vpa-updater:0.10.0
```

If you're pushing the images to a private Amazon ECR repository, then replace `registry.k8s.io` in the manifests with your registry. Replace *`111122223333`* with your account ID. Replace *`region-code`* with the AWS Region that your cluster is in. The following commands assume that you named your repository the same as the repository name in the manifest. If you named your repository something different, then you'll need to change it too.

```
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/admission-controller-deployment.yaml
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/recommender-deployment.yaml
```

```
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/updater-deployment.yaml
```

6. Deploy the Vertical Pod Autoscaler to your cluster with the following command.

```
./hack/vpa-up.sh
```

7. Verify that the Vertical Pod Autoscaler Pods have been created successfully.

```
kubectl get pods -n kube-system
```

An example output is as follows.

```
NAME                                        READY   STATUS    RESTARTS   AGE
[...]
metrics-server-8459fc497-kfj8w              1/1     Running   0          83m
vpa-admission-controller-68c748777d-ppspd   1/1     Running   0          7s
vpa-recommender-6fc8c67d85-gljpl            1/1     Running   0          8s
vpa-updater-786b96955c-bgp9d                1/1     Running   0          8s
```

Test your Vertical Pod Autoscaler installation

In this section, you deploy a sample application to verify that the Vertical Pod Autoscaler is working.

1. Deploy the `hamster.yaml` Vertical Pod Autoscaler example with the following command.

```
kubectl apply -f examples/hamster.yaml
```

2. Get the Pods from the `hamster` example application.

```
kubectl get pods -l app=hamster
```

An example output is as follows.

```
hamster-c7d89d6db-rglf5   1/1     Running   0          48s
hamster-c7d89d6db-znvz5   1/1     Running   0          48s
```

3. Describe one of the Pods to view its `cpu` and `memory` reservation. Replace *c7d89d6db-rglf5* with one of the IDs returned in your output from the previous step.

```
kubectl describe pod hamster-c7d89d6db-rglf5
```

An example output is as follows.

```
[...]
Containers:
  hamster:
    Container ID:  docker://
e76c2413fc720ac395c33b64588c82094fc8e5d590e373d5f818f3978f577e24
    Image:         registry.k8s.io/ubuntu-slim:0.1
    Image ID:      docker-pullable://registry.k8s.io/ubuntu-
slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
    Args:
      -c
      while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
    State:          Running
      Started:      Fri, 27 Sep 2019 10:35:16 -0700
    Ready:          True
    Restart Count:  0
    Requests:
      cpu:        100m
      memory:     50Mi
[...]
```

You can see that the original Pod reserves 100 millicpu of CPU and 50 mebibytes of memory. For this example application, 100 millicpu is less than the Pod needs to run, so it is CPU-constrained. It also reserves much less memory than it needs. The Vertical Pod Autoscaler `vpa-recommender` deployment analyzes the hamster Pods to see if the CPU and memory requirements are appropriate. If adjustments are needed, the `vpa-updater` relaunches the Pods with updated values.

4. Wait for the `vpa-updater` to launch a new hamster Pods. This should take a minute or two. You can monitor the Pods with the following command.

> **Note**
>
> If you are not sure that a new Pod has launched, compare the Pod names with your previous list. When the new Pod launches, you will see a new Pod name.

```
kubectl get --watch Pods -l app=hamster
```

5. When a new hamster Pods is started, describe it and view the updated CPU and memory reservations.

```
kubectl describe pod hamster-c7d89d6db-jxgfv
```

An example output is as follows.

```
[...]
Containers:
  hamster:
    Container ID:
 docker://2c3e7b6fb7ce0d8c86444334df654af6fb3fc88aad4c5d710eac3b1e7c58f7db
    Image:          registry.k8s.io/ubuntu-slim:0.1
    Image ID:       docker-pullable://registry.k8s.io/ubuntu-
slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
    Port:           <none>
    Host Port:      <none>
    Command:
      /bin/sh
    Args:
      -c
      while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
    State:           Running
      Started:       Fri, 27 Sep 2019 10:37:08 -0700
    Ready:           True
    Restart Count:   0
    Requests:
      cpu:        587m
      memory:     262144k
[...]
```

In the previous output, you can see that the cpu reservation increased to 587 millicpu, which is over five times the original value. The memory increased to 262,144 Kilobytes, which is around 250 mebibytes, or five times the original value. This Pod was under-resourced, and the Vertical Pod Autoscaler corrected the estimate with a much more appropriate value.

6. Describe the hamster-vpa resource to view the new recommendation.

```
kubectl describe vpa/hamster-vpa
```

An example output is as follows.

```
Name:         hamster-vpa
Namespace:    default
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"autoscaling.k8s.io/
v1beta2","kind":"VerticalPodAutoscaler","metadata":{"annotations":{},"name":"hamster-
vpa","namespace":"d...
API Version:  autoscaling.k8s.io/v1beta2
Kind:         VerticalPodAutoscaler
Metadata:
  Creation Timestamp:  2019-09-27T18:22:51Z
  Generation:          23
  Resource Version:    14411
  Self Link:           /apis/autoscaling.k8s.io/v1beta2/namespaces/default/
verticalpodautoscalers/hamster-vpa
  UID:                 d0d85fb9-e153-11e9-ae53-0205785d75b0
Spec:
  Target Ref:
    API Version:  apps/v1
    Kind:         Deployment
    Name:         hamster
Status:
  Conditions:
    Last Transition Time:  2019-09-27T18:23:28Z
    Status:                True
    Type:                  RecommendationProvided
  Recommendation:
    Container Recommendations:
      Container Name:  hamster
      Lower Bound:
        Cpu:     550m
```

```
          Memory:  262144k
        Target:
          Cpu:     587m
          Memory:  262144k
        Uncapped Target:
          Cpu:     587m
          Memory:  262144k
        Upper Bound:
          Cpu:     21147m
          Memory:  387863636
Events:            <none>
```

7. When you finish experimenting with the example application, you can delete it with the following command.

```
kubectl delete -f examples/hamster.yaml
```

Scale pod deployments with Horizontal Pod Autoscaler

The Kubernetes Horizontal Pod Autoscaler automatically scales the number of Pods in a deployment, replication controller, or replica set based on that resource's CPU utilization. This can help your applications scale out to meet increased demand or scale in when resources are not needed, thus freeing up your nodes for other applications. When you set a target CPU utilization percentage, the Horizontal Pod Autoscaler scales your application in or out to try to meet that target.

The Horizontal Pod Autoscaler is a standard API resource in Kubernetes that simply requires that a metrics source (such as the Kubernetes metrics server) is installed on your Amazon EKS cluster to work. You do not need to deploy or install the Horizontal Pod Autoscaler on your cluster to begin scaling your applications. For more information, see Horizontal Pod Autoscaler in the Kubernetes documentation.

Use this topic to prepare the Horizontal Pod Autoscaler for your Amazon EKS cluster and to verify that it is working with a sample application.

Note

This topic is based on the Horizontal Pod autoscaler walkthrough in the Kubernetes documentation.

- You have an existing Amazon EKS cluster. If you don't, see *Get started*.
- You have the Kubernetes Metrics Server installed. For more information, see the section called "Metrics server".
- You are using a `kubectl` client that is configured to communicate with your Amazon EKS cluster.

Run a Horizontal Pod Autoscaler test application

In this section, you deploy a sample application to verify that the Horizontal Pod Autoscaler is working.

> **Note**
>
> This example is based on the Horizontal Pod autoscaler walkthrough in the Kubernetes documentation.

1. Deploy a simple Apache web server application with the following command.

```
kubectl apply -f https://k8s.io/examples/application/php-apache.yaml
```

   This Apache web server Pod is given a 500 millicpu CPU limit and it is serving on port 80.

2. Create a Horizontal Pod Autoscaler resource for the `php-apache` deployment.

```
kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
```

   This command creates an autoscaler that targets 50 percent CPU utilization for the deployment, with a minimum of one Pod and a maximum of ten Pods. When the average CPU load is lower than 50 percent, the autoscaler tries to reduce the number of Pods in the deployment, to a minimum of one. When the load is greater than 50 percent, the autoscaler tries to increase the number of Pods in the deployment, up to a maximum of ten. For more information, see How does a HorizontalPodAutoscaler work? in the Kubernetes documentation.

3. Describe the autoscaler with the following command to view its details.

```
kubectl get hpa
```

   An example output is as follows.

```
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache    0%/50%    1         10        1          51s
```

As you can see, the current CPU load is `0%`, because there's no load on the server yet. The Pod count is already at its lowest boundary (one), so it cannot scale in.

4. Create a load for the web server by running a container.

```
kubectl run -i \
    --tty load-generator \
    --rm --image=busybox \
    --restart=Never \
    -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
```

5. To watch the deployment scale out, periodically run the following command in a separate terminal from the terminal that you ran the previous step in.

```
kubectl get hpa php-apache
```

An example output is as follows.

```
NAME         REFERENCE                TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache    250%/50%   1         10        5          4m44s
```

It may take over a minute for the replica count to increase. As long as actual CPU percentage is higher than the target percentage, then the replica count increases, up to 10. In this case, it's `250%`, so the number of `REPLICAS` continues to increase.

> **Note**
>
> It may take a few minutes before you see the replica count reach its maximum. If only 6 replicas, for example, are necessary for the CPU load to remain at or under 50%, then the load won't scale beyond 6 replicas.

6. Stop the load. In the terminal window you're generating the load in, stop the load by holding down the `Ctrl+C` keys. You can watch the replicas scale back to 1 by running the following command again in the terminal that you're watching the scaling in.

```
kubectl get hpa
```

An example output is as follows.

```
NAME          REFERENCE                  TARGETS    MINPODS    MAXPODS    REPLICAS    AGE
php-apache    Deployment/php-apache      0%/50%     1          10         1           25m
```

> **Note**
>
> The default timeframe for scaling back down is five minutes, so it will take some time before you see the replica count reach 1 again, even when the current CPU percentage is 0 percent. The timeframe is modifiable. For more information, see Horizontal Pod Autoscaler in the Kubernetes documentation.

7. When you are done experimenting with your sample application, delete the `php-apache` resources.

```
kubectl delete deployment.apps/php-apache service/php-apache
 horizontalpodautoscaler.autoscaling/php-apache
```

Route TCP and UDP traffic with Network Load Balancers

> **Note**
>
> **New:** Amazon EKS Auto Mode automates routine tasks for load balancing. For more information, see:
>
> - the section called "Deploy load balancer"
> - the section called "Create service"

Network traffic is load balanced at `L4` of the OSI model. To load balance application traffic at `L7`, you deploy a Kubernetes `ingress`, which provisions an AWS Application Load Balancer. For more information, see the section called "Application load balancing". To learn more about the differences between the two types of load balancing, see Elastic Load Balancing features on the AWS website.

When you create a Kubernetes `Service` of type `LoadBalancer`, the AWS cloud provider load balancer controller creates AWS Classic Load Balancers by default, but can also create AWS Network Load Balancers. This controller is only receiving critical bug fixes in the future. For more information about using the AWS cloud provider load balancer , see AWS cloud provider load balancer controller in the Kubernetes documentation. Its use is not covered in this topic.

We recommend that you use version `2.7.2` or later of the AWS Load Balancer Controller instead of the AWS cloud provider load balancer controller. The AWS Load Balancer Controller creates AWS Network Load Balancers, but doesn't create AWS Classic Load Balancers. The remainder of this topic is about using the AWS Load Balancer Controller.

An AWS Network Load Balancer can load balance network traffic to Pods deployed to Amazon EC2 IP and instance targets, to AWS Fargate IP targets, or to Amazon EKS Hybrid Nodes as IP targets. For more information, see AWS Load Balancer Controller on GitHub.

Prerequisites

Before you can load balance network traffic using the AWS Load Balancer Controller, you must meet the following requirements.

- Have an existing cluster. If you don't have an existing cluster, see *Get started*. If you need to update the version of an existing cluster, see the section called "Update Kubernetes version".
- Have the AWS Load Balancer Controller deployed on your cluster. For more information, see the section called " AWS Load Balancer Controller". We recommend version `2.7.2` or later.
- At least one subnet. If multiple tagged subnets are found in an Availability Zone, the controller chooses the first subnet whose subnet ID comes first lexicographically. The subnet must have at least eight available IP addresses.
- If you're using the AWS Load Balancer Controller version `2.1.1` or earlier, subnets must be tagged as follows. If using version `2.1.2` or later, this tag is optional. You might want to tag a subnet if you have multiple clusters running in the same VPC, or multiple AWS services sharing subnets in a VPC, and want more control over where load balancers are provisioned for each cluster. If you explicitly specify subnet IDs as an annotation on a service object, then Kubernetes and the AWS Load Balancer Controller use those subnets directly to create the load balancer. Subnet tagging isn't required if you choose to use this method for provisioning load balancers and you can skip the following private and public subnet tagging requirements. Replace *`my-cluster`* with your cluster name.
  - **Key** – `kubernetes.io/cluster/<my-cluster>`

  - **Value** – `shared` or `owned`
- Your public and private subnets must meet the following requirements, unless you explicitly specify subnet IDs as an annotation on a service or ingress object. If you provision load balancers by explicitly specifying subnet IDs as an annotation on a service or ingress object, then Kubernetes and the AWS Load Balancer Controller use those subnets directly to create the load balancer and the following tags aren't required.
  - **Private subnets** – Must be tagged in the following format. This is so that Kubernetes and the AWS Load Balancer Controller know that the subnets can be used for internal load balancers. If you use `eksctl` or an Amazon EKS AWS AWS CloudFormation template to create your VPC after March 26, 2020, then the subnets are tagged appropriately when they're created. For more information about the Amazon EKS AWS AWS CloudFormation VPC templates, see the section called "Create a VPC".
    - **Key** – `kubernetes.io/role/internal-elb`
    - **Value** – `1`
  - **Public subnets** – Must be tagged in the following format. This is so that Kubernetes knows to use only those subnets for external load balancers instead of choosing a public subnet in each Availability Zone (based on the lexicographical order of the subnet IDs). If you use `eksctl` or an Amazon EKS AWS CloudFormation template to create your VPC after March 26, 2020, then the subnets are tagged appropriately when they're created. For more information about the Amazon EKS AWS CloudFormation VPC templates, see the section called "Create a VPC".
    - **Key** – `kubernetes.io/role/elb`
    - **Value** – `1`

  If the subnet role tags aren't explicitly added, the Kubernetes service controller examines the route table of your cluster VPC subnets to determine if the subnet is private or public. We recommend that you don't rely on this behavior, and instead explicitly add the private or public role tags. The AWS Load Balancer Controller doesn't examine route tables, and requires the private and public tags to be present for successful auto discovery.

Considerations

- The configuration of your load balancer is controlled by annotations that are added to the manifest for your service. Service annotations are different when using the AWS Load Balancer Controller than they are when using the AWS cloud provider load balancer controller. Make sure to review the annotations for the AWS Load Balancer Controller before deploying services.

- When using the Amazon VPC CNI plugin for Kubernetes, the AWS Load Balancer Controller can load balance to Amazon EC2 IP or instance targets and Fargate IP targets. When using Alternate compatible CNI plugins, the controller can only load balance to instance targets, unless you are load balancing to Amazon EKS Hybrid Nodes. For hybrid nodes, the controller can load balance IP targets. For more information about Network Load Balancer target types, see Target type in the User Guide for Network Load Balancers
- If you want to add tags to the load balancer when or after it's created, add the following annotation in your service specification. For more information, see AWS Resource Tags in the AWS Load Balancer Controller documentation.

```
service.beta.kubernetes.io/aws-load-balancer-additional-resource-tags
```

- You can assign Elastic IP addresses to the Network Load Balancer by adding the following annotation. Replace the *example values* with the `Allocation IDs` of your Elastic IP addresses. The number of `Allocation IDs` must match the number of subnets that are used for the load balancer. For more information, see the AWS Load Balancer Controller documentation.

```
service.beta.kubernetes.io/aws-load-balancer-eip-allocations: eipalloc-
xxxxxxxxxxxxxxxxx,eipalloc-yyyyyyyyyyyyyyyyy
```

- Amazon EKS adds one inbound rule to the node's security group for client traffic and one rule for each load balancer subnet in the VPC for health checks for each Network Load Balancer that you create. Deployment of a service of type `LoadBalancer` can fail if Amazon EKS attempts to create rules that exceed the quota for the maximum number of rules allowed for a security group. For more information, see Security groups in Amazon VPC quotas in the Amazon VPC User Guide. Consider the following options to minimize the chances of exceeding the maximum number of rules for a security group:
  - Request an increase in your rules per security group quota. For more information, see Requesting a quota increase in the Service Quotas User Guide.
  - Use IP targets, rather than instance targets. With IP targets, you can share rules for the same target ports. You can manually specify load balancer subnets with an annotation. For more information, see Annotations on GitHub.
  - Use an ingress, instead of a service of type `LoadBalancer`, to send traffic to your service. The AWS Application Load Balancer requires fewer rules than Network Load Balancers. You can share an ALB across multiple ingresses. For more information, see the section called "Application load balancing". You can't share a Network Load Balancer across multiple services.

  - Deploy your clusters to multiple accounts.
- If your Pods run on Windows in an Amazon EKS cluster, a single service with a load balancer can support up to 1024 back-end Pods. Each Pod has its own unique IP address.
- We recommend only creating new Network Load Balancers with the AWS Load Balancer Controller. Attempting to replace existing Network Load Balancers created with the AWS cloud provider load balancer controller can result in multiple Network Load Balancers that might cause application downtime.

Create a network load balancer

You can create a network load balancer with IP or instance targets.

Create network load balancer — IP Targets

- You can use IP targets with Pods deployed to Amazon EC2 nodes, Fargate, or Amazon EKS Hybrid Nodes. Your Kubernetes service must be created as type `LoadBalancer`. For more information, see Type LoadBalancer in the Kubernetes documentation.

  To create a load balancer that uses IP targets, add the following annotations to a service manifest and deploy your service. The `external` value for `aws-load-balancer-type` is what causes the AWS Load Balancer Controller, rather than the AWS cloud provider load balancer controller, to create the Network Load Balancer. You can view a sample service manifest with the annotations.

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip"
```

> **Note**
>
> If you're load balancing to `IPv6` Pods, add the following annotation. You can only load balance over `IPv6` to IP targets, not instance targets. Without this annotation, load balancing is over `IPv4`.

```
service.beta.kubernetes.io/aws-load-balancer-ip-address-type: dualstack
```

Network Load Balancers are created with the `internal` `aws-load-balancer-scheme`, by default. You can launch Network Load Balancers in any subnet in your cluster's VPC, including subnets that weren't specified when you created your cluster.

Kubernetes examines the route table for your subnets to identify whether they are public or private. Public subnets have a route directly to the internet using an internet gateway, but private subnets do not.

If you want to create a Network Load Balancer in a public subnet to load balance to Amazon EC2 nodes (Fargate can only be private), specify `internet-facing` with the following annotation:

```
service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
```

> **Note**
>
> The `service.beta.kubernetes.io/aws-load-balancer-type: "nlb-ip"` annotation is still supported for backwards compatibility. However, we recommend using the previous annotations for new load balancers instead of `service.beta.kubernetes.io/aws-load-balancer-type: "nlb-ip"`.

> **Important**
>
> Do not edit the annotations after creating your service. If you need to modify it, delete the service object and create it again with the desired value for this annotation.

Create network load balancer — Instance Targets

- The AWS cloud provider load balancer controller creates Network Load Balancers with instance targets only. Version `2.2.0` and later of the AWS Load Balancer Controller also creates Network Load Balancers with instance targets. We recommend using it, rather than the AWS cloud provider load balancer controller, to create new Network Load Balancers. You can use Network Load Balancer instance targets with Pods deployed to Amazon EC2 nodes, but not to Fargate. To load balance network traffic across Pods deployed to Fargate, you must use IP targets.

To deploy a Network Load Balancer to a private subnet, your service specification must have the following annotations. You can view a sample service manifest with the annotations. The `external` value for `aws-load-balancer-type` is what causes the AWS Load Balancer Controller, rather than the AWS cloud provider load balancer controller, to create the Network Load Balancer.

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "instance"
```

Network Load Balancers are created with the `internal` `aws-load-balancer-scheme`, by default. For internal Network Load Balancers, your Amazon EKS cluster must be configured to use at least one private subnet in your VPC. Kubernetes examines the route table for your subnets to identify whether they are public or private. Public subnets have a route directly to the internet using an internet gateway, but private subnets do not.

If you want to create an Network Load Balancer in a public subnet to load balance to Amazon EC2 nodes, specify `internet-facing` with the following annotation:

```
service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
```

Important

Do not edit the annotations after creating your service. If you need to modify it, delete the service object and create it again with the desired value for this annotation.

(Optional) Deploy a sample application

- At least one public or private subnet in your cluster VPC.
- Have the AWS Load Balancer Controller deployed on your cluster. For more information, see the section called " AWS Load Balancer Controller". We recommend version `2.7.2` or later.

1. If you're deploying to Fargate, make sure you have an available private subnet in your VPC and create a Fargate profile. If you're not deploying to Fargate, skip this step. You can create the profile by running the following command or in the AWS Management Console using the same values for `name` and `namespace` that are in the command. Replace the *`example values`* with your own.

```
eksctl create fargateprofile \
    --cluster my-cluster \
    --region region-code \
    --name nlb-sample-app \
    --namespace nlb-sample-app
```

2. Deploy a sample application.

   a. Create a namespace for the application.

   ```
   kubectl create namespace nlb-sample-app
   ```

   b. Save the following contents to a file named `sample-deployment.yaml` file on your computer.

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: nlb-sample-app
     namespace: nlb-sample-app
   spec:
     replicas: 3
     selector:
       matchLabels:
         app: nginx
     template:
       metadata:
         labels:
           app: nginx
       spec:
         containers:
           - name: nginx
             image: public.ecr.aws/nginx/nginx:1.23
             ports:
               - name: tcp
                 containerPort: 80
   ```

   c. Apply the manifest to the cluster.

   ```
   kubectl apply -f sample-deployment.yaml
   ```

3. Create a service with an internet-facing Network Load Balancer that load balances to IP targets.

   a. Save the following contents to a file named `sample-service.yaml` file on your computer. If you're deploying to Fargate nodes, remove the `service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing` line.

```
apiVersion: v1
kind: Service
metadata:
  name: nlb-sample-service
  namespace: nlb-sample-app
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: LoadBalancer
  selector:
    app: nginx
```

   b. Apply the manifest to the cluster.

```
kubectl apply -f sample-service.yaml
```

4. Verify that the service was deployed.

```
kubectl get svc nlb-sample-service -n nlb-sample-app
```

An example output is as follows.

```
NAME            TYPE           CLUSTER-IP         EXTERNAL-IP
                                                PORT(S)        AGE
sample-service  LoadBalancer   10.100.240.137   k8s-nlbsampl-nlbsampl-xxxxxxxxxx-
xxxxxxxxxxxxxxxx.elb.region-code.amazonaws.com  80:32400/TCP   16h
```

> **Note**
>
> The values for *10.100.240.137* and *xxxxxxxxxx*-*xxxxxxxxxxxxxxxx* will be different than the example output (they will be unique to your load balancer) and *us-west-2* may be different for you, depending on which AWS Region that your cluster is in.

5. Open the [Amazon EC2 AWS Management Console](). Select **Target Groups** (under **Load Balancing**) in the left navigation pane. In the **Name** column, select the target group's name where the value in the **Load balancer** column matches a portion of the name in the `EXTERNAL-IP` column of the output in the previous step. For example, you'd select the target group named `k8s-default-samplese-`*xxxxxxxxxx* if your output were the same as the previous output. The **Target type** is `IP` because that was specified in the sample service manifest.

6. Select the **Target group** and then select the **Targets** tab. Under **Registered targets**, you should see three IP addresses of the three replicas deployed in a previous step. Wait until the status of all targets is **healthy** before continuing. It might take several minutes before all targets are `healthy`. The targets might be in an `unhealthy` state before changing to a `healthy` state.

7. Send traffic to the service replacing *xxxxxxxxxx-xxxxxxxxxxxxxxxx* and *us-west-2* with the values returned in the output for a [previous step]() for `EXTERNAL-IP`. If you deployed to a private subnet, then you'll need to view the page from a device within your VPC, such as a bastion host. For more information, see [Linux Bastion Hosts on AWS]().

```
curl k8s-default-samplese-xxxxxxxxxx-xxxxxxxxxxxxxxxx.elb.region-code.amazonaws.com
```

An example output is as follows.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

8. When you're finished with the sample deployment, service, and namespace, remove them.

```
kubectl delete namespace nlb-sample-app
```

Route application and HTTP traffic with Application Load Balancers

> **Note**
>
> **New:** Amazon EKS Auto Mode automates routine tasks for load balancing. For more information, see:
>
> - the section called "Deploy load balancer"
> - the section called "Create ingress class"

When you create a Kubernetes `ingress`, an AWS Application Load Balancer (ALB) is provisioned that load balances application traffic. To learn more, see What is an Application Load Balancer? in the *Application Load Balancers User Guide* and Ingress in the Kubernetes documentation. ALBs can be used with Pods that are deployed to nodes or to AWS Fargate. You can deploy an ALB to public or private subnets.

Application traffic is balanced at `L7` of the OSI model. To load balance network traffic at `L4`, you deploy a Kubernetes `service` of the `LoadBalancer` type. This type provisions an AWS Network Load Balancer. For more information, see the section called "Network load balancing". To learn more about the differences between the two types of load balancing, see Elastic Load Balancing features on the AWS website.

Prerequisites

Before you can load balance application traffic to an application, you must meet the following requirements.

- Have an existing cluster. If you don't have an existing cluster, see *Get started*. If you need to update the version of an existing cluster, see the section called "Update Kubernetes version".
- Have the AWS Load Balancer Controller deployed on your cluster. For more information, see the section called " AWS Load Balancer Controller". We recommend version `2.7.2` or later.

- At least two subnets in different Availability Zones. The AWS Load Balancer Controller chooses one subnet from each Availability Zone. When multiple tagged subnets are found in an Availability Zone, the controller chooses the subnet whose subnet ID comes first lexicographically. Each subnet must have at least eight available IP addresses.

  If you're using multiple security groups attached to worker node, exactly one security group must be tagged as follows. Replace *my-cluster* with your cluster name.

  - **Key** – `kubernetes.io/cluster/<my-cluster>`
  - **Value** – `shared` or `owned`
- If you're using the AWS Load Balancer Controller version `2.1.1` or earlier, subnets must be tagged in the format that follows. If you're using version `2.1.2` or later, tagging is optional. However, we recommend that you tag a subnet if any of the following is the case. You have multiple clusters that are running in the same VPC, or have multiple AWS services that share subnets in a VPC. Or, you want more control over where load balancers are provisioned for each cluster. Replace *my-cluster* with your cluster name.
  - **Key** – `kubernetes.io/cluster/<my-cluster>`
  - **Value** – `shared` or `owned`
- Your public and private subnets must meet the following requirements. This is unless you explicitly specify subnet IDs as an annotation on a service or ingress object. Assume that you provision load balancers by explicitly specifying subnet IDs as an annotation on a service or ingress object. In this situation, Kubernetes and the AWS load balancer controller use those subnets directly to create the load balancer and the following tags aren't required.
  - **Private subnets** – Must be tagged in the following format. This is so that Kubernetes and the AWS load balancer controller know that the subnets can be used for internal load balancers. If you use `eksctl` or an Amazon EKS AWS CloudFormation template to create your VPC after March 26, 2020, the subnets are tagged appropriately when created. For more information about the Amazon EKS AWS CloudFormation VPC templates, see the section called "Create a VPC".
    - **Key** – `kubernetes.io/role/internal-elb`
    - **Value** – `1`
  - **Public subnets** – Must be tagged in the following format. This is so that Kubernetes knows to use only the subnets that were specified for external load balancers. This way, Kubernetes doesn't choose a public subnet in each Availability Zone (lexicographically based on their subnet ID). If you use `eksctl` or an Amazon EKS AWS CloudFormation template to create your VPC after March 26, 2020, the subnets are tagged appropriately when created. For more

information about the Amazon EKS AWS CloudFormation VPC templates, see the section called "Create a VPC".

- **Key** – kubernetes.io/role/elb
- **Value** – 1

If the subnet role tags aren't explicitly added, the Kubernetes service controller examines the route table of your cluster VPC subnets. This is to determine if the subnet is private or public. We recommend that you don't rely on this behavior. Rather, explicitly add the private or public role tags. The AWS Load Balancer Controller doesn't examine route tables. It also requires the private and public tags to be present for successful auto discovery.

- The AWS Load Balancer Controller creates ALBs and the necessary supporting AWS resources whenever a Kubernetes ingress resource is created on the cluster with the `kubernetes.io/ingress.class: alb` annotation. The ingress resource configures the ALB to route HTTP or HTTPS traffic to different Pods within the cluster. To ensure that your ingress objects use the AWS Load Balancer Controller, add the following annotation to your Kubernetes ingress specification. For more information, see Ingress specification on GitHub.

```
annotations:
    kubernetes.io/ingress.class: alb
```

> **Note**
>
> If you're load balancing to IPv6 Pods, add the following annotation to your ingress spec. You can only load balance over IPv6 to IP targets, not instance targets. Without this annotation, load balancing is over IPv4.

```
alb.ingress.kubernetes.io/ip-address-type: dualstack
```

- The AWS Load Balancer Controller supports the following traffic modes:
  - **Instance** – Registers nodes within your cluster as targets for the ALB. Traffic reaching the ALB is routed to `NodePort` for your service and then proxied to your Pods. This is the default traffic mode. You can also explicitly specify it with the `alb.ingress.kubernetes.io/target-type: instance` annotation.

> **Note**
>
> Your Kubernetes service must specify the `NodePort` or "LoadBalancer" type to use this traffic mode.

  - **IP** – Registers Pods as targets for the ALB. Traffic reaching the ALB is directly routed to Pods for your service. You must specify the `alb.ingress.kubernetes.io/target-type: ip` annotation to use this traffic mode. The IP target type is required when target Pods are running on Fargate or Amazon EKS Hybrid Nodes.
- To tag ALBs created by the controller, add the following annotation to the controller: `alb.ingress.kubernetes.io/tags`. For a list of all available annotations supported by the AWS Load Balancer Controller, see Ingress annotations on GitHub.
- Upgrading or downgrading the ALB controller version can introduce breaking changes for features that rely on it. For more information about the breaking changes that are introduced in each release, see the ALB controller release notes on GitHub.

Reuse ALBs with Ingress Groups

You can share an application load balancer across multiple service resources using `IngressGroups`.

To join an ingress to a group, add the following annotation to a Kubernetes ingress resource specification.

```
alb.ingress.kubernetes.io/group.name: my-group
```

The group name must:

- Be 63 or fewer characters in length.
- Consist of lower case letters, numbers, `-`, and `.`
- Start and end with a letter or number.

The controller automatically merges ingress rules for all ingresses in the same ingress group. It supports them with a single ALB. Most annotations that are defined on an ingress only apply to the paths defined by that ingress. By default, ingress resources don't belong to any ingress group.

> **Warning**
>
> **Potential security risk**
>
> Specify an ingress group for an ingress only when all the Kubernetes users that have RBAC permission to create or modify ingress resources are within the same trust boundary.
>
> If you add the annotation with a group name, other Kubernetes users might create or modify their ingresses to belong to the same ingress group. Doing so can cause undesirable behavior, such as overwriting existing rules with higher priority rules.

You can add an order number of your ingress resource.

```
alb.ingress.kubernetes.io/group.order: '10'
```

The number can be 1-1000. The lowest number for all ingresses in the same ingress group is evaluated first. All ingresses without this annotation are evaluated with a value of zero. Duplicate rules with a higher number can overwrite rules with a lower number. By default, the rule order between ingresses within the same ingress group is determined lexicographically based namespace and name.

> **Important**
>
> Ensure that each ingress in the same ingress group has a unique priority number. You can't have duplicate order numbers across ingresses.

(Optional) Deploy a sample application

- At least one public or private subnet in your cluster VPC.
- Have the AWS Load Balancer Controller deployed on your cluster. For more information, see the section called " AWS Load Balancer Controller". We recommend version `2.7.2` or later.

You can run the sample application on a cluster that has Amazon EC2 nodes, Fargate Pods, or both.

1. If you're not deploying to Fargate, skip this step. If you're deploying to Fargate, create a Fargate profile. You can create the profile by running the following command or in the AWS

Management Console using the same values for name and namespace that are in the command. Replace the *example values* with your own.

```
eksctl create fargateprofile \
    --cluster my-cluster \
    --region region-code \
    --name alb-sample-app \
    --namespace game-2048
```

2. Deploy the game 2048 as a sample application to verify that the AWS Load Balancer Controller creates an AWS ALB as a result of the ingress object. Complete the steps for the type of subnet you're deploying to.

   a. If you're deploying to Pods in a cluster that you created with the IPv6 family, skip to the next step.

      - **Public**::

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-
balancer-controller/v2.11.0/docs/examples/2048/2048_full.yaml
```

      - **Private**::

        A. Download the manifest.

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/v2.11.0/docs/examples/2048/2048_full.yaml
```

        B. Edit the file and find the line that says alb.ingress.kubernetes.io/scheme: internet-facing.

        C. Change *internet-facing* to internal and save the file.

        D. Apply the manifest to your cluster.

```
kubectl apply -f 2048_full.yaml
```

   b. If you're deploying to Pods in a cluster that you created with the IPv6 family, complete the following steps.

      i. Download the manifest.

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/v2.11.0/docs/examples/2048/2048_full.yaml
```

ii. Open the file in an editor and add the following line to the annotations in the ingress spec.

```
alb.ingress.kubernetes.io/ip-address-type: dualstack
```

iii. If you're load balancing to internal Pods, rather than internet facing Pods, change the line that says `alb.ingress.kubernetes.io/scheme: internet-facing` to `alb.ingress.kubernetes.io/scheme: internal`

iv. Save the file.

v. Apply the manifest to your cluster.

```
kubectl apply -f 2048_full.yaml
```

3. After a few minutes, verify that the ingress resource was created with the following command.

```
kubectl get ingress/ingress-2048 -n game-2048
```

An example output is as follows.

```
NAME           CLASS    HOSTS   ADDRESS
                        PORTS   AGE
ingress-2048   <none>   *       k8s-game2048-ingress2-xxxxxxxxxx-yyyyyyyyyy.region-
code.elb.amazonaws.com   80      2m32s
```

Note

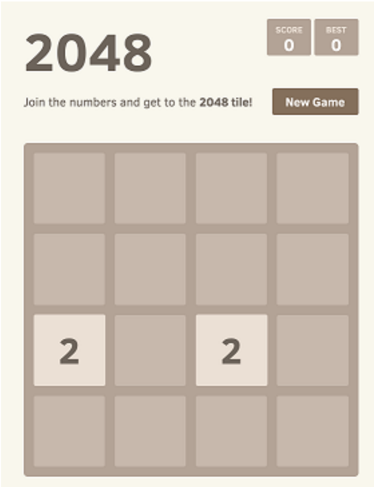
If you created the load balancer in a private subnet, the value under ADDRESS in the previous output is prefaced with `internal-`.

If your ingress wasn't successfully created after several minutes, run the following command to view the AWS Load Balancer Controller logs. These logs might contain error messages that you can use to diagnose issues with your deployment.

```
kubectl logs -f -n kube-system -l app.kubernetes.io/instance=aws-load-balancer-
controller
```

1. If you deployed to a public subnet, open a browser and navigate to the ADDRESS URL from the previous command output to see the sample application. If you don't see anything, refresh your

browser and try again. If you deployed to a private subnet, then you'll need to view the page from a device within your VPC, such as a bastion host. For more information, see Linux Bastion Hosts on AWS.



2. When you finish experimenting with your sample application, delete it by running one of the the following commands.

   - If you applied the manifest, rather than applying a copy that you downloaded, use the following command.

   ```
   kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/examples/2048/2048_full.yaml
   ```

   - If you downloaded and edited the manifest, use the following command.

   ```
   kubectl delete -f 2048_full.yaml
   ```

Restrict external IP addresses that can be assigned to services

Kubernetes services can be reached from inside of a cluster through:

- A cluster IP address that is assigned automatically by Kubernetes
- Any IP address that you specify for the `externalIPs` property in a service spec. External IP addresses are not managed by Kubernetes and are the responsibility of the cluster administrator. External IP addresses specified with `externalIPs` are different than the external IP address assigned to a service of type `LoadBalancer` by a cloud provider.

To learn more about Kubernetes services, see Service in the Kubernetes documentation. You can restrict the IP addresses that can be specified for `externalIPs` in a service spec.

1. Deploy `cert-manager` to manage webhook certificates. For more information, see the cert-manager documentation.

```
kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.5.4/
cert-manager.yaml
```

2. Verify that the `cert-manager` Pods are running.

```
kubectl get pods -n cert-manager
```

An example output is as follows.

```
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-58c8844bb8-nlx7q              1/1     Running   0          15s
cert-manager-cainjector-745768f6ff-696h5   1/1     Running   0          15s
cert-manager-webhook-67cc76975b-4v4nk      1/1     Running   0          14s
```

3. Review your existing services to ensure that none of them have external IP addresses assigned to them that aren't contained within the CIDR block you want to limit addresses to.

```
kubectl get services -A
```

An example output is as follows.

```
NAMESPACE                      NAME                                      TYPE
 CLUSTER-IP       EXTERNAL-IP     PORT(S)         AGE
cert-manager                   cert-manager                              ClusterIP
 10.100.102.137   <none>          9402/TCP        20m
cert-manager                   cert-manager-webhook                      ClusterIP
 10.100.6.136     <none>          443/TCP         20m
default                        kubernetes                                ClusterIP
 10.100.0.1       <none>          443/TCP         2d1h
externalip-validation-system   externalip-validation-webhook-service     ClusterIP
 10.100.234.179   <none>          443/TCP         16s
kube-system                    kube-dns                                  ClusterIP
 10.100.0.10      <none>          53/UDP,53/TCP   2d1h
```

```
my-namespace                    my-service                              ClusterIP
 10.100.128.10     192.168.1.1     80/TCP          149m
```

If any of the values are IP addresses that are not within the block you want to restrict access to, you'll need to change the addresses to be within the block, and redeploy the services. For example, the `my-service` service in the previous output has an external IP address assigned to it that isn't within the CIDR block example in step 5.

4. Download the external IP webhook manifest. You can also view the source code for the webhook on GitHub.

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/docs/externalip-webhook.yaml
```

5. Specify CIDR blocks. Open the downloaded file in your editor and remove the \# at the start of the following lines.

```
#args:
#- --allowed-external-ip-cidrs=10.0.0.0/8
```

Replace `10.0.0.0/8` with your own CIDR block. You can specify as many blocks as you like. If specifying mutiple blocks, add a comma between blocks.

6. If your cluster is not in the `us-west-2` AWS Region, then replace `us-west-2`, `602401143452`, and `amazonaws.com` in the file with the following commands. Before running the commands, replace *`region-code`* and *`111122223333`* with the value for your AWS Region from the list in View Amazon container image registries for Amazon EKS add-ons.

```
sed -i.bak -e 's|602401143452|111122223333|' externalip-webhook.yaml
sed -i.bak -e 's|us-west-2|region-code|' externalip-webhook.yaml
sed -i.bak -e 's|amazonaws.com||' externalip-webhook.yaml
```

7. Apply the manifest to your cluster.

```
kubectl apply -f externalip-webhook.yaml
```

An attempt to deploy a service to your cluster with an IP address specified for `externalIPs` that is not contained in the blocks that you specified in the Specify CIDR blocks step will fail.

Copy a container image from one repository to another repository

This topic describes how to pull a container image from a repository that your nodes don't have access to and push the image to a repository that your nodes have access to. You can push the image to Amazon ECR or an alternative repository that your nodes have access to.

- The Docker engine installed and configured on your computer. For instructions, see Install Docker Engine in the Docker documentation.
- Version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version, use `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*. The AWS CLI version that is installed in AWS CloudShell might also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the *AWS CloudShell User Guide*.
- An interface VPC endpoint for Amazon ECR if you want your nodes to pull container images from or push container images to a private Amazon ECR repository over Amazon's network. For more information, see Create the VPC endpoints for Amazon ECR in the Amazon Elastic Container Registry User Guide.

Complete the following steps to pull a container image from a repository and push it to your own repository. In the following examples that are provided in this topic, the image for the Amazon VPC CNI plugin for Kubernetes metrics helper is pulled. When you follow these steps, make sure to replace the *`example values`* with your own values.

1. If you don't already have an Amazon ECR repository or another repository, then create one that your nodes have access to. The following command creates an Amazon ECR private repository. An Amazon ECR private repository name must start with a letter. It can only contain lowercase letters, numbers, hyphens (-), underscores (_), and forward slashes (/). For more information, see Creating a private repository in the Amazon Elastic Container Registry User Guide.

   You can replace *`cni-metrics-helper`* with whatever you choose. As a best practice, create a separate repository for each image. We recommend this because image tags must be unique within a repository. Replace *`region-code`* with an AWS Region supported by Amazon ECR.

```
aws ecr create-repository --region region-code --repository-name cni-metrics-helper
```

2. Determine the registry, repository, and tag (optional) of the image that your nodes need to pull. This information is in the `registry/repository[:tag]` format.

   Many of the Amazon EKS topics about installing images require that you apply a manifest file or install the image using a Helm chart. However, before you apply a manifest file or install a Helm chart, first view the contents of the manifest or chart's `values.yaml` file. That way, you can determine the registry, repository, and tag to pull.

   For example, you can find the following line in the manifest file for the Amazon VPC CNI plugin for Kubernetes metrics helper. The registry is `602401143452.dkr.ecr.us-west-2.amazonaws.com`, which is an Amazon ECR private registry. The repository is `cni-metrics-helper`.

```
image: "602401143452.dkr.ecr.us-west-2.amazonaws.com/cni-metrics-helper:v1.12.6"
```

   You may see the following variations for an image location:

   - Only `repository-name:tag`. In this case, `docker.io` is usually the registry, but not specified since Kubernetes prepends it to a repository name by default if no registry is specified.
   - `repository-name/repository-namespace/repository:tag`. A repository namespace is optional, but is sometimes specified by the repository owner for categorizing images. For example, all Amazon EC2 images in the Amazon ECR Public Gallery use the `aws-ec2` namespace.

     Before installing an image with Helm, view the Helm `values.yaml` file to determine the image location. For example, the values.yaml file for the Amazon VPC CNI plugin for Kubernetes metrics helper includes the following lines.

```
image:
  region: us-west-2
  tag: v1.12.6
  account: "602401143452"
  domain: "amazonaws.com"
```

3. Pull the container image specified in the manifest file.

a. If you're pulling from a public registry, such as the Amazon ECR Public Gallery, you can skip to the next sub-step, because authentication isn't required. In this example, you authenticate to an Amazon ECR private registry that contains the repository for the CNI metrics helper image. Amazon EKS maintains the image in each registry listed in View Amazon container image registries for Amazon EKS add-ons. You can authenticate to any of the registries by replacing *602401143452* and *region-code* with the information for a different registry. A separate registry exists for each AWS Region that Amazon EKS is supported in.

```
aws ecr get-login-password --region region-code | docker login --username AWS --
password-stdin 602401143452.dkr.ecr.region-code.amazonaws.com
```

b. Pull the image. In this example, you pull from the registry that you authenticated to in the previous sub-step. Replace *602401143452* and *region-code* with the information that you provided in the previous sub-step.

```
docker pull 602401143452.dkr.ecr.region-code.amazonaws.com/cni-metrics-
helper:v1.12.6
```

4. Tag the image that you pulled with your registry, repository, and tag. The following example assumes that you pulled the image from the manifest file and are going to push it to the Amazon ECR private repository that you created in the first step. Replace *111122223333* with your account ID. Replace *region-code* with the AWS Region that you created your Amazon ECR private repository in.

```
docker tag cni-metrics-helper:v1.12.6 111122223333.dkr.ecr.region-code.amazonaws.com/
cni-metrics-helper:v1.12.6
```

5. Authenticate to your registry. In this example, you authenticate to the Amazon ECR private registry that you created in the first step. For more information, see Registry authentication in the Amazon Elastic Container Registry User Guide.

```
aws ecr get-login-password --region region-code | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.region-code.amazonaws.com
```

6. Push the image to your repository. In this example, you push the image to the Amazon ECR private repository that you created in the first step. For more information, see Pushing a Docker image in the Amazon Elastic Container Registry User Guide.

```
docker push 111122223333.dkr.ecr.region-code.amazonaws.com/cni-metrics-helper:v1.12.6
```

7. Update the manifest file that you used to determine the image in a previous step with the `registry/repository:tag` for the image that you pushed. If you're installing with a Helm chart, there's often an option to specify the `registry/repository:tag`. When installing the chart, specify the `registry/repository:tag` for the image that you pushed to your repository.

View Amazon container image registries for Amazon EKS add-ons

When you deploy AWS Amazon EKS add-ons to your cluster, your nodes pull the required container images from the registry specified in the installation mechanism for the add-on, such as an installation manifest or a Helm `values.yaml` file. The images are pulled from an Amazon EKS Amazon ECR private repository. Amazon EKS replicates the images to a repository in each Amazon EKS supported AWS Region. Your nodes can pull the container image over the internet from any of the following registries. Alternatively, your nodes can pull the image over Amazon's network if you created an interface VPC endpoint for Amazon ECR (AWS PrivateLink) in your VPC. The registries require authentication with an AWS IAM account. Your nodes authenticate using the Amazon EKS node IAM role, which has the permissions in the AmazonEC2ContainerRegistryReadOnly managed IAM policy associated to it.

AWS Region	Registry
af-south-1	877085696533.dkr.ecr.af-south-1.amazonaws.com
ap-east-1	800184023465.dkr.ecr.ap-east-1.amazonaws.com
ap-southeast-3	296578399912.dkr.ecr.ap-southeast-3.amazonaws.com
ap-south-2	900889452093.dkr.ecr.ap-south-2.amazonaws.com
ap-southeast-4	491585149902.dkr.ecr.ap-southeast-4.amazonaws.com

AWS Region	Registry
ap-south-1	602401143452.dkr.ecr.ap-south-1.amazonaws.com
ap-northeast-3	602401143452.dkr.ecr.ap-northeast-3.amazonaws.com
ap-northeast-2	602401143452.dkr.ecr.ap-northeast-2.amazonaws.com
ap-southeast-1	602401143452.dkr.ecr.ap-southeast-1.amazonaws.com
ap-southeast-2	602401143452.dkr.ecr.ap-southeast-2.amazonaws.com
ap-southeast-7	121268973566.dkr.ecr.ap-southeast-7.amazonaws.com
ap-northeast-1	602401143452.dkr.ecr.ap-northeast-1.amazonaws.com
cn-north-1	918309763551.dkr.ecr.cn-north-1.amazonaws.com.cn
cn-northwest-1	961992271922.dkr.ecr.cn-northwest-1.amazonaws.com.cn
eu-central-1	602401143452.dkr.ecr.eu-central-1.amazonaws.com
eu-west-1	602401143452.dkr.ecr.eu-west-1.amazonaws.com
eu-west-2	602401143452.dkr.ecr.eu-west-2.amazonaws.com
eu-south-1	590381155156.dkr.ecr.eu-south-1.amazonaws.com

AWS Region	Registry
eu-west-3	602401143452.dkr.ecr.eu-west-3.amazonaws.com
eu-south-2	455263428931.dkr.ecr.eu-south-2.amazonaws.com
eu-north-1	602401143452.dkr.ecr.eu-north-1.amazonaws.com
eu-central-2	900612956339.dkr.ecr.eu-central-2.amazonaws.com
il-central-1	066635153087.dkr.ecr.il-central-1.amazonaws.com
mx-central-1	730335286997.dkr.ecr.mx-central-1.amazonaws.com
me-south-1	558608220178.dkr.ecr.me-south-1.amazonaws.com
me-central-1	759879836304.dkr.ecr.me-central-1.amazonaws.com
us-east-1	602401143452.dkr.ecr.us-east-1.amazonaws.com
us-east-2	602401143452.dkr.ecr.us-east-2.amazonaws.com
us-west-1	602401143452.dkr.ecr.us-west-1.amazonaws.com
us-west-2	602401143452.dkr.ecr.us-west-2.amazonaws.com
ca-central-1	602401143452.dkr.ecr.ca-central-1.amazonaws.com

AWS Region	Registry
ca-west-1	761377655185.dkr.ecr.ca-west-1.amazonaws.com
sa-east-1	602401143452.dkr.ecr.sa-east-1.amazonaws.com
us-gov-east-1	151742754352.dkr.ecr.us-gov-east-1.amazonaws.com
us-gov-west-1	013241004608.dkr.ecr.us-gov-west-1.amazonaws.com

Amazon EKS add-ons

An add-on is software that provides supporting operational capabilities to Kubernetes applications, but is not specific to the application. This includes software like observability agents or Kubernetes drivers that allow the cluster to interact with underlying AWS resources for networking, compute, and storage. Add-on software is typically built and maintained by the Kubernetes community, cloud providers like AWS, or third-party vendors. Amazon EKS automatically installs self-managed add-ons such as the Amazon VPC CNI plugin for Kubernetes, `kube-proxy`, and CoreDNS for every cluster. Note that the VPC CNI add-on isn't compatible with Amazon EKS Hybrid Nodes and doesn't deploy to hybrid nodes. You can change the default configuration of the add-ons and update them when desired.

Amazon EKS add-ons provide installation and management of a curated set of add-ons for Amazon EKS clusters. All Amazon EKS add-ons include the latest security patches, bug fixes, and are validated by AWS to work with Amazon EKS. Amazon EKS add-ons allow you to consistently ensure that your Amazon EKS clusters are secure and stable and reduce the amount of work that you need to do in order to install, configure, and update add-ons. If a self-managed add-on, such as `kube-proxy` is already running on your cluster and is available as an Amazon EKS add-on, then you can install the `kube-proxy` Amazon EKS add-on to start benefiting from the capabilities of Amazon EKS add-ons.

You can update specific Amazon EKS managed configuration fields for Amazon EKS add-ons through the Amazon EKS API. You can also modify configuration fields not managed by Amazon EKS directly within the Kubernetes cluster once the add-on starts. This includes defining specific configuration fields for an add-on where applicable. These changes are not overridden by Amazon EKS once they are made. This is made possible using the Kubernetes server-side apply feature. For more information, see the section called "Fields you can customize".

You can use Amazon EKS add-ons with any Amazon EKS node type. For more information, see *Manage compute*.

You can add, update, or delete Amazon EKS add-ons using the Amazon EKS API, AWS Management Console, AWS CLI, and `eksctl`. You can also create Amazon EKS add-ons using AWS CloudFormation.

Considerations

Consider the following when you use Amazon EKS add-ons:

- To configure add-ons for the cluster your IAM principal must have IAM permissions to work with add-ons. For more information, see the actions with `Addon` in their name in Actions defined by Amazon Elastic Kubernetes Service.
- Amazon EKS add-ons run on the nodes that you provision or configure for your cluster. Node types include Amazon EC2 instances, Fargate, and hybrid nodes.
- You can modify fields that aren't managed by Amazon EKS to customize the installation of an Amazon EKS add-on. For more information, see the section called "Fields you can customize".
- If you create a cluster with the AWS Management Console, the Amazon EKS `kube-proxy`, Amazon VPC CNI plugin for Kubernetes, and CoreDNS Amazon EKS add-ons are automatically added to your cluster. If you use `eksctl` to create your cluster with a `config` file, `eksctl` can also create the cluster with Amazon EKS add-ons. If you create your cluster using `eksctl` without a `config` file or with any other tool, the self-managed `kube-proxy`, Amazon VPC CNI plugin for Kubernetes, and CoreDNS add-ons are installed, rather than the Amazon EKS add-ons. You can either manage them yourself or add the Amazon EKS add-ons manually after cluster creation. Regardless of the method that you use to create your cluster, the VPC CNI add-on doesn't install on hybrid nodes.
- The `eks:addon-cluster-admin` `ClusterRoleBinding` binds the `cluster-admin` `ClusterRole` to the `eks:addon-manager` Kubernetes identity. The role has the necessary permissions for the `eks:addon-manager` identity to create Kubernetes namespaces and install add-ons into namespaces. If the `eks:addon-cluster-admin` `ClusterRoleBinding` is removed, the Amazon EKS cluster will continue to function, however Amazon EKS is no longer

able to manage any add-ons. All clusters starting with the following platform versions use the new `ClusterRoleBinding`.

- A subset of EKS add-ons from AWS have been validated for compatibility with Amazon EKS Hybrid Nodes. For more information, see the compatibility table on the section called " AWS Add-ons".

Required platform version

Review the table to determine the minimum required platform version to use this feature with your cluster. You can use the listed platform version, or a more recent platform version. For example, if the table lists "eks.14" you can use platform version "eks.15". For more information, see the section called "Platform versions".

Kubernetes version	EKS platform version
1.25 or newer	All platform versions
1.20	eks.12
1.21	eks.14
1.22	eks.9
1.23	eks.5
1.24	eks.3

Considerations for Amazon EKS Auto Mode

Amazon EKS Auto mode includes capabilities that deliver essential cluster functionality, including:

- Pod networking
- Service networking
- Cluster DNS
- Autoscaling
- Block storage
- Load balancer controller

- Pod Identity agent
- Node monitoring agent

With Auto mode compute, many commonly used EKS add-ons become redundant, such as:

- Amazon VPC CNI
- kube-proxy
- CoreDNS
- Amazon EBS CSI Driver
- EKS Pod Identity Agent

However, if your cluster combines Auto mode with other compute options like self-managed EC2 instances, Managed Node Groups, or AWS Fargate, these add-ons remain necessary. AWS has enhanced EKS add-ons with anti-affinity rules that automatically ensure add-on pods are scheduled only on supported compute types. Furthermore, users can now leverage the EKS add-ons `DescribeAddonVersions` API to verify the supported computeTypes for each add-on and its specific versions. Additionally, with EKS Auto mode, the controllers listed above run on AWS owned infrastructure. So, you many not even see them in your accounts unless you are using EKS auto mode with other types of compute in which case, you will see the controllers you installed on your cluster.

If you are planning to enable EKS Auto Mode on an existing cluster, you may need to upgrade the version of certain addons. For more information, see the section called “Required Add-on Versions” for EKS Auto Mode.

Support

AWS publishes multiple types of add-ons with different levels of support.

- **AWS Add-ons:** These add-ons are built and fully supported by AWS.
  - Use an AWS add-on to work with other AWS services, such as Amazon EFS.
  - For more information, see the section called “ AWS Add-ons”.
- **AWS Marketplace Add-ons:** These add-ons are scanned by AWS and supported by an independent AWS partner.
  - Use a marketplace add-on to add valuable and sophisticated features to your cluster, such as monitoring with Splunk.

  - For more information, see the section called “Marketplace add-ons”.
- **Community Add-ons**: These add-ons are scanned by AWS but supported by the open source community.
  - Use a community add-on to reduce the complexity of installing common open source software, such as Kubernetes Metrics Server.
  - Community add-ons are built by AWS. AWS only validates community add-ons for version compatibility.
  - For more information, see the section called “Community add-ons”.

The following table details the scope of support for each add-on type:

Category	Feature	AWS add-ons	AWS Marketplace add-ons	Community add-ons
Development	Built by AWS	Yes	No	Yes
Development	Validated by AWS	Yes	No	Yes*
Development	Validated by AWS Partner	No	Yes	No
Maintenance	Scanned by AWS	Yes	Yes	Yes
Maintenance	Patched by AWS	Yes	No	Yes
Maintenance	Patched by AWS Partner	No	Yes	No
Distribution	Published by AWS	Yes	No	Yes
Distribution	Published by AWS Partner	No	Yes	No
Support	Basic Install Support by AWS	Yes	Yes	Yes

Category	Feature	AWS add-ons	AWS Marketplace add-ons	Community add-ons
Support	Full AWS Support	Yes	No	No
Support	Full AWS Partner Support	No	Yes	No

*: Validation for community add-ons only includes Kubernetes version compatibility. For example, if you install a community add-on on a cluster, AWS checks if it is compatible with the Kubernetes version of your cluster.

AWS Marketplace add-ons can download additional software dependencies from external sources outside of AWS. These external dependencies are not scanned or validated by AWS. Consider your security requirements when deploying AWS Marketplace add-ons that fetch external dependencies.

AWS Add-ons

The following Amazon EKS add-ons are available to create on your cluster. You can view the most current list of available add-ons using `eksctl`, the AWS Management Console, or the AWS CLI. To see all available add-ons or to install an add-on, see the section called "Create an add-on". If an add-on requires IAM permissions, then you must have an IAM OpenID Connect (OIDC) provider for your cluster. To determine whether you have one, or to create one, see the section called "IAM OIDC provider". You can an create or delete an add-on after you've installed it. For more information, see the section called "Update an add-on" or the section called "Remove an add-on". For more information about considerations specific to running EKS add-ons with Amazon EKS Hybrid Nodes, see the section called "Configure add-ons".

You can use any of the following Amazon EKS add-ons.

Description	Learn more	Compatible compute types
Provide native VPC networking for your cluster	the section called "Amazon VPC CNI plugin for Kubernetes"	EC2

Description	Learn more	Compatible compute types
A flexible, extensible DNS server that can serve as the Kubernetes cluster DNS	the section called "CoreDNS"	EC2, Fargate, EKS Auto Mode, Amazon EKS Hybrid Nodes
Maintain network rules on each Amazon EC2 node	the section called "`Kube-proxy`"	EC2, Amazon EKS Hybrid Nodes
Provide Amazon EBS storage for your cluster	the section called "Amazon EBS CSI driver"	EC2
Provide Amazon EFS storage for your cluster	the section called "Amazon EFS CSI driver"	EC2, EKS Auto Mode
Provide Amazon S3 storage for your cluster	the section called "Mountpoint for Amazon S3 CSI Driver"	EC2, EKS Auto Mode
Detect additional node health issues	the section called "Node monitoring agent"	EC2
Enable the use of snapshot functionality in compatible CSI drivers, such as the Amazon EBS CSI driver	the section called "CSI snapshot controller"	EC2, Fargate, EKS Auto Mode, Amazon EKS Hybrid Nodes
SageMaker HyperPod task governance optimizes compute resource allocation and usage across teams in Amazon EKS clusters, addressing inefficiencies in task prioritization and resource sharing.	the section called "Amazon SageMaker HyperPod task governance"	EC2, EKS Auto Mode,

Description	Learn more	Compatible compute types
A Kubernetes agent that collects and reports network flow data to Amazon CloudWatch, enabling comprehensive monitoring of TCP connections across cluster nodes.	the section called “AWS Network Flow Monitor Agent”	EC2, EKS Auto Mode
Secure, production-ready, AWS supported distribution of the OpenTelemetry project	the section called “AWS Distro for OpenTelemetry”	EC2, Fargate, EKS Auto Mode, Amazon EKS Hybrid Nodes
Security monitoring service that analyzes and processes foundational data sources including AWS CloudTrai l management events and Amazon VPC flow logs. Amazon GuardDuty also processes features, such as Kubernetes audit logs and runtime monitoring	the section called “Amazon GuardDuty agent”	EC2, EKS Auto Mode
Monitoring and observability service provided by AWS. This add-on installs the CloudWatc h Agent and enables both CloudWatch Application Signals and CloudWatc h Container Insights with enhanced observability for Amazon EKS	the section called “Amazon CloudWatch Observability agent”	EC2, EKS Auto Mode, Amazon EKS Hybrid Nodes

Description	Learn more	Compatible compute types
Ability to manage credentials for your applications, similar to the way that EC2 instance profiles provide credentials to EC2 instances	the section called “EKS Pod Identity Agent”	EC2, Amazon EKS Hybrid Nodes

Amazon VPC CNI plugin for Kubernetes

The Amazon VPC CNI plugin for Kubernetes Amazon EKS add-on is a Kubernetes container network interface (CNI) plugin that provides native VPC networking for your cluster. The self-managed or managed type of this add-on is installed on each Amazon EC2 node, by default. For more information, see Kubernetes container network interface (CNI) plugin.

> **Note**
>
> You do not need to install this add-on on Amazon EKS Auto Mode clusters. For more information, see the section called “Considerations for Amazon EKS Auto Mode”.

The Amazon EKS add-on name is `vpc-cni`.

Required IAM permissions

This add-on uses the IAM roles for service accounts capability of Amazon EKS. For more information, see the section called “Credentials with IRSA”.

If your cluster uses the `IPv4` family, the permissions in the AmazonEKS_CNI_Policy are required. If your cluster uses the `IPv6` family, you must create an IAM policy with the permissions in IPv6 mode. You can create an IAM role, attach one of the policies to it, and annotate the Kubernetes service account used by the add-on with the following command.

Replace *`my-cluster`* with the name of your cluster and *`AmazonEKSVPCCNIRole`* with the name for your role. If your cluster uses the `IPv6` family, then replace *`AmazonEKS_CNI_Policy`* with the name of the policy that you created. This command requires that you have eksctl installed on your device. If you need to use a different tool to create the role, attach the policy to it, and annotate the Kubernetes service account, see the section called “Assign IAM role”.

```
eksctl create iamserviceaccount --name aws-node --namespace kube-system --cluster my-
cluster --role-name AmazonEKSVPCCNIRole \
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy --
approve
```

Update information

You can only update one minor version at a time. For example, if your current version is `1.28.x-eksbuild.y` and you want to update to `1.30.x-eksbuild.y`, then you must update your current version to `1.29.x-eksbuild.y` and then update it again to `1.30.x-eksbuild.y`. For more information about updating the add-on, see the section called "Update (EKS add-on)".

CoreDNS

The CoreDNS Amazon EKS add-on is a flexible, extensible DNS server that can serve as the Kubernetes cluster DNS. The self-managed or managed type of this add-on was installed, by default, when you created your cluster. When you launch an Amazon EKS cluster with at least one node, two replicas of the CoreDNS image are deployed by default, regardless of the number of nodes deployed in your cluster. The CoreDNS Pods provide name resolution for all Pods in the cluster. You can deploy the CoreDNS Pods to Fargate nodes if your cluster includes a Fargate profile with a namespace that matches the namespace for the CoreDNS deployment. For more information, see the section called "Define profiles"

> **Note**
>
> You do not need to install this add-on on Amazon EKS Auto Mode clusters. For more information, see the section called "Considerations for Amazon EKS Auto Mode".

The Amazon EKS add-on name is `coredns`.

Required IAM permissions

This add-on doesn't require any permissions.

Additional information

To learn more about CoreDNS, see Using CoreDNS for Service Discovery and Customizing DNS Service in the Kubernetes documentation.

Kube-proxy

The Kube-proxy Amazon EKS add-on maintains network rules on each Amazon EC2 node. It enables network communication to your Pods. The self-managed or managed type of this add-on is installed on each Amazon EC2 node in your cluster, by default.

> **Note**
>
> You do not need to install this add-on on Amazon EKS Auto Mode clusters. For more information, see the section called "Considerations for Amazon EKS Auto Mode".

The Amazon EKS add-on name is kube-proxy.

Required IAM permissions

This add-on doesn't require any permissions.

Update information

Before updating your current version, consider the following requirements:

- Kube-proxy on an Amazon EKS cluster has the same compatibility and skew policy as Kubernetes.

Additional information

To learn more about kube-proxy, see kube-proxy in the Kubernetes documentation.

Amazon EBS CSI driver

The Amazon EBS CSI driver Amazon EKS add-on is a Kubernetes Container Storage Interface (CSI) plugin that provides Amazon EBS storage for your cluster.

> **Note**
>
> You do not need to install this add-on on Amazon EKS Auto Mode clusters. Auto Mode includes a block storage capability. For more information, see the section called "Deploy stateful workload".

The Amazon EKS add-on name is `aws-ebs-csi-driver`.

Required IAM permissions

This add-on utilizes the IAM roles for service accounts capability of Amazon EKS. For more information, see the section called "Credentials with IRSA". The permissions in the AmazonEBSCSIDriverPolicy AWS managed policy are required. You can create an IAM role and attach the managed policy to it with the following command. Replace *my-cluster* with the name of your cluster and *AmazonEKS_EBS_CSI_DriverRole* with the name for your role. This command requires that you have eksctl installed on your device. If you need to use a different tool or you need to use a custom KMS key for encryption, see the section called "Step 1: Create an IAM role".

```
eksctl create iamserviceaccount \
    --name ebs-csi-controller-sa \
    --namespace kube-system \
    --cluster my-cluster \
    --role-name AmazonEKS_EBS_CSI_DriverRole \
    --role-only \
    --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
    --approve
```

Additional information

To learn more about the add-on, see the section called "Amazon EBS".

Amazon EFS CSI driver

The Amazon EFS CSI driver Amazon EKS add-on is a Kubernetes Container Storage Interface (CSI) plugin that provides Amazon EFS storage for your cluster.

The Amazon EKS add-on name is `aws-efs-csi-driver`.

Required IAM permissions

Required IAM permissions – This add-on utilizes the IAM roles for service accounts capability of Amazon EKS. For more information, see the section called "Credentials with IRSA". The permissions in the AmazonEFSCSIDriverPolicy AWS managed policy are required. You can create an IAM role and attach the managed policy to it with the following commands. Replace *my-cluster* with the name of your cluster and *AmazonEKS_EFS_CSI_DriverRole* with the name for your role. These commands require that you have eksctl installed on your device. If you need to use a different tool, see the section called “Step 1: Create an IAM role”.

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
eksctl create iamserviceaccount \
    --name efs-csi-controller-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name $role_name \
    --role-only \
    --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
    --approve
TRUST_POLICY=$(aws iam get-role --role-name $role_name --query
 'Role.AssumeRolePolicyDocument' | \
    sed -e 's/efs-csi-controller-sa/efs-csi-*/' -e 's/StringEquals/StringLike/')
aws iam update-assume-role-policy --role-name $role_name --policy-document
 "$TRUST_POLICY"
```

Additional information

To learn more about the add-on, see the section called “Amazon EFS”.

Mountpoint for Amazon S3 CSI Driver

The Mountpoint for Amazon S3 CSI Driver Amazon EKS add-on is a Kubernetes Container Storage Interface (CSI) plugin that provides Amazon S3 storage for your cluster.

The Amazon EKS add-on name is `aws-mountpoint-s3-csi-driver`.

Required IAM permissions

This add-on uses the IAM roles for service accounts capability of Amazon EKS. For more information, see the section called “Credentials with IRSA”.

The IAM role that is created will require a policy that gives access to S3. Follow the Mountpoint IAM permissions recommendations when creating the policy. Alternatively, you may use the AWS managed policy AmazonS3FullAccess, but this managed policy grants more permissions than are needed for Mountpoint.

You can create an IAM role and attach your policy to it with the following commands. Replace *`my-cluster`* with the name of your cluster, *`region-code`* with the correct

AWS Region code, *AmazonEKS_S3_CSI_DriverRole* with the name for your role, and *AmazonEKS_S3_CSI_DriverRole_ARN* with the role ARN. These commands require that you have eksctl installed on your device. For instructions on using the IAM console or AWS CLI, see the section called “Create an IAM role”.

```
CLUSTER_NAME=my-cluster
REGION=region-code
ROLE_NAME=AmazonEKS_S3_CSI_DriverRole
POLICY_ARN=AmazonEKS_S3_CSI_DriverRole_ARN
eksctl create iamserviceaccount \
    --name s3-csi-driver-sa \
    --namespace kube-system \
    --cluster $CLUSTER_NAME \
    --attach-policy-arn $POLICY_ARN \
    --approve \
    --role-name $ROLE_NAME \
    --region $REGION \
    --role-only
```

Additional information

To learn more about the add-on, see the section called “Mountpoint for Amazon S3”.

CSI snapshot controller

The Container Storage Interface (CSI) snapshot controller enables the use of snapshot functionality in compatible CSI drivers, such as the Amazon EBS CSI driver.

The Amazon EKS add-on name is `snapshot-controller`.

Required IAM permissions

This add-on doesn’t require any permissions.

Additional information

To learn more about the add-on, see the section called “CSI snapshot controller”.

Amazon SageMaker HyperPod task governance

SageMaker HyperPod task governance is a robust management system designed to streamline resource allocation and ensure efficient utilization of compute resources across teams and projects for your Amazon EKS clusters. This provides administrators with the capability to set:

- Priority levels for various tasks
- Compute allocation for each team
- How each team lends and borrows idle compute
- If a team preempts their own tasks

HyperPod task governance also provides Amazon EKS cluster Observability, offering real-time visibility into cluster capacity. This includes compute availability and usage, team allocation and utilization, and task run and wait time information, setting you up for informed decision-making and proactive resource management.

The Amazon EKS add-on name is `amazon-sagemaker-hyperpod-taskgovernance`.

Required IAM permissions

This add-on doesn't require any permissions.

Additional information

To learn more about the add-on, see SageMaker HyperPod task governance

AWS Network Flow Monitor Agent

The Amazon CloudWatch Network Flow Monitor Agent is a Kubernetes application that collects TCP connection statistics from all nodes in a cluster and publishes network flow reports to Amazon CloudWatch Network Flow Monitor Ingestion APIs.

The Amazon EKS add-on name is `aws-network-flow-monitoring-agent`.

Required IAM permissions

This add-on does require IAM permissions.

You need to attach the `CloudWatchNetworkFlowMonitorAgentPublishPolicy` managed policy to the add-on.

For more information on the required IAM setup, see IAM Policy on the Amazon CloudWatch Network Flow Monitor Agent GitHub repo.

For more information about the managed policy, see CloudWatchNetworkFlowMonitorAgentPublishPolicy in the Amazon CloudWatch User Guide.

Additional information

To learn more about the add-on, see the Amazon CloudWatch Network Flow Monitor Agent GitHub repo.

Node monitoring agent

The node monitoring agent Amazon EKS add-on can detect additional node health issues. These extra health signals can also be leveraged by the optional node auto repair feature to automatically replace nodes as needed.

> **Note**
>
> You do not need to install this add-on on Amazon EKS Auto Mode clusters. For more information, see the section called "Considerations for Amazon EKS Auto Mode".

The Amazon EKS add-on name is `eks-node-monitoring-agent`.

Required IAM permissions

This add-on doesn't require additional permissions.

Additional information

For more information, see the section called "Node health".

AWS Distro for OpenTelemetry

The AWS Distro for OpenTelemetry Amazon EKS add-on is a secure, production-ready, AWS supported distribution of the OpenTelemetry project. For more information, see AWS Distro for OpenTelemetry on GitHub.

The Amazon EKS add-on name is `adot`.

Required IAM permissions

This add-on only requires IAM permissions if you're using one of the preconfigured custom resources that can be opted into through advanced configuration.

Additional information

For more information, see Getting Started with AWS Distro for OpenTelemetry using EKS Add-Ons in the AWS Distro for OpenTelemetry documentation.

ADOT requires that `cert-manager` is deployed on the cluster as a prerequisite, otherwise this add-on won't work if deployed directly using the https://registry.terraform.io/modules/terraform-aws-modules/eks/aws/latest `cluster_addons` property. For more requirements, see Requirements for Getting Started with AWS Distro for OpenTelemetry using EKS Add-Ons in the AWS Distro for OpenTelemetry documentation.

Amazon GuardDuty agent

The Amazon GuardDuty agent Amazon EKS add-on is a security monitoring service that analyzes and processes foundational data sources including AWS CloudTrail management events and Amazon VPC flow logs. Amazon GuardDuty also processes features, such as Kubernetes audit logs and runtime monitoring.

The Amazon EKS add-on name is `aws-guardduty-agent`.

Required IAM permissions

This add-on doesn't require any permissions.

Additional information

For more information, see Runtime Monitoring for Amazon EKS clusters in Amazon GuardDuty.

- To detect potential security threats in your Amazon EKS clusters, enable Amazon GuardDuty runtime monitoring and deploy the GuardDuty security agent to your Amazon EKS clusters.

Amazon CloudWatch Observability agent

The Amazon CloudWatch Observability agent Amazon EKS add-on the monitoring and observability service provided by AWS. This add-on installs the CloudWatch Agent and enables both CloudWatch Application Signals and CloudWatch Container Insights with enhanced observability for Amazon EKS. For more information, see Amazon CloudWatch Agent.

The Amazon EKS add-on name is `amazon-cloudwatch-observability`.

Required IAM permissions

This add-on uses the IAM roles for service accounts capability of Amazon EKS. For more information, see the section called "Credentials with IRSA". The permissions in the AWSXrayWriteOnlyAccess and CloudWatchAgentServerPolicy AWS managed policies are required. You can create an IAM role, attach the managed policies to it, and annotate the Kubernetes service account used by the add-on with the following command. Replace *my-cluster* with the name of your cluster and *AmazonEKS_Observability_role* with the name for your role. This command requires that you have eksctl installed on your device. If you need to use a different tool to create the role, attach the policy to it, and annotate the Kubernetes service account, see the section called "Assign IAM role".

```
eksctl create iamserviceaccount \
    --name cloudwatch-agent \
    --namespace amazon-cloudwatch \
    --cluster my-cluster \
    --role-name AmazonEKS_Observability_Role \
    --role-only \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSXrayWriteOnlyAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
    --approve
```

Additional information

For more information, see Install the CloudWatch agent.

EKS Pod Identity Agent

The Amazon EKS Pod Identity Agent Amazon EKS add-on provides the ability to manage credentials for your applications, similar to the way that EC2 instance profiles provide credentials to EC2 instances.

> **Note**
>
> You do not need to install this add-on on Amazon EKS Auto Mode clusters. Amazon EKS Auto Mode integrates with EKS Pod Identity. For more information, see the section called "Considerations for Amazon EKS Auto Mode".

The Amazon EKS add-on name is `eks-pod-identity-agent`.

Required IAM permissions

This add-on users permissions from the Amazon EKS node IAM role.

Update information

You can only update one minor version at a time. For example, if your current version is `1.28.x-eksbuild.y` and you want to update to `1.30.x-eksbuild.y`, then you must update your current version to `1.29.x-eksbuild.y` and then update it again to `1.30.x-eksbuild.y`. For more information about updating the add-on, see the section called “Update (EKS add-on)”.

Community add-ons

You can use AWS APIs to install community add-ons, such as the Kubernetes Metrics Server. You may choose to install community add-ons as Amazon EKS Add-ons to reduce the complexity of maintaining the software on multiple clusters.

For example, you can use the AWS API, CLI, or Management Console to install community add-ons. You can install a community add-on during cluster creation.

You manage community add-ons just like existing Amazon EKS Add-ons. Community add-ons are different from existing add-ons in that they have a unique scope of support.

> **Note**
>
> Using community add-ons is at your discretion. As part of the shared responsibility model between you and AWS, you are expected to understand what you are installing into your workspace for these third party plugins. You are also responsible for the community add-ons meeting your security needs.

Community add-ons are built by AWS. AWS only validates community add-ons for version compatibility. For example, if you install a community add-on on a cluster, AWS checks if it is compatible with the Kubernetes version of your cluster.

Importantly, AWS does not provide full support for community add-ons. AWS supports only lifecycle operations done using AWS APIs, such as installing add-ons or deleting add-ons.

If you require support for a community add-on, utilize the existing project resources. For example, you may create a GitHub issue on the repo for the project.

Determine add-on type

You can use the AWS CLI to determine the type of an Amazon EKS Add-on.

Use the following CLI command to retrieve information about an add-on. You can replace `metrics-server` with the name of any add-on.

```
aws eks describe-addon-versions --addon-name metrics-server
```

Review the CLI output for the `owner` field.

```
{
    "addons": [
        {
            "addonName": "metrics-server",
            "type": "observability",
            "owner": "community",
            "addonVersions": [
```

If the value of `owner` is `community`, then the add-on is a community add-on. AWS only provides support for installing, updating, and removing the add-on. If you have questions about the functionality and operation of the add-on itself, use community resources like GitHub issues.

Install or update community add-on

You install or update community add-ons in the same way as other Amazon EKS Add-ons.

- the section called "Create an add-on"
- the section called "Update an add-on"
- the section called "Remove an add-on"

Available community add-ons

The following community add-ons are availalbe from Amazon EKS.

Kubernetes Metrics Server

The Kubernetes Metrics Server is a scalable and efficient source of container resource metrics for Kubernetes built-in autoscaling pipelines. It collects resource metrics from Kubelets and exposes them in Kubernetes apiserver through Metrics API for use by Horizontal Pod Autoscaler and Vertical Pod Autoscaler.

Property	Value
Add-on name	`metrics-server`
Namespace	`kube-system`
Documentation	GitHub Readme
Service account name	None
Managed IAM policy	None
Custom IAM permissions	None

kube-state-metrics

Add-on agent to generate and expose cluster-level metrics.

The state of Kubernetes objects in the Kubernetes API can be exposed as metrics. An add-on agent called kube-state-metrics can connect to the Kubernetes API server and expose a HTTP endpoint with metrics generated from the state of individual objects in the cluster. It exposes various information about the state of objects like labels and annotations, startup and termination times, status or the phase the object currently is in.

Property	Value
Add-on name	`kube-state-metrics`
Namespace	`kube-state-metrics`
Documentation	Metrics for Kubernetes Object States in Kubernetes Docs
Service account name	None
Managed IAM policy	None

Property	Value
Custom IAM permissions	None

Prometheus Node exporter

Prometheus exporter for hardware and OS metrics exposed by *NIX kernels, written in Go with pluggable metric collectors. The Prometheus Node Exporter exposes a wide variety of hardware- and kernel-related metrics.

Property	Value
Add-on name	`prometheus-node-exporter`
Namespace	`prometheus-node-exporter`
Documentation	Monitoring Linux host metrics with the Node Exporter in Prometheus Docs
Service account name	None
Managed IAM policy	None
Custom IAM permissions	None

View Attributions

You can download the open source attribtions and license information for community add-ons.

1. Determine the name and version of the add-on you want to download attributions for.
2. Update the following command with the name and version:

```
curl -O https://amazon-eks-docs.s3.amazonaws.com/attributions/<add-on-name>/<add-on-version>/attributions.zip
```

   For example:

```
curl -O https://amazon-eks-docs.s3.amazonaws.com/attributions/kube-state-metrics/
v2.14.0-eksbuild.1/attributions.zip
```

3. Use the command to download the file.

Use this zip file to view information about the license attributions.

AWS Marketplace add-ons

In addition to the previous list of Amazon EKS add-ons, you can also add a wide selection of operational software Amazon EKS add-ons from independent software vendors. Choose an add-on to learn more about it and its installation requirements.

Accuknox

The add-on name is `accuknox_kubearmor` and the namespace is `kubearmor`. Accuknox publishes the add-on.

For information about the add-on, see Getting Started with KubeArmor in the KubeArmor documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Akuity

The add-on name is `akuity_agent` and the namespace is `akuity`. Akuity publishes the add-on.

For information about how the add-on, see Installing the Akuity Agent on Amazon EKS with the Akuity EKS add-on in the Akuity Platform documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Calyptia

The add-on name is `calyptia_fluent-bit` and the namespace is `calytia-fluentbit`. Calyptia publishes the add-on.

For information about the add-on, see Getting Started with Calyptia Core Agent on the Calyptia documentation website.

Service account name

The service account name is `clyptia-fluentbit`.

AWS managed IAM policy

This add-on uses the `AWSMarketplaceMeteringRegisterUsage` managed policy. For more information, see AWSMarketplaceMeteringRegisterUsage in the AWS Managed Policy Reference Guide.

Command to create required IAM role

The following command requires that you have an IAM OpenID Connect (OIDC) provider for your cluster. To determine whether you have one, or to create one, see the section called "IAM OIDC provider". Replace *`my-cluster`* with the name of your cluster and *`my-calyptia-role`* with the name for your role. This command requires that you have eksctl installed on your device. If you need to use a different tool to create the role and annotate the Kubernetes service account, see the section called "Assign IAM role".

```
eksctl create iamserviceaccount --name service-account-name  --namespace calyptia-
fluentbit --cluster my-cluster --role-name my-calyptia-role \
```

```
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/
AWSMarketplaceMeteringRegisterUsage --approve
```

Cisco Observability Collector

The add-on name is `cisco_cisco-cloud-observability-collectors` and the namespace is `appdynamics`. Cisco pubishes the add-on.

For information about the add-on, see Use the Cisco Cloud Observability AWS Marketplace Add-Ons in the Cisco AppDynamics documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Cisco Observability Operator

The add-on name is `cisco_cisco-cloud-observability-operators` and the namespace is `appdynamics`. Cisco publishes the add-on.

For information about the add-on, see Use the Cisco Cloud Observability AWS Marketplace Add-Ons in the Cisco AppDynamics documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

CLOUDSOFT

The add-on name is `cloudsoft_cloudsoft-amp` and the namespace is `cloudsoft-amp`. CLOUDSOFT publishes the add-on.

For information about the add-on, see [Amazon EKS ADDON]() in the CLOUDSOFT documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Cribl

The add-on name is `cribl_cribledge` and the namespace is `cribledge`. Cribl publishes the add-on.

For information about the add-on, see [Installing the Cribl Amazon EKS Add-on for Edge]() in the Cribl documentation

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Dynatrace

The add-on name is `dynatrace_dynatrace-operator` and the namespace is `dynatrace`. Dynatrace publishes the add-on.

For information about the add-on, see Kubernetes monitoring in the dynatrace documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Datree

The add-on name is `datree_engine-pro` and the namespace is `datree`. Datree publishes the add-on.

For information about the add-on, see Amazon EKS-intergration in the Datree documentation.

Service account name

The service account name is datree-webhook-server-awsmp.

AWS managed IAM policy

The managed policy is AWSLicenseManagerConsumptionPolicy. For more information, see AWSLicenseManagerConsumptionPolicy in the AWS Managed Policy Reference Guide..

Command to create required IAM role

The following command requires that you have an IAM OpenID Connect (OIDC) provider for your cluster. To determine whether you have one, or to create one, see the section called "IAM OIDC provider". Replace *`my-cluster`* with the name of your cluster and *`my-datree-role`* with the name for your role. This command requires that you have eksctl installed on your device. If you need to use a different tool to create the role and annotate the Kubernetes service account, see the section called "Assign IAM role".

```
eksctl create iamserviceaccount --name datree-webhook-server-awsmp --namespace datree
 --cluster my-cluster --role-name my-datree-role \
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

Custom permissions

Custom permissions aren't used with this add-on.

Datadog

The add-on name is `datadog_operator` and the namespace is `datadog-agent`. Datadog publishes the add-on.

For information about the add-on, see Installing the Datadog Agent on Amazon EKS with the Datadog Operator Add-on in the Datadog documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Groundcover

The add-on name is `groundcover_agent` and the namespace is `groundcover`. groundcover publishes the add-on.

For information about the add-on, see Installing the groundcover Amazon EKS Add-on in the groundcover documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Grafana Labs

The add-on name is grafana-labs_kubernetes-monitoring and the namespace is monitoring. Grafana Labs publishes the add-on.

For information about the add-on, see Configure Kubernetes Monitoring as an Add-on with Amazon EKS in the Grafana Labs documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Guance

- **Publisher** – GUANCE
- **Name** – guance_datakit
- **Namespace** – datakit
- **Service account name** – A service account isn't used with this add-on.
- **AWS managed IAM policy** – A managed policy isn't used with this add-on.
- **Custom IAM permissions** – Custom permissions aren't used with this add-on.
- **Setup and usage instructions** – See Using Amazon EKS add-on in the Guance documentation.

HA Proxy

The name is haproxy-technologies_kubernetes-ingress-ee and the namespace is haproxy-controller. HA Proxy publishes the add-on.

For information about the add-on, see Amazon EKS-intergration in the Datree documentation.

Service account name

The service account name is customer defined.

AWS managed IAM policy

The managed policy is AWSLicenseManagerConsumptionPolicy. For more information, see AWSLicenseManagerConsumptionPolicy in the AWS Managed Policy Reference Guide..

Command to create required IAM role

The following command requires that you have an IAM OpenID Connect (OIDC) provider for your cluster. To determine whether you have one, or to create one, see the section called "IAM OIDC provider". Replace *my-cluster* with the name of your cluster and *my-haproxy-role* with the name for your role. This command requires that you have eksctl installed on your device. If you need to use a different tool to create the role and annotate the Kubernetes service account, see the section called "Assign IAM role".

```
eksctl create iamserviceaccount --name service-account-name  --namespace haproxy-
controller --cluster my-cluster --role-name my-haproxy-role \
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

Custom permissions

Custom permissions aren't used with this add-on.

Kpow

The add-on name is `factorhouse_kpow` and the namespace is `factorhouse`. Factorhouse publishes the add-on.

For information about the add-on, see AWS Marketplace LM in the Kpow documentation.

Service account name

The service account name is `kpow`.

AWS managed IAM policy

The managed policy is AWSLicenseManagerConsumptionPolicy. For more information, see AWSLicenseManagerConsumptionPolicy in the AWS Managed Policy Reference Guide..

Command to create required IAM role

The following command requires that you have an IAM OpenID Connect (OIDC) provider for your cluster. To determine whether you have one, or to create one, see the section called "IAM OIDC provider". Replace *my-cluster* with the name of your cluster and *my-kpow-role* with the name for your role. This command requires that you have eksctl installed on your device. If you need to use a different tool to create the role and annotate the Kubernetes service account, see the section called "Assign IAM role".

```
eksctl create iamserviceaccount --name kpow --namespace factorhouse --cluster my-
cluster --role-name my-kpow-role \
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

Custom permissions

Custom permissions aren't used with this add-on.

Kubecost

The add-on name is `kubecost_kubecost` and the namespace is `kubecost`. Kubecost publishes the add-on.

For information about the add-on, see AWS Cloud Billing Integration in the Kubecost documentation.

If your cluster is version `1.23` or later, you must have the Store Kubernetes volumes with Amazon EBS installed on your cluster. otherwise you will receive an error.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Kasten

The add-on name is `kasten_k10` and the namespace is `kasten-io`. Kasten by Veeam publishes the add-on.

For information about the add-on, see Installing K10 on AWS using Amazon EKS Add-on in the Kasten documentation.

If your Amazon EKS cluster is version Kubernetes `1.23` or later, you must have the Amazon EBS CSI driver installed on your cluster with a default `StorageClass`.

Service account name

The service account name is `k10-k10`.

AWS managed IAM policy

The managed policy is AWSLicenseManagerConsumptionPolicy. For more information, see AWSLicenseManagerConsumptionPolicy in the AWS Managed Policy Reference Guide..

Command to create required IAM role

The following command requires that you have an IAM OpenID Connect (OIDC) provider for your cluster. To determine whether you have one, or to create one, see the section called "IAM OIDC provider". Replace *my-cluster* with the name of your cluster and *my-kasten-role* with the name for your role. This command requires that you have eksctl installed on your device. If you need to use a different tool to create the role and annotate the Kubernetes service account, see the section called "Assign IAM role".

```
eksctl create iamserviceaccount --name k10-k10 --namespace kasten-io --cluster my-
cluster --role-name my-kasten-role \
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

Custom permissions

Custom permissions aren't used with this add-on.

Kong

The add-on name is `kong_konnect-ri` and the namespace is `kong`. Kong publishes the add-on.

For information about the add-on, see Installing the Kong Gateway EKS Add-on in the Kong documentation.

If your cluster is version `1.23` or later, you must have the Store Kubernetes volumes with Amazon EBS installed on your cluster. otherwise you will receive an error.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

LeakSignal

The add-on name is `leaksignal_leakagent` and the namespace is `leakagent`. LeakSignal publishes the add-on.

For information about the add-on, see https://www.leaksignal.com/docs/LeakAgent/Deployment/ AWS%20EKS%20Addon/[Install the LeakAgent add-on] in the LeakSignal documentation

If your cluster is version `1.23` or later, you must have the Store Kubernetes volumes with Amazon EBS installed on your cluster. otherwise you will receive an error.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

NetApp

The add-on name is `netapp_trident-operator` and the namespace is `trident`. NetApp publishes the add-on.

For information about the add-on, see Configure the Trident EKS add-on in the NetApp documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

New Relic

The add-on name is `new-relic_kubernetes-operator` and the namespace is `newrelic`. New Relic publishes the add-on.

For information about the add-on, see [Installing the New Relic Add-on for EKS] in the New Relic documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Rafay

The add-on name is `rafay-systems_rafay-operator` and the namespace is `rafay-system`. Rafay publishes the add-on.

For information about the add-on, see [Installing the Rafay Amazon EKS Add-on] in the Rafay documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Rad Security

- **Publisher** – RAD SECURITY
- **Name** – `rad-security_rad-security`
- **Namespace** – `ksoc`
- **Service account name** – A service account isn't used with this add-on.
- **AWS managed IAM policy** – A managed policy isn't used with this add-on.
- **Custom IAM permissions** – Custom permissions aren't used with this add-on.
- **Setup and usage instructions** – See Installing Rad Through The AWS Marketplace in the Rad Security documentation.

SolarWinds

- **Publisher** – SOLARWINDS
- **Name** – `solarwinds_swo-k8s-collector-addon`
- **Namespace** – `solarwinds`
- **Service account name** – A service account isn't used with this add-on.
- **AWS managed IAM policy** – A managed policy isn't used with this add-on.
- **Custom IAM permissions** – Custom permissions aren't used with this add-on.
- **Setup and usage instructions** – See Monitor an Amazon EKS cluster in the SolarWinds documentation.

Solo

The add-on name is `solo-io_istio-distro` and the namespace is `istio-system`. Solo publishes the add-on.

For information about the add-on, see Installing Istio in the Solo.io documentation..

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Snyk

- **Publisher** – SNYK
- **Name** – `snyk_runtime-sensor`
- **Namespace** – `snyk_runtime-sensor`
- **Service account name** – A service account isn't used with this add-on.
- **AWS managed IAM policy** – A managed policy isn't used with this add-on.
- **Custom IAM permissions** – Custom permissions aren't used with this add-on.
- **Setup and usage instructions** – See Snyk runtime sensor in the Snyk user docs.

Stormforge

The add-on name is `stormforge_optimize-Live` and the namespace is `stormforge-system`. Stormforge publishes the add-on.

For information about the add-on, see Installing the StormForge Agent in the StormForge documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Splunk

The add-on name is `splunk_splunk-otel-collector-chart` and the namespace is `splunk-monitoring`. Splunk publishes the add-on.

For information about the add-on, see [Install the Splunk add-on for Amazon EKS](#) in the Splunk documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Teleport

The add-on name is `teleport_teleport` and the namespace is `teleport`. Teleport publishes the add-on.

For information about the add-on, see [How Teleport Works](#) in the Teleport documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Tetrate

The add-on name is `tetrate-io_istio-distro` and the namespace is `istio-system`. Tetrate Io publishes the add-on.

For information about the add-on, see the Tetrate Istio Distro website.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Upbound Universal Crossplane

The add-on name is `upbound_universal-crossplane` and the namespace is `upbound-system`. Upbound publishes the add-on.

For information about the add-on, see Upbound Universal Crossplane (UXP) in the Upbound documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Upwind

The add-on name is `upwind` and the namespace is `upwind`. Upwind publishes the add-on.

For information about the add-on, see Upwind documentation.

Service account name

A service account isn't used with this add-on.

AWS managed IAM policy

A managed policy isn't used with this add-on.

Custom IAM permissions

Custom permissions aren't used with this add-on.

Create an Amazon EKS add-on

Amazon EKS add-ons are add-on software for Amazon EKS clusters. All Amazon EKS add-ons:

- Include the latest security patches and bug fixes.
- Are validated by AWS to work with Amazon EKS.
- Reduce the amount of work required to manage the add-on software.

You can create an Amazon EKS add-on using `eksctl`, the AWS Management Console, or the AWS CLI. If the add-on requires an IAM role, see the details for the specific add-on in Amazon EKS add-ons for details about creating the role.

Prerequisites

Complete the following before you create an add-on:

- The cluster must exist before you create an add-on for it. For more information, see the section called "Create a cluster".
- Check if your add-on requires an IAM role. For more information, see the section called "Verify compatibility".
- Verify that the Amazon EKS add-on version is compatabile with your cluster. For more information, see the section called "Verify compatibility".
- Verify that version 0.190.0 or later of the `eksctl` command line tool installed on your computer or AWS CloudShell. For more information, see Installation on the `eksctl` website.

Procedure

You can create an Amazon EKS add-on using `eksctl`, the AWS Management Console, or the AWS CLI. If the add-on requires an IAM role, see the details for the specific add-on in Available Amazon EKS add-ons from AWS for details about creating the role.

Create add-on (eksctl)

1. View the names of add-ons available for a cluster version. Replace *1.32* with the version of your cluster.

```
eksctl utils describe-addon-versions --kubernetes-version 1.32 | grep AddonName
```

   An example output is as follows.

```
"AddonName": "aws-ebs-csi-driver",
                        "AddonName": "coredns",
                        "AddonName": "kube-proxy",
                        "AddonName": "vpc-cni",
                        "AddonName": "adot",
                        "AddonName": "dynatrace_dynatrace-operator",
                        "AddonName": "upbound_universal-crossplane",
                        "AddonName": "teleport_teleport",
                        "AddonName": "factorhouse_kpow",
                        [...]
```

2. View the versions available for the add-on that you would like to create. Replace *1.32* with the version of your cluster. Replace *name-of-addon* with the name of the add-on you want to view the versions for. The name must be one of the names returned in the previous step.

```
eksctl utils describe-addon-versions --kubernetes-version 1.32 --name name-of-addon |
 grep AddonVersion
```

   The following output is an example of what is returned for the add-on named `vpc-cni`. You can see that the add-on has several available versions.

```
"AddonVersions": [
    "AddonVersion": "v1.12.0-eksbuild.1",
    "AddonVersion": "v1.11.4-eksbuild.1",
    "AddonVersion": "v1.10.4-eksbuild.1",
```

```
    "AddonVersion": "v1.9.3-eksbuild.1",
```

a. Determine whether the add-on you want to create is an Amazon EKS or AWS Marketplace add-on. The AWS Marketplace has third party add-ons that require you to complete additional steps to create the add-on.

```
eksctl utils describe-addon-versions --kubernetes-version 1.32 --name name-of-addon | grep ProductUrl
```

If no output is returned, then the add-on is an Amazon EKS. If output is returned, then the add-on is an AWS Marketplace add-on. The following output is for an add-on named `teleport_teleport`.

```
"ProductUrl": "https://aws.amazon.com/marketplace/pp?sku=3bda70bb-566f-4976-806c-f96faef18b26"
```

You can learn more about the add-on in the AWS Marketplace with the returned URL. If the add-on requires a subscription, you can subscribe to the add-on through the AWS Marketplace. If you're going to create an add-on from the AWS Marketplace, then the IAM principal that you're using to create the add-on must have permission to create the AWSServiceRoleForAWSLicenseManagerRole service-linked role. For more information about assigning permissions to an IAM entity, see Adding and removing IAM identity permissions in the IAM User Guide.

3. Create an Amazon EKS add-on. Copy the command and replace the *user-data* as follows:

- Replace *my-cluster* with the name of your cluster.
- Replace *name-of-addon* with the name of the add-on that you want to create.
- If you want a version of the add-on that's earlier than the latest version, then replace *latest* with the version number returned in the output of a previous step that you want to use.
- If the add-on uses a service account role, replace *111122223333* with your account ID and replace *role-name* with the name of the role. For instructions on creating a role for your service account, see the documentation for the add-on that you're creating. For a list of add-ons, see the section called " AWS Add-ons". Specifying a service account role requires that you have an IAM OpenID Connect (OIDC) provider for your cluster. To determine whether you have one for your cluster, or to create one, see the section called "IAM OIDC provider".

If the add-on doesn't use a service account role, delete `--service-account-role-arnarn:aws:iam::111122223333:role/role-name`.

- This example command overwrites the configuration of any existing self-managed version of the add-on, if there is one. If you don't want to overwrite the configuration of an existing self-managed add-on, remove the *`--force`* option. If you remove the option, and the Amazon EKS add-on needs to overwrite the configuration of an existing self-managed add-on, then creation of the Amazon EKS add-on fails with an error message to help you resolve the conflict. Before specifying this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to manage, because those settings are overwritten with this option.

```
eksctl create addon --cluster my-cluster --name name-of-addon --version latest \
    --service-account-role-arn arn:aws:iam::111122223333:role/role-name --force
```

You can see a list of all available options for the command.

```
eksctl create addon --help
```

For more information about available options see Addons in the `eksctl` documentation.

Create add-on (AWS Console)

1. Open the Amazon EKS console.
2. In the left navigation pane, choose **Clusters**.
3. Choose the name of the cluster that you want to create the add-on for.
4. Choose the **Add-ons** tab.
5. Choose **Get more add-ons**.
6. On the **Select add-ons** page, choose the add-ons that you want to add to your cluster. You can add as many **Amazon EKS add-ons** and **AWS Marketplace add-ons** as you require.

   For **AWS Marketplace** add-ons the IAM principal that you're using to create the add-on must have permissions to read entitlements for the add-on from the AWS LicenseManager. AWS LicenseManager requires AWSServiceRoleForAWSLicenseManagerRole service-linked role (SLR) that allows AWS resources to manage licenses on your behalf. The SLR is a one time requirement, per account, and you will not have to create separate SLR's for each add-on nor each cluster. For more information about assigning permissions to an IAM principal see Adding and removing IAM identity permissions in the IAM User Guide.

If the **AWS Marketplace add-ons** that you want to install aren't listed, you can click the page numbering to view additional page results or search in the search box. In the **Filtering options**, you can also filter by **category**, **vendor**, or **pricing model** and then choose the add-ons from the search results. Once you've selected the add-ons that you want to install, choose **Next**.

7. On the **Configure selected add-ons settings** page, do the following:

   a. Choose **View subscription options** to open the **Subscription options** form. Review the **Pricing details** and **Legal** sections, then choose the **Subscribe** button to continue.

   b. For **Version**, choose the version that you want to install. We recommend the version marked **latest**, unless the individual add-on that you're creating recommends a different version. To determine whether an add-on has a recommended version, see the documentation for the add-on that you're creating. For a list of add-ons, see the section called " AWS Add-ons".

   c. You have two options for configuring roles for add-ons: EKS Pod Identities IAM role and IAM roles for service accounts (IRSA). Follow the appropriate step below for your preferred option. If all of the add-ons that you selected have **Requires subscription** under **Status**, choose **Next**. You can't configure those add-ons further until you've subscribed to them after your cluster is created. For the add-ons that don't have **Requires subscription** under **Status**, do the following:

      i. For **Pod Identity IAM role for service account**, you can either use an existing EKS Pod Identity IAM role or create one using the **Create Recommended Role** button. This field will only provide options with the appropriate trust policy. If there's no role to select, then you don't have an existing role with a matching trust policy. To configure an EKS Pod Identity IAM role for service accounts of the selected add-on, choose **Create recommended role**. The role creation wizard opens in a separate window. The wizard will automatically populate the role information as follows. For each add-on where you want to create the EKS Pod Identity IAM role, complete the steps in the IAM wizard as follows.

         - On the **Select trusted entity** step, the AWS service option for **EKS** and the use case for **EKS - Pod Identity** are preselected, and the appropriate trust policy will be automatically populated for the add-on. For example, the role will be created with the appropriate trust policy containing the pods.eks.amazonaws.com IAM Principal as detailed in the section called "Benefits of EKS Pod Identities". Choose **Next**.
         - On the **Add permissions** step, the appropriate managed policy for the role policy is preselected for the add-on. For example, for the Amazon VPC CNI add-on, the role will

be created with the managed policy `AmazonEKS_CNI_Policy` as detailed in the section called "Amazon VPC CNI plugin for Kubernetes". Choose **Next**.

- On the **Name, review, and create** step, in **Role name**, the default role name is automatically populated for the add-on. For example, for the **Amazon VPC CNI** add-on, the role will be created with the name **AmazonEKSPodIdentityAmazonVPCCNIRole**. In **Description**, the default description is automatically populated with the appropriate description for the add-on. For example, for the Amazon VPC CNI add-on, the role will be created with the description **Allows pods running in Amazon EKS cluster** to access AWS resources. In **Trust policy**, view the populated trust policy for the add-on. Choose **Create role**.

  NOTE: Retaining the default role name enables EKS to pre-select the role for add-ons in new clusters or when adding add-ons to existing clusters. You can still override this name and the role will be available for the add-on across your clusters, but the role will need to be manually selected from the drop down.

ii. For add-ons that do not have **Requires subscription** under **Status** and where you want to configure roles using IRSA, see the documentation for the add-on that you're creating to create an IAM policy and attach it to a role. For a list of add-ons, see the section called " AWS Add-ons". Selecting an IAM role requires that you have an IAM OpenID Connect (OIDC) provider for your cluster. To determine whether you have one for your cluster, or to create one, see the section called "IAM OIDC provider".

iii. Choose **Optional configuration settings**.

iv. If the add-on requires configuration, enter it in the **Configuration values** box. To determine whether the add-on requires configuration information, see the documentation for the add-on that you're creating. For a list of add-ons, see the section called " AWS Add-ons".

v. Choose one of the available options for **Conflict resolution method**. If you choose **Override** for the **Conflict resolution method**, one or more of the settings for the existing add-on can be overwritten with the Amazon EKS add-on settings. If you don't enable this option and there's a conflict with your existing settings, the operation fails. You can use the resulting error message to troubleshoot the conflict. Before choosing this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to self-manage.

vi. Choose **Next**.

8. On the **Review and add** page, choose **Create**. After the add-on installation is complete, you see your installed add-ons.

9. If any of the add-ons that you installed require a subscription, complete the following steps:

a. Choose the **Subscribe** button in the lower right corner for the add-on. You're taken to the page for the add-on in the AWS Marketplace. Read the information about the add-on such as its **Product Overview** and **Pricing Information**.

b. Select the **Continue to Subscribe** button on the top right of the add-on page.

c. Read through the **Terms and Conditions**. If you agree to them, choose **Accept Terms**. It may take several minutes to process the subscription. While the subscription is processing, the **Return to Amazon EKS Console** button is grayed out.

d. Once the subscription has finished processing, the **Return to Amazon EKS Console** button is no longer grayed out. Choose the button to go back to the Amazon EKS console **Add-ons** tab for your cluster.

e. For the add-on that you subscribed to, choose **Remove and reinstall** and then choose **Reinstall add-on**. Installation of the add-on can take several minutes. When Installation is complete, you can configure the add-on.

Create add-on (AWS CLI)

1. You need version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version, use `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*. The AWS CLI version that is installed in AWS CloudShell might also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the *AWS CloudShell User Guide*.

2. Determine which add-ons are available. You can see all available add-ons, their type, and their publisher. You can also see the URL for add-ons that are available through the AWS Marketplace. Replace *1.32* with the version of your cluster.

```
aws eks describe-addon-versions --kubernetes-version 1.32 \
    --query 'addons[].{MarketplaceProductUrl: marketplaceInformation.productUrl,
 Name: addonName, Owner: owner Publisher: publisher, Type: type}' --output table
```

An example output is as follows.

```
---------------------------------------------------------------------------------------
```

```
|
 DescribeAddonVersions
  |
+---------------------------------------------------------------
+-------------------------------+------------------+-------------
+---------------------+
|                    MarketplaceProductUrl                    |             Name
          |      Owner        |  Publisher   |        Type        |
+---------------------------------------------------------------
+-------------------------------+------------------+-------------
+---------------------+
|  None                                                       |  aws-ebs-csi-driver
          |  aws               |  eks          |  storage            |
|  None                                                       |  coredns
          |  aws               |  eks          |  networking         |
|  None                                                       |  kube-proxy
          |  aws               |  eks          |  networking         |
|  None                                                       |  vpc-cni
          |  aws               |  eks          |  networking         |
|  None                                                       |  adot
          |  aws               |  eks          |  observability      |
| https://aws.amazon.com/marketplace/pp/prodview-brb73nceicv7u |
 dynatrace_dynatrace-operator |  aws-marketplace |  dynatrace    |  monitoring
  |
| https://aws.amazon.com/marketplace/pp/prodview-uhc2iwi5xysoc |  upbound_universal-
crossplane |  aws-marketplace |  upbound      |  infra-management   |
| https://aws.amazon.com/marketplace/pp/prodview-hd2ydsrgqy4li |  teleport_teleport
         |  aws-marketplace |  teleport     |  policy-management  |
| https://aws.amazon.com/marketplace/pp/prodview-vgghgqdsplhvc |  factorhouse_kpow
         |  aws-marketplace |  factorhouse |  monitoring         |
|  [...]                                                      |  [...]
          |  [...]             |  [...]        |  [...]              |
+---------------------------------------------------------------
+-------------------------------+------------------+-------------
+---------------------+
```

Your output might be different. In this example output, there are three different add-ons available of type `networking` and five add-ons with a publisher of type `eks`. The add-ons with `aws-marketplace` in the `Owner` column may require a subscription before you can install them. You can visit the URL to learn more about the add-on and to subscribe to it.

3. You can see which versions are available for each add-on. Replace *`1.32`* with the version of your cluster and replace *`vpc-cni`* with the name of an add-on returned in the previous step.

```
aws eks describe-addon-versions --kubernetes-version 1.32 --addon-name vpc-cni \
    --query 'addons[].addonVersions[].{Version: addonVersion, Defaultversion:
 compatibilities[0].defaultVersion}' --output table
```

An example output is as follows.

```
------------------------------------------
|          DescribeAddonVersions         |
+-----------------+----------------------+
| Defaultversion  |       Version        |
+-----------------+----------------------+
|  False          |  v1.12.0-eksbuild.1  |
|  True           |  v1.11.4-eksbuild.1  |
|  False          |  v1.10.4-eksbuild.1  |
|  False          |  v1.9.3-eksbuild.1   |
+-----------------+----------------------+
```

The version with `True` in the `Defaultversion` column is the version that the add-on is created with, by default.

4. (Optional) Find the configuration options for your chosen add-on by running the following command:

```
aws eks describe-addon-configuration --addon-name vpc-cni --addon-version v1.12.0-
eksbuild.1
```


```
{
    "addonName": "vpc-cni",
    "addonVersion": "v1.12.0-eksbuild.1",
    "configurationSchema": "{\"$ref\":\"#/definitions/VpcCni\",\"$schema\":\"http://
json-schema.org/draft-06/schema#\",\"definitions\":{\"Cri\":{\"additionalProperties
\":false,\"properties\":{\"hostPath\":{\"$ref\":\"#/definitions/HostPath\"}},\"title
\":\"Cri\",\"type\":\"object\"},\"Env\":{\"additionalProperties\":false,\"properties
\":{\"ADDITIONAL_ENI_TAGS\":{\"type\":\"string\"},\"AWS_VPC_CNI_NODE_PORT_SUPPORT
\":{\"format\":\"boolean\",\"type\":\"string\"},\"AWS_VPC_ENI_MTU\":{\"format\":
\"integer\",\"type\":\"string\"},\"AWS_VPC_K8S_CNI_CONFIGURE_RPFILTER\":{\"format
\":\"boolean\",\"type\":\"string\"},\"AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG\":{\"format
\":\"boolean\",\"type\":\"string\"},\"AWS_VPC_K8S_CNI_EXTERNALSNAT\":{\"format\":
\"boolean\",\"type\":\"string\"},\"AWS_VPC_K8S_CNI_LOGLEVEL\":{\"type\":\"string\"},
\"AWS_VPC_K8S_CNI_LOG_FILE\":{\"type\":\"string\"},\"AWS_VPC_K8S_CNI_RANDOMIZESNAT
```

```
\":{\"type\":\"string\"},\"AWS_VPC_K8S_CNI_VETHPREFIX\":{\"type\":\"string\"},
\"AWS_VPC_K8S_PLUGIN_LOG_FILE\":{\"type\":\"string\"},\"AWS_VPC_K8S_PLUGIN_LOG_LEVEL
\":{\"type\":\"string\"},\"DISABLE_INTROSPECTION\":{\"format\":\"boolean\",\"type
\":\"string\"},\"DISABLE_METRICS\":{\"format\":\"boolean\",\"type\":\"string
\"},\"DISABLE_NETWORK_RESOURCE_PROVISIONING\":{\"format\":\"boolean\",\"type
\":\"string\"},\"ENABLE_POD_ENI\":{\"format\":\"boolean\",\"type\":\"string
\"},\"ENABLE_PREFIX_DELEGATION\":{\"format\":\"boolean\",\"type\":\"string\"},
\"WARM_ENI_TARGET\":{\"format\":\"integer\",\"type\":\"string\"},\"WARM_PREFIX_TARGET
\":{\"format\":\"integer\",\"type\":\"string\"}},\"title\":\"Env\",\"type\":
\"object\"},\"HostPath\":{\"additionalProperties\":false,\"properties\":{\"path
\":{\"type\":\"string\"}},\"title\":\"HostPath\",\"type\":\"object\"},\"Limits
\":{\"additionalProperties\":false,\"properties\":{\"cpu\":{\"type\":\"string
\"},\"memory\":{\"type\":\"string\"}},\"title\":\"Limits\",\"type\":\"object\"},
\"Resources\":{\"additionalProperties\":false,\"properties\":{\"limits\":{\"$ref\":
\"#/definitions/Limits\"},\"requests\":{\"$ref\":\"#/definitions/Limits\"}},\"title
\":\"Resources\",\"type\":\"object\"},\"VpcCni\":{\"additionalProperties\":false,
\"properties\":{\"cri\":{\"$ref\":\"#/definitions/Cri\"},\"env\":{\"$ref\":\"#/
definitions/Env\"},\"resources\":{\"$ref\":\"#/definitions/Resources\"}},\"title\":
\"VpcCni\",\"type\":\"object\"}}}"
}
```

The output is a standard JSON schema.

Here is an example of valid configuration values, in JSON format, that works with the schema above.

```
{
  "resources": {
    "limits": {
      "cpu": "100m"
    }
  }
}
```

Here is an example of valid configuration values, in YAML format, that works with the schema above.

```
  resources:
    limits:
      cpu: 100m
```

5. Determine if the add-on requires IAM permissions. If so, you need to (1) determine if you want to use EKS Pod Identities or IAM Roles for Service Accounts (IRSA), (2) determine the ARN of the IAM role to use with the add-on, and (3) determine the name of the Kubernetes service account used by the add-on. For more information, see the section called "Retrieve IAM information".

   - Amazon EKS suggests using EKS Pod Identities if the add-on supports it. This requires the Pod Identity Agent is installed on your cluster. For more information about using Pod Identities with Add-ons, see the section called "IAM roles".
   - If the add-on or your cluster is not setup for EKS Pod Identities, use IRSA. Confirm IRSA is setup on your cluster.
   - Review the Amazon EKS Add-ons documentation to determine if the add-on requires IAM permissions and the name of the associated Kubernetes service account.

   a. Create an Amazon EKS add-on. Copy the command that follows to your device. Make the following modifications to the command as needed and then run the modified command:

   - Replace *my-cluster* with the name of your cluster.
   - Replace *vpc-cni* with an add-on name returned in the output of the previous step that you want to create.
   - Replace *version-number* with the version returned in the output of the previous step that you want to use.
   - If the add-on doesn't require IAM permissions, delete *<service-account-configuration>*.
   - Do one of the following:
     - If the add-on (1) requires IAM permissions, and (2) your cluster uses EKS Pod Identities, replace *<service-account-configuration>* with the following pod identity association. Replace *<service-account-name>* with the service account name used by the add-on. Replace *<role-arn>* with the ARN of an IAM role. The role must have the trust policy required by EKS Pod Identities.

       ```
       --pod-identity-associations 'serviceAccount=<service-account-name>,roleArn=<role-arn>'
       ```

     - If the add-on (1) requires IAM permissions, and (2) your cluster uses IRSA, replace *<service-account-configuration>* with the following IRSA configuration. Replace *111122223333* with your account ID and *role-name* with the name of an existing IAM role that you've created. For instructions on creating the role, see the documentation for the add-on that you're creating. For a list of add-ons, see the section called " AWS Add-ons".

Specifying a service account role requires that you have an IAM OpenID Connect (OIDC) provider for your cluster. To determine whether you have one for your cluster, or to create one, see the section called “IAM OIDC provider”.

```
--service-account-role-arn arn:aws::iam::111122223333:role/role-name
```

- These example commands overwrites the `--configuration-values` option of any existing self-managed version of the add-on, if there is one. Replace this with the desired configuration values, such as a string or a file input. If you don't want to provide configuration values, then delete the `--configuration-values` option. If you don't want the AWS CLI to overwrite the configuration of an existing self-managed add-on, remove the *`--resolve-conflicts OVERWRITE`* option. If you remove the option, and the Amazon EKS add-on needs to overwrite the configuration of an existing self-managed add-on, then creation of the Amazon EKS add-on fails with an error message to help you resolve the conflict. Before specifying this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to manage, because those settings are overwritten with this option.

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 version-number \
     <service-account-configuration> --configuration-values '{"resources":
{"limits":{"cpu":"100m"}}}' --resolve-conflicts OVERWRITE
```

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 version-number \
    <service-account-configuration> --configuration-values 'file://example.yaml' --
resolve-conflicts OVERWRITE
```

For a full list of available options, see `create-addon` in the Amazon EKS Command Line Reference. If the add-on that you created has `aws-marketplace` listed in the `Owner` column of a previous step, then creation may fail, and you may receive an error message similar to the following error.

```
{
    "addon": {
        "addonName": "addon-name",
        "clusterName": "my-cluster",
        "status": "CREATE_FAILED",
        "addonVersion": "version",
        "health": {
```

```
            "issues": [
                {
                    "code": "AddonSubscriptionNeeded",
                    "message": "You are currently not subscribed to this add-on. To
 subscribe, visit the AWS Marketplace console, agree to the seller EULA, select the
 pricing type if required, then re-install the add-on"
                }
            ]
        }
    }
}
```

If you receive an error similar to the error in the previous output, visit the URL in the output of a previous step to subscribe to the add-on. Once subscribed, run the `create-addon` command again.

Update an Amazon EKS add-on

Amazon EKS doesn't automatically update an add-on when new versions are released or after you update your cluster to a new Kubernetes minor version. To update an add-on for an existing cluster, you must initiate the update. After you initiate the update, Amazon EKS updates the add-on for you. Before updating an add-on, review the current documentation for the add-on. For a list of available add-ons, see the section called " AWS Add-ons". If the add-on requires an IAM role, see the details for the specific add-on in Available Amazon EKS add-ons from AWS for details about creating the role.

Prerequisites

Complete the following before you create an add-on:

- Check if your add-on requires an IAM role. For more information, see the section called "Amazon EKS add-ons".
- Verify that the Amazon EKS add-on version is compatible with your cluster. For more information, see the section called "Verify compatibility".

Procedure

You can update an Amazon EKS add-on using `eksctl`, the AWS Management Console, or the AWS CLI.

Update add-on (eksctl)

1. Determine the current add-ons and add-on versions installed on your cluster. Replace *my-cluster* with the name of your cluster.

```
eksctl get addon --cluster my-cluster
```

An example output is as follows.

```
NAME        VERSION              STATUS   ISSUES   IAMROLE   UPDATE AVAILABLE
coredns     v1.8.7-eksbuild.2    ACTIVE   0
kube-proxy  v1.23.7-eksbuild.1   ACTIVE   0                  v1.23.8-eksbuild.2
vpc-cni     v1.10.4-eksbuild.1   ACTIVE   0                  v1.12.0-eksbuild.1,v1.11.4-
eksbuild.1,v1.11.3-eksbuild.1,v1.11.2-eksbuild.1,v1.11.0-eksbuild.1
```

Your output might look different, depending on which add-ons and versions that you have on your cluster. You can see that in the previous example output, two existing add-ons on the cluster have newer versions available in the `UPDATE AVAILABLE` column.

2. Update the add-on.

   a. Copy the command that follows to your device. Make the following modifications to the command as needed:

      - Replace *my-cluster* with the name of your cluster.
      - Replace *region-code* with the AWS Region that your cluster is in.
      - Replace *vpc-cni* with the name of an add-on returned in the output of the previous step that you want to update.
      - If you want to update to a version earlier than the latest available version, then replace *latest* with the version number returned in the output of the previous step that you want to use. Some add-ons have recommended versions. For more information, see the documentation for the add-on that you're updating. For a list of add-ons, see the section called " AWS Add-ons".* If the add-on uses a Kubernetes service account and IAM role, replace *111122223333* with your account ID and *role-name* with the name of an existing IAM role that you've created. For instructions on creating the role, see the documentation for the add-on that you're creating. For a list of add-ons, see the section called " AWS Add-ons". Specifying a service account role requires that you have an IAM OpenID Connect (OIDC) provider for your cluster. To determine whether you have one for your cluster, or to create one, see the section called "IAM OIDC provider".

If the add-on doesn't use a Kubernetes service account and IAM role, delete the `serviceAccountRoleARN: arn:aws:iam::111122223333:role/role-name` line.

- The *preserve* option preserves existing values for the add-on. If you have set custom values for add-on settings, and you don't use this option, Amazon EKS overwrites your values with its default values. If you use this option, then we recommend that you test any field and value changes on a non-production cluster before updating the add-on on your production cluster. If you change this value to `overwrite`, all settings are changed to Amazon EKS default values. If you've set custom values for any settings, they might be overwritten with Amazon EKS default values. If you change this value to none, Amazon EKS doesn't change the value of any settings, but the update might fail. If the update fails, you receive an error message to help you resolve the conflict.

```
cat >update-addon.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-cluster
  region: region-code

addons:
- name: vpc-cni
  version: latest
  serviceAccountRoleARN: arn:aws:iam::111122223333:role/role-name
  resolveConflicts: preserve
EOF
```

b. Run the modified command to create the `update-addon.yaml` file.

c. Apply the config file to your cluster.

```
eksctl update addon -f update-addon.yaml
```

For more information about updating add-ons, see Updating addons in the `eksctl` documentation.

Update add-on (AWS Console)

1. Open the Amazon EKS console.

2. In the left navigation pane, choose **Clusters**.
3. Choose the name of the cluster that you want to update the add-on for.
4. Choose the **Add-ons** tab.
5. Choose the add-on that you want to update.
6. Choose **Edit**.
7. On the **Configure** ***name of addon*** page, do the following:

   a. Choose the **Version** that you'd like to use. The add-on might have a recommended version. For more information, see the documentation for the add-on that you're updating. For a list of add-ons, see the section called " AWS Add-ons".

   b. You have two options for configuring roles for add-ons: EKS Pod Identities IAM role and IAM roles for service accounts (IRSA). Follow the appropriate step below for your preferred option. If all of the add-ons that you selected have **Requires subscription** under **Status**, choose **Next**. For the add-ons that don't have **Requires subscription** under **Status**, do the following:

      i. For **Pod Identity IAM role for service account**, you can either use an existing EKS Pod Identity IAM role or create one using the **Create Recommended Role** button. This field will only provide options with the appropriate trust policy. If there's no role to select, then you don't have an existing role with a matching trust policy. To configure an EKS Pod Identity IAM role for service accounts of the selected add-on, choose **Create recommended role**. The role creation wizard opens in a separate window. The wizard will automatically populate the role information as follows. For each add-on where you want to create the EKS Pod Identity IAM role, complete the steps in the IAM wizard as follows.

         - On the **Select trusted entity** step, the AWS service option for **EKS** and the use case for **EKS - Pod Identity** are preselected, and the appropriate trust policy will be automatically populated for the add-on. For example, the role will be created with the appropriate trust policy containing the pods.eks.amazonaws.com IAM Principal as detailed in the section called "Benefits of EKS Pod Identities". Choose **Next**.
         - On the **Add permissions** step, the appropriate managed policy for the role policy is preselected for the add-on. For example, for the Amazon VPC CNI add-on, the role will be created with the managed policy `AmazonEKS_CNI_Policy` as detailed in the section called "Amazon VPC CNI plugin for Kubernetes". Choose **Next**.
         - On the **Name, review, and create** step, in **Role name**, the default role name is automatically populated for the add-on. For example, for the **Amazon VPC CNI** add-on, the role will be created with the name **AmazonEKSPodIdentityAmazonVPCCNIRole**. In **Description**, the default description is automatically populated with the appropriate

description for the add-on. For example, for the Amazon VPC CNI add-on, the role will be created with the description **Allows pods running in Amazon EKS cluster** to access AWS resources. In **Trust policy**, view the populated trust policy for the add-on. Choose **Create role**.

> **Note**
>
> Retaining the default role name enables EKS to pre-select the role for add-ons in new clusters or when adding add-ons to existing clusters. You can still override this name and the role will be available for the add-on across your clusters, but the role will need to be manually selected from the drop down.

ii. For add-ons that do not have **Requires subscription** under **Status** and where you want to configure roles using IRSA, see the documentation for the add-on that you're creating to create an IAM policy and attach it to a role. For a list of add-ons, see the section called " AWS Add-ons". Selecting an IAM role requires that you have an IAM OpenID Connect (OIDC) provider for your cluster. To determine whether you have one for your cluster, or to create one, see the section called "IAM OIDC provider".

c. Expand the **Optional configuration settings**.

d. In **Configuration values**, enter any add-on specific configuration information. For more information, see the documentation for the add-on that you're updating. For a list of add-ons, see the section called " AWS Add-ons"… For **Conflict resolution method**, select one of the options. If you have set custom values for add-on settings, we recommend the **Preserve** option. If you don't choose this option, Amazon EKS overwrites your values with its default values. If you use this option, then we recommend that you test any field and value changes on a non-production cluster before updating the add-on on your production cluster. If you change this value to overwrite, all settings are changed to Amazon EKS default values. If you've set custom values for any settings, they might be overwritten with Amazon EKS default values. If you change this value to none, Amazon EKS doesn't change the value of any settings, but the update might fail. If the update fails, you receive an error message to help you resolve the conflict.

8. Choose **Save changes**.

Update add-on (AWS CLI)

1. You need version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version, use `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*. The AWS CLI version that is installed in AWS CloudShell might also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the *AWS CloudShell User Guide*.
2. See a list of installed add-ons. Replace *`my-cluster`* with the name of your cluster.

```
aws eks list-addons --cluster-name my-cluster
```

   An example output is as follows.

```
{
    "addons": [
        "coredns",
        "kube-proxy",
        "vpc-cni"
    ]
}
```

3. View the current version of the add-on that you want to update. Replace *`my-cluster`* with your cluster name and *`vpc-cni`* with the name of the add-on that you want to update.

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
 "addon.addonVersion" --output text
```

   An example output is as follows.

```
v1.10.4-eksbuild.1
```

4. Determine which versions of the add-on are available for your cluster's version. Replace *`1.32`* with your cluster's version and *`vpc-cni`* with the name of the add-on that you want to update.

```
aws eks describe-addon-versions --kubernetes-version 1.32 --addon-name vpc-cni \
```

```
    --query 'addons[].addonVersions[].{Version: addonVersion, Defaultversion: compatibilities[0].defaultVersion}' --output table
```

An example output is as follows.

```
------------------------------------------
|          DescribeAddonVersions         |
+-----------------+----------------------+
| Defaultversion  |       Version        |
+-----------------+----------------------+
|  False          |  v1.12.0-eksbuild.1  |
|  True           |  v1.11.4-eksbuild.1  |
|  False          |  v1.10.4-eksbuild.1  |
|  False          |  v1.9.3-eksbuild.1   |
+-----------------+----------------------+
```

The version with `True` in the `Defaultversion` column is the version that the add-on is created with, by default.

5. Update your add-on. Copy the command that follows to your device. Make the following modifications to the command, as needed, and then run the modified command. For more information about this command, see update-addon in the Amazon EKS Command Line Reference.

   - Replace *my-cluster* with the name of your cluster.
   - Replace *vpc-cni* with the name of the add-on that you want to update that was returned in the output of a previous step.
   - Replace *version-number* with the version returned in the output of the previous step that you want to update to. Some add-ons have recommended versions. For more information, see the documentation for the add-on that you're updating. For a list of add-ons, see the section called " AWS Add-ons".* If the add-on uses a Kubernetes service account and IAM role, replace *111122223333* with your account ID and *role-name* with the name of an existing IAM role that you've created. For instructions on creating the role, see the documentation for the add-on that you're creating. For a list of add-ons, see the section called " AWS Add-ons". Specifying a service account role requires that you have an IAM OpenID Connect (OIDC) provider for your cluster. To determine whether you have one for your cluster, or to create one, see the section called "IAM OIDC provider".

     If the add-on doesn't use a Kubernetes service account and IAM role, delete the `serviceAccountRoleARN: arn:aws:iam::111122223333:role/role-name` line.

- The `--resolve-conflicts PRESERVE` option preserves existing values for the add-on. If you have set custom values for add-on settings, and you don't use this option, Amazon EKS overwrites your values with its default values. If you use this option, then we recommend that you test any field and value changes on a non-production cluster before updating the add-on on your production cluster. If you change this value to `OVERWRITE`, all settings are changed to Amazon EKS default values. If you've set custom values for any settings, they might be overwritten with Amazon EKS default values. If you change this value to `NONE`, Amazon EKS doesn't change the value of any settings, but the update might fail. If the update fails, you receive an error message to help you resolve the conflict.
- If you want to remove all custom configuration then perform the update using the *`--configuration-values '{}'`* option. This sets all custom configuration back to the default values. If you don't want to change your custom configuration, don't provide the *`--configuration-values`* flag. If you want to adjust a custom configuration then replace *`{}`* with the new parameters.

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 version-number \
    --service-account-role-arn arn:aws:iam::111122223333:role/role-name --
configuration-values '{}' --resolve-conflicts PRESERVE
```

6. Check the status of the update. Replace *`my-cluster`* with the name of your cluster and *`vpc-cni`* with the name of the add-on you're updating.

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni
```

An example output is as follows.

```
{
    "addon": {
        "addonName": "vpc-cni",
        "clusterName": "my-cluster",
        "status": "UPDATING",
    }
}
```

The update is complete when the status is `ACTIVE`.

Verify Amazon EKS add-on version compatibility with a cluster

Before you create an Amazon EKS add-on you need to verify that the Amazon EKS add-on version is compatible with your cluster.

Use the describe-addon-versions API to list the available versions of EKS add-ons, and which Kubernetes versions each addon version supports.

1. Verify the AWS CLI is installed and working with `aws sts get-caller-identity`. If this command doesn't work, learn how to Get started with the AWS CLI.
2. Determine the name of the add-on you want to retrieve version compatibility information for, such as `amazon-cloudwatch-observability`.
3. Determine the Kubernetes version of your cluster, such as `1.32`.
4. Use the AWS CLI to retrieve the addon versions that are compatible with the Kubernetes version of your cluster.

```
aws eks describe-addon-versions --addon-name amazon-cloudwatch-observability --
kubernetes-version 1.32
```

An example output is as follows.

```
{
    "addons": [
        {
            "addonName": "amazon-cloudwatch-observability",
            "type": "observability",
            "addonVersions": [
                {
                    "addonVersion": "vX.X.X-eksbuild.X",
                    "architecture": [
                        "amd64",
                        "arm64"
                    ],
                    "computeTypes": [
                        "ec2",
                        "auto",
                        "hybrid"
                    ],
                    "compatibilities": [
                        {
```

```
                                "clusterVersion": "1.32",
                                "platformVersions": [
                                    "*"
                                ],
                                "defaultVersion": true
                            }
                        ],
                    }
                ]
            }
        ]
}
```

This output shows that addon version `vX.X.X-eksbuild.X` is compatible with Kubernetes cluster version `1.32`.

Add-on compatibility with compute types

The `computeTypes` field in the `describe-addon-versions` output indicates an add-on's compatibility with EKS Auto Mode Managed Nodes or Hybrid Nodes. Add-ons marked `auto` work with EKS Auto Mode's cloud-based, AWS-managed infrastructure, while those marked `hybrid` can run on on-premises nodes connected to the EKS cloud control plane.

For more information, see the section called "Considerations for Amazon EKS Auto Mode".

Remove an Amazon EKS add-on from a cluster

You can remove an Amazon EKS add-on from your cluster using `eksctl`, the AWS Management Console, or the AWS CLI.

When you remove an Amazon EKS add-on from a cluster:

- There is no downtime for the functionality that the add-on provides.
- If you are using IAM Roles for Service Accounts (IRSA) and the add-on has an IAM role associated with it, the IAM role isn't removed.
- If you are using Pod Identities, any Pod Identity Associations owned by the add-on are removed. If you specify the `--preserve` option to the AWS CLI, the associations are preserved.
- Amazon EKS stops managing settings for the add-on.
- The console stops notifying you when new versions are available.

- You can't update the add-on using any AWS tools or APIs.
- You can choose to leave the add-on software on your cluster so that you can self-manage it, or you can remove the add-on software from your cluster. You should only remove the add-on software from your cluster if there are no resources on your cluster are dependent on the functionality that the add-on provides.

Prerequisites

Complete the following before you create an add-on:

- An existing Amazon EKS cluster. To deploy one, see *Get started*.
- Check if your add-on requires an IAM role. For more information, see
- Version `0.204.0` or later of the `eksctl` command line tool installed on your device or AWS CloudShell. To install or update `eksctl`, see Installation in the `eksctl` documentation..

Procedure

You have two options when removing an Amazon EKS add-on.

- **Preserve add-on software on your cluster** – This option removes Amazon EKS management of any settings. It also removes the ability for Amazon EKS to notify you of updates and automatically update the Amazon EKS add-on after you initiate an update. However, it preserves the add-on software on your cluster. This option makes the add-on a self-managed installation, rather than an Amazon EKS add-on. With this option, there's no downtime for the add-on.
- **Remove add-on software entirely from your cluster** – We recommend that you remove the Amazon EKS add-on from your cluster only if there are no resources on your cluster that are dependent on it.

You can remove an Amazon EKS add-on using `eksctl`, the AWS Management Console, or the AWS CLI.

Remove add-on (eksctl)

1. Determine the current add-ons installed on your cluster. Replace *`my-cluster`* with the name of your cluster.

```
eksctl get addon --cluster my-cluster
```

An example output is as follows.

```
NAME        VERSION              STATUS   ISSUES   IAMROLE   UPDATE AVAILABLE
coredns     v1.8.7-eksbuild.2    ACTIVE   0
kube-proxy  v1.23.7-eksbuild.1   ACTIVE   0
vpc-cni     v1.10.4-eksbuild.1   ACTIVE   0
[...]
```

Your output might look different, depending on which add-ons and versions that you have on your cluster.

2. Remove the add-on. Replace *my-cluster* with the name of your cluster and *name-of-add-on* with the name of the add-on returned in the output of the previous step that you want to remove. If you remove the *--preserve* option, in addition to Amazon EKS no longer managing the add-on, the add-on software is deleted from your cluster.

```
eksctl delete addon --cluster my-cluster --name name-of-addon --preserve
```

For more information about removing add-ons, see Deleting addons in the `eksctl` documentation.

Remove add-on (AWS Console)

1. Open the Amazon EKS console.
2. In the left navigation pane, choose **Clusters**.
3. Choose the name of the cluster that you want to remove the Amazon EKS add-on for.
4. Choose the **Add-ons** tab.
5. Choose the add-on that you want to remove.
6. Choose **Remove**.
7. In the **Remove: *name of addon*** confirmation dialog box, do the following:
   a. If you want Amazon EKS to stop managing settings for the add-on, select **Preserve on cluster**. Do this if you want to retain the add-on software on your cluster. This is so that you can manage all of the settings of the add-on on your own.

b. Enter the add-on name.

c. Choose **Remove**.

Remove add-on (AWS CLI)

1. You need version `0.204.0` or later of the `eksctl` command line tool installed on your device or AWS CloudShell. To install or update `eksctl`, see Installation in the `eksctl` documentation.

2. See a list of installed add-ons. Replace *my-cluster* with the name of your cluster.

```
aws eks list-addons --cluster-name my-cluster
```

An example output is as follows.

```
{
    "addons": [
        "coredns",
        "kube-proxy",
        "vpc-cni",
        "name-of-addon"
    ]
}
```

3. Remove the installed add-on. Replace *my-cluster* with the name of your cluster and *name-of-add-on* with the name of the add-on that you want to remove. Removing *--preserve* deletes the add-on software from your cluster.

```
aws eks delete-addon --cluster-name my-cluster --addon-name name-of-addon --preserve
```

The abbreviated example output is as follows.

```
{
    "addon": {
        "addonName": "name-of-add-on",
        "clusterName": "my-cluster",
        "status": "DELETING",
    }
}
```

4. Check the status of the removal. Replace *my-cluster* with the name of your cluster and *name-of-addon* with the name of the add-on that you're removing.

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

After the add-on is removed, the example output is as follows.

```
An error occurred (ResourceNotFoundException) when calling the DescribeAddon
 operation: No addon: name-of-addon found in cluster: my-cluster
```

IAM roles for Amazon EKS add-ons

Certain Amazon EKS add-ons need IAM roles and permissions to call AWS APIs. For example, the Amazon VPC CNI add-on calls certain AWS APIs to configure networking resources in your account. These add-ons need to be granted permission using IAM. More specifically, the service account of the pod running the add-on needs to be associated with an IAM role with a specific IAM policy.

The recommended way to grant AWS permissions to cluster workloads is using the Amazon EKS feature Pod Identities. You can use a **Pod Identity Association** to map the service account of an add-on to an IAM role. If a pod uses a service account that has an association, Amazon EKS sets environment variables in the containers of the pod. The environment variables configure the AWS SDKs, including the AWS CLI, to use the EKS Pod Identity credentials. For more information, see the section called "Pod Identity"

Amazon EKS add-ons can help manage the life cycle of pod identity associations corresponding to the add-on. For example, you can create or update an Amazon EKS add-on and the necessary pod identity association in a single API call. Amazon EKS also provides an API for retrieving suggested IAM policies.

1. Confirm that Amazon EKS pod identity agent is setup on your cluster.
2. Determine if the add-on you want to install requires IAM permissions using the `describe-addon-versions` AWS CLI operation. If the `requiresIamPermissions` flag is `true`, then you should use the `describe-addon-configurations` operation to determine the permissions needed by the addon. The response includes a list of suggested managed IAM policies.
3. Retrieve the name of the Kubernetes Service Account and the IAM policy using the `describe-addon-configuration` CLI operation. Evaluate the scope of the suggested policy against your security requirements.

4. Create an IAM role using the suggested permissions policy, and the trust policy required by Pod Identity. For more information, see the section called “Create a Pod Identity association (AWS Console)”.
5. Create or update an Amazon EKS add-on using the CLI. Specify at least one pod identity association. A pod identity association is the name of a Kubernetes service account, and the ARN of the IAM role.
    - Pod identity associations created using the add-on APIs are owned by the respective add-on. If you delete the add-on, the pod identity association is also deleted. You can prevent this cascading delete by using the `preserve` option when deleting an addon using the AWS CLI or API. You also can directly update or delete the pod identity association if necessary. Add-ons can’t assume ownership of existing pod identity associations. You must delete the existing association and re-create it using an add-on create or update operation.
    - Amazon EKS recommends using pod identity associations to manage IAM permissions for add-ons. The previous method, IAM roles for service accounts (IRSA), is still supported. You can specify both an IRSA `serviceAccountRoleArn` and a pod identity association for an add-on. If the EKS pod identity agent is installed on the cluster, the `serviceAccountRoleArn` will be ignored, and EKS will use the provided pod identity association. If Pod Identity is not enabled, the `serviceAccountRoleArn` will be used.
    - If you update the pod identity associations for an existing add-on, Amazon EKS initiates a rolling restart of the add-on pods.

Retrieve IAM information about an Amazon EKS add-on

Before you create an add-on, use the AWS CLI to determine:

- If the add-on requires IAM permissions
- The suggested IAM policy to use

Procedure

1. Determine the name of the add-on you want to install, and the Kubernetes version of your cluster. For more information about add-ons, see the section called “Amazon EKS add-ons”.
2. Use the AWS CLI to determine if the add-on requires IAM permissions.

```
aws eks describe-addon-versions \
--addon-name <addon-name> \
```

```
--kubernetes-version <kubernetes-version>
```

For example:

```
aws eks describe-addon-versions \
--addon-name aws-ebs-csi-driver \
--kubernetes-version 1.30
```

Review the following sample output. Note that `requiresIamPermissions` is `true`, and the default add-on version. You need to specify the add-on version when retrieving the recommended IAM policy.

```
{
    "addons": [
        {
            "addonName": "aws-ebs-csi-driver",
            "type": "storage",
            "addonVersions": [
                {
                    "addonVersion": "v1.31.0-eksbuild.1",
                    "architecture": [
                        "amd64",
                        "arm64"
                    ],
                    "compatibilities": [
                        {
                            "clusterVersion": "1.30",
                            "platformVersions": [
                                "*"
                            ],
                            "defaultVersion": true
                        }
                    ],
                    "requiresConfiguration": false,
                    "requiresIamPermissions": true
                },
[...]
```

3. If the add-on requires IAM permissions, use the AWS CLI to retrieve a recommended IAM policy.

```
aws eks describe-addon-configuration \
--query podIdentityConfiguration \
```

```
--addon-name <addon-name> \
--addon-version <addon-version>
```

For example:

```
aws eks describe-addon-configuration \
--query podIdentityConfiguration \
--addon-name aws-ebs-csi-driver \
--addon-version v1.31.0-eksbuild.1
```

Review the following output. Note the `recommendedManagedPolicies`.

```
[
    {
        "serviceAccount": "ebs-csi-controller-sa",
        "recommendedManagedPolicies": [
            "arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy"
        ]
    }
]
```

4. Create an IAM role and attach the recommended Managed Policy. Alternatively, review the managed policy and scope down the permissions as appropriate. For more information see the section called “Create a Pod Identity association (AWS Console)”.

Pod Identity Support Reference

The following table indicates if certain Amazon EKS add-ons support EKS Pod Identity.

Add-on Name	Pod Identity Support	Minimum Version Required
Amazon EBS CSI Driver	Yes	v1.26.0-eksbuild.1
Amazon VPC CNI	Yes	v1.15.5-eksbuild.1
Amazon EFS CSI Driver	Yes	v2.0.5-eksbuild.1
AWS Distro for OpenTelemetry	Yes	v0.94.1-eksbuild.1

Add-on Name	Pod Identity Support	Minimum Version Required
Mountpoint for Amazon S3 CSI Driver	No	N/A
Amazon CloudWatch Observability agent	Yes	v3.1.0-eksbuild.1

This table was last updated on October 28, 2024.

Use Pod Identities to assign an IAM role to an Amazon EKS add-on

Certain Amazon EKS add-ons need IAM roles and permissions. Before you add update an Amazon EKS add-on to use a Pod Identity association, verify the role and policy to use. For more information, see the section called “Retrieve IAM information”.

1. Determine:
   - `cluster-name` – The name of the cluster to install the add-on onto.
   - `addon-name` – The name of the add-on to install.
   - `service-account-name` – The name of the Kubernetes Service Account used by the add-on.
   - `iam-role-arn` – The ARN of an IAM role with sufficient permissions for the add-on. The role must have the required trust policy for EKS Pod Identity. For more information see the section called “Create a Pod Identity association (AWS Console)”.
2. Update the add-on using the AWS CLI. You can also specify Pod Identity associations when creating an add-on, using the same `--pod-identity-assocations` syntax. Note that when you specify pod identity associations while updating an add-on, all previous pod identity associations are overwritten.

```
aws eks update-addon --cluster-name <cluster-name> \
--addon-name <addon-name> \
--pod-identity-associations 'serviceAccount=<service-account-name>,roleArn=<role-
arn>'
```

   For example:

```
aws eks update-addon --cluster-name mycluster \
```

```
--addon-name aws-ebs-csi-driver \
--pod-identity-associations 'serviceAccount=ebs-csi-controller-
sa,roleArn=arn:aws:iam::123456789012:role/StorageDriver'
```

3. Validate the Pod Identity association was created:

```
aws eks list-pod-identity-associations --cluster-name <cluster-name>
```

If successful, you should see output similar to the following. Note the OwnerARN of the EKS add-on.

```
{
    "associations": [
        {
            "clusterName": "mycluster",
            "namespace": "kube-system",
            "serviceAccount": "ebs-csi-controller-sa",
            "associationArn": "arn:aws:eks:us-
west-2:123456789012:podidentityassociation/mycluster/a-4wvljrezsukshq1bv",
            "associationId": "a-4wvljrezsukshq1bv",
            "ownerArn": "arn:aws:eks:us-west-2:123456789012:addon/mycluster/aws-ebs-
csi-driver/9cc7ce8c-2e15-b0a7-f311-426691cd8546"
        }
    ]
}
```

Remove Pod Identity associations from an Amazon EKS add-on

Remove the Pod Identity associations from an Amazon EKS add-on.

1. Determine:
   - `cluster-name` - The name of the EKS cluster to install the add-on onto.
   - `addon-name` - The name of the Amazon EKS add-on to install.
2. Update the addon to specify an empty array of pod identity associations.

```
aws eks update-addon --cluster-name <cluster-name> \
--addon-name <addon-name> \
--pod-identity-associations "[]"
```

Troubleshoot Pod Identities for EKS add-ons

If your add-ons are encountering errors while attempting AWS API, SDK, or CLI operations, confirm the following:

- The Pod Identity Agent is installed in your cluster.
  - For information about how to install the Pod Identity Agent, see the section called "Set up the Agent".
- The Add-on has a valid Pod Identity association.
  - Use the AWS CLI to retrieve the associations for the service account name used by the add-on.

```
aws eks list-pod-identity-associations --cluster-name <cluster-name>
```

- The IAM role has the required trust policy for Pod Identities.
  - Use the AWS CLI to retrieve the trust policy for an add-on.

```
aws iam get-role --role-name <role-name> --query Role.AssumeRolePolicyDocument
```

- The IAM role has the necessary permissions for the add-on.
  - Use AWS CloudTrail to review `AccessDenied` or `UnauthorizedOperation` events .
- The service account name in the pod identity association matches the service account name used by the add-on.
  - For information about the available add-ons, see the section called " AWS Add-ons".

Determine fields you can customize for Amazon EKS add-ons

Amazon EKS add-ons are installed to your cluster using standard, best practice configurations. For more information about adding an Amazon EKS add-on to your cluster, see the section called "Amazon EKS add-ons".

You may want to customize the configuration of an Amazon EKS add-on to enable advanced features. Amazon EKS uses the Kubernetes server-side apply feature to enable management of an add-on by Amazon EKS without overwriting your configuration for settings that aren't managed by Amazon EKS. For more information, see Server-Side Apply in the Kubernetes documentation. To achieve this, Amazon EKS manages a minimum set of fields for every add-on that it installs. You can modify all fields that aren't managed by Amazon EKS, or another Kubernetes control plane process such as `kube-controller-manager`, without issue.

> **Important**
>
> Modifying a field managed by Amazon EKS prevents Amazon EKS from managing the add-on and may result in your changes being overwritten when an add-on is updated.

Field management syntax

When you view details for a Kubernetes object, both managed and unmanaged fields are returned in the output. Managed fields can be either of the following types:

- **Fully managed** – All keys for the field are managed by Amazon EKS. Modifications to any value causes a conflict.
- **Partially managed** – Some keys for the field are managed by Amazon EKS. Only modifications to the keys explicitly managed by Amazon EKS cause a conflict.

Both types of fields are tagged with `manager: eks`.

Each key is either a `.` representing the field itself, which always maps to an empty set, or a string that represents a sub-field or item. The output for field management consists of the following types of declarations:

- `f:`*`name`* , where *`name`* is the name of a field in a list.
- `k:`*`keys`* , where *`keys`* is a map of a list item's fields.
- `v:`*`value`* , where *`value`* is the exact JSON formatted value of a list item.
- `i:`*`index`* , where *`index`* is position of an item in the list.

The following portions of output for the CoreDNS add-on illustrate the previous declarations:

- **Fully managed fields** – If a managed field has an `f:` (field) specified, but no `k:` (key), then the entire field is managed. Modifications to any values in this field cause a conflict.

  In the following output, you can see that the container named `coredns` is managed by `eks`. The `args`, `image`, and `imagePullPolicy` sub-fields are also managed by `eks`. Modifications to any values in these fields cause a conflict.

  ```
  [...]
  ```

```
f:containers:
  k:{"name":"coredns"}:
  .: {}
  f:args: {}
  f:image: {}
  f:imagePullPolicy: {}
[...]
manager: eks
[...]
```

- **Partially managed fields** – If a managed key has a value specified, the declared keys are managed for that field. Modifying the specified keys cause a conflict.

  In the following output, you can see that `eks` manages the `config-volume` and `tmp` volumes set with the `name` key.

```
[...]
f:volumes:
  k:{"name":"config-volume"}:
    .: {}
    f:configMap:
      f:items: {}
      f:name: {}
    f:name: {}
  k:{"name":"tmp"}:
    .: {}
    f:name: {}
[...]
manager: eks
[...]
```

- **Adding keys to partially managed fields** – If only a specific key value is managed, you can safely add additional keys, such as arguments, to a field without causing a conflict. If you add additional keys, make sure that the field isn't managed first. Adding or modifying any value that is managed causes a conflict.

  In the following output, you can see that both the `name` key and `name` field are managed. Adding or modifying any container name causes a conflict with this managed key.

```
[...]
f:containers:
  k:{"name":"coredns"}:
```

```
[...]
    f:name: {}
[...]
manager: eks
[...]
```

Procedure

You can use kubectl to see which fields are managed by Amazon EKS for any Amazon EKS add-on.

You can modify all fields that aren't managed by Amazon EKS, or another Kubernetes control plane process such as kube-controller-manager, without issue.

1. Determine which add-on that you want to examine. To see all of the deployments and DaemonSets deployed to your cluster, see the section called "Access cluster resources".
2. View the managed fields for an add-on by running the following command:

   ```
   kubectl get type/add-on-name -n add-on-namespace -o yaml
   ```

   For example, you can see the managed fields for the CoreDNS add-on with the following command.

   ```
   kubectl get deployment/coredns -n kube-system -o yaml
   ```

   Field management is listed in the following section in the returned output.

   ```
   [...]
   managedFields:
     - apiVersion: apps/v1
       fieldsType: FieldsV1
       fieldsV1:
   [...]
   ```

> **Note**
>
> If you don't see `managedFields` in the output, add `--show-managed-fields` to the command and run it again. The version of `kubectl` that you're using determines whether managed fields are returned by default.

Next steps

Customize the fields not owned by AWS for you add-on.

Validate container image signatures during deployment

If you use AWS Signer and want to verify signed container images at the time of deployment, you can use one of the following solutions:

- Gatekeeper and Ratify – Use Gatekeeper as the admission controller and Ratify configured with an AWS Signer plugin as a web hook for validating signatures.
- Kyverno – A Kubernetes policy engine configured with an AWS Signer plugin for validating signatures.

> **Note**
>
> Before verifying container image signatures, configure the Notation trust store and trust policy, as required by your selected admission controller.

Organize and monitor cluster resources

This chapter includes the following topics to help you manage your cluster. You can also view information about your Kubernetes resources with the AWS Management Console.

- The Kubernetes Dashboard is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage applications running in the cluster and troubleshoot them, as well as manage the cluster itself. For more information, see The Kubernetes Dashboard GitHub repository.
- the section called "Metrics server" – The Kubernetes Metrics Server is an aggregator of resource usage data in your cluster. It isn't deployed by default in your cluster, but is used by Kubernetes add-ons, such as the Kubernetes Dashboard and the section called "Horizontal Pod Autoscaler". In this topic you learn how to install the Metrics Server.
- the section called "Deploy apps with Helm" – The Helm package manager for Kubernetes helps you install and manage applications on your Kubernetes cluster. This topic helps you install and run the Helm binaries so that you can install and manage charts using the Helm CLI on your local computer.
- the section called "Tagging your resources" – To help you manage your Amazon EKS resources, you can assign your own metadata to each resource in the form of *tags*. This topic describes tags and shows you how to create them.
- the section called "Service quotas" – Your AWS account has default quotas, formerly referred to as limits, for each AWS service. Learn about the quotas for Amazon EKS and how to increase them.

Monitor and optimize Amazon EKS cluster costs

Cost monitoring is an essential aspect of managing your Kubernetes clusters on Amazon EKS. By gaining visibility into your cluster costs, you can optimize resource utilization, set budgets, and make data-driven decisions about your deployments. Amazon EKS provides two cost monitoring solutions, each with its own unique advantages, to help you track and allocate your costs effectively:

AWS Billing split cost allocation data for Amazon EKS — This native feature integrates seamlessly with the AWS Billing Console, allowing you to analyze and allocate costs using the same familiar interface and workflows you use for other AWS services. With split cost allocation, you can gain insights into your Kubernetes costs directly alongside your other AWS spend, making it easier to optimize costs holistically across your AWS environment. You can also leverage existing AWS Billing features like Cost Categories and Cost Anomaly Detection to further enhance your cost management capabilities. For more information, see Understanding split cost allocation data in the AWS Billing User Guide.

Kubecost — Amazon EKS supports Kubecost, a Kubernetes cost monitoring tool. Kubecost offers a feature-rich, Kubernetes-native approach to cost monitoring, providing granular cost breakdowns by Kubernetes resources, cost optimization recommendations, and out-of-the-box dashboards and reports. Kubecost also retrieves accurate pricing data by integrating with the AWS Cost and Usage Report, ensuring you get a precise view of your Amazon EKS costs. Learn how to Install Kubecost.

View costs by Pod in AWS billing with split cost allocation

Cost monitoring using AWS split cost allocation data for Amazon EKS

You can use AWS split cost allocation data for Amazon EKS to get granular cost visibility for your Amazon EKS clusters. This enables you to analyze, optimize, and chargeback cost and usage for your Kubernetes applications. You allocate application costs to individual business units and teams based on Amazon EC2 CPU and memory resources consumed by your Kubernetes application. Split cost allocation data for Amazon EKS gives visibility into cost per Pod, and enables you to aggregate the cost data per Pod using namespace, cluster, and other Kubernetes primitives. The following are examples of Kubernetes primitives that you can use to analyze Amazon EKS cost allocation data.

- Cluster name
- Deployment
- Namespace
- Node
- Workload Name
- Workload Type

For more information about using split cost allocation data, see Understanding split cost allocation data in the AWS Billing User Guide.

Set up Cost and Usage Reports

You can turn on Split Cost Allocation Data for EKS in the Cost Management Console, AWS Command Line Interface, or the AWS SDKs.

Use the following for *Split Cost Allocation Data*:

1. Opt in to Split Cost Allocation Data. For more information, see Enabling split cost allocation data in the AWS Cost and Usage Report User Guide.
2. Include the data in a new or existing report.
3. View the report. You can use the Billing and Cost Management console or view the report files in Amazon Simple Storage Service.

Install Kubecost and access dashboard

Amazon EKS supports Kubecost, which you can use to monitor your costs broken down by Kubernetes resources including Pods, nodes, namespaces, and labels. This topic covers installing Kubecost, and accessing the Kubecost dashboard.

Amazon EKS provides an AWS optimized bundle of Kubecost for cluster cost visibility. You can use your existing AWS support agreements to obtain support. For more information about the available versions of Kubecost, see the section called "Learn more about Kubecost".

As a Kubernetes platform administrator and finance leader, you can use Kubecost to visualize a breakdown of Amazon EKS charges, allocate costs, and charge back organizational units such as application teams. You can provide your internal teams and business units with transparent and accurate cost data based on their actual AWS bill. Moreover, you can also get customized recommendations for cost optimization based on their infrastructure environment and usage patterns within their clusters.

> **Note**
>
> Kubecost v2 introduces several major new features. Learn more about Kubecost v2.

For more information about Kubecost, see the Kubecost documentation.

Install Kubecost using Amazon EKS Add-ons

Note

Install Kubecost as an Amazon EKS Add-on and benefit from additional features at no additional cost with the Amazon EKS optimized Kubecost bundle. For more information, see the section called "Kubecost v2".

Amazon EKS Add-ons reduce the complexity of upgrading Kubecost, and managing licenses. EKS Add-ons are integrated with the AWS marketplace.

1. View Kubecost in the AWS Marketplace console and subscribe.
2. Determine the name of your cluster, and the region. Verify you are logged into the AWS CLI with sufficient permissions to manage EKS.
3. Create the Kubecost addon.

```
aws eks create-addon --addon-name kubecost_kubecost --cluster-name $YOUR_CLUSTER_NAME
 --region $AWS_REGION
```

Learn how to remove an EKS Add-on, such as Kubecost.

Install Kubecost using Helm

- An existing Amazon EKS cluster. To deploy one, see *Get started*. The cluster must have Amazon EC2 nodes because you can't run Kubecost on Fargate nodes.
- The `kubectl` command line tool is installed on your device or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. For example, if your cluster version is `1.29`, you can use `kubectl` version `1.28`, `1.29`, or `1.30` with it. To install or upgrade `kubectl`, see the section called "Set up kubectl and eksctl".
- Helm version 3.9.0 or later configured on your device or AWS CloudShell. To install or update Helm, see the section called "Deploy apps with Helm".
- If your cluster is version `1.23` or later, you must have the Store Kubernetes volumes with Amazon EBS installed on your cluster.

  1. Determine the version of Kubecost to install. You can see the available versions at kubecost/cost-analyzer in the Amazon ECR Public Gallery. For more information about the compatibility

of Kubecost versions and Amazon EKS, see the Environment Requirements in the Kubecost documentation.

2. Install Kubecost with the following command. Replace *kubecost-version* with the value retrieved from ECR, such as *1.108.1*.

```
helm upgrade -i kubecost oci://public.ecr.aws/kubecost/cost-analyzer --version
 kubecost-version \
    --namespace kubecost --create-namespace \
    -f https://raw.githubusercontent.com/kubecost/cost-analyzer-helm-chart/develop/
cost-analyzer/values-eks-cost-monitoring.yaml
```

   Kubecost releases new versions regularly. You can update your version using helm upgrade. By default, the installation includes a local Prometheus server and `kube-state-metrics`. You can customize your deployment to use Amazon Managed Service for Prometheus by following the documentation in Integrating with Amazon EKS cost monitoring. For a list of all other settings that you can configure, see the sample configuration file on GitHub.

   You can remove Kubecost from your cluster with the following commands.

```
helm uninstall kubecost --namespace kubecost
kubectl delete ns kubecost
```

Access Kubecost Dashboard

1. Make sure the required Pods are running.

```
kubectl get pods -n kubecost
```

   An example output is as follows.

```
NAME                                            READY   STATUS    RESTARTS   AGE
kubecost-cost-analyzer-b9788c99f-5vj5b          2/2     Running   0          3h27m
kubecost-kube-state-metrics-99bb8c55b-bn2br     1/1     Running   0          3h27m
kubecost-prometheus-server-7d9967bfc8-9c8p7     2/2     Running   0          3h27m
```

2. On your device, enable port-forwarding to expose the Kubecost dashboard.

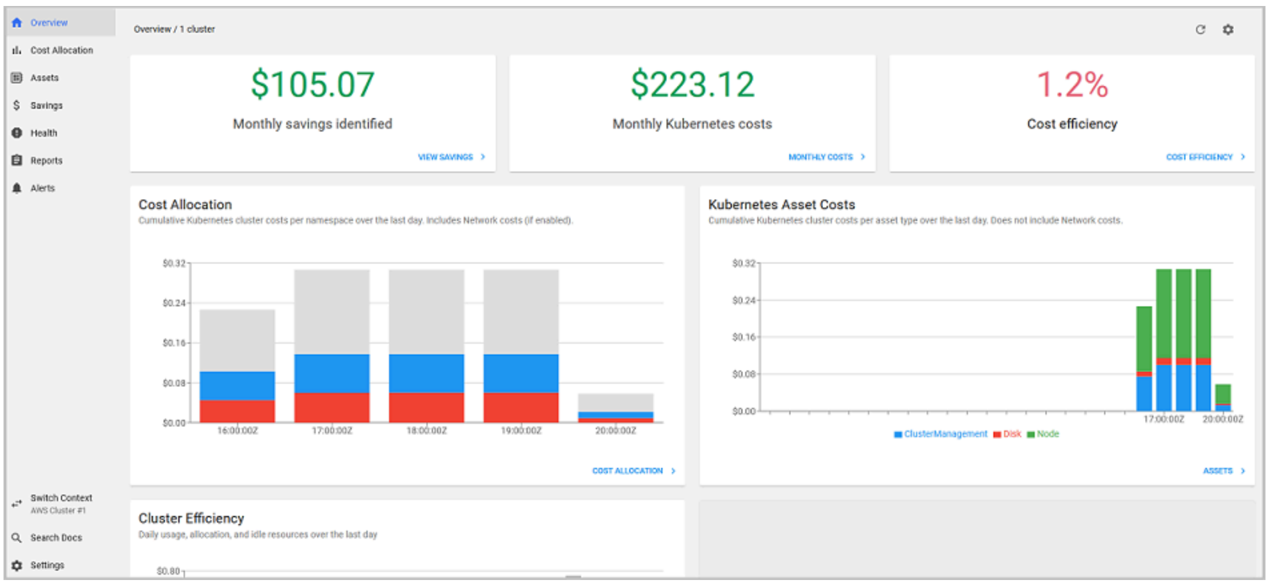
```
kubectl port-forward --namespace kubecost deployment/kubecost-cost-analyzer 9090
```

Alternatively, you can use the AWS Load Balancer Controller to expose Kubecost and use Amazon Cognito for authentication, authorization, and user management. For more information, see How to use Application Load Balancer and Amazon Cognito to authenticate users for your Kubernetes web apps.

3. On the same device that you completed the previous step on, open a web browser and enter the following address.

```
http://localhost:9090
```

You see the Kubecost Overview page in your browser. It might take 5–10 minutes for Kubecost to gather metrics. You can see your Amazon EKS spend, including cumulative cluster costs, associated Kubernetes asset costs, and monthly aggregated spend.



4. To track costs at a cluster level, tag your Amazon EKS resources for billing. For more information, see the section called "Tagging your resources for billing".

   - **Cost allocation** – View monthly Amazon EKS costs and cumulative costs for each of your namespaces and other dimensions over the past seven days. This is helpful for understanding which parts of your application are contributing to Amazon EKS spend.
   - **Assets** – View the costs of the AWS infrastructure assets that are associated with your Amazon EKS resources.

Learn more about Kubecost

Amazon EKS provides an AWS optimized bundle of Kubecost for cluster cost visibility. Amazon EKS supports Kubecost, which you can use to monitor your costs broken down by Kubernetes resources including Pods, nodes, namespaces, and labels.

This topic covers the available versions of Kubecost, and the differences between the available tiers. EKS supports Kubecost Version 1 and Version 2. Each version is available in different tiers. You can use *Amazon EKS optimized Kubecost custom bundle* for your EKS clusters at no additional cost. You may be charged for use of associated AWS services, such as Amazon Managed Service for Prometheus. Also, you can use your existing AWS support agreements to obtain support.

As a Kubernetes platform administrator and finance leader, you can use Kubecost to visualize a breakdown of Amazon EKS charges, allocate costs, and charge back organizational units such as application teams. You can provide your internal teams and business units with transparent and accurate cost data based on their actual AWS bill. Moreover, you can also get customized recommendations for cost optimization based on their infrastructure environment and usage patterns within their clusters. For more information about Kubecost, see the Kubecost documentation.

What is the difference between the custom bundle of Kubecost and the free version of Kubecost (also known as OpenCost)?

AWS and Kubecost collaborated to offer a customized version of Kubecost. This version includes a subset of commercial features at no additional charge. See the tables below for features that are included with in the custom bundle of Kubecost.

Kubecost v2

What is the difference between Kubecost v1 and v2?

Kubecost 2.0 is a major upgrade from previous versions and includes major new features including a brand new API Backend. Note the Allocation and Assets APIs are fully backwards compatible. Please review the Kubecost documentation to ensure a smooth transition. For the full list of enhancements, please see the Kubecost release notes

> **Important**
>
> Review the Kubecost documentation before upgrading. Upgrading may impact report availability.

Core features comparison:

Feature	Kubecost free tier 2.0	Amazon EKS optimized Kubecost bundle 2.0	Kubecost Enterprise 2.0
Cluster cost visibility	Single clusters up to 250 cores	Unified multi-clu ster without core limits when integrate d with Amazon Managed Service for Prometheus	Unified and unlimited number of clusters across unlimited numbers of environments (i.e. multi-cloud)
Deployment	User hosted	User hosted	User hosted, Kubecost hosted (dedicated tenant), SaaS
Databases supported	Local Prometheus	Amazon Managed Service for Prometheus or Local Prometheus	Any prometheus flavor and custom databases
Database retention support (raw metrics)	15 days	Unlimited historical data	Unlimited historical data
Kubecost API and UI retention (ETL)	15 days	15 days	Unlimited
Hybrid cloud visibility	-	Amazon EKS and Amazon EKS Anywhere clusters	Multi-cloud and hybrid cloud
Alerts and recurring reports	Only supported on the primary cluster, limited to 250 cores	Efficiency alerts, budget alerts, spend change alerts, and more supported across all clusters	Efficiency alerts, budget alerts, spend change alerts, and more supported across all clusters

Feature	Kubecost free tier 2.0	Amazon EKS optimized Kubecost bundle 2.0	Kubecost Enterprise 2.0
Saved reports	-	Reports using 15 days of metrics	Reports using unlimited historical data and metrics
Cloud billing integration	Only supported on the primary cluster, limited to 250 cores	Custom pricing support for AWS (including multiple clusters and multiple accounts)	Custom pricing support for any cloud
Savings recommendations	Only supported on the primary cluster, limited to 250 cores	Primary cluster insights, but there is no 250 core limit	Multi-cluster insights
Governance: Audits	-	-	Audit historical cost events
Single sign-on (SSO) support	-	Amazon Cognito supported	Okta, Auth0, PingID, KeyCloak, and anything else custom
Role-based access control (RBAC) with SAML 2.0	-	-	Okta, Auth0, PingID, KeyCloak, and anything else custom
Enterprise training and onboarding	-	-	Full-service training and FinOps onboarding
Teams	-	-	Yes

New Features:

The following features have metric limits:

- Kubecost Aggregator
- Network Monitoring
- Kubecost Actions
- Collections
- Anomaly detection
- Container Request Right-Sizing
- Kubecost Forecasting
- Autocomplete for filtering and aggregation

Metric limits:

Metric	Kubecost Free Tier 2.0	Amazon EKS Optimized Kubecost Custom Bundle 2.0	Kubecost Enterprise 2.0
Cluster size	Limited to 250 cores	Unlimited	Unlimited
Metric retention	15 days	15 days	Unlimited
Multi-cluster support	Not available	Available	Available
Core limits	250 cores per cluster	No core limits	No core limits

Kubecost v1

Feature	Kubecost free tier	Amazon EKS optimized Kubecost custom bundle	Kubecost Enterprise
Deployment	User hosted	User hosted	User hosted or Kubecost hosted (SaaS)
Number of clusters supported	Unlimited	Unlimited	Unlimited

Feature	Kubecost free tier	Amazon EKS optimized Kubecost custom bundle	Kubecost Enterprise
Databases supported	Local Prometheus	Local Prometheus or Amazon Managed Service for Prometheus	Prometheus, Amazon Managed Service for Prometheus, Cortex, or Thanos
Database retention support	15 days	Unlimited historical data	Unlimited historical data
Kubecost API retention (ETL)	15 days	15 days	Unlimited historical data
Cluster cost visibility	Single clusters	Unified multi-cluster	Unified multi-cluster
Hybrid cloud visibility	-	Amazon EKS and Amazon EKS Anywhere clusters	Multi-cloud and hybrid-cloud support
Alerts and recurring reports	-	Efficiency alerts, budget alerts, spend change alerts, and more supported	Efficiency alerts, budget alerts, spend change alerts, and more supported
Saved reports	-	Reports using 15 days data	Reports using unlimited historical data
Cloud billing integration	Required for each individual cluster	Custom pricing support for AWS (including multiple clusters and multiple accounts)	Custom pricing support for AWS (including multiple clusters and multiple accounts)
Savings recommendations	Single cluster insights	Single cluster insights	Multi-cluster insights

Feature	Kubecost free tier	Amazon EKS optimized Kubecost custom bundle	Kubecost Enterprise
Governance: Audits	-	-	Audit historical cost events
Single sign-on (SSO) support	-	Amazon Cognito supported	Okta, Auth0, PingID, KeyCloak
Role-based access control (RBAC) with SAML `2.0`	-	-	Okta, Auth0, PingID, Keycloak
Enterprise training and onboarding	-	-	Full-service training and FinOps onboarding

Frequently asked questions

See the following common questions and answers about using Kubecost with Amazon EKS.

What is the Kubecost API retention (ETL) feature?

The Kubecost ETL feature aggregates and organizes metrics to surface cost visibility at various levels of granularity (such as `namespace-level`, `pod-level`, and `deployment-level`). For the custom Kubecost bundle, customers get data and insights from metrics for the last 15 days.

What is the alerts and recurring reports feature? What alerts and reports does it include?

Kubecost alerts allow teams to receive updates on real-time Kubernetes spend as well as cloud spend. Recurring reports enable teams to receive customized views of historical Kubernetes and cloud spend. Both are configurable using the Kubecost UI or Helm values. They support email, Slack, and Microsoft Teams.

What do saved reports include?

Kubecost saved reports are predefined views of cost and efficiency metrics. They include cost by cluster, namespace, label, and more.

What is cloud billing integration?

Integration with AWS billing APIs allows Kubecost to display out-of-cluster costs (such as Amazon S3). Additionally, it allows Kubecost to reconcile Kubecost's in-cluster predictions with actual billing data to account for spot usage, savings plans, and enterprise discounts.

What do savings recommendations include?

Kubecost provides insights and automation to help users optimize their Kubernetes infrastructure and spend.

Is there a charge for this functionality?

No. You can use this version of Kubecost at no additional charge. If you want additional Kubecost capabilities that aren't included in this bundle, you can buy an enterprise license of Kubecost through the AWS Marketplace, or from Kubecost directly.

Is support available?

Yes. You can open a support case with the AWS Support team at Contact AWS.

Do I need a license to use Kubecost features provided by the Amazon EKS integration?

No.

Can I integrate Kubecost with AWS Cost and Usage Report for more accurate reporting?

Yes. You can configure Kubecost to ingest data from AWS Cost and Usage Report to get accurate cost visibility, including discounts, Spot pricing, reserved instance pricing, and others. For more information, see AWS Cloud Billing Integration in the Kubecost documentation.

Does this version support cost management of self-managed Kubernetes clusters on Amazon EC2?

No. This version is only compatible with Amazon EKS clusters.

Can Kubecost track costs for Amazon EKS on AWS Fargate?

Kubecost provides best effort to show cluster cost visibility for Amazon EKS on Fargate, but with lower accuracy than with Amazon EKS on Amazon EC2. This is primarily due to the difference in how you're billed for your usage. With Amazon EKS on Fargate, you're billed for consumed resources. With Amazon EKS on Amazon EC2 nodes, you're billed for provisioned resources. Kubecost calculates the cost of an Amazon EC2 node based on the node specification, which includes CPU, RAM, and ephemeral storage. With Fargate, costs are calculated based on the requested resources for the Fargate Pods.

How can I get updates and new versions of Kubecost?

You can upgrade your Kubecost version using standard Helm upgrade procedures. The latest versions are in the Amazon ECR Public Gallery.

Is the `kubectl-cost` CLI supported? How do I install it?

Yes. `Kubectl-cost` is an open source tool by Kubecost (Apache 2.0 License) that provides CLI access to Kubernetes cost allocation metrics. To install `kubectl-cost`, see Installation on GitHub.

Is the Kubecost user interface supported? How do I access it?

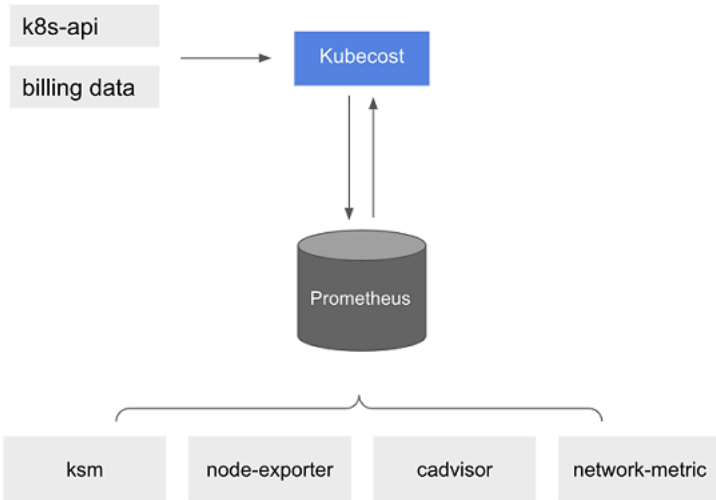
Kubecost provides a web dashboard that you can access through `kubectl` port forwarding, an ingress, or a load balancer. You can also use the AWS Load Balancer Controller to expose Kubecost and use Amazon Cognito for authentication, authorization, and user management. For more information, see How to use Application Load Balancer and Amazon Cognito to authenticate users for your Kubernetes web apps on the AWS blog.

Is Amazon EKS Anywhere supported?

No.

Additional Kubecost Features

- The following features are available in both Kubecost v1 and v2.
- **Export cost metrics** – Amazon EKS optimized cost monitoring is deployed with Kubecost and Prometheus, which is an open-source monitoring system and time series database. Kubecost reads metric from Prometheus and then performs cost allocation calculations and writes the metrics back to Prometheus. The Kubecost front-end reads metrics from Prometheus and shows them on the Kubecost user interface. The architecture is illustrated in the following diagram.



With Prometheus pre-installed, you can write queries to ingest Kubecost data into your current business intelligence system for further analysis. You can also use it as a data source for your current Grafana dashboard to display Amazon EKS cluster costs that your internal teams are familiar with. To learn more about how to write Prometheus queries, see the https://github.com/opencost/opencost/blob/develop/PROMETHEUS.md `readme` file on GitHub or use the example Grafana JSON models in the Kubecost Github repository as references.

- **AWS Cost and Usage Report integration** – To perform cost allocation calculations for your Amazon EKS cluster, Kubecost retrieves the public pricing information of AWS services and AWS resources from the AWS Price List API. You can also integrate Kubecost with **AWS Cost and Usage Report** to enhance the accuracy of the pricing information specific to your AWS account. This information includes enterprise discount programs, reserved instance usage, savings plans, and spot usage. To learn more about how the AWS Cost and Usage Report integration works, see AWS Cloud Billing Integration in the Kubecost documentation.

View resource usage with the Kubernetes Metrics Server

The Kubernetes Metrics Server is an aggregator of resource usage data in your cluster, and it isn't deployed by default in Amazon EKS clusters. For more information, see Kubernetes Metrics Server on GitHub. The Metrics Server is commonly used by other Kubernetes add ons, such as the Scale pod deployments with Horizontal Pod Autoscaler or the Kubernetes Dashboard. For more information, see Resource metrics pipeline in the Kubernetes documentation. This topic explains how to deploy the Kubernetes Metrics Server on your Amazon EKS cluster.

> **Important**
>
> The metrics are meant for point-in-time analysis and aren't an accurate source for historical analysis. They can't be used as a monitoring solution or for other non-auto scaling purposes. For information about monitoring tools, see *Monitor clusters*.

Deploy as community add-on with Amazon EKS Add-ons

New: You can now deploy Metrics Server as a community add-on using the AWS console or Amazon EKS APIs.

Deploy with AWS console

1. Open your EKS cluster in the AWS console
2. From the "Add-ons" tab, select **Get More Add-ons**.
3. From the "Community add-ons" section, select **Metrics Server** and then **Next**
4. EKS determines the appropriate version of the add-on for your cluster. You can change the version using the **Version** dropdown menu.
5. Select **Next** and then **Create** to install the add-on.

Additional resources

Learn more about the section called "Community add-ons".

You install or update community add-ons in the same way as other Amazon EKS Add-ons.

- the section called "Create an add-on"
- the section called "Update an add-on"
- the section called "Remove an add-on"

Deploy with manifest

New: You can now deploy Metrics Server as a community add-on using the AWS console or Amazon EKS APIs. These manifest install instructions will be archived.

1. Deploy the Metrics Server with the following command:

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/
download/components.yaml
```

If you are using Fargate, you will need to change this file. In the default configuration, the metrics server uses port 10250. This port is reserved on Fargate. Replace references to port 10250 in components.yaml with another port, such as 10251.

2. Verify that the `metrics-server` deployment is running the desired number of Pods with the following command.

```
kubectl get deployment metrics-server -n kube-system
```

An example output is as follows.

```
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
metrics-server   1/1     1            1           6m
```

3. Test the metrics server is working by displaying resource (CPU/memory) usage of nodes.

```
kubectl top nodes
```

4. If you receive the error message `Error from server (Forbidden)`, you need to update your Kubernetes RBAC configuration. Your Kubernetes RBAC identity needs sufficent permissions to read cluster metrics. Review the minimum required Kubernetes API permissions for reading metrics on GitHub. Learn how to grant AWS IAM Identities such as Roles access to Kubernetes APIs.

Deploy applications with Helm on Amazon EKS

The Helm package manager for Kubernetes helps you install and manage applications on your Kubernetes cluster. For more information, see the Helm documentation. This topic helps you install and run the Helm binaries so that you can install and manage charts using the Helm CLI on your local system.

Important

Before you can install Helm charts on your Amazon EKS cluster, you must configure `kubectl` to work for Amazon EKS. If you have not already done this, see the section called

"Access cluster with kubectl" before proceeding. If the following command succeeds for your cluster, you're properly configured.

```
kubectl get svc
```

1. Run the appropriate command for your client operating system.
   - If you're using macOS with Homebrew, install the binaries with the following command.

   ```
   brew install helm
   ```

   - If you're using Windows with Chocolatey, install the binaries with the following command.

   ```
   choco install kubernetes-helm
   ```

   - If you're using Linux, install the binaries with the following commands.

   ```
   curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 >
    get_helm.sh
   chmod 700 get_helm.sh
   ./get_helm.sh
   ```

   Note

   If you get a message that `openssl` must first be installed, you can install it with the following command.

```
sudo yum install openssl
```

1. To pick up the new binary in your `PATH`, Close your current terminal window and open a new one.
2. See the version of Helm that you installed.

   ```
   helm version | cut -d + -f 1
   ```

   An example output is as follows.

```
v3.9.0
```

3. At this point, you can run any Helm commands (such as `helm install chart-name`) to install, modify, delete, or query Helm charts in your cluster. If you're new to Helm and don't have a specific chart to install, you can:

   - Experiment by installing an example chart. See Install an example chart in the Helm Quickstart guide.
   - Create an example chart and push it to Amazon ECR. For more information, see Pushing a Helm chart in the *Amazon Elastic Container Registry User Guide*.
   - Install an Amazon EKS chart from the eks-chartsGitHub repo or from ArtifactHub.

Organize Amazon EKS resources with tags

You can use *tags* to help you manage your Amazon EKS resources. This topic provides an overview of the tags function and shows how you can create tags.

Topics

> **Note**
>
> Tags are a type of metadata that's separate from Kubernetes labels and annotations. For more information about these other metadata types, see the following sections in the Kubernetes documentation:
>
> - Labels and Selectors
> - Annotations

Tag basics

A tag is a label that you assign to an AWS resource. Each tag consists of a *key* and an optional *value*.

With tags, you can categorize your AWS resources. For example, you can categorize resources by purpose, owner, or environment. When you have many resources of the same type, you can use the tags that you assigned to a specific resource to quickly identify that resource. For example, you can define a set of tags for your Amazon EKS clusters to help you track each cluster's owner and stack level. We recommend that you devise a consistent set of tag keys for each resource type. You can then search and filter the resources based on the tags that you add.

After you add a tag, you can edit tag keys and values or remove tags from a resource at any time. If you delete a resource, any tags for the resource are also deleted.

Tags don't have any semantic meaning to Amazon EKS and are interpreted strictly as a string of characters. You can set the value of a tag to an empty string. However, you can't set the value of a tag to null. If you add a tag that has the same key as an existing tag on that resource, the new value overwrites the earlier value.

If you use AWS Identity and Access Management (IAM), you can control which users in your AWS account have permission to manage tags.

Tagging your resources

The following Amazon EKS resources support tags:

- clusters
- managed node groups
- Fargate profiles

You can tag these resources using the following:

- If you're using the Amazon EKS console, you can apply tags to new or existing resources at any time. You can do this by using the **Tags** tab on the relevant resource page. For more information, see the section called "Working with tags using the console".
- If you're using `eksctl`, you can apply tags to resources when they're created using the `--tags` option.

- If you're using the AWS CLI, the Amazon EKS API, or an AWS SDK, you can apply tags to new resources using the `tags` parameter on the relevant API action. You can apply tags to existing resources using the `TagResource` API action. For more information, see TagResource.

When you use some resource-creating actions, you can also specify tags for the resource at the same time that you create it. If tags can't be applied while the resource is being created, the resource fails to be created. This mechanism ensures that resources that you intend to tag are either created with the tags that you specify or not created at all. If you tag resources when you create them, you don't need to run custom tagging scripts after you create the resource.

Tags don't propagate to other resources that are associated with the resource that you create. For example, Fargate profile tags don't propagate to other resources that are associated with the Fargate profile, such as the Pods that are scheduled with it.

Tag restrictions

The following restrictions apply to tags:

- A maximum of 50 tags can be associated with a resource.
- Tag keys can't be repeated for one resource. Each tag key must be unique, and can only have one value.
- Keys can be up to 128 characters long in UTF-8.
- Values can be up to 256 characters long in UTF-8.
- If multiple AWS services and resources use your tagging schema, limit the types of characters you use. Some services might have restrictions on allowed characters. Generally, allowed characters are letters, numbers, spaces, and the following characters: `+ - = . _ : / @`.
- Tag keys and values are case sensitive.
- Don't use `aws:`, `AWS:`, or any upper or lowercase combination of such as a prefix for either keys or values. These are reserved only for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix don't count against your tags-per-resource limit.

Tagging your resources for billing

When you apply tags to Amazon EKS clusters, you can use them for cost allocation in your **Cost & Usage Reports**. The metering data in your **Cost & Usage Reports** shows usage across all of your

Amazon EKS clusters. For more information, see AWS cost and usage report in the *AWS Billing User Guide*.

The AWS generated cost allocation tag, specifically `aws:eks:cluster-name`, lets you break down Amazon EC2 instance costs by individual Amazon EKS cluster in **Cost Explorer**. However, this tag doesn't capture the control plane expenses. The tag is automatically added to Amazon EC2 instances that participate in an Amazon EKS cluster. This behavior happens regardless of whether the instances are provisioned using Amazon EKS managed node groups, Karpenter, or directly with Amazon EC2. This specific tag doesn't count towards the 50 tags limit. To use the tag, the account owner must activate it in the AWS Billing console or by using the API. When an AWS Organizations management account owner activates the tag, it's also activated for all organization member accounts.

You can also organize your billing information based on resources that have the same tag key values. For example, you can tag several resources with a specific application name, and then organize your billing information. That way, you can see the total cost of that application across several services. For more information about setting up a cost allocation report with tags, see The Monthly Cost Allocation Report in the *AWS Billing User Guide*.

> **Note**
>
> If you just enabled reporting, data for the current month is available for viewing after 24 hours.

Cost Explorer is a reporting tool that's available as part of the AWS Free Tier. You can use **Cost Explorer** to view charts of your Amazon EKS resources from the last 13 months. You can also forecast how much you're likely to spend for the next three months. You can see patterns in how much you spend on AWS resources over time. For example, you can use it to identify areas that need further inquiry and see trends that you can use to understand your costs. You also can specify time ranges for the data, and view time data by day or by month.

Working with tags using the console

Using the Amazon EKS console, you can manage the tags that are associated with new or existing clusters and managed node groups.

When you select a resource-specific page in the Amazon EKS console, the page displays a list of those resources. For example, if you select **Clusters** from the left navigation pane, the console displays a list of Amazon EKS clusters. When you select a resource from one of these lists (for example, a specific cluster) that supports tags, you can view and manage its tags on the **Tags** tab.

You can also use **Tag Editor** in the AWS Management Console, which provides a unified way to manage your tags. For more information, see Tagging your AWS resources with Tag Editor in the *AWS Tag Editor User Guide*.

Adding tags on a resource on creation

You can add tags to Amazon EKS clusters, managed node groups, and Fargate profiles when you create them. For more information, see the section called “Create a cluster”.

Adding and deleting tags on a resource

You can add or delete the tags that are associated with your clusters directly from the resource’s page.

1. Open the Amazon EKS console.
2. On the navigation bar, select the AWS Region to use.
3. In the left navigation pane, choose **Clusters**.
4. Choose a specific cluster.
5. Choose the **Tags** tab, and then choose **Manage tags**.
6. On the **Manage tags** page, add or delete your tags as necessary.
   - To add a tag, choose **Add tag**. Then specify the key and value for each tag.
   - To delete a tag, choose **Remove tag**.
7. Repeat this process for each tag that you want to add or delete.
8. Choose **Update** to finish.

Working with tags using the CLI, API, or `eksctl`

Use the following AWS CLI commands or Amazon EKS API operations to add, update, list, and delete the tags for your resources. You can only use `eksctl` to add tags while simultaneously creating the new resources with one command.

Task	AWS CLI	AWS Tools for Windows PowerShell	API action
Add or overwrite one or more tags.	tag-resource	Add-EKSResourceTag	TagResource
Delete one or more tags.	untag-resource	Remove-EKSResource Tag	UntagResource

The following examples show how to tag or untag resources using the AWS CLI.

Example 1: Tag an existing cluster

The following command tags an existing cluster.

```
aws eks tag-resource --resource-arn resource_ARN --tags team=devs
```

Example 2: Untag an existing cluster

The following command deletes a tag from an existing cluster.

```
aws eks untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

Example 3: List tags for a resource

The following command lists the tags that are associated with an existing resource.

```
aws eks list-tags-for-resource --resource-arn resource_ARN
```

When you use some resource-creating actions, you can specify tags at the same time that you create the resource. The following actions support specifying a tag when you create a resource.

Task	AWS CLI	AWS Tools for Windows PowerShell	API action	eksctl
Create a cluster	create-cluster	New-EKSCluster	CreateCluster	`create cluster`

Task	AWS CLI	AWS Tools for Windows PowerShell	API action	eksctl
Create a managed node group*	create-no degroup	New-EKSNo degroup	CreateNod egroup	`create nodegroup`
Create a Fargate profile	create-fargate-profile	New-EKSFa rgateProfile	CreateFar gateProfile.html	`create fargatepr ofile`

- If you want to also tag the Amazon EC2 instances when you create a managed node group, create the managed node group using a launch template. For more information, see the section called "Tagging Amazon EC2 instances". If your instances already exist, you can manually tag the instances. For more information, see Tagging your resources in the Amazon EC2 User Guide.

View and manage Amazon EKS and Fargate service quotas

Amazon EKS has integrated with Service Quotas, an AWS service that you can use to view and manage your quotas from a central location. For more information, see What Is Service Quotas? in the *Service Quotas User Guide*. With Service Quotas integration, you can quickly look up the value of your Amazon EKS and AWS Fargate service quotas using the AWS Management Console and AWS CLI.

View EKS service quotas in the AWS Management Console

1. Open the Service Quotas console.
2. In the left navigation pane, choose **AWS services**.
3. From the **AWS services** list, search for and select **Amazon Elastic Kubernetes Service (Amazon EKS)** or **AWS Fargate**.

   In the **Service quotas** list, you can see the service quota name, applied value (if it's available), AWS default quota, and whether the quota value is adjustable.
4. To view additional information about a service quota, such as the description, choose the quota name.

5. (Optional) To request a quota increase, select the quota that you want to increase, select **Request quota increase**, enter or select the required information, and select **Request**.

To work more with service quotas using the AWS Management Console, see the Service Quotas User Guide. To request a quota increase, see Requesting a Quota Increase in the *Service Quotas User Guide*.

View EKS service quotas with the AWS CLI

Run the following command to view your Amazon EKS quotas.

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code eks \
    --output table
```

Run the following command to view your Fargate quotas.

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code fargate \
    --output table
```

Note

The quota returned is the number of Amazon ECS tasks or Amazon EKS Pods that can run concurrently on Fargate in this account in the current AWS Region.

To work more with service quotas using the AWS CLI, see service-quotas in the *AWS CLI Command Reference*. To request a quota increase, see the request-service-quota-increase command in the *AWS CLI Command Reference*.

Amazon EKS service quotas

AWS recommends using the AWS Management Console to view your current quotas. For more information, see the section called “View EKS service quotas in the AWS Management Console”.

To view the default EKS service quotas, see Amazon Elastic Kubernetes Service endpoints and quotas in the *AWS General Reference*.

These service quotas are listed under **Amazon Elastic Kubernetes Service (Amazon EKS)** in the Service Quotas console. To request a quota increase for values that are shown as adjustable, see Requesting a quota increase in the *Service Quotas User Guide*.

> **Note**
>
> The following quotas aren't available in Service Quotas:
>
> - Pod Identity associations per cluster is 1000 in each supported region and this quota isn't adjustable.
> - You can use up to 15 CIDRs for Remote Node Networks and 15 CIDRs for Remote Pod Networks per cluster for hybrid nodes. This quota isn't adjustable.

AWS Fargate service quotas

The **AWS Fargate** service in the Service Quotas console lists several service quotas. You can configure alarms that alert you when your usage approaches a service quota. For more information, see the section called "Creating a CloudWatch alarm to monitor Fargate resource usage metrics".

New AWS accounts might have lower initial quotas that can increase over time. Fargate constantly monitors the account usage within each AWS Region, and then automatically increases the quotas based on the usage. You can also request a quota increase for values that are shown as adjustable. For more information, see Requesting a quota increase in the *Service Quotas User Guide*.

AWS recomends using the AWS Management Console to view your current quotas. For more information, see the section called "View EKS service quotas in the AWS Management Console".

To view default AWS Fargate on EKS service quotas, see Fargate service quotas in the *AWS General Reference*.

> **Note**
>
> Fargate additionally enforces Amazon ECS tasks and Amazon EKS Pods launch rate quotas. For more information, see AWS Fargate throttling quotas in the *Amazon ECS guide*.

Security in Amazon EKS

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. For Amazon EKS, AWS is responsible for the Kubernetes control plane, which includes the control plane nodes and `etcd` database. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon EKS, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility includes the following areas.
  - The security configuration of the data plane, including the configuration of the security groups that allow traffic to pass from the Amazon EKS control plane into the customer VPC
  - The configuration of the nodes and the containers themselves
  - The node's operating system (including updates and security patches)
  - Other associated application software:
    - Setting up and managing network controls, such as firewall rules
    - Managing platform-level identity and access management, either with or in addition to IAM
  - The sensitivity of your data, your company's requirements, and applicable laws and regulations

This documentation helps you understand how to apply the shared responsibility model when using Amazon EKS. The following topics show you how to configure Amazon EKS to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon EKS resources.

> **Note**
>
> Linux containers are made up of control groups (cgroups) and namespaces that help limit what a container can access, but all containers share the same Linux kernel as the host Amazon EC2 instance. Running a container as the root user (UID 0) or granting a container

access to host resources or namespaces such as the host network or host PID namespace are strongly discouraged, because doing so reduces the effectiveness of the isolation that containers provide.

Topics

Secure Amazon EKS clusters with best practices

The Amazon EKS security best practices are in the Best Practices for Security in the *Amazon EKS Best Practices Guide*.

Analyze vulnerabilities in Amazon EKS

Security is a critical consideration for configuring and maintaining Kubernetes clusters and applications. The following lists resources for you to analyze the security configuration of your EKS clusters, resources for you to check for vulnerabilities, and integrations with AWS services that can do that analysis for you.

The Center for Internet Security (CIS) benchmark for Amazon EKS

The Center for Internet Security (CIS) Kubernetes Benchmark provides guidance for Amazon EKS security configurations. The benchmark:

- Is applicable to Amazon EC2 nodes (both managed and self-managed) where you are responsible for security configurations of Kubernetes components.
- Provides a standard, community-approved way to ensure that you have configured your Kubernetes cluster and nodes securely when using Amazon EKS.

- Consists of four sections; control plane logging configuration, node security configurations, policies, and managed services.
- Supports all of the Kubernetes versions currently available in Amazon EKS and can be run using kube-bench, a standard open source tool for checking configuration using the CIS benchmark on Kubernetes clusters.

To learn more, see Introducing The CIS Amazon EKS Benchmark.

For an automated `aws-sample` pipeline to update your node group with a CIS benchmarked AMI, see EKS-Optimized AMI Hardening Pipeline.

Amazon EKS platform versions

Amazon EKS *platform versions* represent the capabilities of the cluster control plane, including which Kubernetes API server flags are enabled and the current Kubernetes patch version. New clusters are deployed with the latest platform version. For details, see the section called "Platform versions".

You can update an Amazon EKS cluster to newer Kubernetes versions. As new Kubernetes versions become available in Amazon EKS, we recommend that you proactively update your clusters to use the latest available version. For more information about Kubernetes versions in EKS, see the section called "Kubernetes versions".

Operating system vulnerability list

AL2023 vulnerability list

Track security or privacy events for Amazon Linux 2023 at the Amazon Linux Security Center or subscribe to the associated RSS feed. Security and privacy events include an overview of the issue affected, packages, and instructions for updating your instances to correct the issue.

Amazon Linux 2 vulnerability list

Track security or privacy events for Amazon Linux 2 at the Amazon Linux Security Center or subscribe to the associated RSS feed. Security and privacy events include an overview of the issue affected, packages, and instructions for updating your instances to correct the issue.

Node detection with Amazon Inspector

You can use Amazon Inspector to check for unintended network accessibility of your nodes and for vulnerabilities on those Amazon EC2 instances.

Cluster and node detection with Amazon GuardDuty

Amazon GuardDuty threat detection service that helps protect your accounts, containers, workloads, and the data within your AWS environment. Among other features, GuardDuty offers the following two features that detect potential threats to your EKS clusters: *EKS Protection* and *Runtime Monitoring*.

For more information, see the section called "Amazon GuardDuty".

Compliance validation for Amazon EKS clusters

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

> **Note**
>
> Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Security considerations for Amazon Elastic Kubernetes Service

The following are considerations for security of the cloud, as they affect Amazon EKS.

Topics

Infrastructure security in Amazon EKS

As a managed service, Amazon Elastic Kubernetes Service is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well‐Architected Framework*.

You use AWS published API calls to access Amazon EKS through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

When you create an Amazon EKS cluster, you specify the VPC subnets for your cluster to use. Amazon EKS requires subnets in at least two Availability Zones. We recommend a VPC with public and private subnets so that Kubernetes can create public load balancers in the public subnets that load balance traffic to Pods running on nodes that are in private subnets.

For more information about VPC considerations, see the section called “VPC and subnet requirements”.

If you create your VPC and node groups with the AWS CloudFormation templates provided in the Get started with Amazon EKS walkthrough, then your control plane and node security groups are configured with our recommended settings.

For more information about security group considerations, see the section called “Security group requirements”.

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

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

For more information about modifying cluster endpoint access, see the section called “Modifying cluster endpoint access”.

You can implement Kubernetes *network policies* with the Amazon VPC CNI or third-party tools such as Project Calico. For more information about using the Amazon VPC CNI for network policies, see the section called “Kubernetes policies”. Project Calico is a third party open source project. For more information, see the Project Calico documentation.

Access Amazon EKS using AWS PrivateLink

You can use AWS PrivateLink to create a private connection between your VPC and Amazon Elastic Kubernetes Service. You can access Amazon EKS as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don’t need public IP addresses to access Amazon EKS.

You establish this private connection by creating an interface endpoint powered by AWS PrivateLink. We create an endpoint network interface in each subnet that you enable for the interface endpoint. These are requester-managed network interfaces that serve as the entry point for traffic destined for Amazon EKS.

For more information, see Access AWS services through AWS PrivateLink in the *AWS PrivateLink Guide*.

Considerations for Amazon EKS

- Before you set up an interface endpoint for Amazon EKS, review Considerations in the *AWS PrivateLink Guide*.
- Amazon EKS supports making calls to all of its API actions through the interface endpoint, but not to the Kubernetes APIs. The Kubernetes API server already supports a private endpoint. The Kubernetes API server private endpoint creates a private endpoint for the Kubernetes API server that you use to communicate with your cluster (using Kubernetes management tools such as `kubectl`). You can enable private access to the Kubernetes API server so that all communication between your nodes and the API server stays within your VPC. AWS PrivateLink for the Amazon EKS API helps you call the Amazon EKS APIs from your VPC without exposing traffic to the public internet.
- You can’t configure Amazon EKS to only be accessed through an interface endpoint.
- Standard pricing for AWS PrivateLink applies for interface endpoints for Amazon EKS. You are billed for every hour that an interface endpoint is provisioned in each Availability Zone and for data processed through the interface endpoint. For more information, see AWS PrivateLink pricing.

- VPC endpoint policies are supported for Amazon EKS. You can use these policies to control access to Amazon EKS through the interface endpoint. Additionally, you can associate a security group with the endpoint network interfaces to control traffic to Amazon EKS through the interface endpoint. For more information, see Control access to VPC endpoints using endpoint policies in the Amazon VPC docs.
- You can use VPC flow logs to capture information about IP traffic going to and from network interfaces, including interface endpoints. You can publish flow log data to Amazon CloudWatch or Amazon S3. For more information, see Logging IP traffic using VPC Flow Logs in the Amazon VPC User Guide.
- You can access the Amazon EKS APIs from an on-premises data center by connecting it to a VPC that has an interface endpoint. You can use AWS Direct Connect or AWS Site-to-Site VPN to connect your on-premises sites to a VPC.
- You can connect other VPCs to the VPC with an interface endpoint using an AWS Transit Gateway or VPC peering. VPC peering is a networking connection between two VPCs. You can establish a VPC peering connection between your VPCs, or with a VPC in another account. The VPCs can be in different AWS Regions. Traffic between peered VPCs stays on the AWS network. The traffic doesn't traverse the public internet. A Transit Gateway is a network transit hub that you can use to interconnect VPCs. Traffic between a VPC and a Transit Gateway remains on the AWS global private network. The traffic isn't exposed to the public internet.
- Before August 2024, VPC interface endpoints for Amazon EKS were only accessible over `IPv4` using `eks.`*`region`*`.amazonaws.com`. New VPC interface endpoints that are made after August 2024 use dual-stack of `IPv4` and `IPv6` IP addresses and both DNS names: `eks.`*`region`*`.amazonaws.com` and `eks.`*`region`*`.api.aws`.
- AWS PrivateLink support for the EKS API isn't available in the Asia Pacific (Malaysia) (`ap-southeast-5`), Asia Pacific (Thailand) (`ap-southeast-7`), and Mexico (Central) (`mx-central-1`) AWS Regions. AWS PrivateLink support for `eks-auth` for EKS Pod Identity is available in the the Asia Pacific (Malaysia) (`ap-southeast-5`) Region.

Create an interface endpoint for Amazon EKS

You can create an interface endpoint for Amazon EKS using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Create a VPC endpoint in the *AWS PrivateLink Guide*.

Create an interface endpoint for Amazon EKS using the following service names:

- EKS API

```
com.amazonaws.region-code.eks
```

- EKS Auth API (EKS Pod Identity)

```
com.amazonaws.region-code.eks-auth
```

The private DNS feature is enabled by default when creating an interface endpoint for Amazon EKS and other AWS services. To use the private DNS feature, you must ensure that the following VPC attributes are set to `true`: `enableDnsHostnames` and `enableDnsSupport`. For more information, see [View and update DNS attributes for your VPC] in the Amazon VPC User Guide. With the private DNS feature enabled for the interface endpoint:

- You can make any API request to Amazon EKS using its default Regional DNS name. After August 2024, any new VPC interface endpoint for the Amazon EKS API have two default Regional DNS names and you can choose the `dualstack` for the IP address type. The first DNS name is `eks.`*`region`*`.api.aws` which is dual-stack. It resolves to both `IPv4` addresses and `IPv6` addresses. Before August 2024, Amazon EKS only used `eks.`*`region`*`.amazonaws.com` which resolved to `IPv4` addresses only. If you want to use `IPv6` and dual-stack IP addresses with an existing VPC interface endpoint, you can update the endpoint to use the `dualstack` type of IP address, but it will only have the `eks.`*`region`*`.amazonaws.com` DNS name. In this configuration, the existing endpoint updates to point that name to both `IPv4` and `IPv6` IP addresses. For a list of APIs, see Actions in the Amazon EKS API Reference.
- You don't need to make any changes to your applications that call the EKS APIs.

  However, To use the dual-stack endpoints with the AWS CLI, see the Dual-stack and FIPS endpoints configuration in the *AWS SDKs and Tools Reference Guide*.
- Any call made to the Amazon EKS default service endpoint is automatically routed through the interface endpoint over the private AWS network.

Understand resilience in Amazon EKS clusters

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

Amazon EKS runs and scales the Kubernetes control plane across multiple AWS Availability Zones to ensure high availability. Amazon EKS automatically scales control plane instances based on load, detects and replaces unhealthy control plane instances, and automatically patches the control plane. After you initiate a version update, Amazon EKS updates your control plane for you, maintaining high availability of the control plane during the update.

This control plane consists of at least two API server instances and three `etcd` instances that run across three Availability Zones within an AWS Region. Amazon EKS:

- Actively monitors the load on control plane instances and automatically scales them to ensure high performance.
- Automatically detects and replaces unhealthy control plane instances, restarting them across the Availability Zones within the AWS Region as needed.
- Leverages the architecture of AWS Regions in order to maintain high availability. Because of this, Amazon EKS is able to offer an SLA for API server endpoint availability.

For more information about AWS Regions and Availability Zones, see AWS global infrastructure.

Security considerations for Kubernetes

The following are considerations for security in the cloud, as they affect Kubernetes in Amazon EKS clusters. For an in-depth review of security controls and practices in Kubernetes, see Cloud Native Security and Kubernetes in the Kubernetes documentation.

Topics

Secure workloads with Kubernetes certificates

The Kubernetes Certificates API automates X.509 credential provisioning. The API features a command line interface for Kubernetes API clients to request and obtain X.509 certificates from a Certificate Authority (CA). You can use the `CertificateSigningRequest` (CSR) resource to request that a denoted signer sign the certificate. Your requests are either approved or denied before they're signed. Kubernetes supports both built-in signers and custom signers with well-defined behaviors. This way, clients can predict what happens to their CSRs. To learn more about certificate signing, see signing requests.

One of the built-in signers is `kubernetes.io/legacy-unknown`. The `v1beta1` API of CSR resource honored this legacy-unknown signer. However, the stable `v1` API of CSR doesn't allow the `signerName` to be set to `kubernetes.io/legacy-unknown`.

Amazon EKS version `1.21` and earlier allowed the `legacy-unknown` value as the `signerName` in `v1beta1` CSR API. This API enables the Amazon EKS Certificate Authority (CA) to generate certificates. However, in Kubernetes version `1.22`, the `v1beta1` CSR API was replaced by the `v1` CSR API. This API doesn't support the signerName of "legacy-unknown." If you want to use Amazon EKS CA for generating certificates on your clusters, you must use a custom signer. It was introduced in Amazon EKS version `1.22`. To use the CSR `v1` API version and generate a new certificate, you must migrate any existing manifests and API clients. Existing certificates that were created with the existing `v1beta1` API are valid and function until the certificate expires. This includes the following:

- Trust distribution: None. There's no standard trust or distribution for this signer in a Kubernetes cluster.
- Permitted subjects: Any
- Permitted x509 extensions: Honors subjectAltName and key usage extensions and discards other extensions
- Permitted key usages: Must not include usages beyond ["key encipherment", "digital signature", "server auth"]

> **Note**
>
> Client certificate signing is not supported.

- Expiration/certificate lifetime: 1 year (default and maximum)

- CA bit allowed/disallowed: Not allowed

Example CSR generation with signerName

These steps shows how to generate a serving certificate for DNS name `myserver.default.svc` using `signerName: beta.eks.amazonaws.com/app-serving`. Use this as a guide for your own environment.

1. Run the `openssl genrsa -out myserver.key 2048` command to generate an RSA private key.

```
openssl genrsa -out myserver.key 2048
```

2. Run the following command to generate a certificate request.

```
openssl req -new -key myserver.key -out myserver.csr -subj "/CN=myserver.default.svc"
```

3. Generate a `base64` value for the CSR request and store it in a variable for use in a later step.

```
base_64=$(cat myserver.csr | base64 -w 0 | tr -d "
")
```

4. Run the following command to create a file named `mycsr.yaml`. In the following example, `beta.eks.amazonaws.com/app-serving` is the `signerName`.

```
cat >mycsr.yaml <<EOF
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: myserver
spec:
  request: $base_64
  signerName: beta.eks.amazonaws.com/app-serving
  usages:
    - digital signature
    - key encipherment
    - server auth
EOF
```

5. Submit the CSR.

```
kubectl apply -f mycsr.yaml
```

6. Approve the serving certificate.

```
kubectl certificate approve myserver
```

7. Verify that the certificate was issued.

```
kubectl get csr myserver
```

An example output is as follows.

```
NAME       AGE     SIGNERNAME                           REQUESTOR          CONDITION
myserver   3m20s   beta.eks.amazonaws.com/app-serving   kubernetes-admin
 Approved,Issued
```

8. Export the issued certificate.

```
kubectl get csr myserver -o jsonpath='{.status.certificate}'| base64 -d >
 myserver.crt
```

Certificate signing considerations before upgrading your cluster to Kubernetes 1.24

In Kubernetes `1.23` and earlier, `kubelet` serving certificates with unverifiable IP and DNS Subject Alternative Names (SANs) are automatically issued with unverifiable SANs. The SANs are omitted from the provisioned certificate. In `1.24` and later clusters, `kubelet` serving certificates aren't issued if a SAN can't be verified. This prevents the `kubectl exec` and `kubectl logs` commands from working.

Before upgrading your cluster to `1.24`, determine whether your cluster has certificate signing requests (CSR) that haven't been approved by completing the following steps:

1. Run the following command.

```
kubectl get csr -A
```

An example output is as follows.

```
NAME        AGE   SIGNERNAME                            REQUESTOR
                        REQUESTEDDURATION   CONDITION
csr-7znmf   90m   kubernetes.io/kubelet-serving
 system:node:ip-192-168-42-149.region.compute.internal      <none>
 Approved
csr-9xx5q   90m   kubernetes.io/kubelet-serving
 system:node:ip-192-168-65-38.region.compute.internal      <none>
 Approved, Issued
```

If the returned output shows a CSR with a kubernetes.io/kubelet-serving signer that's `Approved` but not `Issued` for a node, then you need to approve the request.

2. Manually approve the CSR. Replace `csr-7znmf` with your own value.

```
kubectl certificate approve csr-7znmf
```

To auto-approve CSRs in the future, we recommend that you write an approving controller that can automatically validate and approve CSRs that contain IP or DNS SANs that Amazon EKS can't verify.

Understand Amazon EKS created RBAC roles and users

When you create a Kubernetes cluster, several default Kubernetes identities are created on that cluster for the proper functioning of Kubernetes. Amazon EKS creates Kubernetes identities for each of its default components. The identities provide Kubernetes role-based authorization control (RBAC) for the cluster components. For more information, see Using RBAC Authorization in the Kubernetes documentation.

When you install optional add-ons to your cluster, additional Kubernetes identities might be added to your cluster. For more information about identities not addressed by this topic, see the documentation for the add-on.

You can view the list of Amazon EKS created Kubernetes identities on your cluster using the AWS Management Console or `kubectl` command line tool. All of the user identities appear in the kube audit logs available to you through Amazon CloudWatch.

AWS Management Console

Prerequisite

The IAM principal that you use must have the permissions described in Required permissions.

To view Amazon EKS created identities using the AWS Management Console

1. Open the Amazon EKS console.
2. In the **Clusters** list, choose the cluster that contains the identities that you want to view.
3. Choose the **Resources** tab.
4. Under **Resource types**, choose **Authorization**.
5. Choose, **ClusterRoles**, **ClusterRoleBindings**, **Roles**, or **RoleBindings**. All resources prefaced with **eks** are created by Amazon EKS. Additional Amazon EKS created identity resources are:
   - The **ClusterRole** and **ClusterRoleBinding** named **aws-node**. The **aws-node** resources support the Amazon VPC CNI plugin for Kubernetes, which Amazon EKS installs on all clusters.
   - A **ClusterRole** named **vpc-resource-controller-role** and a **ClusterRoleBinding** named **vpc-resource-controller-rolebinding**. These resources support the Amazon VPC resource controller, which Amazon EKS installs on all clusters.

   In addition to the resources that you see in the console, the following special user identities exist on your cluster, though they're not visible in the cluster's configuration:
   - **`eks:cluster-bootstrap`** – Used for `kubectl` operations during cluster bootstrap.
   - **`eks:support-engineer`** – Used for cluster management operations.
6. Choose a specific resource to view details about it. By default, you're shown information in **Structured view**. In the top-right corner of the details page you can choose **Raw view** to see all information for the resource.

Kubectl

Prerequisite

The entity that you use (AWS Identity and Access Management (IAM) or OpenID Connect (OIDC)) to list the Kubernetes resources on the cluster must be authenticated by IAM or your OIDC identity provider. The entity must be granted permissions to use the Kubernetes `get` and `list` verbs for the `Role`, `ClusterRole`, `RoleBinding`, and `ClusterRoleBinding` resources on your cluster that you want the entity to work with. For more information about granting IAM entities access to your cluster, see the section called "Kubernetes API access". For more information about granting entities authenticated by your own OIDC provider access to your cluster, see the section called "Link OIDC provider".

To view Amazon EKS created identities using `kubectl`

Run the command for the type of resource that you want to see. All returned resources that are prefaced with **eks** are created by Amazon EKS. In addition to the resources returned in the output from the commands, the following special user identities exist on your cluster, though they're not visible in the cluster's configuration:

- **`eks:cluster-bootstrap`** – Used for `kubectl` operations during cluster bootstrap.
- **`eks:support-engineer`** – Used for cluster management operations.

ClusterRoles – `ClusterRoles` are scoped to your cluster, so any permission granted to a role applies to resources in any Kubernetes namespace on the cluster.

The following command returns all of the Amazon EKS created Kubernetes `ClusterRoles` on your cluster.

```
kubectl get clusterroles | grep eks
```

In addition to the `ClusterRoles` returned in the output that are prefaced with, the following `ClusterRoles` exist.

- **`aws-node`** – This `ClusterRole` supports the Amazon VPC CNI plugin for Kubernetes, which Amazon EKS installs on all clusters.
- **`vpc-resource-controller-role`** – This `ClusterRole` supports the Amazon VPC resource controller, which Amazon EKS installs on all clusters.

To see the specification for a `ClusterRole`, replace *`eks:k8s-metrics`* in the following command with a `ClusterRole` returned in the output of the previous command. The following example returns the specification for the *`eks:k8s-metrics`* `ClusterRole`.

```
kubectl describe clusterrole eks:k8s-metrics
```

An example output is as follows.

```
Name:         eks:k8s-metrics
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources         Non-Resource URLs  Resource Names  Verbs
  ---------         -----------------  --------------  -----
                    [/metrics]         []              [get]
  endpoints         []                 []              [list]
  nodes             []                 []              [list]
  pods              []                 []              [list]
  deployments.apps  []                 []              [list]
```

ClusterRoleBindings – `ClusterRoleBindings` are scoped to your cluster.

The following command returns all of the Amazon EKS created Kubernetes `ClusterRoleBindings` on your cluster.

```
kubectl get clusterrolebindings | grep eks
```

In addition to the `ClusterRoleBindings` returned in the output, the following `ClusterRoleBindings` exist.

- **`aws-node`** – This `ClusterRoleBinding` supports the [Amazon VPC CNI plugin for Kubernetes](), which Amazon EKS installs on all clusters.
- **`vpc-resource-controller-rolebinding`** – This `ClusterRoleBinding` supports the [Amazon VPC resource controller](), which Amazon EKS installs on all clusters.

To see the specification for a `ClusterRoleBinding`, replace *`eks:k8s-metrics`* in the following command with a `ClusterRoleBinding` returned in the output of the previous command. The following example returns the specification for the *`eks:k8s-metrics`* `ClusterRoleBinding`.

```
kubectl describe clusterrolebinding eks:k8s-metrics
```

An example output is as follows.

```
Name:         eks:k8s-metrics
Labels:       <none>
Annotations:  <none>
```

```
Role:
  Kind:  ClusterRole
  Name:  eks:k8s-metrics
Subjects:
  Kind  Name             Namespace
  ----  ----             ---------
  User  eks:k8s-metrics
```

Roles – Roles are scoped to a Kubernetes namespace. All Amazon EKS created Roles are scoped to the kube-system namespace.

The following command returns all of the Amazon EKS created Kubernetes Roles on your cluster.

```
kubectl get roles -n kube-system | grep eks
```

To see the specification for a Role, replace *eks:k8s-metrics* in the following command with the name of a Role returned in the output of the previous command. The following example returns the specification for the *eks:k8s-metrics* Role.

```
kubectl describe role eks:k8s-metrics -n kube-system
```

An example output is as follows.

```
Name:         eks:k8s-metrics
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources         Non-Resource URLs  Resource Names              Verbs
  ---------         -----------------  --------------              -----
  daemonsets.apps   []                 [aws-node]                  [get]
  deployments.apps  []                 [vpc-resource-controller]   [get]
```

RoleBindings – RoleBindings are scoped to a Kubernetes namespace. All Amazon EKS created RoleBindings are scoped to the kube-system namespace.

The following command returns all of the Amazon EKS created Kubernetes RoleBindings on your cluster.

```
kubectl get rolebindings -n kube-system | grep eks
```

To see the specification for a `RoleBinding`, replace *`eks:k8s-metrics`* in the following command with a `RoleBinding` returned in the output of the previous command. The following example returns the specification for the *`eks:k8s-metrics`* `RoleBinding`.

```
kubectl describe rolebinding eks:k8s-metrics -n kube-system
```

An example output is as follows.

```
Name:         eks:k8s-metrics
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  eks:k8s-metrics
Subjects:
  Kind  Name             Namespace
  ----  ----             ---------
  User  eks:k8s-metrics
```

Understand Amazon EKS created Pod security policies (PSP)

The Kubernetes Pod security policy admission controller validates Pod creation and update requests against a set of rules. By default, Amazon EKS clusters ship with a fully permissive security policy with no restrictions. For more information, see Pod Security Policies in the Kubernetes documentation.

> **Note**
>
> The `PodSecurityPolicy` (PSP) was deprecated in Kubernetes version `1.21` and removed in Kubernetes `1.25`. PSPs are being replaced with Pod Security Admission (PSA), a built-in admission controller that implements the security controls outlined in the Pod Security Standards (PSS). PSA and PSS have both reached beta feature states, and are enabled in Amazon EKS by default. To address PSP removal in `1.25`, we recommend that you implement PSS in Amazon EKS. For more information, see Implementing Pod Security Standards in Amazon EKS on the AWS blog.

Amazon EKS default Pod security policy

Amazon EKS clusters with Kubernetes version `1.13` or higher have a default Pod security policy named `eks.privileged`. This policy has no restriction on what kind of Pod can be accepted into the system, which is equivalent to running Kubernetes with the `PodSecurityPolicy` controller disabled.

> **Note**
>
> This policy was created to maintain backwards compatibility with clusters that did not have the `PodSecurityPolicy` controller enabled. You can create more restrictive policies for your cluster and for individual namespaces and service accounts and then delete the default policy to enable the more restrictive policies.

You can view the default policy with the following command.

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

An example output is as follows.

```
NAME             PRIV   CAPS   SELINUX    RUNASUSER   FSGROUP    SUPGROUP
 READONLYROOTFS   VOLUMES
eks.privileged   true   *      RunAsAny   RunAsAny    RunAsAny   RunAsAny   false
      *
```

For more details, you can describe the policy with the following command.

```
kubectl describe psp eks.privileged
```

An example output is as follows.

```
Name:  eks.privileged

Settings:
  Allow Privileged:                        true
  Allow Privilege Escalation:              0xc0004ce5f8
  Default Add Capabilities:                <none>
  Required Drop Capabilities:              <none>
```

```
  Allowed Capabilities:                    *
  Allowed Volume Types:                    *
  Allow Host Network:                      true
  Allow Host Ports:                        0-65535
  Allow Host PID:                          true
  Allow Host IPC:                          true
  Read Only Root Filesystem:               false
  SELinux Context Strategy: RunAsAny
    User:                                  <none>
    Role:                                  <none>
    Type:                                  <none>
    Level:                                 <none>
  Run As User Strategy: RunAsAny
    Ranges:                                <none>
  FSGroup Strategy: RunAsAny
    Ranges:                                <none>
  Supplemental Groups Strategy: RunAsAny
    Ranges:                                <none>
```

You can view the full YAML file for the `eks.privileged` Pod security policy, its cluster role, and cluster role binding in Install or restore the default Pod security policy.

Delete the default Amazon EKS Pod security policy

If you create more restrictive policies for your Pods, then after doing so, you can delete the default Amazon EKS `eks.privileged` Pod security policy to enable your custom policies.

> **Important**
>
> If you are using version `1.7.0` or later of the CNI plugin and you assign a custom Pod security policy to the `aws-node` Kubernetes service account used for the `aws-node` Pods deployed by the Daemonset, then the policy must have `NET_ADMIN` in its `allowedCapabilities` section along with `hostNetwork: true` and `privileged: true` in the policy's `spec`.

1. Create a file named *`privileged-podsecuritypolicy.yaml`* with the contents in the example file in Install or restore the default Pod security policy.
2. Delete the YAML with the following command. This deletes the default Pod security policy, the `ClusterRole`, and the `ClusterRoleBinding` associated with it.

```
kubectl delete -f privileged-podsecuritypolicy.yaml
```

Install or restore the default Pod security policy

If you are upgrading from an earlier version of Kubernetes, or have modified or deleted the default Amazon EKS `eks.privileged` Pod security policy, you can restore it with the following steps.

1. Create a file called `privileged-podsecuritypolicy.yaml` with the following contents.

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: eks.privileged
  annotations:
    kubernetes.io/description: 'privileged allows full unrestricted access to
      Pod features, as if the PodSecurityPolicy controller was not enabled.'
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
spec:
  privileged: true
  allowPrivilegeEscalation: true
  allowedCapabilities:
  - '*'
  volumes:
  - '*'
  hostNetwork: true
  hostPorts:
  - min: 0
    max: 65535
  hostIPC: true
  hostPID: true
  runAsUser:
    rule: 'RunAsAny'
  seLinux:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'
```

```
  readOnlyRootFilesystem: false

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: eks:podsecuritypolicy:privileged
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
rules:
- apiGroups:
  - policy
  resourceNames:
  - eks.privileged
  resources:
  - podsecuritypolicies
  verbs:
  - use

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks:podsecuritypolicy:authenticated
  annotations:
    kubernetes.io/description: 'Allow all authenticated users to create privileged
 Pods.'
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:podsecuritypolicy:privileged
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:authenticated
```

2. Apply the YAML with the following command.

```
kubectl apply -f privileged-podsecuritypolicy.yaml
```

Migrate from legacy Pod security policies (PSP)

`PodSecurityPolicy` was deprecated in Kubernetes1.21, and has been removed in Kubernetes `1.25`. If you are using PodSecurityPolicy in your cluster, **then you must migrate to the built-in Kubernetes Pod Security Standards (PSS) or to a policy-as-code solution before upgrading your cluster to version *`1.25` to avoid interruptions to your workloads.* Select any frequently asked question to learn more.**

What is a PSP?

PodSecurityPolicy is a built-in admission controller that allows a cluster administrator to control security-sensitive aspects of Pod specification. If a Pod meets the requirements of its PSP, the Pod is admitted to the cluster as usual. If a Pod doesn't meet the PSP requirements, the Pod is rejected and can't run.

Is the PSP removal specific to Amazon EKS or is it being removed in upstream Kubernetes?

This is an upstream change in the Kubernetes project, and not a change made in Amazon EKS. PSP was deprecated in Kubernetes `1.21` and removed in Kubernetes `1.25`. The Kubernetes community identified serious usability problems with PSP. These included accidentally granting broader permissions than intended and difficulty in inspecting which PSPs apply in a given situation. These issues couldn't be addressed without making breaking changes. This is the primary reason why the Kubernetes community decided to remove PSP.

How can I check if I'm using PSPs in my Amazon EKS clusters?

To check if you're using PSPs in your cluster, you can run the following command:

```
kubectl get psp
```

To see the Pods that the PSPs in your cluster are impacting, run the following command. This command outputs the Pod name, namespace, and PSPs:

```
kubectl get pod -A -o jsonpath='{range.items[?(@.metadata.annotations.kubernetes\.io/
psp)]}{.metadata.name}{" "}{.metadata.namespace}{" "}{.metadata.annotations.kubernetes
\.io/psp}{"
"}'
```

If I'm using PSPs in my Amazon EKS cluster, what can I do?

Before upgrading your cluster to `1.25`, you must migrate your PSPs to either one of these alternatives:

- Kubernetes PSS.
- Policy-as-code solutions from the Kubernetes environment.

In response to the PSP deprecation and the ongoing need to control Pod security from the start, the Kubernetes community created a built-in solution with (PSS) and Pod Security Admission (PSA). The PSA webhook implements the controls that are defined in the PSS.

You can review best practices for migrating PSPs to the built-in PSS in the EKS Best Practices Guide. We also recommend reviewing our blog on Implementing Pod Security Standards in Amazon EKS. Additional references include Migrate from PodSecurityPolicy to the Built-In PodSecurity Admission Controller and Mapping PodSecurityPolicies to Pod Security Standards.

Policy-as-code solutions provide guardrails to guide cluster users and prevents unwanted behaviors through prescribed automated controls. Policy-as-code solutions typically use Kubernetes Dynamic Admission Controllers to intercept the Kubernetes API server request flow using a webhook call. Policy-as-code solutions mutate and validate request payloads based on policies written and stored as code.

There are several open source policy-as-code solutions available for Kubernetes. To review best practices for migrating PSPs to a policy-as-code solution, see the Policy-as-code section of the Pod Security page on GitHub.

I see a PSP called `eks.privileged` in my cluster. What is it and what can I do about it?

Amazon EKS clusters with Kubernetes version `1.13` or higher have a default PSP that's named `eks.privileged`. This policy is created in `1.24` and earlier clusters. It isn't used in `1.25` and later clusters. Amazon EKS automatically migrates this PSP to a PSS-based enforcement. No action is needed on your part.

Will Amazon EKS make any changes to PSPs present in my existing cluster when I update my cluster to version `1.25`?

No. Besides `eks.privileged`, which is a PSP created by Amazon EKS, no changes are made to other PSPs in your cluster when you upgrade to `1.25`.

Will Amazon EKS prevent a cluster update to version `1.25` if I haven't migrated off of PSP?

No. Amazon EKS won't prevent a cluster update to version `1.25` if you didn't migrate off of PSP yet.

What if I forget to migrate my PSPs to PSS/PSA or to a policy-as-code solution before I update my cluster to version `1.25`? Can I migrate after updating my cluster?

When a cluster that contains a PSP is upgraded to Kubernetes version `1.25`, the API server doesn't recognize the PSP resource in `1.25`. This might result in Pods getting incorrect security scopes. For an exhaustive list of implications, see Migrate from PodSecurityPolicy to the Built-In PodSecurity Admission Controller.

How does this change impact pod security for Windows workloads?

We don't expect any specific impact to Windows workloads. PodSecurityContext has a field called `windowsOptions` in the `PodSpec v1` API for Windows Pods. This uses PSS in Kubernetes `1.25`. For more information and best practices about enforcing PSS for Windows workloads, see the EKS Best Practices Guide and Kubernetes documentation.

Encrypt Kubernetes secrets with {aws} KMS on existing clusters

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

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

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

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

eksctl

You can enable encryption in two ways:

- Add encryption to your cluster with a single command.

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

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

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

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

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

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

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

secretsEncryption:
  keyARN: arn:aws:kms:region-code:account:key/key
```

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

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

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

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

AWS Management Console

a. Open the Amazon EKS console.

b. Choose the cluster that you want to add KMS encryption to.

c. Choose the **Overview** tab (this is selected by default).

d. Scroll down to the **Secrets encryption** section and choose **Enable**.

e. Select a key from the dropdown list and choose the **Enable** button. If no keys are listed, you must create one first. For more information, see Creating keys

f. Choose the **Confirm** button to use the chosen key.

AWS CLI

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

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

An example output is as follows.

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

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

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

An example output is as follows.

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

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

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

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

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

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

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

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

Use AWS Secrets Manager secrets with Amazon EKS Pods

To show secrets from Secrets Manager and parameters from Parameter Store as files mounted in Amazon EKS Pods, you can use the AWS Secrets and Configuration Provider (ASCP) for the Kubernetes Secrets Store CSI Driver.

With the ASCP, you can store and manage your secrets in Secrets Manager and then retrieve them through your workloads running on Amazon EKS. You can use IAM roles and policies to limit access to your secrets to specific Kubernetes Pods in a cluster. The ASCP retrieves the Pod identity and exchanges the identity for an IAM role. ASCP assumes the IAM role of the Pod, and then it can retrieve secrets from Secrets Manager that are authorized for that role.

If you use Secrets Manager automatic rotation for your secrets, you can also use the Secrets Store CSI Driver rotation reconciler feature to ensure you are retrieving the latest secret from Secrets Manager.

> **Note**
>
> AWS Fargate (Fargate) node groups are not supported.

For more information, see Using Secrets Manager secrets in Amazon EKS in the AWS Secrets Manager User Guide.

Security considerations for Amazon EKS Auto Mode

This topic describes the security architecture, controls, and best practices for Amazon EKS Auto Mode. As organizations deploy containerized applications at scale, maintaining a strong security posture becomes increasingly complex. EKS Auto Mode implements automated security controls and integrates with AWS security services to help you protect your cluster infrastructure, workloads, and data. Through built-in security features like enforced node lifecycle management and automated patch deployment, EKS Auto Mode helps you maintain security best practices while reducing operational overhead.

Before proceeding with this topic, make sure that you're familiar with basic EKS Auto Mode concepts and have reviewed the prerequisites for enabling EKS Auto Mode on your clusters. For general information about Amazon EKS security, see *Security*.

Amazon EKS Auto Mode builds upon the existing security foundations of Amazon EKS while introducing additional automated security controls for EC2 managed instances.

API security and authentication

Amazon EKS Auto Mode uses AWS platform security mechanisms to secure and authenticate calls to the Amazon EKS API.

- Access to the Kubernetes API is secured through EKS access entries, which integrate with AWS IAM identities.
  - For more information, see the section called "Grant permissions".
- Customers can implement fine-grained access control to the Kubernetes API endpoint through configuration of EKS access entries.

Network security

Amazon EKS Auto Mode supports multiple layers of network security:

- **VPC integration**
  - Operates within your Amazon Virtual Private Cloud (VPC)
  - Supports custom VPC configurations and subnet layouts
  - Enables private networking between cluster components
  - For more information, see Managing security responsibilities for Amazon Virtual Private Cloud
- **Network Policies**
  - Native support for Kubernetes Network Policies
  - Ability to define granular network traffic rules
  - For more information, see the section called "Kubernetes policies"

EC2 managed instance security

Amazon EKS Auto Mode operates EC2 managed instances with the following security controls:

EC2 security

- EC2 managed instances maintain the security features of Amazon EC2.
- For more information about EC2 managed instances, see Security in Amazon EC2.

Instance lifecycle management

EC2 managed instances operated by EKS Auto Mode have maximum lifetime of 21 days. Amazon EKS Auto Mode automatically terminates instances exceeding this lifetime. This lifecycle limit helps prevent configuration drift and maintains security posture.

Data protection

- Amazon EC2 Instance Storage is encrypted, this is storage directly attached to the instance. For more information, see Data protection in Amazon EC2.
- EKS Auto Mode manages the volumes attached to EC2 instances at creation time, including root and data volumes. EKS Auto Mode does not fully manage EBS volumes created using Kubernetes persistent storage features.

Patch management

- Amazon EKS Auto Mode automatically applies patches to managed instances.
- Patches include:
  - Operating system updates
  - Security patches
  - Amazon EKS Auto Mode components

> **Note**
>
> Customers retain responsibility for securing and updating workloads running on these instances.

Access controls

- Direct instance access is restricted:
  - SSH access is not available.
  - AWS Systems Manager Session Manager (SSM) access is not available.
- Management operations are performed through the Amazon EKS API and Kubernetes API.

Automated resource management

Amazon EKS Auto Mode does not fully manage Amazon Elastic Block Store (Amazon EBS) Volumes created using Kubernetes persistent storage features. EKS Auto Mode also does not manage Elastic Load Balancers (ELB). Amazon EKS Auto Mode automates routine tasks for these resources.

Storage security

- AWS recommends that you enable encryption for EBS Volumes provisioned by Kubernetes persistent storage features. For more information, see the section called "Create StorageClass".
- Encryption at rest using AWS KMS
- You can configure your AWS account to enforce the encryption of the new EBS volumes and snapshot copies that you create. For more information, see Enable Amazon EBS encryption by default in the Amazon EBS User Guide.
- For more information, see Security in Amazon EBS.

Load balancer security

- Automated configuration of Elastic Load Balancers
- SSL/TLS certificate management through AWS Certificate Manager integration
- Security group automation for load balancer access control
- For more information, see Security in Elastic Load Balancing.

Security best practices

The following section describes security best practices for Amazon EKS Auto Mode.

- Regularly review AWS IAM policies and EKS access entries.
- Implement least privilege access patterns for workloads.
- Monitor cluster activity through AWS CloudTrail and Amazon CloudWatch. For more information, see the section called " AWS CloudTrail" and the section called "Amazon CloudWatch".
- Use AWS Security Hub for security posture assessment.
- Implement pod security standards appropriate for your workloads.

Identity and access management for Amazon EKS

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon EKS resources. IAM is an AWS service that you can use with no additional charge.

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon EKS.

Service user – If you use the Amazon EKS service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon EKS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon EKS, see the section called "Troubleshooting".

Service administrator – If you're in charge of Amazon EKS resources at your company, you probably have full access to Amazon EKS. It's your job to determine which Amazon EKS features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon EKS, see the section called "Amazon EKS and IAM".

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon EKS. To view example Amazon EKS identity-based policies that you can use in IAM, see the section called "Identity-based policies".

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An IAM group is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

  - **Service role –** A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role –** A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2 –** You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon EKS works with IAM

Before you use IAM to manage access to Amazon EKS, you should understand what IAM features are available to use with Amazon EKS. To get a high-level view of how Amazon EKS and other AWS services work with IAM, see AWS services that work with IAM in the *IAM User Guide*.

Topics

- Amazon EKS identity-based policies
- Amazon EKS resource-based policies
- Authorization based on Amazon EKS tags
- Amazon EKS IAM roles

Amazon EKS identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon EKS supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon EKS use the following prefix before the action: `eks:`. For example, to grant someone permission to get descriptive information about an Amazon EKS cluster, you include the `DescribeCluster` action in their policy. Policy statements must include either an `Action` or `NotAction` element.

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

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

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

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

To see a list of Amazon EKS actions, see Actions defined by Amazon Elastic Kubernetes Service in the *Service Authorization Reference*.

Resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

The Amazon EKS cluster resource has the following ARN.

```
            arn:aws:eks:region-code:account-id:cluster/cluster-name
```

For more information about the format of ARNs, see Amazon resource names (ARNs) and AWS service namespaces.

For example, to specify the cluster with the name *`my-cluster`* in your statement, use the following ARN:

```
"Resource": "arn:aws:eks:region-code:111122223333:cluster/my-cluster"
```

To specify all clusters that belong to a specific account and AWS Region, use the wildcard (*):

```
"Resource": "arn:aws:eks:region-code:111122223333:cluster/*"
```

Some Amazon EKS actions, such as those for creating resources, can't be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

To see a list of Amazon EKS resource types and their ARNs, see Resources defined by Amazon Elastic Kubernetes Service in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Amazon Elastic Kubernetes Service.

Condition keys

Amazon EKS defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the *IAM User Guide*.

You can set condition keys when associating an OpenID Connect provider to your cluster. For more information, see the section called “Example IAM policy”.

All Amazon EC2 actions support the `aws:RequestedRegion` and `ec2:Region` condition keys. For more information, see Example: Restricting Access to a Specific AWS Region.

For a list of Amazon EKS condition keys, see Conditions defined by Amazon Elastic Kubernetes Service in the *Service Authorization Reference*. To learn which actions and resources you can use a condition key with, see Actions defined by Amazon Elastic Kubernetes Service.

Examples

To view examples of Amazon EKS identity-based policies, see the section called “Identity-based policies”.

When you create an Amazon EKS cluster, the IAM principal that creates the cluster is automatically granted `system:masters` permissions in the cluster's role-based access control (RBAC) configuration in the Amazon EKS control plane. This principal doesn't appear in any visible configuration, so make sure to keep track of which principal originally created the cluster. To grant additional IAM principals the ability to interact with your cluster, edit the `aws-auth ConfigMap` within Kubernetes and create a Kubernetes `rolebinding` or `clusterrolebinding` with the name of a `group` that you specify in the `aws-auth ConfigMap`.

For more information about working with the ConfigMap, see the section called “Kubernetes API access”.

Amazon EKS resource-based policies

Amazon EKS does not support resource-based policies.

Authorization based on Amazon EKS tags

You can attach tags to Amazon EKS resources or pass tags in a request to Amazon EKS. To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys. For more information about tagging Amazon EKS resources, see the section called "Tagging your resources". For more information about which actions that you can use tags in condition keys with, see Actions defined by Amazon EKS in the Service Authorization Reference.

Amazon EKS IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Using temporary credentials with Amazon EKS

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

Amazon EKS supports using temporary credentials.

Service-linked roles

```
link:IAM/latest/UserGuide/id_roles.html#iam-term-service-linked-role[Service-linked
 roles,type="documentation"] allow {aws} services to access resources in other services
 to complete an action on your behalf. Service-linked roles appear in your IAM account
 and are owned by the service. An administrator can view but can't edit the permissions
 for service-linked roles.
```

Amazon EKS supports service-linked roles. For details about creating or managing Amazon EKS service-linked roles, see the section called "Service-linked roles".

Service roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Amazon EKS supports service roles. For more information, see the section called "Cluster IAM role" and the section called "Node IAM role".

Choosing an IAM role in Amazon EKS

When you create a cluster resource in Amazon EKS, you must choose a role to allow Amazon EKS to access several other AWS resources on your behalf. If you have previously created a service role, then Amazon EKS provides you with a list of roles to choose from. It's important to choose a role that has the Amazon EKS managed policies attached to it. For more information, see the section called "Check for an existing cluster role" and the section called "Check for an existing node role".

Amazon EKS identity-based policy examples

By default, IAM users and roles don't have permission to create or modify Amazon EKS resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating policies on the JSON tab in the *IAM User Guide*.

When you create an Amazon EKS cluster, the IAM principal that creates the cluster is automatically granted `system:masters` permissions in the cluster's role-based access control (RBAC) configuration in the Amazon EKS control plane. This principal doesn't appear in any visible configuration, so make sure to keep track of which principal originally created the cluster. To grant additional IAM principals the ability to interact with your cluster, edit the `aws-auth ConfigMap` within Kubernetes and create a Kubernetes `rolebinding` or `clusterrolebinding` with the name of a `group` that you specify in the `aws-auth ConfigMap`.

For more information about working with the ConfigMap, see the section called "Kubernetes API access".

Topics

- Policy best practices
- Using the Amazon EKS console
- Allow IAM users to view their own permissions
- Create a Kubernetes cluster on the AWS Cloud
- Create a local Kubernetes cluster on an Outpost
- Update a Kubernetes cluster
- List or describe all clusters

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon EKS resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see IAM Access Analyzer policy validation in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the Amazon EKS console

To access the Amazon EKS console, an IAM principal, must have a minimum set of permissions. These permissions allow the principal to list and view details about the Amazon EKS resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for principals with that policy attached to them.

To ensure that your IAM principals can still use the Amazon EKS console, create a policy with your own unique name, such as `AmazonEKSAdminPolicy`. Attach the policy to the principals. For more information, see Adding and removing IAM identity permissions in the *IAM User Guide*.

Important

The following example policy allows a principal to view information on the **Configuration** tab in the console. To view information on the **Overview** and **Resources** tabs in the AWS Management Console, the principal also needs Kubernetes permissions. For more information, see the section called "Required permissions".

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "eks.amazonaws.com"
                }
            }
        }
    ]
```

```
}
```

You don't need to allow minimum console permissions for principals that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Allow IAM users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

Create a Kubernetes cluster on the AWS Cloud

This example policy includes the minimum permissions required to create an Amazon EKS cluster named *my-cluster* in the *us-west-2* AWS Region. You can replace the AWS Region with the AWS Region that you want to create a cluster in. If you see a warning that says **The actions in your policy do not support resource-level permissions and require you to choose `All resources`** in the AWS Management Console, it can be safely ignored. If your account already has the *AWSServiceRoleForAmazonEKS* role, you can remove the `iam:CreateServiceLinkedRole` action from the policy. If you've ever created an Amazon EKS cluster in your account then this role already exists, unless you deleted it.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:CreateCluster",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::111122223333:role/aws-service-role/
eks.amazonaws.com/AWSServiceRoleForAmazonEKS",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iam:AWSServiceName": "eks"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::111122223333:role/cluster-role-name"
        }
    ]
}
```

Create a local Kubernetes cluster on an Outpost

This example policy includes the minimum permissions required to create an Amazon EKS local cluster named *my-cluster* on an Outpost in the *us-west-2* AWS Region. You can replace the AWS Region with the AWS Region that you want to create a cluster in. If you see a warning that says **The actions in your policy do not support resource-level permissions and require you to choose `All resources`** in the AWS Management Console, it can be safely ignored. If your account already has the `AWSServiceRoleForAmazonEKSLocalOutpost` role, you can remove the `iam:CreateServiceLinkedRole` action from the policy. If you've ever created an Amazon EKS local cluster on an Outpost in your account then this role already exists, unless you deleted it.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:CreateCluster",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
        },
        {
            "Action": [
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "iam:GetRole"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::111122223333:role/aws-service-role/outposts.eks-
local.amazonaws.com/AWSServiceRoleForAmazonEKSLocalOutpost"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole",
                "iam:ListAttachedRolePolicies"
            ]
            "Resource": "arn:aws:iam::111122223333:role/cluster-role-name"
        },
```

```
        {
            "Action": [
                "iam:CreateInstanceProfile",
                "iam:TagInstanceProfile",
                "iam:AddRoleToInstanceProfile",
                "iam:GetInstanceProfile",
                "iam:DeleteInstanceProfile",
                "iam:RemoveRoleFromInstanceProfile"
            ],
            "Resource": "arn:aws:iam::*:instance-profile/eks-local-*",
            "Effect": "Allow"
        },
    ]
}
```

Update a Kubernetes cluster

This example policy includes the minimum permission required to update a cluster named *my-cluster* in the us-west-2 AWS Region.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:UpdateClusterVersion",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
        }
    ]
}
```

List or describe all clusters

This example policy includes the minimum permissions required to list and describe all clusters in your account. An IAM principal must be able to list and describe clusters to use the `update-kubeconfig` AWS CLI command.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:ListClusters"
            ],
            "Resource": "*"
        }
    ]
}
```

Using service-linked roles for Amazon EKS

Amazon Elastic Kubernetes Service uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

Topics

Using roles for Amazon EKS clusters

Amazon Elastic Kubernetes Service uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon EKS easier because you don't have to manually add the necessary permissions. Amazon EKS defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon EKS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Amazon EKS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Amazon EKS

Amazon EKS uses the service-linked role named `AWSServiceRoleForAmazonEKS`. The role allows Amazon EKS to manage clusters in your account. The attached policies allow the role to manage the following resources: network interfaces, security groups, logs, and VPCs.

> **Note**
>
> The `AWSServiceRoleForAmazonEKS` service-linked role is distinct from the role required for cluster creation. For more information, see the section called "Cluster IAM role".

The `AWSServiceRoleForAmazonEKS` service-linked role trusts the following services to assume the role:

- `eks.amazonaws.com`

The role permissions policy allows Amazon EKS to complete the following actions on the specified resources:

- AmazonEKSServiceRolePolicy

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for Amazon EKS

You don't need to manually create a service-linked role. When you create a cluster in the AWS Management Console, the AWS CLI, or the AWS API, Amazon EKS creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a cluster, Amazon EKS creates the service-linked role for you again.

Editing a service-linked role for Amazon EKS

Amazon EKS does not allow you to edit the `AWSServiceRoleForAmazonEKS` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for Amazon EKS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

> **Note**
>
> If the Amazon EKS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

1. Open the Amazon EKS console.
2. In the left navigation pane, choose **Clusters**.
3. If your cluster has any node groups or Fargate profiles, you must delete them before you can delete the cluster. For more information, see the section called "Delete" and the section called "Delete profiles".
4. On the **Clusters** page, choose the cluster that you want to delete and choose **Delete**.
5. Type the name of the cluster in the deletion confirmation window, and then choose **Delete**.
6. Repeat this procedure for any other clusters in your account. Wait for all of the delete operations to finish.

Manually delete the service-linked role

Use the IAM console, the AWS CLI, or the AWS API to delete the `AWSServiceRoleForAmazonEKS` service-linked role. For more information, see [Deleting a service-linked role](#) in the *IAM User Guide*.

Supported regions for Amazon EKS service-linked roles

Amazon EKS supports using service-linked roles in all of the regions where the service is available. For more information, see [Amazon EKS endpoints and quotas](#).

Using roles for Amazon EKS node groups

Amazon EKS uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon EKS easier because you don't have to manually add the necessary permissions. Amazon EKS defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon EKS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Amazon EKS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see [AWS services that work with IAM](#) and look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Amazon EKS

Amazon EKS uses the service-linked role named `AWSServiceRoleForAmazonEKSNodegroup`. The role allows Amazon EKS to manage node groups in your account. The attached `AWSServiceRoleForAmazonEKSNodegroup` policy allows the role to manage the following resources: Auto Scaling groups, security groups, launch templates, and IAM instance profiles. For more information, see [the section called "AWS managed policy: AWSServiceRoleForAmazonEKSNodegroup"](#).

The `AWSServiceRoleForAmazonEKSNodegroup` service-linked role trusts the following services to assume the role:

- `eks-nodegroup.amazonaws.com`

The role permissions policy allows Amazon EKS to complete the following actions on the specified resources:

- AWSServiceRoleForAmazonEKSNodegroup

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for Amazon EKS

You don't need to manually create a service-linked role. When you CreateNodegroup in the AWS Management Console, the AWS CLI, or the AWS API, Amazon EKS creates the service-linked role for you.

> **Important**
>
> This service-linked role can appear in your account if you completed an action in another service that uses the features supported by this role. If you were using the Amazon EKS service before January 1, 2017, when it began supporting service-linked roles, then Amazon EKS created the AWSServiceRoleForAmazonEKSNodegroup role in your account. To learn more, see A new role appeared in my IAM account.

Creating a service-linked role in Amazon EKS (AWS API)

You don't need to manually create a service-linked role. When you create a managed node group in the AWS Management Console, the AWS CLI, or the AWS API, Amazon EKS creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create another managed node group, Amazon EKS creates the service-linked role for you again.

Editing a service-linked role for Amazon EKS

Amazon EKS does not allow you to edit the `AWSServiceRoleForAmazonEKSNodegroup` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for Amazon EKS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

> **Note**
>
> If the Amazon EKS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

1. Open the Amazon EKS console.
2. In the left navigation pane, choose **Clusters**.
3. Select the **Compute** tab.
4. In the **Node groups** section, choose the node group to delete.
5. Type the name of the node group in the deletion confirmation window, and then choose **Delete**.
6. Repeat this procedure for any other node groups in the cluster. Wait for all of the delete operations to finish.

Manually delete the service-linked role

Use the IAM console, the AWS CLI, or the AWS API to delete the `AWSServiceRoleForAmazonEKSNodegroup` service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported regions for Amazon EKS service-linked roles

Amazon EKS supports using service-linked roles in all of the regions where the service is available. For more information, see Amazon EKS endpoints and quotas.

Using roles for Amazon EKS Fargate profiles

Amazon EKS uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon EKS easier because you don't have to manually add the necessary permissions. Amazon EKS defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon EKS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Amazon EKS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Amazon EKS

Amazon EKS uses the service-linked role named `AWSServiceRoleForAmazonEKSForFargate`. The role allows Amazon EKS Fargate to configure VPC networking required for Fargate Pods. The attached policies allow the role to create and delete elastic network interfaces and describe elastic network Interfaces and resources.

The `AWSServiceRoleForAmazonEKSForFargate` service-linked role trusts the following services to assume the role:

- `eks-fargate.amazonaws.com`

The role permissions policy allows Amazon EKS to complete the following actions on the specified resources:

- AmazonEKSForFargateServiceRolePolicy

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for Amazon EKS

You don't need to manually create a service-linked role. When you create a Fargate profile in the AWS Management Console, the AWS CLI, or the AWS API, Amazon EKS creates the service-linked role for you.

> **Important**
>
> This service-linked role can appear in your account if you completed an action in another service that uses the features supported by this role. If you were using the Amazon EKS service before December 13, 2019, when it began supporting service-linked roles, then Amazon EKS created the AWSServiceRoleForAmazonEKSForFargate role in your account. To learn more, see A New role appeared in my IAM account.

Creating a service-linked role in Amazon EKS (AWS API)

You don't need to manually create a service-linked role. When you create a Fargate profile in the AWS Management Console, the AWS CLI, or the AWS API, Amazon EKS creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create another managed node group, Amazon EKS creates the service-linked role for you again.

Editing a service-linked role for Amazon EKS

Amazon EKS does not allow you to edit the `AWSServiceRoleForAmazonEKSForFargate` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for Amazon EKS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

> **Note**
>
> If the Amazon EKS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

1. Open the Amazon EKS console.
2. In the left navigation pane, choose **Clusters**.
3. On the **Clusters** page, select your cluster.
4. Select the **Compute** tab.
5. If there are any Fargate profiles in the **Fargate profiles** section, select each one individually, and then choose **Delete**.
6. Type the name of the profile in the deletion confirmation window, and then choose **Delete**.
7. Repeat this procedure for any other Fargate profiles in the cluster and for any other clusters in your account.

Manually delete the service-linked role

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAmazonEKSForFargate service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported regions for Amazon EKS service-linked roles

Amazon EKS supports using service-linked roles in all of the regions where the service is available. For more information, see Amazon EKS endpoints and quotas.

Using roles to connect a Kubernetes cluster to Amazon EKS

Amazon EKS uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon EKS easier because you don't have to manually add the necessary permissions. Amazon EKS defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon EKS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Amazon EKS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Amazon EKS

Amazon EKS uses the service-linked role named `AWSServiceRoleForAmazonEKSConnector`. The role allows Amazon EKS to connect Kubernetes clusters. The attached policies allow the role to manage necessary resources to connect to your registered Kubernetes cluster.

The `AWSServiceRoleForAmazonEKSConnector` service-linked role trusts the following services to assume the role:

- `eks-connector.amazonaws.com`

The role permissions policy allows Amazon EKS to complete the following actions on the specified resources:

- AmazonEKSConnectorServiceRolePolicy

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for Amazon EKS

You don't need to manually create a service-linked role to connect a cluster. When you connect a cluster in the AWS Management Console, the AWS CLI, `eksctl`, or the AWS API, Amazon EKS creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you connect a cluster, Amazon EKS creates the service-linked role for you again.

Editing a service-linked role for Amazon EKS

Amazon EKS does not allow you to edit the `AWSServiceRoleForAmazonEKSConnector` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for Amazon EKS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

> **Note**
>
> If the Amazon EKS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

1. Open the Amazon EKS console.
2. In the left navigation pane, choose **Clusters**.
3. On the **Clusters** page, select your cluster.
4. Select the **Deregister** tab and then select the **Ok** tab.

Manually delete the service-linked role

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAmazonEKSConnector service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Using roles for Amazon EKS local clusters on Outpost

Amazon Elastic Kubernetes Service uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon EKS easier because you don't have to manually add the necessary permissions. Amazon EKS defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon EKS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Amazon EKS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Amazon EKS

Amazon EKS uses the service-linked role named `AWSServiceRoleForAmazonEKSLocalOutpost`. The role allows Amazon EKS to manage local clusters in your account. The attached policies allow the role to manage the following resources: network interfaces, security groups, logs, and Amazon EC2 instances.

> **Note**
>
> The `AWSServiceRoleForAmazonEKSLocalOutpost` service-linked role is distinct from the role required for cluster creation. For more information, see the section called "Cluster IAM role".

The `AWSServiceRoleForAmazonEKSLocalOutpost` service-linked role trusts the following services to assume the role:

- `outposts.eks-local.amazonaws.com`

The role permissions policy allows Amazon EKS to complete the following actions on the specified resources:

- AmazonEKSServiceRolePolicy

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for Amazon EKS

You don't need to manually create a service-linked role. When you create a cluster in the AWS Management Console, the AWS CLI, or the AWS API, Amazon EKS creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a cluster, Amazon EKS creates the service-linked role for you again.

Editing a service-linked role for Amazon EKS

Amazon EKS does not allow you to edit the `AWSServiceRoleForAmazonEKSLocalOutpost` service-linked role. After you create a service-linked role, you can't change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for Amazon EKS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

> **Note**
>
> If the Amazon EKS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

1. Open the Amazon EKS console.
2. In the left navigation pane, choose Amazon EKS **Clusters**.
3. If your cluster has any node groups or Fargate profiles, you must delete them before you can delete the cluster. For more information, see the section called “Delete” and the section called “Delete profiles”.
4. On the **Clusters** page, choose the cluster that you want to delete and choose **Delete**.
5. Type the name of the cluster in the deletion confirmation window, and then choose **Delete**.
6. Repeat this procedure for any other clusters in your account. Wait for all of the delete operations to finish.

Manually delete the service-linked role

Use the IAM console, the AWS CLI, or the AWS API to delete the `AWSServiceRoleForAmazonEKSLocalOutpost` service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported regions for Amazon EKS service-linked roles

Amazon EKS supports using service-linked roles in all of the regions where the service is available. For more information, see Amazon EKS endpoints and quotas.

Amazon EKS Pod execution IAM role

The Amazon EKS Pod execution role is required to run Pods on AWS Fargate infrastructure.

When your cluster creates Pods on AWS Fargate infrastructure, the components running on the Fargate infrastructure must make calls to AWS APIs on your behalf. This is so that they can do actions such as pull container images from Amazon ECR or route logs to other AWS services. The Amazon EKS Pod execution role provides the IAM permissions to do this.

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

Note

The Fargate profile must have a different IAM role than Amazon EC2 node groups.

Important

The containers running in the Fargate Pod can't assume the IAM permissions associated with a Pod execution role. To give the containers in your Fargate Pod permissions to access other AWS services, you must use IAM roles for service accounts.

Before you create a Fargate profile, you must create an IAM role with the AmazonEKSFargatePodExecutionRolePolicy.

Check for a correctly configured existing Pod execution role

You can use the following procedure to check and see if your account already has a correctly configured Amazon EKS Pod execution role. To avoid a confused deputy security problem, it's important that the role restricts access based on `SourceArn`. You can modify the execution role as needed to include support for Fargate profiles on other clusters.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Roles**.
3. On the **Roles** page, search the list of roles for **AmazonEKSFargatePodExecutionRole**. If the role doesn't exist, see the section called "Creating the Amazon EKS Pod execution role" to create the role. If the role does exist, choose the role.
4. On the **AmazonEKSFargatePodExecutionRole** page, do the following:
   a. Choose **Permissions**.

b. Ensure that the **AmazonEKSFargatePodExecutionRolePolicy** Amazon managed policy is attached to the role.

c. Choose **Trust relationships**.

d. Choose **Edit trust policy**.

5. On the **Edit trust policy** page, verify that the trust relationship contains the following policy and has a line for Fargate profiles on your cluster. If so, choose **Cancel**.

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

If the policy matches but doesn't have a line specifying the Fargate profiles on your cluster, you can add the following line at the top of the `ArnLike` object. Replace *`region-code`* with the AWS Region that your cluster is in, *`111122223333`* with your account ID, and *`my-cluster`* with the name of your cluster.

```
"aws:SourceArn": "arn:aws:eks:region-code:111122223333:fargateprofile/my-cluster/*",
```

If the policy doesn't match, copy the full previous policy into the form and choose **Update policy**. Replace *`region-code`* with the AWS Region that your cluster is in. If you want to use the same role in all AWS Regions in your account, replace *`region-code`* with `*`. Replace *`111122223333`* with your account ID and *`my-cluster`* with the name of your cluster. If you want to use the same role for all clusters in your account, replace *`my-cluster`* with `*`.

Creating the Amazon EKS Pod execution role

If you don't already have the Amazon EKS Pod execution role for your cluster, you can use the AWS Management Console or the AWS CLI to create it.

AWS Management Console

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

b. In the left navigation pane, choose **Roles**.

c. On the **Roles** page, choose **Create role**.

d. On the **Select trusted entity** page, do the following:

   i. In the **Trusted entity type** section, choose **AWS service**.

   ii. From the **Use cases for other AWS services** dropdown list, choose **EKS**.

   iii. Choose **EKS - Fargate Pod**.

   iv. Choose **Next**.

e. On the **Add permissions** page, choose **Next**.

f. On the **Name, review, and create** page, do the following:

   i. For **Role name**, enter a unique name for your role, such as `AmazonEKSFargatePodExecutionRole`.

   ii. Under **Add tags (Optional)**, add metadata to the role by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM resources in the *IAM User Guide*.

   iii. Choose **Create role**.

g. On the **Roles** page, search the list of roles for **AmazonEKSFargatePodExecutionRole**. Choose the role.

h. On the **AmazonEKSFargatePodExecutionRole** page, do the following:

   i. Choose **Trust relationships**.

   ii. Choose **Edit trust policy**.

i. On the **Edit trust policy** page, do the following:

   i. Copy and paste the following contents into the **Edit trust policy** form. Replace *`region-code`* with the AWS Region that your cluster is in. If you want to use the same role in all AWS Regions in your account, replace *`region-code`* with `*`. Replace *`111122223333`* with your account ID and *`my-cluster`* with the name of your cluster. If you want to use the same role for all clusters in your account, replace *`my-cluster`* with `*`.

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

ii. Choose **Update policy**.

AWS CLI

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

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

```
        "Service": "eks-fargate-pods.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. Create a Pod execution IAM role.

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

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

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

Amazon EKS connector IAM role

You can connect Kubernetes clusters to view them in your AWS Management Console. To connect to a Kubernetes cluster, create an IAM role.

Check for an existing EKS connector role

You can use the following procedure to check and see if your account already has the Amazon EKS connector role.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Roles**.
3. Search the list of roles for `AmazonEKSConnectorAgentRole`. If a role that includes `AmazonEKSConnectorAgentRole` doesn't exist, then see the section called "Creating the Amazon EKS connector agent role" to create the role. If a role that includes `AmazonEKSConnectorAgentRole` does exist, then select the role to view the attached policies.
4. Choose **Permissions**.
5. Ensure that the **AmazonEKSConnectorAgentPolicy** managed policy is attached to the role. If the policy is attached, your Amazon EKS connector role is properly configured.
6. Choose **Trust relationships**, and then choose **Edit trust policy**.

7. Verify that the trust relationship contains the following policy. If the trust relationship matches the following policy, choose **Cancel**. If the trust relationship doesn't match, copy the policy into the **Edit trust policy** window and choose **Update policy**.

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

Creating the Amazon EKS connector agent role

You can use the AWS Management Console or AWS CloudFormation to create the connector agent role.

AWS CLI

a. Create a file named `eks-connector-agent-trust-policy.json` that contains the following JSON to use for the IAM role.

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

```
}
```

b. Create a file named `eks-connector-agent-policy.json` that contains the following JSON to use for the IAM role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SsmControlChannel",
            "Effect": "Allow",
            "Action": [
                "ssmmessages:CreateControlChannel"
            ],
            "Resource": "arn:aws:eks:*:*:cluster/*"
        },
        {
            "Sid": "ssmDataplaneOperations",
            "Effect": "Allow",
            "Action": [
                "ssmmessages:CreateDataChannel",
                "ssmmessages:OpenDataChannel",
                "ssmmessages:OpenControlChannel"
            ],
            "Resource": "*"
        }
    ]
}
```

c. Create the Amazon EKS Connector agent role using the trust policy and policy you created in the previous list items.

```
aws iam create-role \
     --role-name AmazonEKSConnectorAgentRole \
     --assume-role-policy-document file://eks-connector-agent-trust-policy.json
```

d. Attach the policy to your Amazon EKS Connector agent role.

```
aws iam put-role-policy \
     --role-name AmazonEKSConnectorAgentRole \
     --policy-name AmazonEKSConnectorAgentPolicy \
     --policy-document file://eks-connector-agent-policy.json
```

AWS CloudFormation

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

> **Note**
>
> This template also creates the service-linked role that would otherwise be created when the `registerCluster` API is called. See the section called "Cluster connector role" for details.

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Provisions necessary resources needed to register clusters in EKS'
Parameters: {}
Resources:
  EKSConnectorSLR:
    Type: AWS::IAM::ServiceLinkedRole
    Properties:
      AWSServiceName: eks-connector.amazonaws.com

  EKSConnectorAgentRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Action: [ 'sts:AssumeRole' ]
            Principal:
              Service: 'ssm.amazonaws.com'

  EKSConnectorAgentPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyName: EKSConnectorAgentPolicy
      Roles:
        - {Ref: 'EKSConnectorAgentRole'}
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: 'Allow'
```

```
            Action: [ 'ssmmessages:CreateControlChannel' ]
            Resource:
            - Fn::Sub: 'arn:${AWS::Partition}:eks:*:*:cluster/*'
          - Effect: 'Allow'
            Action: [ 'ssmmessages:CreateDataChannel',
 'ssmmessages:OpenDataChannel', 'ssmmessages:OpenControlChannel' ]
            Resource: "*"
Outputs:
  EKSConnectorAgentRoleArn:
    Description: The agent role that EKS connector uses to communicate with AWS
 services.
    Value: !GetAtt EKSConnectorAgentRole.Arn
```

b. Open the AWS CloudFormation console.
c. Choose **Create stack** with new resources (standard).
d. For **Specify template**, select **Upload a template file**, and then choose **Choose file**.
e. Choose the file you created earlier, and then choose **Next**.
f. For **Stack name**, enter a name for your role, such as `eksConnectorAgentRole`, and then choose **Next**.
g. On the **Configure stack options** page, choose **Next**.
h. On the **Review** page, review your information, acknowledge that the stack might create IAM resources, and then choose **Create stack**.

AWS managed policies for Amazon Elastic Kubernetes Service

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

AWS managed policy: AmazonEKS_CNI_Policy

You can attach the `AmazonEKS_CNI_Policy` to your IAM entities. Before you create an Amazon EC2 node group, this policy must be attached to either the node IAM role, or to an IAM role that's used specifically by the Amazon VPC CNI plugin for Kubernetes. This is so that it can perform actions on your behalf. We recommend that you attach the policy to a role that's used only by the plugin. For more information, see the section called "Amazon VPC CNI" and the section called "Configure for IRSA".

Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks:

- **`ec2:*NetworkInterface` and `ec2:*PrivateIpAddresses`** – Allows the Amazon VPC CNI plugin to perform actions such as provisioning Elastic Network Interfaces and IP addresses for Pods to provide networking for applications that run in Amazon EKS.
- **`ec2` read actions** – Allows the Amazon VPC CNI plugin to perform actions such as describe instances and subnets to see the amount of free IP addresses in your Amazon VPC subnets. The VPC CNI can use the free IP addresses in each subnet to pick the subnets with the most free IP addresses to use when creating an elastic network interface.

To view the latest version of the JSON policy document, see AmazonEKS_CNI_Policy in the AWS Managed Policy Reference Guide.

AWS managed policy: AmazonEKSClusterPolicy

You can attach `AmazonEKSClusterPolicy` to your IAM entities. Before creating a cluster, you must have a cluster IAM role with this policy attached. Kubernetes clusters that are managed by Amazon EKS make calls to other AWS services on your behalf. They do this to manage the resources that you use with the service.

This policy includes the following permissions that allow Amazon EKS to complete the following tasks:

- **autoscaling** – Read and update the configuration of an Auto Scaling group. These permissions aren't used by Amazon EKS but remain in the policy for backwards compatibility.
- **ec2** – Work with volumes and network resources that are associated to Amazon EC2 nodes. This is required so that the Kubernetes control plane can join instances to a cluster and dynamically provision and manage Amazon EBS volumes that are requested by Kubernetes persistent volumes.
- **elasticloadbalancing** – Work with Elastic Load Balancers and add nodes to them as targets. This is required so that the Kubernetes control plane can dynamically provision Elastic Load Balancers requested by Kubernetes services.
- **iam** – Create a service-linked role. This is required so that the Kubernetes control plane can dynamically provision Elastic Load Balancers that are requested by Kubernetes services.
- **kms** – Read a key from AWS KMS. This is required for the Kubernetes control plane to support secrets encryption of Kubernetes secrets stored in `etcd`.

To view the latest version of the JSON policy document, see AmazonEKSClusterPolicy in the AWS Managed Policy Reference Guide.

AWS managed policy: AmazonEKSFargatePodExecutionRolePolicy

You can attach `AmazonEKSFargatePodExecutionRolePolicy` to your IAM entities. Before you can create a Fargate profile, you must create a Fargate Pod execution role and attach this policy to it. For more information, see the section called "Step 2: Create a Fargate Pod execution role" and the section called "Define profiles".

This policy grants the role the permissions that provide access to other AWS service resources that are required to run Amazon EKS Pods on Fargate.

Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks:

- **ecr** – Allows Pods that are running on Fargate to pull container images that are stored in Amazon ECR.

To view the latest version of the JSON policy document, see AmazonEKSFargatePodExecutionRolePolicy in the AWS Managed Policy Reference Guide.

AWS managed policy: AmazonEKSForFargateServiceRolePolicy

You can't attach `AmazonEKSForFargateServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role that allows Amazon EKS to perform actions on your behalf. For more information, see `AWSServiceRoleforAmazonEKSForFargate`.

This policy grants necessary permissions to Amazon EKS to run Fargate tasks. The policy is only used if you have Fargate nodes.

Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks.

- **ec2** – Create and delete Elastic Network Interfaces and describe Elastic Network Interfaces and resources. This is required so that the Amazon EKS Fargate service can configure the VPC networking that's required for Fargate Pods.

To view the latest version of the JSON policy document, see AmazonEKSForFargateServiceRolePolicy in the AWS Managed Policy Reference Guide.

AWS managed policy: AmazonEKSComputePolicy

You can attach `AmazonEKSComputePolicy` to your IAM entities. You may attach this policy to your cluster IAM role to expand the resources EKS can manage in your account.

This policy grants the permissions required for Amazon EKS to create and manage EC2 instances for the EKS cluster, as well as the necessary IAM permissions to configure EC2.

Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks:

- **ec2 Permissions**:
  - `ec2:CreateFleet` and `ec2:RunInstances` - Allows creating EC2 instances and using specific EC2 resources (images, security groups, subnets) for EKS cluster nodes.
  - `ec2:CreateLaunchTemplate` - Allows creating EC2 launch templates for EKS cluster nodes.
  - The policy also includes conditions to restrict the use of these EC2 permissions to resources tagged with the EKS cluster name and other relevant tags.

  - `ec2:CreateTags` - Allows adding tags to EC2 resources created by the `CreateFleet`, `RunInstances`, and `CreateLaunchTemplate` actions.
- **`iam` Permissions**:
  - `iam:AddRoleToInstanceProfile` - Allows adding an IAM role to the EKS compute instance profile.
  - `iam:PassRole` - Allows passing the necessary IAM roles to the EC2 service.

To view the latest version of the JSON policy document, see AmazonEKSComputePolicy in the AWS Managed Policy Reference Guide.

AWS managed policy: AmazonEKSNetworkingPolicy

You can attach `AmazonEKSNetworkingPolicy` to your IAM entities. You may attach this policy to your cluster IAM role to expand the resources EKS can manage in your account.

This policy is designed to grant the necessary permissions for Amazon EKS to create and manage network interfaces for the EKS cluster, allowing the control plane and worker nodes to communicate and function properly.

Permissions details

This policy grants the following permissions to allow Amazon EKS to manage network interfaces for the cluster:

- **ec2 Network Interface Permissions**:
  - `ec2:CreateNetworkInterface` - Allows creating EC2 network interfaces.
  - The policy includes conditions to restrict the use of this permission to network interfaces tagged with the EKS cluster name and the Kubernetes CNI node name.
  - `ec2:CreateTags` - Allows adding tags to the network interfaces created by the `CreateNetworkInterface` action.
- **ec2 Network Interface Management Permissions**:
  - `ec2:AttachNetworkInterface`, `ec2:DetachNetworkInterface` - Allows attaching and detaching network interfaces to EC2 instances.
  - `ec2:UnassignPrivateIpAddresses`, `ec2:UnassignIpv6Addresses`, `ec2:AssignPrivateIpAddresses`, `ec2:AssignIpv6Addresses` - Allows managing the IP address assignments of the network interfaces.

- These permissions are restricted to network interfaces tagged with the EKS cluster name.

To view the latest version of the JSON policy document, see AmazonEKSNetworkingPolicy in the AWS Managed Policy Reference Guide.

AWS managed policy: AmazonEKSBlockStoragePolicy

You can attach `AmazonEKSBlockStoragePolicy` to your IAM entities. You may attach this policy to your cluster IAM role to expand the resources EKS can manage in your account.

This policy grants the necessary permissions for Amazon EKS to create, manage, and maintain EC2 volumes and snapshots for the EKS cluster, enabling the control plane and worker nodes to provision and use persistent storage as required by Kubernetes workloads.

Permissions details

This IAM policy grants the following permissions to allow Amazon EKS to manage EC2 volumes and snapshots:

- **ec2 Volume Management Permissions**:
  - `ec2:AttachVolume`, `ec2:DetachVolume`, `ec2:ModifyVolume`, `ec2:EnableFastSnapshotRestores` - Allows attaching, detaching, modifying, and enabling fast snapshot restores for EC2 volumes.
  - These permissions are restricted to volumes tagged with the EKS cluster name.
  - `ec2:CreateTags` - Allows adding tags to the EC2 volumes and snapshots created by the `CreateVolume` and `CreateSnapshot` actions.
- **ec2 Volume Creation Permissions**:
  - `ec2:CreateVolume` - Allows creating new EC2 volumes.
  - The policy includes conditions to restrict the use of this permission to volumes tagged with the EKS cluster name and other relevant tags.
  - `ec2:CreateSnapshot` - Allows creating new EC2 volume snapshots.
  - The policy includes conditions to restrict the use of this permission to snapshots tagged with the EKS cluster name and other relevant tags.

To view the latest version of the JSON policy document, see AmazonEKSBlockStoragePolicy in the AWS Managed Policy Reference Guide.

AWS managed policy: AmazonEKSLoadBalancingPolicy

You can attach `AmazonEKSLoadBalancingPolicy` to your IAM entities. You may attach this policy to your cluster IAM role to expand the resources EKS can manage in your account.

This IAM policy grants the necessary permissions for Amazon EKS to work with various AWS services to manage Elastic Load Balancers (ELBs) and related resources.

Permissions details

The key permissions granted by this policy are:

- **`elasticloadbalancing`** : Allows creating, modifying, and managing Elastic Load Balancers and Target Groups. This includes permissions to create, update, and delete load balancers, target groups, listeners, and rules.
- **`ec2`** : Allows creating and managing security groups, which are required for the Kubernetes control plane to join instances to a cluster and manage Amazon EBS volumes. Also allows describing and listing EC2 resources such as instances, VPCs, Subnets, Security Groups, and other networking resources.
- **`iam`** : Allows creating a service-linked role for Elastic Load Balancing, which is required for the Kubernetes control plane to dynamically provision ELBs.
- **`kms`** : Allows reading a key from AWS KMS, which is required for the Kubernetes control plane to support encryption of Kubernetes secrets stored in etcd.
- **`wafv2`** and **`shield`** : Allows associating and disassociating Web ACLs and creating/deleting AWS Shield protections for the Elastic Load Balancers.
- **`cognito-idp`** , **`acm`** , and **`elasticloadbalancing`** : Grants permissions to describe user pool clients, list and describe certificates, and describe target groups, which are required for the Kubernetes control plane to manage the Elastic Load Balancers.

The policy also includes several condition checks to ensure that the permissions are scoped to the specific EKS cluster being managed, using the `eks:eks-cluster-name` tag.

To view the latest version of the JSON policy document, see AmazonEKSLoadBalancingPolicy in the AWS Managed Policy Reference Guide.

AWS managed policy: AmazonEKSServicePolicy

You can attach `AmazonEKSServicePolicy` to your IAM entities. Clusters that were created before April 16, 2020, required you to create an IAM role and attach this policy to it. Clusters that were created on or after April 16, 2020, don't require you to create a role and don't require you to assign this policy. When you create a cluster using an IAM principal that has the `iam:CreateServiceLinkedRole` permission, the AWSServiceRoleforAmazonEKS service-linked role is automatically created for you. The service-linked role has the managed policy: AmazonEKSServiceRolePolicy attached to it.

This policy allows Amazon EKS to create and manage the necessary resources to operate Amazon EKS clusters.

Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks.

- **`eks`** – Update the Kubernetes version of your cluster after you initiate an update. This permission isn't used by Amazon EKS but remains in the policy for backwards compatibility.
- **`ec2`** – Work with Elastic Network Interfaces and other network resources and tags. This is required by Amazon EKS to configure networking that facilitates communication between nodes and the Kubernetes control plane. Read information about security groups. Update tags on security groups.
- **`route53`** – Associate a VPC with a hosted zone. This is required by Amazon EKS to enable private endpoint networking for your Kubernetes cluster API server.
- **`logs`** – Log events. This is required so that Amazon EKS can ship Kubernetes control plane logs to CloudWatch.
- **`iam`** – Create a service-linked role. This is required so that Amazon EKS can create the the section called "Service-linked role permissions for Amazon EKS" service-linked role on your behalf.

To view the latest version of the JSON policy document, see AmazonEKSServicePolicy in the AWS Managed Policy Reference Guide.

AWS managed policy: AmazonEKSServiceRolePolicy

You can't attach `AmazonEKSServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role that allows Amazon EKS to perform actions on your behalf. For more information, see the section called "Service-linked role permissions for Amazon EKS". When you create a cluster using an IAM principal that has the `iam:CreateServiceLinkedRole` permission, the AWSServiceRoleforAmazonEKS service-linked role is automatically created for you and this policy is attached to it.

This policy allows the service-linked role to call AWS services on your behalf.

Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks.

- **`ec2`** – Create and describe Elastic Network Interfaces and Amazon EC2 instances, the cluster security group, and VPC that are required to create a cluster. For more information, see the section called "Security group requirements". Read information about security groups. Update tags on security groups.
- **`iam`** – List all of the managed policies that attached to an IAM role. This is required so that Amazon EKS can list and validate all managed policies and permissions required to create a cluster.
- **Associate a VPC with a hosted zone** – This is required by Amazon EKS to enable private endpoint networking for your Kubernetes cluster API server.
- **Log event** – This is required so that Amazon EKS can ship Kubernetes control plane logs to CloudWatch.
- **Put metric** – This is required so that Amazon EKS can ship Kubernetes control plane logs to CloudWatch.
- **`eks`** - Manage cluster access entries and policies, allowing fine-grained control over who can access EKS resources and what actions they can perform. This includes associating standard access policies for compute, networking, load balancing, and storage operations.
- **`elasticloadbalancing`** - Create, manage, and delete load balancers and their components (listeners, target groups, certificates) that are associated with EKS clusters. View load balancer attributes and health status.

- **`events`** - Create and manage EventBridge rules for monitoring EC2 and AWS Health events related to EKS clusters, enabling automated responses to infrastructure changes and health alerts.
- **`iam`** - Manage EC2 instance profiles with the "eks" prefix, including creation, deletion, and role association, which is necessary for EKS node management.
- **`pricing`** & **`shield`** - Access AWS pricing information and Shield protection status, enabling cost management and advanced security features for EKS resources.
- **Resource cleanup** - Safely delete EKS-tagged resources including volumes, snapshots, launch templates, and network interfaces during cluster cleanup operations.

To view the latest version of the JSON policy document, see AmazonEKSServiceRolePolicy in the AWS Managed Policy Reference Guide.

AWS managed policy: AmazonEKSVPCResourceController

You can attach the `AmazonEKSVPCResourceController` policy to your IAM identities. If you're using security groups for Pods, you must attach this policy to your Amazon EKS cluster IAM role to perform actions on your behalf.

This policy grants the cluster role permissions to manage Elastic Network Interfaces and IP addresses for nodes.

Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks:

- **`ec2`** – Manage Elastic Network Interfaces and IP addresses to support Pod security groups and Windows nodes.

To view the latest version of the JSON policy document, see AmazonEKSVPCResourceController in the AWS Managed Policy Reference Guide.

AWS managed policy: AmazonEKSWorkerNodePolicy

You can attach the `AmazonEKSWorkerNodePolicy` to your IAM entities. You must attach this policy to a node IAM role that you specify when you create Amazon EC2 nodes that allow Amazon

EKS to perform actions on your behalf. If you create a node group using `eksctl`, it creates the node IAM role and attaches this policy to the role automatically.

This policy grants Amazon EKS Amazon EC2 nodes permissions to connect to Amazon EKS clusters.

Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks:

- **`ec2`** – Read instance volume and network information. This is required so that Kubernetes nodes can describe information about Amazon EC2 resources that are required for the node to join the Amazon EKS cluster.
- **`eks`** – Optionally describe the cluster as part of node bootstrapping.
- **`eks-auth:AssumeRoleForPodIdentity`** – Allow retrieving credentials for EKS workloads on the node. This is required for EKS Pod Identity to function properly.

To view the latest version of the JSON policy document, see [AmazonEKSWorkerNodePolicy] in the AWS Managed Policy Reference Guide.

AWS managed policy: AmazonEKSWorkerNodeMinimalPolicy

You can attach the AmazonEKSWorkerNodeMinimalPolicy to your IAM entities. You may attach this policy to a node IAM role that you specify when you create Amazon EC2 nodes that allow Amazon EKS to perform actions on your behalf.

This policy grants Amazon EKS Amazon EC2 nodes permissions to connect to Amazon EKS clusters. This policy has fewer permissions compared to AmazonEKSWorkerNodePolicy.

Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks:

- `eks-auth:AssumeRoleForPodIdentity` - Allow retrieving credentials for EKS workloads on the node. This is required for EKS Pod Identity to function properly.

To view the latest version of the JSON policy document, see [AmazonEKSWorkerNodePolicy] in the AWS Managed Policy Reference Guide.

AWS managed policy: AWSServiceRoleForAmazonEKSNodegroup

You can't attach `AWSServiceRoleForAmazonEKSNodegroup` to your IAM entities. This policy is attached to a service-linked role that allows Amazon EKS to perform actions on your behalf. For more information, see the section called "Service-linked role permissions for Amazon EKS".

This policy grants the `AWSServiceRoleForAmazonEKSNodegroup` role permissions that allow it to create and manage Amazon EC2 node groups in your account.

Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks:

- **`ec2`** – Work with security groups, tags, capacity reservations, and launch templates. This is required for Amazon EKS managed node groups to enable remote access configuration and to describe capacity reservations that can be used in managed node groups. Additionally, Amazon EKS managed node groups create a launch template on your behalf. This is to configure the Amazon EC2 Auto Scaling group that backs each managed node group.
- **`iam`** – Create a service-linked role and pass a role. This is required by Amazon EKS managed node groups to manage instance profiles for the role being passed when creating a managed node group. This instance profile is used by Amazon EC2 instances launched as part of a managed node group. Amazon EKS needs to create service-linked roles for other services such as Amazon EC2 Auto Scaling groups. These permissions are used in the creation of a managed node group.
- **`autoscaling`** – Work with security Auto Scaling groups. This is required by Amazon EKS managed node groups to manage the Amazon EC2 Auto Scaling group that backs each managed node group. It's also used to support functionality such as evicting Pods when nodes are terminated or recycled during node group updates.

To view the latest version of the JSON policy document, see AWSServiceRoleForAmazonEKSNodegroup in the AWS Managed Policy Reference Guide.

AWS managed policy: AmazonEBSCSIDriverPolicy

The `AmazonEBSCSIDriverPolicy` policy allows the Amazon EBS Container Storage Interface (CSI) driver to create, modify, attach, detach, and delete volumes on your behalf. This includes modifying tags on existing volumes and enabling Fast Snapshot Restore (FSR) on EBS volumes. It also grants the EBS CSI driver permissions to create, restore, and delete snapshots, and to list your instances, volumes, and snapshots.

To view the latest version of the JSON policy document, see AmazonEBSCSIDriverServiceRolePolicy in the AWS Managed Policy Reference Guide.

AWS managed policy: AmazonEFSCSIDriverPolicy

The `AmazonEFSCSIDriverPolicy` policy allows the Amazon EFS Container Storage Interface (CSI) to create and delete access points on your behalf. It also grants the Amazon EFS CSI driver permissions to list your access points file systems, mount targets, and Amazon EC2 availability zones.

To view the latest version of the JSON policy document, see AmazonEFSCSIDriverServiceRolePolicy in the AWS Managed Policy Reference Guide.

AWS managed policy: AmazonEKSLocalOutpostClusterPolicy

You can attach this policy to IAM entities. Before creating a local cluster, you must attach this policy to your cluster role. Kubernetes clusters that are managed by Amazon EKS make calls to other AWS services on your behalf. They do this to manage the resources that you use with the service.

The `AmazonEKSLocalOutpostClusterPolicy` includes the following permissions:

- **`ec2` read actions** – Allows control plane instances to describe Availability Zone, route table, instance, and network interface properties. Required permissions for Amazon EC2 instances to successfully join the cluster as control plane instances.
- **`ssm`** – Allows Amazon EC2 Systems Manager connection to the control plane instance, which is used by Amazon EKS to communicate and manage the local cluster in your account.
- **`logs`** – Allows instances to push logs to Amazon CloudWatch.
- **`secretsmanager`** – Allows instances to get and delete bootstrap data for the control plane instances securely from AWS Secrets Manager.
- **`ecr`** – Allows Pods and containers that are running on the control plane instances to pull container images that are stored in Amazon Elastic Container Registry.

To view the latest version of the JSON policy document, see AmazonEKSLocalOutpostClusterPolicy in the AWS Managed Policy Reference Guide.

AWS managed policy: AmazonEKSLocalOutpostServiceRolePolicy

You can't attach this policy to your IAM entities. When you create a cluster using an IAM principal that has the `iam:CreateServiceLinkedRole` permission, Amazon EKS automatically creates the AWSServiceRoleforAmazonEKSLocalOutpost service-linked role for you and attaches this policy to it. This policy allows the service-linked role to call AWS services on your behalf for local clusters.

The `AmazonEKSLocalOutpostServiceRolePolicy` includes the following permissions:

- **`ec2`** – Allows Amazon EKS to work with security, network, and other resources to successfully launch and manage control plane instances in your account.
- **`ssm`** – Allows Amazon EC2 Systems Manager connection to the control plane instances, which is used by Amazon EKS to communicate and manage the local cluster in your account.
- **`iam`** – Allows Amazon EKS to manage the instance profile associated with the control plane instances.
- **`secretsmanager`** - Allows Amazon EKS to put bootstrap data for the control plane instances into AWS Secrets Manager so it can be securely referenced during instance bootstrapping.
- **`outposts`** – Allows Amazon EKS to get Outpost information from your account to successfully launch a local cluster in an Outpost.

To view the latest version of the JSON policy document, see AmazonEKSLocalOutpostServiceRolePolicy in the AWS Managed Policy Reference Guide.

Amazon EKS updates to AWS managed policies

View details about updates to AWS managed policies for Amazon EKS since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon EKS Document history page.

Change	Description	Date
Added permissions to AmazonEBSCSIDriverPolicy.	Added a new statement authorizing the EBS CSI Driver to restore all snapshots. This was previously allowed by the existing policy but a new explicit statement is	January 13, 2025

Change	Description	Date
	required due to a change in the handling of IAM for `CreateVolume` . Added the ability for the EBS CSI Driver to modify tags on existing volumes. The EBS CSI Driver can modify tags of existing volumes via a parameters in Kubernetes VolumeAttributesClasses. Added the ability for the EBS CSI Driver to enable Fast Snapshot Restore (FSR) on EBS volumes. The EBS CSI Driver can enable FSR on new volumes via parameters in Kubernetes storage classes.	
Added permissions to the section called “AWS managed policy: AmazonEKSLoadBalancingPolicy”.	Updated `AmazonEKSLoadBalancingPolicy` to allow listing and describing networking and IP address resources.	December 26, 2024
Added permissions to the section called “AWS managed policy: AWSServiceRoleForAmazonEKSNodegroup”.	Updated `AWSServiceRoleForAmazonEKSNodegroup` for compatibility with China regions.	November 22, 2024

Change	Description	Date
Added permissions to the section called "AWS managed policy: AmazonEKSLocalOutpostClusterPolicy"	Added `ec2:DescribeAvailabilityZones` permission to `AmazonEKSLocalOutpostClusterPolicy` so the AWS Cloud Controller Manager on the cluster control plane can identify the Availability Zone that each node is in.	November 21, 2024
Added permissions to the section called "AWS managed policy: AWSServiceRoleForAmazonEKSNodegroup".	Updated `AWSServiceRoleForAmazonEKSNodegroup` policy to allow `ec2:RebootInstances` for instances created by Amazon EKS managed node groups. Restricted the `ec2:CreateTags` permissions for Amazon EC2 resources.	November 20, 2024
Added permissions to the section called "AWS managed policy: AmazonEKSServiceRolePolicy".	EKS updated AWS managed policy `AmazonEKSServiceRolePolicy`. Added permissions for EKS access policies, load balancer management, and automated cluster resource cleanup.	November 16, 2024
Introduced the section called "AWS managed policy: AmazonEKSComputePolicy".	EKS updated AWS managed policy `AmazonEKSComputePolicy`. Updated resource permissions for the `iam:AddRoleToInstanceProfile` action.	November 7, 2024

Change	Description	Date
Introduced the section called "AWS managed policy: AmazonEKSComputePolicy".	AWS introduced the `AmazonEKSComputePolicy` .	November 1, 2024
Added permissions to `AmazonEKSClusterPolicy`	Added `ec2:DescribeInstanceTopology` permission to allow Amazon EKS to attach topology information to the node as labels.	November 1, 2024
Introduced the section called "AWS managed policy: AmazonEKSBlockStoragePolicy".	AWS introduced the `AmazonEKSBlockStoragePolicy` .	October 30, 2024
Introduced the section called "AWS managed policy: AmazonEKSLoadBalancingPolicy".	AWS introduced the `AmazonEKSLoadBalancingPolicy` .	October 30, 2024
Added permissions to AmazonEKSServiceRolePolicy.	Added `cloudwatch:PutMetricData` permissions to allow Amazon EKS to publish metrics to Amazon CloudWatch.	October 29, 2024
Introduced the section called "AWS managed policy: AmazonEKSNetworkingPolicy".	AWS introduced the `AmazonEKSNetworkingPolicy` .	October 28, 2024

Change	Description	Date
Added permissions to AmazonEKSServicePolicy and AmazonEKSServiceRolePolicy	Added ec2:GetSecurityGroupsForVpc and associated tag permissions to allow EKS to read security group information and update related tags.	October 10, 2024
Introduced AmazonEKSWorkerNodeMinimalPolicy.	AWS introduced the AmazonEKSWorkerNodeMinimalPolicy .	October 3, 2024
Added permissions to AWSServiceRoleForAmazonEKSNodegroup.	Added autoscaling:ResumeProcesses and autoscaling:SuspendProcesses permissions to allow Amazon EKS to suspend and resume AZRebalance in Amazon EKS-managed Auto Scaling groups.	August 21, 2024
Added permissions to AWSServiceRoleForAmazonEKSNodegroup.	Added ec2:DescribeCapacityReservations permission to allow Amazon EKS to describe capacity reservation in user's account. Added autoscaling:PutScheduledUpdateGroupAction permission to enable setting scheduled scaling on CAPACITY_BLOCK node groups.	June 27, 2024

Change	Description	Date
AmazonEKS_CNI_Policy – Update to an existing policy	Amazon EKS added new `ec2:DescribeSubnets` permissions to allow the Amazon VPC CNI plugin for Kubernetes to see the amount of free IP addresses in your Amazon VPC subnets. The VPC CNI can use the free IP addresses in each subnet to pick the subnets with the most free IP addresses to use when creating an elastic network interface.	March 4, 2024
AmazonEKSWorkerNodePolicy – Update to an existing policy	Amazon EKS added new permissions to allow EKS Pod Identities. The Amazon EKS Pod Identity Agent uses the node role.	November 26, 2023
Introduced AmazonEFSCSIDriverPolicy.	AWS introduced the `AmazonEFSCSIDriverPolicy` .	July 26, 2023
Added permissions to AmazonEKSClusterPolicy.	Added `ec2:DescribeAvailabilityZones` permission to allow Amazon EKS to get the AZ details during subnet auto-discovery while creating load balancers.	February 7, 2023

Change	Description	Date
Updated policy conditions in AmazonEBSCSIDriverPolicy.	Removed invalid policy conditions with wildcard characters in the `StringLike` key field. Also added a new condition `ec2:ResourceTag/kubernetes.io/created-for/pvc/name: "*"` to `ec2:DeleteVolume`, which allows the EBS CSI driver to delete volumes created by the in-tree plugin.	November 17, 2022
Added permissions to AmazonEKSLocalOutpostServiceRolePolicy.	Added `ec2:DescribeVPCAttribute`, `ec2:GetConsoleOutput` and `ec2:DescribeSecret` to allow better prerequisite validation and managed lifecycle control. Also added `ec2:DescribePlacementGroups` and `"arn:aws:ec2:*:*:placement-group/*"` to `ec2:RunInstances` to support placement control of the control plane Amazon EC2 instances on Outposts.	October 24, 2022

Change	Description	Date
Update Amazon Elastic Container Registry permissions in AmazonEKSLocalOutpostClusterPolicy.	Moved action `ecr:GetDownloadUrlForLayer` from all resource sections to a scoped section. Added resource `arn:aws:ecr:*:*:repository/eks/` . Removed resource `arn:aws:ecr:` . This resource is covered by the added `arn:aws:ecr:*:*:repository/eks/*` resource.	October 20, 2022
Added permissions to AmazonEKSLocalOutpostClusterPolicy.	Added the `arn:aws:ecr:*:*:repository/kubelet-config-updater` Amazon Elastic Container Registry repository so the cluster control plane instances can update some `kubelet` arguments.	August 31, 2022
Introduced AmazonEKSLocalOutpostClusterPolicy.	AWS introduced the `AmazonEKSLocalOutpostClusterPolicy` .	August 24, 2022
Introduced AmazonEKSLocalOutpostServiceRolePolicy.	AWS introduced the `AmazonEKSLocalOutpostServiceRolePolicy` .	August 23, 2022
Introduced AmazonEBSCSIDriverPolicy.	AWS introduced the `AmazonEBSCSIDriverPolicy` .	April 4, 2022

Change	Description	Date
Added permissions to AmazonEKSWorkerNodePolicy.	Added `ec2:DescribeInstanceTypes` to enable Amazon EKS-optimized AMIs that can auto discover instance level properties.	March 21, 2022
Added permissions to AWSServiceRoleForAmazonEKSNodegroup.	Added `autoscaling:EnableMetricsCollection` permission to allow Amazon EKS to enable metrics collection.	December 13, 2021
Added permissions to AmazonEKSClusterPolicy.	Added `ec2:DescribeAccountAttributes`, `ec2:DescribeAddresses`, and `ec2:DescribeInternetGateways` permissions to allow Amazon EKS to create a service-linked role for a Network Load Balancer.	June 17, 2021
Amazon EKS started tracking changes.	Amazon EKS started tracking changes for its AWS managed policies.	June 17, 2021

Troubleshooting IAM

This topic covers some common errors that you may see while using Amazon EKS with IAM and how to work around them.

AccessDeniedException

If you receive an AccessDeniedException when calling an AWS API operation, then the IAM principal credentials that you're using don't have the required permissions to make that call.

```
An error occurred (AccessDeniedException) when calling the DescribeCluster operation:
User: arn:aws:iam::111122223333:user/user_name is not authorized to perform:
eks:DescribeCluster on resource: arn:aws:eks:region:111122223333:cluster/my-cluster
```

In the previous example message, the user does not have permissions to call the Amazon EKS DescribeCluster API operation. To provide Amazon EKS admin permissions to an IAM principal, see the section called "Identity-based policies".

For more general information about IAM, see Controlling access using policies in the *IAM User Guide*.

Can't see Nodes on the Compute tab or anything on the Resources tab and you receive an error in the AWS Management Console

You may see a console error message that says Your current user or role does not have access to Kubernetes objects on this EKS cluster. Make sure that the IAM principal user that you're using the AWS Management Console with has the necessary permissions. For more information, see the section called "Required permissions".

aws-auth ConfigMap does not grant access to the cluster

The AWS IAM Authenticator doesn't permit a path in the role ARN used in the ConfigMap. Therefore, before you specify rolearn, remove the path. For example, change arn:aws:iam::*111122223333*:role/*team*/*developers*/*eks-admin* to arn:aws:iam::*111122223333*:role/*eks-admin* .

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the iam:PassRole action, your policies must be updated to allow you to pass a role to Amazon EKS.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon EKS. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: {arn-aws}iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my Amazon EKS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon EKS supports these features, see the section called "Amazon EKS and IAM".
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Pod containers receive the following error: `An error occurred (SignatureDoesNotMatch) when calling the GetCallerIdentity operation: Credential should be scoped to a valid region`

Your containers receive this error if your application is explicitly making requests to the AWS STS global endpoint (`https://sts.amazonaws`) and your Kubernetes service account is configured to use a regional endpoint. You can resolve the issue with one of the following options:

- Update your application code to remove explicit calls to the AWS STS global endpoint.
- Update your application code to make explicit calls to regional endpoints such as `https://sts.us-west-2.amazonaws.com` . Your application should have redundancy built in to pick a different AWS Region in the event of a failure of the service in the AWS Region. For more information, see Managing AWS STS in an AWS Region in the IAM User Guide.
- Configure your service accounts to use the global endpoint. All versions earlier than `1.22` used the global endpoint by default, but version `1.22` and later clusters use the regional endpoint by default. For more information, see the section called "STS endpoints".

Amazon EKS cluster IAM role

An Amazon EKS cluster IAM role is required for each cluster. Kubernetes clusters managed by Amazon EKS use this role to manage nodes and the legacy Cloud Provider uses this role to create load balancers with Elastic Load Balancing for services.

Before you can create Amazon EKS clusters, you must create an IAM role with either of the following IAM policies:

- AmazonEKSClusterPolicy
- A custom IAM policy. The minimal permissions that follow allows the Kubernetes cluster to manage nodes, but doesn't allow the legacy Cloud Provider to create load balancers with Elastic Load Balancing. Your custom IAM policy must have at least the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateTags"
```

```
      ],
      "Resource": "arn:aws:ec2:*:*:instance/*",
      "Condition": {
        "ForAnyValue:StringLike": {
          "aws:TagKeys": "kubernetes.io/cluster/*"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeInstanceTopology",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    }
  ]
}
```

Note

Prior to October 3, 2023, AmazonEKSClusterPolicy was required on the IAM role for each cluster.

Prior to April 16, 2020, AmazonEKSServicePolicy and AmazonEKSClusterPolicy was required and the suggested name for the role was `eksServiceRole`. With the `AWSServiceRoleForAmazonEKS` service-linked role, the AmazonEKSServicePolicy policy is no longer required for clusters created on or after April 16, 2020.

Check for an existing cluster role

You can use the following procedure to check and see if your account already has the Amazon EKS cluster role.

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

2. In the left navigation pane, choose **Roles**.
3. Search the list of roles for `eksClusterRole`. If a role that includes `eksClusterRole` doesn't exist, then see the section called "Creating the Amazon EKS cluster role" to create the role. If a role that includes `eksClusterRole` does exist, then select the role to view the attached policies.
4. Choose **Permissions**.
5. Ensure that the **AmazonEKSClusterPolicy** managed policy is attached to the role. If the policy is attached, your Amazon EKS cluster role is properly configured.
6. Choose **Trust relationships**, and then choose **Edit trust policy**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the following policy, choose **Cancel**. If the trust relationship doesn't match, copy the policy into the **Edit trust policy** window and choose **Update policy**.

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

Creating the Amazon EKS cluster role

You can use the AWS Management Console or the AWS CLI to create the cluster role.

AWS Management Console

a. Open the IAM console at https://console.aws.amazon.com/iam/.
b. Choose **Roles**, then **Create role**.
c. Under **Trusted entity type**, select **AWS service**.
d. From the **Use cases for other AWS services** dropdown list, choose **EKS**.
e. Choose **EKS - Cluster** for your use case, and then choose **Next**.

f. On the **Add permissions** tab, choose **Next**.

g. For **Role name**, enter a unique name for your role, such as `eksClusterRole`.

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

i. Choose **Create role**.

AWS CLI

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

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

b. Create the role. You can replace *`eksClusterRole`* with any name that you choose.

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

c. Attach the required IAM policy to the role.

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

Amazon EKS node IAM role

The Amazon EKS node `kubelet` daemon makes calls to AWS APIs on your behalf. Nodes receive permissions for these API calls through an IAM instance profile and associated policies. Before you can launch nodes and register them into a cluster, you must create an IAM role for those nodes to use when they are launched. This requirement applies to nodes launched with the Amazon

EKS optimized AMI provided by Amazon, or with any other node AMIs that you intend to use. Additionally, this requirement applies to both managed node groups and self-managed nodes.

> **Note**
>
> You can’t use the same role that is used to create any clusters.

Before you create nodes, you must create an IAM role with the following permissions:

- Permissions for the `kubelet` to describe Amazon EC2 resources in the VPC, such as provided by the AmazonEKSWorkerNodePolicy policy. This policy also provides the permissions for the Amazon EKS Pod Identity Agent.
- Permissions for the `kubelet` to use container images from Amazon Elastic Container Registry (Amazon ECR), such as provided by the AmazonEC2ContainerRegistryPullOnly policy. The permissions to use container images from Amazon Elastic Container Registry (Amazon ECR) are required because the built-in add-ons for networking run pods that use container images from Amazon ECR.
- (Optional) Permissions for the Amazon EKS Pod Identity Agent to use the `eks-auth:AssumeRoleForPodIdentity` action to retrieve credentials for pods. If you don’t use the AmazonEKSWorkerNodePolicy, then you must provide this permission in addition to the EC2 permissions to use EKS Pod Identity.
- (Optional) If you don’t use IRSA or EKS Pod Identity to give permissions to the VPC CNI pods, then you must provide permissions for the VPC CNI on the instance role. You can use either the `AmazonEKS_CNI_Policy` managed policy (if you created your cluster with the `IPv4` family) or an IPv6 policy that you create (if you created your cluster with the `IPv6` family). Rather than attaching the policy to this role however, we recommend that you attach the policy to a separate role used specifically for the Amazon VPC CNI add-on. For more information about creating a separate role for the Amazon VPC CNI add-on, see the section called “Configure for IRSA”.

> **Note**
>
> The Amazon EC2 node groups must have a different IAM role than the Fargate profile. For more information, see the section called “Pod execution IAM role”.

Check for an existing node role

You can use the following procedure to check and see if your account already has the Amazon EKS node role.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Roles**.
3. Search the list of roles for `eksNodeRole`, `AmazonEKSNodeRole`, or `NodeInstanceRole`. If a role with one of those names doesn't exist, then see the section called "Creating the Amazon EKS node IAM role" to create the role. If a role that contains `eksNodeRole`, `AmazonEKSNodeRole`, or `NodeInstanceRole` does exist, then select the role to view the attached policies.
4. Choose **Permissions**.
5. Ensure that the **AmazonEKSWorkerNodePolicy** and **AmazonEC2ContainerRegistryPullOnly** managed policies are attached to the role or a custom policy is attached with the minimal permissions.

> **Note**
>
> If the **AmazonEKS_CNI_Policy** policy is attached to the role, we recommend removing it and attaching it to an IAM role that is mapped to the `aws-node` Kubernetes service account instead. For more information, see the section called "Configure for IRSA".

6. Choose **Trust relationships**, and then choose **Edit trust policy**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the following policy, choose **Cancel**. If the trust relationship doesn't match, copy the policy into the **Edit trust policy** window and choose **Update policy**.

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

```
                        "ec2.amazonaws.com"
                    ]
                }
            }
        ]
}
```

Creating the Amazon EKS node IAM role

You can create the node IAM role with the AWS Management Console or the AWS CLI.

AWS Management Console

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

b. In the left navigation pane, choose **Roles**.

c. On the **Roles** page, choose **Create role**.

d. On the **Select trusted entity** page, do the following:

   i. In the **Trusted entity type** section, choose **AWS service**.

   ii. Under **Use case**, choose **EC2**.

   iii. Choose **Next**.

e. On the **Add permissions** page, attach a custom policy or do the following:

   i. In the **Filter policies** box, enter `AmazonEKSWorkerNodePolicy`.

   ii. Select the check box to the left of **AmazonEKSWorkerNodePolicy** in the search results.

   iii. Choose **Clear filters**.

   iv. In the **Filter policies** box, enter `AmazonEC2ContainerRegistryPullOnly`.

   v. Select the check box to the left of **AmazonEC2ContainerRegistryPullOnly** in the search results.

   Either the **AmazonEKS_CNI_Policy** managed policy, or an IPv6 policy that you create must also be attached to either this role or to a different role that's mapped to the `aws-node` Kubernetes service account. We recommend assigning the policy to the role associated to the Kubernetes service account instead of assigning it to this role. For more information, see the section called "Configure for IRSA".

   vi. Choose **Next**.

f. On the **Name, review, and create** page, do the following:

i. For **Role name**, enter a unique name for your role, such as `AmazonEKSNodeRole`.

ii. For **Description**, replace the current text with descriptive text such as `Amazon EKS - Node role`.

iii. Under **Add tags (Optional)**, add metadata to the role by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM resources in the *IAM User Guide*.

iv. Choose **Create role**.

AWS CLI

a. Run the following command to create the `node-role-trust-relationship.json` file.

```
cat >node-role-trust-relationship.json <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
                ]
            }
        }
    ]
}
EOF
```

b. Create the IAM role.

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

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

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

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly \
  --role-name AmazonEKSNodeRole
```

d. Attach one of the following IAM policies to the IAM role depending on which IP family you created your cluster with. The policy must be attached to this role or to a role associated to the Kubernetes aws-node service account that's used for the Amazon VPC CNI plugin for Kubernetes. We recommend assigning the policy to the role associated to the Kubernetes service account. To assign the policy to the role associated to the Kubernetes service account, see the section called "Configure for IRSA".

- IPv4

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

- IPv6

  A. Copy the following text and save it to a file named vpc-cni-ipv6-policy.json.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AssignIpv6Addresses",
                "ec2:DescribeInstances",
                "ec2:DescribeTags",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeInstanceTypes"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*"
            ]
        }
```

```
    ]
}
```

B. Create the IAM policy.

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-
document file://vpc-cni-ipv6-policy.json
```

C. Attach the IAM policy to the IAM role. Replace *111122223333* with your account ID.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy \
  --role-name AmazonEKSNodeRole
```

Amazon EKS Auto Mode cluster IAM role

An Amazon EKS cluster IAM role is required for each cluster. Kubernetes clusters managed by Amazon EKS use this role to automate routine tasks for storage, networking, and compute autoscaling.

Before you can create Amazon EKS clusters, you must create an IAM role with the policies required for EKS Auto Mode. You can either attach the suggested AWS IAM managed policies, or create custom polices with equivalent permissions.

- AmazonEKSComputePolicy
- AmazonEKSBlockStoragePolicy
- AmazonEKSLoadBalancingPolicy
- AmazonEKSNetworkingPolicy
- AmazonEKSClusterPolicy

Check for an existing cluster role

You can use the following procedure to check and see if your account already has the Amazon EKS cluster role.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Roles**.

3. Search the list of roles for `AmazonEKSAutoClusterRole`. If a role that includes `AmazonEKSAutoClusterRole` doesn't exist, then see the instructions in the next section to create the role. If a role that includes `AmazonEKSAutoClusterRole` does exist, then select the role to view the attached policies.
4. Choose **Permissions**.
5. Ensure that the **AmazonEKSClusterPolicy** managed policy is attached to the role. If the policy is attached, your Amazon EKS cluster role is properly configured.
6. Choose **Trust relationships**, and then choose **Edit trust policy**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the following policy, choose **Cancel**. If the trust relationship doesn't match, copy the policy into the **Edit trust policy** window and choose **Update policy**.

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

Note

AWS does not require the name `AmazonEKSAutoClusterRole` for this role.

Creating the Amazon EKS cluster role

You can use the AWS Management Console or the AWS CLI to create the cluster role.

AWS Management Console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Roles**, then **Create role**.
3. Under **Trusted entity type**, select **AWS service**.
4. From the **Use cases for other AWS services** dropdown list, choose **EKS**.
5. Choose **EKS - Cluster** for your use case, and then choose **Next**.
6. On the **Add permissions** tab, select the policies and then choose **Next**.
   - AmazonEKSComputePolicy
   - AmazonEKSBlockStoragePolicy
   - AmazonEKSLoadBalancingPolicy
   - AmazonEKSNetworkingPolicy
   - AmazonEKSClusterPolicy
7. For **Role name**, enter a unique name for your role, such as `AmazonEKSAutoClusterRole`.
8. For **Description**, enter descriptive text such as `Amazon EKS - Cluster role`.
9. Choose **Create role**.

AWS CLI

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

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

2. Create the role. You can replace *AmazonEKSAutoClusterRole* with any name that you choose.

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

3. Attach the required IAM policies to the role:

AmazonEKSClusterPolicy:

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

AmazonEKSComputePolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSComputePolicy
```

AmazonEKSBlockStoragePolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSBlockStoragePolicy
```

AmazonEKSLoadBalancingPolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSLoadBalancingPolicy
```

AmazonEKSNetworkingPolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSNetworkingPolicy
```

Amazon EKS Auto Mode node IAM role

Note

You can't use the same role that is used to create any clusters.

Before you create nodes, you must create an IAM role with the following policies, or equivalent permissions:

- AmazonEKSWorkerNodeMinimalPolicy
- AmazonEC2ContainerRegistryPullOnly

Check for an existing node role

You can use the following procedure to check and see if your account already has the Amazon EKS node role.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Roles**.
3. Search the list of roles for `AmazonEKSAutoNodeRole`. If a role with one of those names doesn't exist, then see instructions in the next section to create the role. If a role that contains `AmazonEKSAutoNodeRole` does exist, then select the role to view the attached policies.
4. Choose **Permissions**.
5. Ensure that the required policies above are attached, or equivalent custom policies.
6. Choose **Trust relationships**, and then choose **Edit trust policy**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the following policy, choose **Cancel**. If the trust relationship doesn't match, copy the policy into the **Edit trust policy** window and choose **Update policy**.

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

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

Creating the Amazon EKS node IAM role

You can create the node IAM role with the AWS Management Console or the AWS CLI.

AWS Management Console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Roles**.
3. On the **Roles** page, choose **Create role**.
4. On the **Select trusted entity** page, do the following:
   a. In the **Trusted entity type** section, choose **AWS service**.
   b. Under **Use case**, choose **EC2**.
   c. Choose **Next**.
5. On the **Add permissions** page, attach the following policies:
   - AmazonEKSWorkerNodeMinimalPolicy
   - AmazonEC2ContainerRegistryPullOnly
6. On the **Name, review, and create** page, do the following:
   a. For **Role name**, enter a unique name for your role, such as `AmazonEKSAutoNodeRole`.
   b. For **Description**, replace the current text with descriptive text such as `Amazon EKS - Node role`.
   c. Under **Add tags (Optional)**, add metadata to the role by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM resources in the *IAM User Guide*.
   d. Choose **Create role**.

AWS CLI

Create the Node IAM Role

Use the **node-trust-policy.json** file from the previous step to define which entities can assume the role. Run the following command to create the Node IAM Role:

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

Note the Role ARN

After creating the role, retrieve and save the ARN of the Node IAM Role. You will need this ARN in subsequent steps. Use the following command to get the ARN:

```
aws iam get-role --role-name AmazonEKSAutoNodeRole --query "Role.Arn" --output text
```

Attach Required Policies

Attach the following AWS managed policies to the Node IAM Role to provide the necessary permissions:

To attach AmazonEKSWorkerNodeMinimalPolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodeMinimalPolicy
```

To attach AmazonEC2ContainerRegistryPullOnly:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

Monitor your cluster performance and view logs

You can observe your data in Amazon EKS using many available monitoring or logging tools. Your Amazon EKS log data can be streamed to AWS services or to partner tools for data analysis. There are many services available in the AWS Management Console that provide data for troubleshooting your Amazon EKS issues. You can also use an AWS-supported open-source solution for monitoring Amazon EKS infrastructure.

After selecting **Clusters** in the left navigation pane of the Amazon EKS console, you can view cluster health and details by choosing your cluster's name and choosing the **Observability** tab. To view details about any existing Kubernetes resources that are deployed to your cluster, see the section called "Access cluster resources".

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon EKS and your AWS solutions. We recommend that you collect monitoring data from all of the parts of your AWS solution. That way, you can more easily debug a multi-point failure if one occurs. Before you start monitoring Amazon EKS, make sure that your monitoring plan addresses the following questions.

- What are your goals? Do you need real-time notifications if your clusters scale dramatically?
- What resources need to be observed?
- How frequently do you need to observe these resources? Does your company want to respond quickly to risks?
- What tools do you intend to use? If you already run AWS Fargate as part of your launch, then you can use the built-in log router.
- Who do you intend to perform the monitoring tasks?
- Whom do you want notifications to be sent to when something goes wrong?

Monitoring and logging on Amazon EKS

Amazon EKS provides built-in tools for monitoring and logging. For supported versions, the observability dashboard gives visibility into the performance of your cluster. It helps you to quickly detect, troubleshoot, and remediate issues. In addition to monitoring features, it includes lists based on the control plane audit logs. The Kubernetes control plane exposes a number of metrics that that can also be scraped outside of the console.

Control plane logging records all API calls to your clusters, audit information capturing what users performed what actions to your clusters, and role-based information. For more information, see Logging and monitoring on Amazon EKS in the *AWS Prescriptive Guidance*.

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

> **Note**
>
> When you check the Amazon EKS authenticator logs in Amazon CloudWatch, the entries are displayed that contain text similar to the following example text.
>
> ```
> level=info msg="mapping IAM role" groups="[]"
>  role="arn:aws:iam::111122223333:role/XXXXXXXXXXXXXXXXXX-NodeManagerRole-
> XXXXXXXX" username="eks:node-manager"
> ```
>
> Entries that contain this text are expected. The `username` is an Amazon EKS internal service role that performs specific operations for managed node groups and Fargate.
>
> For low-level, customizable logging, then Kubernetes logging is available.

Amazon EKS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon EKS. CloudTrail captures all API calls for Amazon EKS as events. The calls captured include calls from the Amazon EKS console and code calls to the Amazon EKS API operations. For more information, see the section called " AWS CloudTrail".

The Kubernetes API server exposes a number of metrics that are useful for monitoring and analysis. For more information, see the section called "Prometheus metrics".

To configure Fluent Bit for custom Amazon CloudWatch logs, see Setting up Fluent Bit in the *Amazon CloudWatch User Guide*.

Amazon EKS monitoring and logging tools

Amazon Web Services provides various tools that you can use to monitor Amazon EKS. You can configure some tools to set up automatic monitoring, but some require manual calls. We recommend that you automate monitoring tasks as much as your environment and existing toolset allows.

The following table describes various monitoring tool options.

Areas	Tool	Description	Setup
Control plane	Observability dashboard	For supported versions, the observability dashboard gives visibility into the performance of your cluster. It helps you to quickly detect, troubleshoot, and remediate issues.	Setup procedure
Applications / control plane	Prometheus	Prometheus can be used to monitor metrics and alerts for applications and the control plane.	Setup procedure
Applications	CloudWatch Container Insights	CloudWatch Container Insights collects, aggregate s, and summarize s metrics and logs from your container ized applications and microservices.	Setup procedure
Applications	AWS Distro for OpenTelemetry (ADOT)	ADOT can collect and sends correlate d metrics, trace data, and metadata to AWS monitoring	Setup procedure

Areas	Tool	Description	Setup
		services or partners. It can be set up through CloudWatch Container Insights.	
Applications	Amazon DevOps Guru	Amazon DevOps Guru detects node-level operational performance and availability.	Setup procedure
Applications	AWS X-Ray	AWS X-Ray receives trace data about your application. This trace data includes ingoing and outgoing requests and metadata about the requests. For Amazon EKS, the implement ation requires the OpenTelemetry add-on.	Setup procedure

Areas	Tool	Description	Setup
Applications	Amazon CloudWatch	CloudWatch provides some basic Amazon EKS metrics for free on supported versions. You can expand this functionality with the CloudWatc h Observability Operator to handle collecting metrics, logs, and trace data.	Setup procedure

The following table describes various logging tool options.

Areas	Tool	Description	Setup
Control plane	Observability dashboard	For supported versions, the observability dashboard shows lists based on the control plane audit logs. It also includes links to control plane logs in Amazon CloudWatch.	Setup procedure
Applications	Amazon CloudWatch Container Insights	Amazon CloudWatc h Container Insights collects, aggregate s, and summarize s metrics and logs from your container	Setup procedure

Areas	Tool	Description	Setup
		ized applications and microservices.	
Control plane	Amazon CloudWatch Logs	You can send audit and diagnostic logs directly from the Amazon EKS control plane to CloudWatch Logs in your account.	Setup procedure
Control plane	AWS CloudTrail	It logs API calls by a user, role, or service.	Setup procedure
Multiple areas for AWS Fargate instances	AWS Fargate log router	For AWS Fargate instances, the log router streams logs to AWS services or partner tools. It uses AWS for Fluent Bit. Logs can be streamed to other AWS services or partner tools.	Setup procedure

Monitor your cluster with the observability dashboard

The Amazon EKS console includes an observability dashboard that gives visibility into the performance of your cluster. The information in this dashboard helps you to quickly detect, troubleshoot, and remediate issues. You can open the applicable section of the dashboard by choosing an item in the **Health and performance summary**. This summary is included in several places, including the **Observability** tab.

The dashboard is split into several tabs.

Summary

The **Health and performance summary** lists the quantity of items in various categories. Each number acts as a hyperlink to a location in the dashboard with a list for that category.

Cluster health issues

Cluster health issues are important notifications to be aware of, some of which you may need to take action on as soon as possible. With this list, you can see descriptions and the affected resources. To refresh the status, choose the refresh button (↻).

For more information, see the section called “Cluster health FAQs and error codes with resolution paths”.

Control plane monitoring

The **Control plane monitoring** tab is divided into three sections, each of which help you to monitor and troubleshoot your cluster’s control plane.

Metrics

For clusters that are Kubernetes version `1.28` and above, the **Metrics** section shows graphs of several metrics gathered for various control plane components.

You can set the time period used by the X-axis of every graph by making selections at the top of the section. You can refresh data with the refresh button (↻). For each separate graph, the vertical ellipses button (⋮) opens a menu with options from CloudWatch.

These metrics and more are automatically available as basic monitoring metrics in CloudWatch under the `AWS/EKS` namespace. For more information, see Basic monitoring and detailed monitoring in the *Amazon CloudWatch User Guide*. To get more detailed metrics, visualization, and insights, see Container Insights in the *Amazon CloudWatch User Guide*. Or if you prefer Prometheus based monitoring, see the section called “Prometheus metrics”.

The following table describes available metrics.

Metric	Description
APIServer Requests	The requests per minute made to the API server.

Metric	Description
APIServer Total Requests 4XX	The count of API server requests per minute that had HTTP 4XX response codes (client-side errors).
APIServer Total Requests 5XX	The count of API server requests per minute that had HTTP 5XX response codes (server-side errors).
APIServer Total Requests 429	The count of API server requests per minute that had HTTP 429 response codes (too many requests).
Storage size	The storage database (`etcd`) size.
Scheduler attempts	The number of attempts to schedule pods by results "unschedulable" "error", and "scheduled".
Pending pods	The number of pending pods by queue type of "active", "backoff", "unschedulable", and "gated".
API server request latency	The latency for API server requests.
API server current inflight requests	The current in-flight requests for the API server.
Webhook requests	The webhook requests per minute.
Webhook request rejections	The count of webhook requests that were rejected.
Webhook request latency P99	The 99th percentile latency of external, third-party webhook requests.

CloudWatch Log Insights

The **CloudWatch Log Insights** section shows various lists based on the control plane audit logs. The Amazon EKS control plane logs need to be turned on to use this feature, which you can do from the **View control plane logs in CloudWatch** section.

When enough time has passed to collect data, you can **Run all queries** or choose **Run query** for a single list at a time. An additional cost will incur from CloudWatch whenever you run queries. Choose the time period of results you want to view at the top of the section. If you want more advanced control for any query, you can choose **View in CloudWatch**. This will allow you to update a query in CloudWatch to fit your needs.

For more information, see Analyzing log data with CloudWatch Logs Insights in the Amazon CloudWatch Logs User Guide.

View control plane logs in CloudWatch

Choose **Manage logging** to update the log types that are available. It takes several minutes for the logs to appear in CloudWatch Logs after you enable logging. When enough time has passed, choose any of the **View** links in this section to navigate to the applicable log.

For more information, see the section called “Control plane logs”.

Cluster insights

The **Upgrade insights** table both surfaces issues and recommends corrective actions, accelerating the validation process for upgrading to new Kubernetes versions. Amazon EKS automatically scans clusters against a list of potential Kubernetes version upgrade impacting issues. The **Upgrade insights** table lists the insight checks performed by Amazon EKS against this cluster, along with their associated statuses.

Amazon EKS maintains and periodically refreshes the list of insight checks to be performed based on evaluations of changes in the Kubernetes project as well as Amazon EKS service changes tied to new versions. The Amazon EKS console automatically refreshes the status of each insight, which can be seen in the last refresh time column.

For more information, see the section called “Cluster insights”.

Node health issues

The Amazon EKS node monitoring agent automatically reads node logs to detect health issues. Regardless of the auto repair setting, all node health issues are reported so that you can investigate as needed. If an issue type is listed without a description, you can read the description in its popover element.

When you refresh the page, any resolved issues will disappear from the list. If auto repair is enabled, you could temporarily see some health issues that will be resolved without action from you. Issues that are not supported by auto repair may require manual action from you depending on the type.

For node health issues to be reported, your cluster must use Amazon EKS Auto Mode or have the node monitoring agent add-on. For more information, see the section called “Node health”.

Monitor your cluster metrics with Prometheus

Prometheus is a monitoring and time series database that scrapes endpoints. It provides the ability to query, aggregate, and store collected data. You can also use it for alerting and alert aggregation. This topic explains how to set up Prometheus as either a managed or open source option. Monitoring Amazon EKS control plane metrics is a common use case.

Amazon Managed Service for Prometheus is a Prometheus-compatible monitoring and alerting service that makes it easy to monitor containerized applications and infrastructure at scale. It is a fully-managed service that automatically scales the ingestion, storage, querying, and alerting of your metrics. It also integrates with AWS security services to enable fast and secure access to your data. You can use the open-source PromQL query language to query your metrics and alert on them. Also, you can use alert manager in Amazon Managed Service for Prometheus to set up alerting rules for critical alerts. You can then send these critical alerts as notifications to an Amazon SNS topic.

There are several different options for using Prometheus with Amazon EKS:

- You can turn on Prometheus metrics when first creating an Amazon EKS cluster or you can create your own Prometheus scraper for existing clusters. Both of these options are covered by this topic.
- You can deploy Prometheus using Helm. For more information, see the section called “Deploy using Helm”.

- You can view control plane raw metrics in Prometheus format. For more information, see the section called “Control plane”.

Step 1: Turn on Prometheus metrics

> **Important**
>
> Amazon Managed Service for Prometheus resources are outside of the cluster lifecycle and need to be maintained independent of the cluster. When you delete your cluster, make sure to also delete any applicable scrapers to stop applicable costs. For more information, see Find and delete scrapers in the *Amazon Managed Service for Prometheus User Guide*.

Prometheus discovers and collects metrics from your cluster through a pull-based model called scraping. Scrapers are set up to gather data from your cluster infrastructure and containerized applications. When you turn on the option to send Prometheus metrics, Amazon Managed Service for Prometheus provides a fully managed agentless scraper.

If you haven’t created the cluster yet, you can turn on the option to send metrics to Prometheus when first creating the cluster. In the Amazon EKS console, this option is in the **Configure observability** step of creating a new cluster. For more information, see the section called “Create a cluster”.

If you already have an existing cluster, you can create your own Prometheus scraper. To do this in the Amazon EKS console, navigate to your cluster’s **Observability** tab and choose the **Add scraper** button. If you would rather do so with the AWS API or AWS CLI, see Create a scraper in the *Amazon Managed Service for Prometheus User Guide*.

The following options are available when creating the scraper with the Amazon EKS console.

Scraper alias

(Optional) Enter a unique alias for the scraper.

Destination

Choose an Amazon Managed Service for Prometheus workspace. A workspace is a logical space dedicated to the storage and querying of Prometheus metrics. With this workspace, you will be able to view Prometheus metrics across the accounts that have access to it. The

Create new workspace option tells Amazon EKS to create a workspace on your behalf using the **Workspace alias** you provide. With the **Select existing workspace** option, you can select an existing workspace from a dropdown list. For more information about workspaces, see Managing workspaces in the *Amazon Managed Service for Prometheus User Guide*.

Service access

This section summarizes the permissions you grant when sending Prometheus metrics:

- Allow Amazon Managed Service for Prometheus to describe the scraped Amazon EKS cluster
- Allow remote writing to the Amazon Managed Prometheus workspace

If the `AmazonManagedScraperRole` already exists, the scraper uses it. Choose the `AmazonManagedScraperRole` link to see the **Permission details**. If the `AmazonManagedScraperRole` doesn't exist already, choose the **View permission details** link to see the specific permissions you are granting by sending Prometheus metrics.

Subnets

Modify the subnets that the scraper will inherit as needed. If you need to add a grayed out subnet option, go back to the create cluster **Specify networking** step.

Scraper configuration

Modify the scraper configuration in YAML format as needed. To do so, use the form or upload a replacement YAML file. For more information, see Scraper configuration in the *Amazon Managed Service for Prometheus User Guide*.

Amazon Managed Service for Prometheus refers to the agentless scraper that is created alongside the cluster as an AWS managed collector. For more information about AWS managed collectors, see Ingest metrics with AWS managed collectors in the *Amazon Managed Service for Prometheus User Guide*.

> **Important**
>
> - If you create a Prometheus scraper using the AWS CLI or AWS API, you need to adjust its configuration to give the scraper in-cluster permissions. For more information, see Configuring your Amazon EKS cluster in the *Amazon Managed Service for Prometheus User Guide*.
> - If you have a Prometheus scraper created before November 11, 2024 that uses the `aws-auth` `ConfigMap` instead of access entries, you need to update it to access additional

metrics from the Amazon EKS cluster control plane. For the updated configuration, see Manually configuring Amazon EKS for scraper access in the *Amazon Managed Service for Prometheus User Guide*.

Step 2: Use the Prometheus metrics

For more information about how to use the Prometheus metrics after you turn them on for your cluster, see the Amazon Managed Service for Prometheus User Guide.

Step 3: Manage Prometheus scrapers

To manage scrapers, choose the **Observability** tab in the Amazon EKS console. A table shows a list of scrapers for the cluster, including information such as the scraper ID, alias, status, and creation date. You can add more scrapers, delete scrapers, or view more information about the current scrapers.

To see more details about a scraper, choose the scraper ID link. For example, you can view the ARN, environment, workspace ID, IAM role, configuration, and networking information. You can use the scraper ID as input to Amazon Managed Service for Prometheus API operations like `DescribeScraper` and `DeleteScraper`. For more information on using the Prometheus API, see the Amazon Managed Service for Prometheus API Reference.

Deploy Prometheus using Helm

As an alternative to using Amazon Managed Service for Prometheus, you can deploy Prometheus into your cluster with Helm V3. If you already have Helm installed, you can check your version with the `helm version` command. Helm is a package manager for Kubernetes clusters. For more information about Helm and how to install it, see the section called “Deploy apps with Helm”.

After you configure Helm for your Amazon EKS cluster, you can use it to deploy Prometheus with the following steps.

1. Create a Prometheus namespace.

```
kubectl create namespace prometheus
```

2. Add the `prometheus-community` chart repository.

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

3. Deploy Prometheus.

```
helm upgrade -i prometheus prometheus-community/prometheus \
    --namespace prometheus \
    --set alertmanager.persistence.storageClass="gp2" \
    --set server.persistentVolume.storageClass="gp2"
```

> **Note**
>
> If you get the error `Error: failed to download "stable/prometheus" (hint: running helm repo update may help)` when executing this command, run `helm repo update prometheus-community`, and then try running the Step 2 command again.

If you get the error `Error: rendered manifests contain a resource that already exists`, run `helm uninstall your-release-name -n namespace`, then try running the Step 3 command again.

4. Verify that all of the Pods in the `prometheus` namespace are in the READY state.

```
kubectl get pods -n prometheus
```

An example output is as follows.

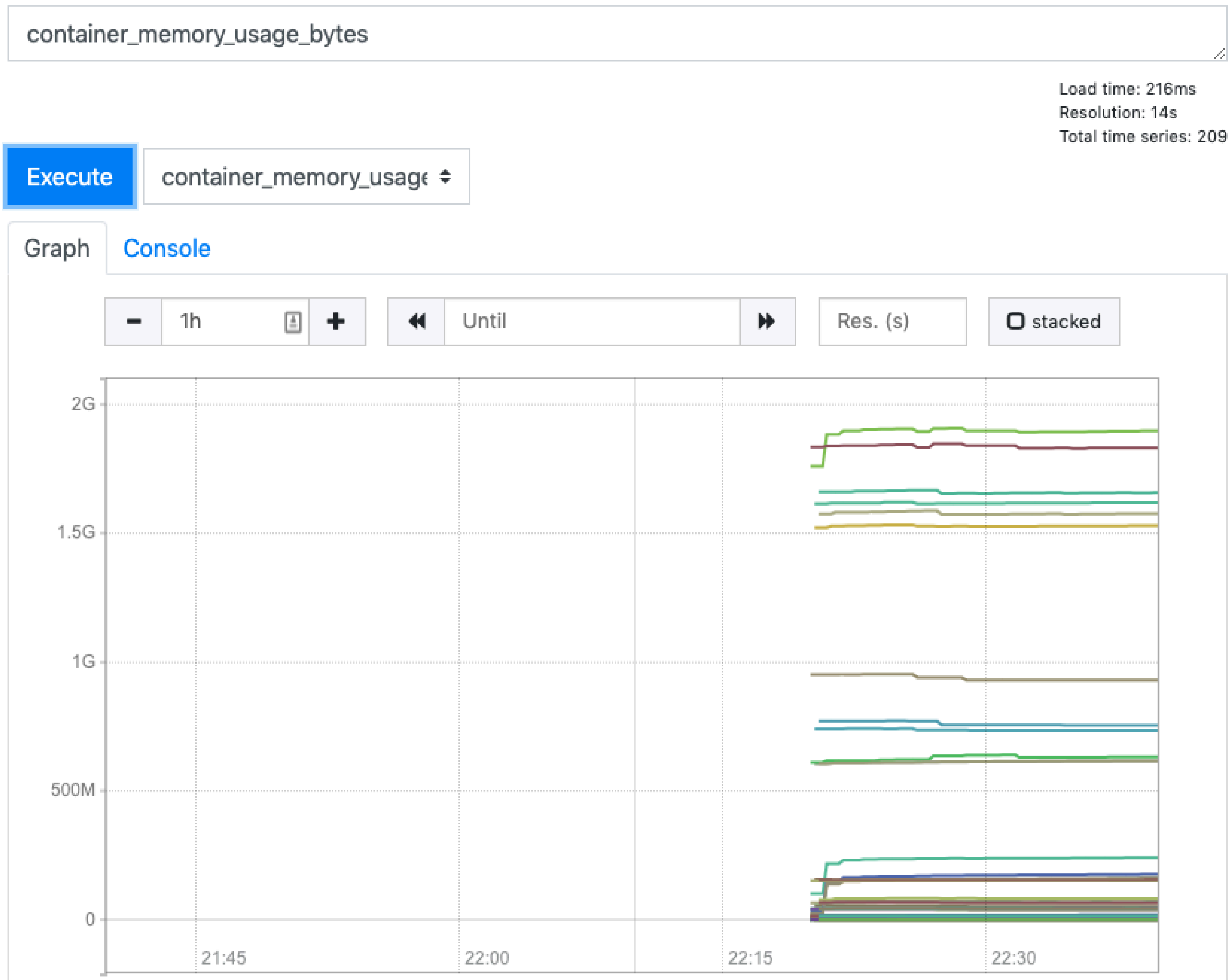
```
NAME                                             READY   STATUS    RESTARTS   AGE
prometheus-alertmanager-59b4c8c744-r7bgp         1/2     Running   0          48s
prometheus-kube-state-metrics-7cfd87cf99-jkz2f   1/1     Running   0          48s
prometheus-node-exporter-jcjqz                   1/1     Running   0          48s
prometheus-node-exporter-jxv2h                   1/1     Running   0          48s
prometheus-node-exporter-vbdks                   1/1     Running   0          48s
prometheus-pushgateway-76c444b68c-82tnw          1/1     Running   0          48s
prometheus-server-775957f748-mmht9               1/2     Running   0          48s
```

5. Use `kubectl` to port forward the Prometheus console to your local machine.

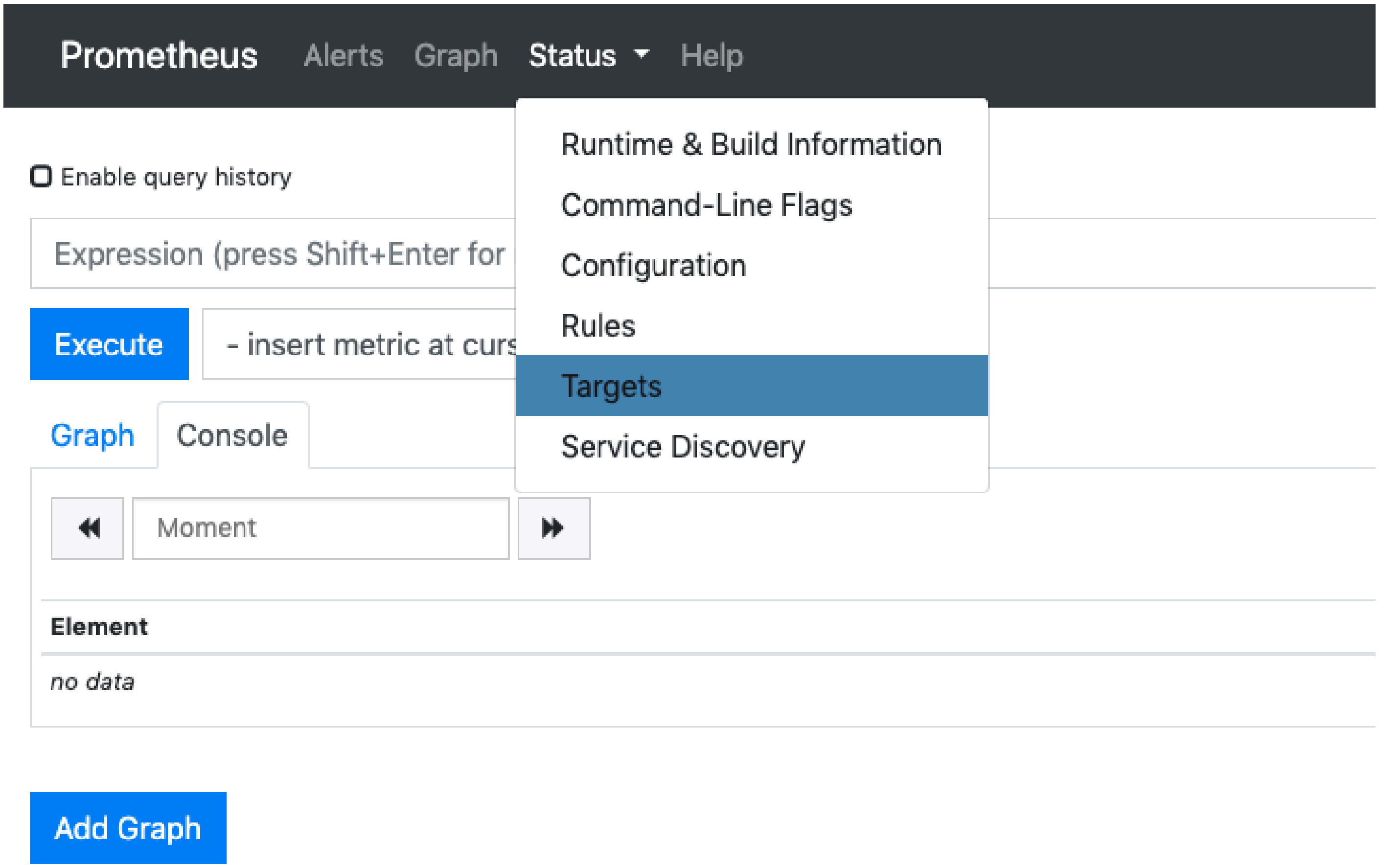
```
kubectl --namespace=prometheus port-forward deploy/prometheus-server 9090
```

6. Point a web browser to http://localhost:9090 to view the Prometheus console.

7. Choose a metric from the **- insert metric at cursor** menu, then choose **Execute**. Choose the **Graph** tab to show the metric over time. The following image shows `container_memory_usage_bytes` over time.



8. From the top navigation bar, choose **Status**, then **Targets**.



All of the Kubernetes endpoints that are connected to Prometheus using service discovery are displayed.

Fetch control plane raw metrics in Prometheus format

The Kubernetes control plane exposes a number of metrics that are represented in a [Prometheus format](). These metrics are useful for monitoring and analysis. They are exposed internally through metrics endpoints, and can be accessed without fully deploying Prometheus. However, deploying Prometheus more easily allows analyzing metrics over time.

To view the raw metrics output, replace `endpoint` and run the following command.

```
kubectl get --raw endpoint
```

This command allows you to pass any endpoint path and returns the raw response. The output lists different metrics line-by-line, with each line including a metric name, tags, and a value.

```
metric_name{tag="value"[,...]} value
```

Fetch metrics from the API server

The general API server endpoint is exposed on the Amazon EKS control plane. This endpoint is primarily useful when looking at a specific metric.

```
kubectl get --raw /metrics
```

An example output is as follows.

```
[...]
# HELP rest_client_requests_total Number of HTTP requests, partitioned by status code,
 method, and host.
# TYPE rest_client_requests_total counter
rest_client_requests_total{code="200",host="127.0.0.1:21362",method="POST"} 4994
rest_client_requests_total{code="200",host="127.0.0.1:443",method="DELETE"} 1
rest_client_requests_total{code="200",host="127.0.0.1:443",method="GET"} 1.326086e+06
rest_client_requests_total{code="200",host="127.0.0.1:443",method="PUT"} 862173
rest_client_requests_total{code="404",host="127.0.0.1:443",method="GET"} 2
rest_client_requests_total{code="409",host="127.0.0.1:443",method="POST"} 3
rest_client_requests_total{code="409",host="127.0.0.1:443",method="PUT"} 8
# HELP ssh_tunnel_open_count Counter of ssh tunnel total open attempts
# TYPE ssh_tunnel_open_count counter
ssh_tunnel_open_count 0
# HELP ssh_tunnel_open_fail_count Counter of ssh tunnel failed open attempts
# TYPE ssh_tunnel_open_fail_count counter
ssh_tunnel_open_fail_count 0
```

This raw output returns verbatim what the API server exposes.

Fetch control plane metrics with `metrics.eks.amazonaws.com`

For clusters that are Kubernetes version `1.28` and above, Amazon EKS also exposes metrics under the API group `metrics.eks.amazonaws.com`. These metrics include control plane components such as `kube-scheduler` and `kube-controller-manager`.

> **Note**
>
> If you have a webhook configuration that could block the creation of the new `APIService` resource `v1.metrics.eks.amazonaws.com` on your cluster, the metrics endpoint feature

might not be available. You can verify that in the `kube-apiserver` audit log by searching for the `v1.metrics.eks.amazonaws.com` keyword.

Fetch kube-scheduler metrics

To retrieve `kube-scheduler` metrics, use the following command.

```
kubectl get --raw "/apis/metrics.eks.amazonaws.com/v1/ksh/container/metrics"
```

An example output is as follows.

```
# TYPE scheduler_pending_pods gauge
scheduler_pending_pods{queue="active"} 0
scheduler_pending_pods{queue="backoff"} 0
scheduler_pending_pods{queue="gated"} 0
scheduler_pending_pods{queue="unschedulable"} 18
# HELP scheduler_pod_scheduling_attempts [STABLE] Number of attempts to successfully
 schedule a pod.
# TYPE scheduler_pod_scheduling_attempts histogram
scheduler_pod_scheduling_attempts_bucket{le="1"} 79
scheduler_pod_scheduling_attempts_bucket{le="2"} 79
scheduler_pod_scheduling_attempts_bucket{le="4"} 79
scheduler_pod_scheduling_attempts_bucket{le="8"} 79
scheduler_pod_scheduling_attempts_bucket{le="16"} 79
scheduler_pod_scheduling_attempts_bucket{le="+Inf"} 81
[...]
```

Fetch kube-controller-manager metrics

To retrieve `kube-controller-manager` metrics, use the following command.

```
kubectl get --raw "/apis/metrics.eks.amazonaws.com/v1/kcm/container/metrics"
```

An example output is as follows.

```
[...]
workqueue_work_duration_seconds_sum{name="pvprotection"} 0
workqueue_work_duration_seconds_count{name="pvprotection"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-08"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-07"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-06"} 0
```

```
workqueue_work_duration_seconds_bucket{name="replicaset",le="9.999999999999999e-06"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="9.999999999999999e-05"} 19
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.001"} 109
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.01"} 139
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.1"} 181
workqueue_work_duration_seconds_bucket{name="replicaset",le="1"} 191
workqueue_work_duration_seconds_bucket{name="replicaset",le="10"} 191
workqueue_work_duration_seconds_bucket{name="replicaset",le="+Inf"} 191
workqueue_work_duration_seconds_sum{name="replicaset"} 4.265655885000002
[...]
```

Understand the scheduler and controller manager metrics

The following table describes the scheduler and controller manager metrics that are made available for Prometheus style scraping. For more information about these metrics, see Kubernetes Metrics Reference in the Kubernetes documentation.

Metric	Control plane component	Description
scheduler_pending_pods	scheduler	The number of Pods that are waiting to be scheduled onto a node for execution.
scheduler_schedule_attempts_total	scheduler	The number of attempts made to schedule Pods.
scheduler_preemption_attempts_total	scheduler	The number of attempts made by the scheduler to schedule higher priority Pods by evicting lower priority ones.
scheduler_preemption_victims	scheduler	The number of Pods that have been selected for eviction to make room for higher priority Pods.
scheduler_pod_scheduling_attempts	scheduler	The number of attempts to successfully schedule a Pod.

Metric	Control plane component	Description
scheduler_scheduling_attempt_duration_seconds	scheduler	Indicates how quickly or slowly the scheduler is able to find a suitable place for a Pod to run based on various factors like resource availability and scheduling rules.
scheduler_pod_scheduling_sli_duration_seconds	scheduler	The end-to-end latency for a Pod being scheduled, from the time the Pod enters the scheduling queue. This might involve multiple scheduling attempts.
kube_pod_resource_request	scheduler	The resources requested by workloads on the cluster, broken down by Pod. This shows the resource usage the scheduler and kubelet expect per Pod for resources along with the unit for the resource if any.
kube_pod_resource_limit	scheduler	The resources limit for workloads on the cluster, broken down by Pod. This shows the resource usage the scheduler and kubelet expect per Pod for resources along with the unit for the resource if any.

Metric	Control plane component	Description
cronjob_controller_job_crea tion_skew_duration_seconds	controller manager	The time between when a cronjob is scheduled to be run, and when the correspon ding job is created.
workqueue_depth	controller manager	The current depth of queue.
workqueue_adds_total	controller manager	The total number of adds handled by workqueue.
workqueue_queue_du ration_seconds	controller manager	The time in seconds an item stays in workqueue before being requested.
workqueue_work_dur ation_seconds	controller manager	The time in seconds processin g an item from workqueue takes.

Deploy a Prometheus scraper to consistently scrape metrics

To deploy a Prometheus scraper to consistently scrape the metrics, use the following configuration:

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-conf
data:
  prometheus.yml: |-
    global:
      scrape_interval: 30s
    scrape_configs:
    # apiserver metrics
    - job_name: apiserver-metrics
      kubernetes_sd_configs:
      - role: endpoints
      scheme: https
      tls_config:
```

```
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels:
          [
            __meta_kubernetes_namespace,
            __meta_kubernetes_service_name,
            __meta_kubernetes_endpoint_port_name,
          ]
        action: keep
        regex: default;kubernetes;https
    # Scheduler metrics
    - job_name: 'ksh-metrics'
      kubernetes_sd_configs:
      - role: endpoints
      metrics_path: /apis/metrics.eks.amazonaws.com/v1/ksh/container/metrics
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels:
          [
            __meta_kubernetes_namespace,
            __meta_kubernetes_service_name,
            __meta_kubernetes_endpoint_port_name,
          ]
        action: keep
        regex: default;kubernetes;https
    # Controller Manager metrics
    - job_name: 'kcm-metrics'
      kubernetes_sd_configs:
      - role: endpoints
      metrics_path: /apis/metrics.eks.amazonaws.com/v1/kcm/container/metrics
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels:
          [
```

```
              __meta_kubernetes_namespace,
              __meta_kubernetes_service_name,
              __meta_kubernetes_endpoint_port_name,
            ]
          action: keep
          regex: default;kubernetes;https
---
apiVersion: v1
kind: Pod
metadata:
  name: prom-pod
spec:
  containers:
  - name: prom-container
    image: prom/prometheus
    ports:
    - containerPort: 9090
    volumeMounts:
    - name: config-volume
      mountPath: /etc/prometheus/
  volumes:
  - name: config-volume
    configMap:
      name: prometheus-conf
```

The permission that follows is required for the Pod to access the new metrics endpoint.

```
{
  "effect": "allow",
  "apiGroups": [
    "metrics.eks.amazonaws.com"
  ],
  "resources": [
    "kcm/metrics",
    "ksh/metrics"
  ],
  "verbs": [
    "get"
  ] },
```

To patch the role being used, you can use the following command.

```
kubectl patch clusterrole <role-name> --type=json -p='[
```

```
  {
    "op": "add",
    "path": "/rules/-",
    "value": {
      "verbs": ["get"],
      "apiGroups": ["metrics.eks.amazonaws.com"],
      "resources": ["kcm/metrics", "ksh/metrics"]
    }
  }
]'
```

Then you can view the Prometheus dashboard by proxying the port of the Prometheus scraper to your local port.

```
kubectl port-forward pods/prom-pod 9090:9090
```

For your Amazon EKS cluster, the core Kubernetes control plane metrics are also ingested into Amazon CloudWatch Metrics under the `AWS/EKS` namespace. To view them, open the [CloudWatch console](#) and select **All metrics** from the left navigation pane. On the **Metrics** selection page, choose the `AWS/EKS` namespace and a metrics dimension for your cluster.

Monitor cluster data with Amazon CloudWatch

Amazon CloudWatch is a monitoring service that collects metrics and logs from your cloud resources. CloudWatch provides some basic Amazon EKS metrics for free when using a new cluster that is version `1.28` and above. However, when using the CloudWatch Observability Operator as an Amazon EKS add-on, you can gain enhanced observability features.

Basic metrics in Amazon CloudWatch

For clusters that are Kubernetes version `1.28` and above, you get CloudWatch vended metrics for free in the `AWS/EKS` namespace. The following table gives a list of the basic metrics that are available for the supported versions. Every metric listed has a frequency of one minute.

Metric name	Description
`scheduler_schedule_attempts_total`	The number of total attempts by the scheduler to schedule Pods in the cluster for a given period. This metric helps monitor

Metric name	Description
	the scheduler's workload and can indicate scheduling pressure or potential issues with Pod placement. **Units:** Count **Valid statistics:** Sum
`scheduler_schedule_attempts_SCHEDULED`	The number of successful attempts by the scheduler to schedule Pods to nodes in the cluster for a given period. **Units:** Count **Valid statistics:** Sum
`scheduler_schedule_attempts_UNSCHEDULABLE`	The number of attempts to schedule Pods that were unschedulable for a given period due to valid constraints, such as insufficient CPU or memory on a node. **Units:** Count **Valid statistics:** Sum
`scheduler_schedule_attempts_ERROR`	The number of attempts to schedule Pods that failed for a given period due to an internal problem with the scheduler itself, such as API Server connectivity issues. **Units:** Count **Valid statistics:** Sum

Metric name	Description
`scheduler_pending_pods`	The number of total pending Pods to be scheduled by the scheduler in the cluster for a given period. **Units:** Count **Valid statistics:** Sum
`scheduler_pending_pods_ACTIVEQ`	The number of pending Pods in activeQ, that are waiting to be scheduled in the cluster for a given period. **Units:** Count **Valid statistics:** Sum
`scheduler_pending_pods_UNSCHEDULABLE`	The number of pending Pods that the scheduler attempted to schedule and failed, and are kept in an unschedulable state for retry. **Units:** Count **Valid statistics:** Sum
`scheduler_pending_pods_BACKOFF`	The number of pending Pods in `backoffQ` in a backoff state that are waiting for their backoff period to expire. **Units:** Count **Valid statistics:** Sum

Metric name	Description
scheduler_pending_pods_GATED	The number of pending Pods that are currently waiting in a gated state as they cannot be scheduled until they meet required conditions. **Units:** Count **Valid statistics:** Sum
apiserver_request_total	The number of HTTP requests made across all the API servers in the cluster. **Units:** Count **Valid statistics:** Sum
apiserver_request_total_4XX	The number of HTTP requests made to all the API servers in the cluster that resulted in 4XX (client error) status codes. **Units:** Count **Valid statistics:** Sum
apiserver_request_total_429	The number of HTTP requests made to all the API servers in the cluster that resulted in 429 status code, which occurs when clients exceed the rate limiting thresholds. **Units:** Count **Valid statistics:** Sum

Metric name	Description
`apiserver_request_total_5XX`	The number of HTTP requests made to all the API servers in the cluster that resulted in 5XX (server error) status codes. **Units:** Count **Valid statistics:** Sum
`apiserver_request_total_LIST_PODS`	The number of `LIST` Pods requests made to all the API servers in the cluster. **Units:** Count **Valid statistics:** Sum
`apiserver_request_duration_seconds_PUT_P99`	The 99th percentile of latency for PUT requests calculated from all requests across all API servers in the cluster. Represents the response time below which 99% of all PUT requests are completed. **Units:** Seconds **Valid statistics:** Average
`apiserver_request_duration_seconds_PATCH_P99`	The 99th percentile of latency for PATCH requests calculated from all requests across all API servers in the cluster. Represents the response time below which 99% of all PATCH requests are completed. **Units:** Seconds **Valid statistics:** Average

Metric name	Description
`apiserver_request_duration_seconds_POST_P99`	The 99th percentile of latency for `POST` requests calculated from all requests across all API servers in the cluster. Represents the response time below which 99% of all `POST` requests are completed. **Units:** Seconds **Valid statistics:** Average
`apiserver_request_duration_seconds_GET_P99`	The 99th percentile of latency for `GET` requests calculated from all requests across all API servers in the cluster. Represents the response time below which 99% of all `GET` requests are completed. **Units:** Seconds **Valid statistics:** Average
`apiserver_request_duration_seconds_LIST_P99`	The 99th percentile of latency for `LIST` requests calculated from all requests across all API servers in the cluster. Represents the response time below which 99% of all `LIST` requests are completed. **Units:** Seconds **Valid statistics:** Average

Metric name	Description
apiserver_request_duration_seconds_DELETE_P99	The 99th percentile of latency for DELETE requests calculated from all requests across all API servers in the cluster. Represents the response time below which 99% of all DELETE requests are completed. **Units:** Seconds **Valid statistics:** Average
apiserver_current_inflight_requests_MUTATING	The number of mutating requests (POST, PUT, DELETE, PATCH) currently being processed across all API servers in the cluster. This metric represents requests that are in-flight and haven't completed processing yet. **Units:** Count **Valid statistics:** Sum
apiserver_current_inflight_requests_READONLY	The number of read-only requests (GET, LIST) currently being processed across all API servers in the cluster. This metric represent s requests that are in-flight and haven't completed processing yet. **Units:** Count **Valid statistics:** Sum
apiserver_admission_webhook_request_total	The number of admission webhook requests made across all API servers in the cluster. **Units:** Count **Valid statistics:** Sum

Metric name	Description
`apiserver_admission_webhook_request_total_ADMIT`	The number of mutating admission webhook requests made across all API servers in the cluster. **Units:** Count **Valid statistics:** Sum
`apiserver_admission_webhook_request_total_VALIDATING`	The number of validating admission webhook requests made across all API servers in the cluster. **Units:** Count **Valid statistics:** Sum
`apiserver_admission_webhook_rejection_count`	The number of admission webhook requests made across all API servers in the cluster that were rejected. **Units:** Count **Valid statistics:** Sum
`apiserver_admission_webhook_rejection_count_ADMIT`	The number of mutating admission webhook requests made across all API servers in the cluster that were rejected. **Units:** Count **Valid statistics:** Sum
`apiserver_admission_webhook_rejection_count_VALIDATING`	The number of validating admission webhook requests made across all API servers in the cluster that were rejected. **Units:** Count **Valid statistics:** Sum

Metric name	Description
`apiserver_admission_webhook_admission_duration_seconds`	The 99th percentile of latency for third-party admission webhook requests calculated from all requests across all API servers in the cluster. Represents the response time below which 99% of all third-party admission webhook requests are completed. **Units:** Seconds **Valid statistics:** Average
`apiserver_admission_webhook_admission_duration_seconds_ADMIT_P99`	The 99th percentile of latency for third-party mutating admission webhook requests calculated from all requests across all API servers in the cluster. Represents the response time below which 99% of all third-party mutating admission webhook requests are completed. **Units:** Seconds **Valid statistics:** Average
`apiserver_admission_webhook_admission_duration_seconds_VALIDATING_P99`	The 99th percentile of latency for third-party validating admission webhook requests calculated from all requests across all API servers in the cluster. Represents the response time below which 99% of all third-party validating admission webhook requests are completed. **Units:** Seconds **Valid statistics:** Average

Metric name	Description
`apiserver_storage_size_bytes`	The physical size in bytes of the etcd storage database file used by the API servers in the cluster. This metric represents the actual disk space allocated for the storage. **Units:** Bytes **Valid statistics:** Maximum

Amazon CloudWatch Observability Operator

Amazon CloudWatch Observability collects real-time logs, metrics, and trace data. It sends them to Amazon CloudWatch and AWS X-Ray. You can install this add-on to enable both CloudWatch Application Signals and CloudWatch Container Insights with enhanced observability for Amazon EKS. This helps you monitor the health and performance of your infrastructure and containerized applications. The Amazon CloudWatch Observability Operator is designed to install and configure the necessary components.

Amazon EKS supports the CloudWatch Observability Operator as an Amazon EKS add-on. The add-on allows Container Insights on both Linux and Windows worker nodes in the cluster. To enable Container Insights on Windows, the Amazon EKS add-on version must be `1.5.0` or higher. Currently, CloudWatch Application Signals isn't supported on Amazon EKS Windows.

The topics below describe how to get started using CloudWatch Observability Operator for your Amazon EKS cluster.

- For instructions on installing this add-on, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart in the *Amazon CloudWatch User Guide*.
- For more information about CloudWatch Application Signals, see Application Signals in the *Amazon CloudWatch User Guide*.
- For more information about Container Insights, see Using Container Insights in the *Amazon CloudWatch User Guide*.

Send control plane logs to CloudWatch Logs

Amazon EKS control plane logging provides audit and diagnostic logs directly from the Amazon EKS control plane to CloudWatch Logs in your account. These logs make it easy for you to secure and run your clusters. You can select the exact log types you need, and logs are sent as log streams to a group for each Amazon EKS cluster in CloudWatch. You can use CloudWatch subscription filters to do real time analysis on the logs or to forward them to other services (the logs will be Base64 encoded and compressed with the gzip format). For more information, see Amazon CloudWatch logging.

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

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

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

API server (`api`)

Your cluster's API server is the control plane component that exposes the Kubernetes API. If you enable API server logs when you launch the cluster, or shortly thereafter, the logs include API server flags that were used to start the API server. For more information, see kube-apiserver and the audit policy in the Kubernetes documentation.

Audit (`audit`)

Kubernetes audit logs provide a record of the individual users, administrators, or system components that have affected your cluster. For more information, see Auditing in the Kubernetes documentation.

Authenticator (`authenticator`)

Authenticator logs are unique to Amazon EKS. These logs represent the control plane component that Amazon EKS uses for Kubernetes Role Based Access Control (RBAC) authentication using IAM credentials. For more information, see *Cluster management*.

Controller manager (`controllerManager`)

The controller manager manages the core control loops that are shipped with Kubernetes. For more information, see kube-controller-manager in the Kubernetes documentation.

Scheduler (`scheduler`)

The scheduler component manages when and where to run Pods in your cluster. For more information, see kube-scheduler in the Kubernetes documentation.

Enable or disable control plane logs

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

To update the control plane logging configuration, Amazon EKS requires up to five available IP addresses in each subnet. When you enable a log type, the logs are sent with a log verbosity level of 2.

You can enable or disable control plane logs with either the AWS Management Console or the AWS CLI.

AWS Management Console

1. Open the Amazon EKS console.
2. Choose the name of the cluster to display your cluster information.
3. Choose the **Observability** tab.
4. In the **Control plane logging** section, choose **Manage logging**.
5. For each individual log type, choose whether the log type should be turned on or turned off. By default, each log type is turned off.

6. Choose **Save changes** to finish.

AWS CLI

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

```
aws --version
```

If your AWS CLI version is earlier than `1.16.139`, you must first update to the latest version. To install or upgrade the AWS CLI, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

2. Update your cluster's control plane log export configuration with the following AWS CLI command. Replace *my-cluster* with your cluster name and specify your desired endpoint access values.

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

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

An example output is as follows.

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

```
        ],
        "createdAt": 1553271814.684,
        "errors": []
    }
}
```

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

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

An example output is as follows.

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

View cluster control plane logs

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

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

1. Open the [CloudWatch console](). The link opens the console and displays your current available log groups and filters them with the `/aws/eks` prefix.
2. Choose the cluster that you want to view logs for. The log group name format is `/aws/eks/my-cluster/cluster`.
3. Choose the log stream to view. The following list describes the log stream name format for each log type.

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

   - **Kubernetes API server component logs (`api`)** – `kube-apiserver-1234567890abcdef01234567890abcde`
   - **Audit (`audit`)** – `kube-apiserver-audit-1234567890abcdef01234567890abcde`
   - **Authenticator (`authenticator`)** – `authenticator-1234567890abcdef01234567890abcde`
   - **Controller manager (`controllerManager`)** – `kube-controller-manager-1234567890abcdef01234567890abcde`
   - **Scheduler (`scheduler`)** – `kube-scheduler-1234567890abcdef01234567890abcde`
4. Look through the events of the log stream.

   For example, you should see the initial API server flags for the cluster when viewing the top of `kube-apiserver-1234567890abcdef01234567890abcde` .

> **Note**
>
> If you don't see the API server logs at the beginning of the log stream, then it is likely that the API server log file was rotated on the server before you enabled API server logging on the server. Any log files that are rotated before API server logging is enabled can't be exported to CloudWatch.

However, you can create a new cluster with the same Kubernetes version and enable the API server logging when you create the cluster. Clusters with the same platform version have the same flags enabled, so your flags should match the new cluster's flags. When you finish viewing the flags for the new cluster in CloudWatch, you can delete the new cluster.

Log API calls as AWS CloudTrail events

Amazon EKS is integrated with AWS CloudTrail. CloudTrail is a service that provides a record of actions by a user, role, or an AWS service in Amazon EKS. CloudTrail captures all API calls for Amazon EKS as events. This includes calls from the Amazon EKS console and from code calls to the Amazon EKS API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket. This includes events for Amazon EKS. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information that CloudTrail collects, you can determine several details about a request. For example, you can determine when the request was made to Amazon EKS, the IP address where the request was made from, and who made the request.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Topics

- View helpful references for AWS CloudTrail
- Analyze AWS CloudTrail log file entries
- View metrics for Amazon EC2 Auto Scaling groups

View helpful references for AWS CloudTrail

When you create your AWS account, CloudTrail is also enabled on your AWS account. When any activity occurs in Amazon EKS, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail event history.

For an ongoing record of events in your AWS account, including events for Amazon EKS, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all AWS Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify.

Additionally, you can configure other AWS services to further analyze and act upon the event data that's collected in CloudTrail logs. For more information, see the following resources.

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

All Amazon EKS actions are logged by CloudTrail and are documented in the Amazon EKS API Reference. For example, calls to the CreateCluster, ListClusters and DeleteCluster sections generate entries in the CloudTrail log files.

Every event or log entry contains information about the type of IAM identity that made the request, and which credentials were used. If temporary credentials were used, the entry shows how the credentials were obtained.

For more information, see the CloudTrail userIdentity element.

Analyze AWS CloudTrail log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action. This include information such as the date and time of the action and the request parameters that were used. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateCluster` action.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws:iam::111122223333:user/username",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
```

```
    "userName": "username"
  },
  "eventTime": "2018-05-28T19:16:43Z",
  "eventSource": "eks.amazonaws.com",
  "eventName": "CreateCluster",
  "awsRegion": "region-code",
  "sourceIPAddress": "205.251.233.178",
  "userAgent": "PostmanRuntime/6.4.0",
  "requestParameters": {
    "resourcesVpcConfig": {
      "subnetIds": [
        "subnet-a670c2df",
        "subnet-4f8c5004"
      ]
    },
    "roleArn": "arn:aws:iam::111122223333:role/AWSServiceRoleForAmazonEKS-
CAC1G1VH3ZKZ",
    "clusterName": "test"
  },
  "responseElements": {
    "cluster": {
      "clusterName": "test",
      "status": "CREATING",
      "createdAt": 1527535003.208,
      "certificateAuthority": {},
      "arn": "arn:aws:eks:region-code:111122223333:cluster/test",
      "roleArn": "arn:aws:iam::111122223333:role/AWSServiceRoleForAmazonEKS-
CAC1G1VH3ZKZ",
      "version": "1.10",
      "resourcesVpcConfig": {
        "securityGroupIds": [],
        "vpcId": "vpc-21277358",
        "subnetIds": [
          "subnet-a670c2df",
          "subnet-4f8c5004"
        ]
      }
    }
  },
  "requestID": "a7a0735d-62ab-11e8-9f79-81ce5b2b7d37",
  "eventID": "eab22523-174a-499c-9dd6-91e7be3ff8e3",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
```

```
}
```

Log Entries for Amazon EKS Service Linked Roles

The Amazon EKS service linked roles make API calls to AWS resources. CloudTrail log entries with `username: AWSServiceRoleForAmazonEKS` and `username: AWSServiceRoleForAmazonEKSNodegroup` appears for calls made by the Amazon EKS service linked roles. For more information about Amazon EKS and service linked roles, see the section called "Service-linked roles".

The following example shows a CloudTrail log entry that demonstrates a `DeleteInstanceProfile` action that's made by the `AWSServiceRoleForAmazonEKSNodegroup` service linked role, noted in the `sessionContext`.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3WHGPEZ7SJ2CW55C5:EKS",
        "arn": "arn:aws:sts::111122223333:assumed-role/
AWSServiceRoleForAmazonEKSNodegroup/EKS",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA3WHGPEZ7SJ2CW55C5",
                "arn": "arn:aws:iam::111122223333:role/aws-service-role/eks-
nodegroup.amazonaws.com/AWSServiceRoleForAmazonEKSNodegroup",
                "accountId": "111122223333",
                "userName": "AWSServiceRoleForAmazonEKSNodegroup"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-02-26T00:56:33Z"
            }
        },
        "invokedBy": "eks-nodegroup.amazonaws.com"
    },
    "eventTime": "2020-02-26T00:56:34Z",
    "eventSource": "iam.amazonaws.com",
```

```
    "eventName": "DeleteInstanceProfile",
    "awsRegion": "region-code",
    "sourceIPAddress": "eks-nodegroup.amazonaws.com",
    "userAgent": "eks-nodegroup.amazonaws.com",
    "requestParameters": {
        "instanceProfileName": "eks-11111111-2222-3333-4444-abcdef123456"
    },
    "responseElements": null,
    "requestID": "11111111-2222-3333-4444-abcdef123456",
    "eventID": "11111111-2222-3333-4444-abcdef123456",
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

View metrics for Amazon EC2 Auto Scaling groups

Amazon EKS managed node groups have Amazon EC2 Auto Scaling group metrics enabled by default with no additional charge. The Auto Scaling group sends sampled data to Amazon CloudWatch every minute. These metrics can be refined by the name of the Auto Scaling groups. They give you continuous visibility into the history of the Auto Scaling group powering your managed node groups, such as changes in the size of the group over time. Auto Scaling group metrics are available in the Amazon CloudWatch or Auto Scaling console. For more information, see Monitor CloudWatch metrics for your Auto Scaling groups and instances.

With Auto Scaling group metrics collection, you're able to monitor the scaling of managed node groups. Auto Scaling group metrics report the minimum, maximum, and desired size of an Auto Scaling group. You can create an alarm if the number of nodes in a node group falls below the minimum size, which would indicate an unhealthy node group. Tracking node group size is also useful in adjusting the maximum count so that your data plane doesn't run out of capacity.

If you would prefer to not have these metrics collected, you can choose to disable all or only some of them. For example, you can do this to avoid noise in your CloudWatch dashboards. For more information, see Amazon CloudWatch metrics for Amazon EC2 Auto Scaling.

Send metric and trace data with ADOT Operator

Amazon EKS supports using the AWS Management Console, AWS CLI and Amazon EKS API to install and manage the AWS Distro for OpenTelemetry (ADOT) Operator. This makes it easier to enable your applications running on Amazon EKS to send metric and trace data to multiple monitoring service options like Amazon CloudWatch, Prometheus, and X-Ray.

For more information, see Getting Started with AWS Distro for OpenTelemetry using EKS Add-Ons in the AWS Distro for OpenTelemetry documentation.

Enhance EKS with integrated AWS services

In addition to the services covered in other sections, Amazon EKS works with more AWS services to provide additional solutions. This topic identifies some of the other services that either use Amazon EKS to add functionality, or services that Amazon EKS uses to perform tasks.

Topics

Create Amazon EKS resources with AWS CloudFormation

Amazon EKS is integrated with AWS CloudFormation, a service that helps you model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the AWS resources that you want, for example an Amazon EKS cluster, and AWS CloudFormation takes care of provisioning and configuring those resources for you.

When you use AWS CloudFormation, you can reuse your template to set up your Amazon EKS resources consistently and repeatedly. Just describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

Amazon EKS and AWS CloudFormation templates

To provision and configure resources for Amazon EKS and related services, you must understand AWS CloudFormation templates. Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use AWS CloudFormation Designer to help you get started with AWS CloudFormation templates. For more information, see What is AWS CloudFormation Designer? in the *AWS CloudFormation User Guide*.

Amazon EKS supports creating clusters and node groups in AWS CloudFormation. For more information, including examples of JSON and YAML templates for your Amazon EKS resources, see Amazon EKS resource type reference in the *AWS CloudFormation User Guide*.

Learn more about AWS CloudFormation

To learn more about AWS CloudFormation, see the following resources:

- AWS CloudFormation
- AWS CloudFormation User Guide
- AWS CloudFormation Command Line Interface User Guide

Analyze security events on EKS with Amazon Detective

Amazon Detective helps you analyze, investigate, and quickly identify the root cause of security findings or suspicious activities. Detective automatically collects log data from your AWS resources. It then uses machine learning, statistical analysis, and graph theory to generate visualizations that help you to conduct faster and more efficient security investigations. The Detective prebuilt data aggregations, summaries, and context help you to quickly analyze and determine the nature and extent of possible security issues. For more information, see the Amazon Detective User Guide.

Detective organizes Kubernetes and AWS data into findings such as:

- Amazon EKS cluster details, including the IAM identity that created the cluster and the service role of the cluster. You can investigate the AWS and Kubernetes API activity of these IAM identities with Detective.
- Container details, such as the image and security context. You can also review details for terminated Pods.
- Kubernetes API activity, including both overall trends in API activity and details on specific API calls. For example, you can show the number of successful and failed Kubernetes API calls that were issued during a selected time range. Additionally, the section on newly observed API calls might be helpful to identify suspicious activity.

Amazon EKS audit logs is an optional data source package that can be added to your Detective behavior graph. You can view the available optional source packages, and their status in your account. For more information, see Amazon EKS audit logs for Detective in the *Amazon Detective User Guide*.

Use Amazon Detective with Amazon EKS

Before you can review findings, Detective must be enabled for at least 48 hours in the same AWS Region that your cluster is in. For more information, see Setting up Amazon Detective in the *Amazon Detective User Guide*.

1. Open the Detective console at https://console.aws.amazon.com/detective/.
2. From the left navigation pane, select **Search**.
3. Select **Choose type** and then select **EKS cluster**.
4. Enter the cluster name or ARN and then choose **Search**.
5. In the search results, choose the name of the cluster that you want to view activity for. For more information about what you can view, see Overall Kubernetes API activity involving an Amazon EKS cluster in the *Amazon Detective User Guide*.

Detect threats with Amazon GuardDuty

Amazon GuardDuty is a threat detection service that helps protect you accounts, containers, workloads, and the data with your AWS environment. Using machine learning (ML) models, and anomaly and threat detection capabilities, GuardDuty continuously monitors different log sources and runtime activity to identify and prioritize potential security risks and malicious activities in your environment.

Among other features, GuardDuty offers the following two features that detect potential threats to your EKS clusters: *EKS Protection* and *Runtime Monitoring*.

> **Note**
>
> **New:** Amazon EKS Auto Mode integrates with GuardDuty.

EKS Protection

This feature provides threat detection coverage to help you protect Amazon EKS clusters by monitoring the associated Kubernetes audit logs. Kubernetes audit logs capture sequential actions within your cluster, including activities from users, applications using the Kubernetes API, and the control plane. For example, GuardDuty can identify that APIs called to potentially tamper with resources in a Kubernetes cluster were invoked by an unauthenticated user.

When you enable EKS Protection, GuardDuty will be able to access your Amazon EKS audit logs only for continuous threat detection. If GuardDuty identifies a potential threat to your cluster, it generates an associated Kubernetes audit log *finding* of a specific type. For more information about the types of findings available from Kubernetes audit logs, see Kubernetes audit logs finding types in the Amazon GuardDuty User Guide.

For more information, see EKS Protection in the Amazon GuardDuty User Guide.

Runtime Monitoring

This feature monitors and analyzes operating system-level, networking, and file events to help you detect potential threats in specific AWS workloads in your environment.

When you enable *Runtime Monitoring* and install the GuardDuty agent in your Amazon EKS clusters, GuardDuty starts monitoring the runtime events associated with this cluster. Note that the GuardDuty agent and *Runtime Monitoring* aren't available for Amazon EKS Hybrid Nodes, so *Runtime Monitoring* isn't available for runtime events that occur on your hybrid nodes. If GuardDuty identifies a potential threat to your cluster, it generates an associated *Runtime Monitoring finding*. For example, a threat can potentially start by compromising a single container that runs a vulnerable web application. This web application might have access permissions to the underlying containers and workloads. In this scenario, incorrectly configured credentials could potentially lead to a broader access to the account, and the data stored within it.

To configure *Runtime Monitoring*, you install the GuardDuty agent to your cluster as an *Amazon EKS add-on*. For more information the add-on, see the section called " AWS Add-ons".

For more information, see Runtime Monitoring in the Amazon GuardDuty User Guide.

Assess EKS cluster resiliency with AWS Resilience Hub

AWS Resilience Hub assesses the resiliency of an Amazon EKS cluster by analyzing its infrastructure. AWS Resilience Hub uses the Kubernetes role-based access control (RBAC) configuration to assess the Kubernetes workloads deployed to your cluster. For more information, see Enabling AWS Resilience Hub access to your Amazon EKS cluster in the AWS Resilience Hub User Guide.

Centralize and analyze EKS security data with Security Lake

Amazon Security Lake is a fully managed security data lake service that allows you to centralize security data from various sources, including Amazon EKS. By integrating Amazon EKS with Security Lake, you can gain deeper insights into the activities performed on your Kubernetes resources and enhance the security posture of your Amazon EKS clusters.

> **Note**
>
> For more information about using Security Lake with Amazon EKS and setting up data sources, refer to the Amazon Security Lake documentation.

Benefits of using Security Lake with Amazon Amazon EKS

Centralized security data — Security Lake automatically collects and centralizes security data from your Amazon EKS clusters, along with data from other AWS services, SaaS providers, on-premises sources, and third-party sources. This provides a comprehensive view of your security posture across your entire organization.

Standardized data format — Security Lake converts the collected data into the Open Cybersecurity Schema Framework (OCSF) format, which is a standard open-source schema. This normalization enables easier analysis and integration with other security tools and services.

Improved threat detection — By analyzing the centralized security data, including Amazon EKS control plane logs, you can detect potentially suspicious activities within your Amazon EKS clusters more effectively. This helps in identifying and responding to security incidents promptly.

Simplified data management — Security Lake manages the lifecycle of your security data with customizable retention and replication settings. This simplifies data management tasks and ensures that you retain the necessary data for compliance and auditing purposes.

Enabling Security Lake for Amazon EKS

1. Enable Amazon EKS control plane logging for your EKS clusters. Refer to Enabling and disabling control plane logs for detailed instructions.
2. Add Amazon EKS Audit Logs as a source in Security Lake. Security Lake will then start collecting in-depth information about the activities performed on the Kubernetes resources running in your EKS clusters.

3. [Configure retention and replication settings](#) for your security data in Security Lake based on your requirements.
4. Use the normalized OCSF data stored in Security Lake for incident response, security analytics, and integration with other AWS services or third-party tools. For example, you can Generate security insights from Amazon Security Lake data using Amazon OpenSearch Ingestion.

Analyzing EKS Logs in Security Lake

Security Lake normalizes EKS log events to the OCSF format, making it easier to analyze and correlate the data with other security events. You can use various tools and services, such as Amazon Athena, Amazon QuickSight, or third-party security analytics tools, to query and visualize the normalized data.

For more information about the OCSF mapping for EKS log events, refer to the https://github.com/ocsf/examples/tree/main/mappings/markdown/ AWS/v1.1.0/EKS Audit Logs[mapping reference] in the OCSF GitHub repository.

Enable secure cross-cluster connectivity with Amazon VPC Lattice

Amazon VPC Lattice is a fully managed application networking service built directly into the AWS networking infrastructure that you can use to connect, secure, and monitor your services across multiple accounts and Virtual Private Clouds (VPCs). With Amazon EKS, you can leverage Amazon VPC Lattice through the use of the AWS Gateway API Controller, an implementation of the Kubernetes Gateway API. Using Amazon VPC Lattice, you can set up cross-cluster connectivity with standard Kubernetes semantics in a simple and consistent manner. To get started using Amazon VPC Lattice with Amazon EKS see the AWS Gateway API Controller User Guide.

Launch low-latency EKS clusters with AWS Local Zones

An AWS Local Zone is an extension of an AWS Region in geographic proximity to your users. Local Zones have their own connections to the internet and support AWS Direct Connect. Resources created in a Local Zone can serve local users with low-latency communications. For more information, see the AWS Local Zones User Guide and Local Zones in the *Amazon EC2 User Guide*.

Amazon EKS supports certain resources in Local Zones. This includes managed node groups, self-managed Amazon EC2 nodes, Amazon EBS volumes, and Application Load Balancers (ALBs). We recommend that you consider the following when using Local Zones as part of your Amazon EKS cluster.

- You can’t create Fargate nodes in Local Zones with Amazon EKS.
- The Amazon EKS managed Kubernetes control plane always runs in the AWS Region. The Amazon EKS managed Kubernetes control plane can’t run in the Local Zone. Because Local Zones appear as a subnet within your VPC, Kubernetes sees your Local Zone resources as part of that subnet.
- The Amazon EKS Kubernetes cluster communicates with the Amazon EC2 instances you run in the AWS Region or Local Zone using Amazon EKS managed elastic network interfaces. To learn more about Amazon EKS networking architecture, see *Configure networking*.
- Unlike regional subnets, Amazon EKS can’t place network interfaces into your Local Zone subnets. This means that you must not specify Local Zone subnets when you create your cluster.

Troubleshoot problems with Amazon EKS clusters and nodes

This chapter covers some common errors that you may see while using Amazon EKS and how to work around them. If you need to troubleshoot specific Amazon EKS areas, see the separate the section called "Troubleshooting", the section called "Troubleshoot EKS Connector", and Troubleshooting for ADOT using EKS Add-Ons topics.

For other troubleshooting information, see Knowledge Center content about Amazon Elastic Kubernetes Service on *AWS re:Post*.

Insufficient capacity

If you receive the following error while attempting to create an Amazon EKS cluster, then one of the Availability Zones you specified doesn't have sufficient capacity to support a cluster.

```
Cannot create cluster 'example-cluster' because region-1d, the targeted
Availability Zone, does not currently have sufficient capacity to support
the cluster. Retry and choose from these Availability Zones: region-1a,
region-1b, region-1c
```

Retry creating your cluster with subnets in your cluster VPC that are hosted in the Availability Zones returned by this error message.

There are Availability Zones that a cluster can't reside in. Compare the Availability Zones that your subnets are in with the list of Availability Zones in the Subnet requirements and considerations.

Nodes fail to join cluster

There are a few common reasons that prevent nodes from joining the cluster:

- If the nodes are managed nodes, Amazon EKS adds entries to the `aws-auth` `ConfigMap` when you create the node group. If the entry was removed or modified, then you need to re-add it. For more information, enter `eksctl create iamidentitymapping --help` in your terminal. You can view your current `aws-auth` `ConfigMap` entries by replacing *`my-cluster`* in the following command with the name of your cluster and then running the modified command: `eksctl get iamidentitymapping --cluster my-cluster` . The ARN of the role that you specify can't include a path other than /. For example, if the name of your role is development/apps/my-role, you'd need to change it to my-role when specifying the ARN for the role. Make sure that you specify the node IAM role ARN (not the instance profile ARN).

If the nodes are self-managed, and you haven't created access entries for the ARN of the node's IAM role, then run the same commands listed for managed nodes. If you have created an access entry for the ARN for your node IAM role, then it might not be configured properly in the access entry. Make sure that the node IAM role ARN (not the instance profile ARN) is specified as the principal ARN in your aws-auth ConfigMap entry or access entry. For more information about access entries, see the section called "Grant permissions".

- The **ClusterName** in your node AWS CloudFormation template doesn't exactly match the name of the cluster you want your nodes to join. Passing an incorrect value to this field results in an incorrect configuration of the node's /var/lib/kubelet/kubeconfig file, and the nodes will not join the cluster.
- The node is not tagged as being *owned* by the cluster. Your nodes must have the following tag applied to them, where *my-cluster* is replaced with the name of your cluster.

Key	Value
kubernetes.io/cluster/ *my-cluster*	owned

- The nodes may not be able to access the cluster using a public IP address. Ensure that nodes deployed in public subnets are assigned a public IP address. If not, you can associate an Elastic IP address to a node after it's launched. For more information, see Associating an Elastic IP address with a running instance or network interface. If the public subnet is not set to automatically assign public IP addresses to instances deployed to it, then we recommend enabling that setting. For more information, see Modifying the public IPv4 addressing attribute for your subnet. If the node is deployed to a private subnet, then the subnet must have a route to a NAT gateway that has a public IP address assigned to it.
- The AWS STS endpoint for the AWS Region that you're deploying the nodes to is not enabled for your account. To enable the region, see Activating and deactivating AWS STS in an AWS Region.
- The node doesn't have a private DNS entry, resulting in the kubelet log containing a node "" not found error. Ensure that the VPC where the node is created has values set for domain-name and domain-name-servers as Options in a DHCP options set. The default values are domain-name:<region>.compute.internal and domain-name-

servers:AmazonProvidedDNS. For more information, see DHCP options sets in the *Amazon VPC User Guide*.

- If the nodes in the managed node group do not connect to the cluster within 15 minutes, a health issue of "NodeCreationFailure" will be emitted and the console status will be set to `Create failed`. For Windows AMIs that have slow launch times, this issue can be resolved using fast launch.

To identify and troubleshoot common causes that prevent worker nodes from joining a cluster, you can use the `AWSSupport-TroubleshootEKSWorkerNode` runbook. For more information, see `AWSSupport-TroubleshootEKSWorkerNode` in the *AWS Systems Manager Automation runbook reference*.

Unauthorized or access denied (`kubectl`)

If you receive one of the following errors while running `kubectl` commands, then you don't have `kubectl` configured properly for Amazon EKS or the credentials for the IAM principal (role or user) that you're using don't map to a Kubernetes username that has sufficient permissions to Kubernetes objects on your Amazon EKS cluster.

- `could not get token: AccessDenied: Access denied`
- `error: You must be logged in to the server (Unauthorized)`
- `error: the server doesn't have a resource type "svc"`

This could be due to one of the following reasons:

- The cluster was created with credentials for one IAM principal and `kubectl` is configured to use credentials for a different IAM principal. To resolve this, update your `kube config` file to use the credentials that created the cluster. For more information, see the section called "Access cluster with kubectl".
- If your cluster meets the minimum platform requirements in the prerequisites section of Grant IAM users access to Kubernetes with EKS access entries, an access entry doesn't exist with your IAM principal. If it exists, it doesn't have the necessary Kubernetes group names defined for it, or doesn't have the proper access policy associated to it. For more information, see the section called "Grant permissions".

- If your cluster doesn't meet the minimum platform requirements in Grant IAM users access to Kubernetes with EKS access entries, an entry with your IAM principal doesn't exist in the `aws-auth` `ConfigMap`. If it exists, it's not mapped to Kubernetes group names that are bound to a Kubernetes `Role` or `ClusterRole` with the necessary permissions. For more information about Kubernetes role-based authorization (RBAC) objects, see Using RBAC authorization in the Kubernetes documentation. You can view your current `aws-auth` `ConfigMap` entries by replacing *my-cluster* in the following command with the name of your cluster and then running the modified command: `eksctl get iamidentitymapping --cluster my-cluster` . If an entry for with the ARN of your IAM principal isn't in the `ConfigMap`, enter `eksctl create iamidentitymapping --help` in your terminal to learn how to create one.

If you install and configure the AWS CLI, you can configure the IAM credentials that you use. For more information, see Configuring the AWS CLI in the *AWS Command Line Interface User Guide*. You can also configure `kubectl` to use an IAM role, if you assume an IAM role to access Kubernetes objects on your cluster. For more information, see the section called "Access cluster with kubectl".

hostname doesn't match

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

getsockopt: no route to host

Docker runs in the `172.17.0.0/16` CIDR range in Amazon EKS clusters. We recommend that your cluster's VPC subnets do not overlap this range. Otherwise, you will receive the following error:

```
Error: : error upgrading connection: error dialing backend: dial tcp
 172.17.<nn>.<nn>:10250: getsockopt: no route to host
```

Instances failed to join the Kubernetes cluster

If you receive the error `Instances failed to join the Kubernetes cluster` in the AWS Management Console, ensure that either the cluster's private endpoint access is enabled, or that you have correctly configured CIDR blocks for public endpoint access. For more information, see the section called "Configure endpoint access".

Managed node group error codes

If your managed node group encounters a hardware health issue, Amazon EKS returns an error code to help you to diagnose the issue. These health checks don't detect software issues because they are based on Amazon EC2 health checks. The following list describes the error codes.

AccessDenied

Amazon EKS or one or more of your managed nodes is failing to authenticate or authorize with your Kubernetes cluster API server. For more information about resolving a common cause, see the section called "Fixing a common cause of `AccessDenied` errors for managed node groups". Private Windows AMIs can also cause this error code alongside the `Not authorized for images` error message. For more information, see the section called "`Not authorized for images`".

AmiIdNotFound

We couldn't find the AMI ID associated with your launch template. Make sure that the AMI exists and is shared with your account.

AutoScalingGroupNotFound

We couldn't find the Auto Scaling group associated with the managed node group. You may be able to recreate an Auto Scaling group with the same settings to recover.

ClusterUnreachable

Amazon EKS or one or more of your managed nodes is unable to communicate with your Kubernetes cluster API server. This can happen if there are network disruptions or if API servers are timing out processing requests.

Ec2SecurityGroupNotFound

We couldn't find the cluster security group for the cluster. You must recreate your cluster.

Ec2SecurityGroupDeletionFailure

We could not delete the remote access security group for your managed node group. Remove any dependencies from the security group.

Ec2LaunchTemplateNotFound

We couldn't find the Amazon EC2 launch template for your managed node group. You must recreate your node group to recover.

Ec2LaunchTemplateVersionMismatch

The Amazon EC2 launch template version for your managed node group doesn't match the version that Amazon EKS created. You may be able to revert to the version that Amazon EKS created to recover.

IamInstanceProfileNotFound

We couldn't find the IAM instance profile for your managed node group. You may be able to recreate an instance profile with the same settings to recover.

IamNodeRoleNotFound

We couldn't find the IAM role for your managed node group. You may be able to recreate an IAM role with the same settings to recover.

AsgInstanceLaunchFailures

Your Auto Scaling group is experiencing failures while attempting to launch instances.

NodeCreationFailure

Your launched instances are unable to register with your Amazon EKS cluster. Common causes of this failure are insufficient node IAM role permissions or lack of outbound internet access for the nodes. Your nodes must meet either of the following requirements:

- Able to access the internet using a public IP address. The security group associated to the subnet the node is in must allow the communication. For more information, see the section called "Subnet requirements and considerations" and the section called "Security group requirements".
- Your nodes and VPC must meet the requirements in Deploy private clusters with limited internet access.

InstanceLimitExceeded

Your AWS account is unable to launch any more instances of the specified instance type. You may be able to request an Amazon EC2 instance limit increase to recover.

InsufficientFreeAddresses

One or more of the subnets associated with your managed node group doesn't have enough available IP addresses for new nodes.

InternalFailure

These errors are usually caused by an Amazon EKS server-side issue.

Fixing a common cause of AccessDenied errors for managed node groups

The most common cause of `AccessDenied` errors when performing operations on managed node groups is missing the `eks:node-manager` `ClusterRole` or `ClusterRoleBinding`. Amazon EKS sets up these resources in your cluster as part of onboarding with managed node groups, and these are required for managing the node groups.

The `ClusterRole` may change over time, but it should look similar to the following example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: eks:node-manager
rules:
- apiGroups:
  - ''
  resources:
  - pods
  verbs:
  - get
  - list
  - watch
  - delete
- apiGroups:
  - ''
  resources:
  - nodes
  verbs:
  - get
  - list
  - watch
  - patch
- apiGroups:
  - ''
  resources:
  - pods/eviction
  verbs:
  - create
```

The `ClusterRoleBinding` may change over time, but it should look similar to the following example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks:node-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:node-manager
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: eks:node-manager
```

Verify that the `eks:node-manager` `ClusterRole` exists.

```
kubectl describe clusterrole eks:node-manager
```

If present, compare the output to the previous `ClusterRole` example.

Verify that the `eks:node-manager` `ClusterRoleBinding` exists.

```
kubectl describe clusterrolebinding eks:node-manager
```

If present, compare the output to the previous `ClusterRoleBinding` example.

If you've identified a missing or broken `ClusterRole` or `ClusterRoleBinding` as the cause of an `AcessDenied` error while requesting managed node group operations, you can restore them. Save the following contents to a file named *`eks-node-manager-role.yaml`*.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: eks:node-manager
rules:
- apiGroups:
  - ''
  resources:
  - pods
  verbs:
  - get
  - list
```

```
  - watch
  - delete
- apiGroups:
  - ''
  resources:
  - nodes
  verbs:
  - get
  - list
  - watch
  - patch
- apiGroups:
  - ''
  resources:
  - pods/eviction
  verbs:
  - create
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks:node-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:node-manager
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: eks:node-manager
```

Apply the file.

```
kubectl apply -f eks-node-manager-role.yaml
```

Retry the node group operation to see if that resolved your issue.

Not authorized for images

One potential cause of a `Not authorized for images` error message is using a private Amazon EKS Windows AMI to launch Windows managed node groups. After releasing new Windows AMIs, AWS makes AMIs that are older than 4 months private, which makes them no longer accessible.

If your managed node group is using a private Windows AMI, consider updating your Windows managed node group. While we can't guarantee that we can provide access to AMIs that have been made private, you can request access by filing a ticket with AWS Support. For more information, see Patches in the *Amazon EC2 User Guide*.

Node is in NotReady state

If your node enters a `NotReady` status, this likely indicates that the node is unhealthy and unavailable to schedule new Pods. This can occur for various reasons, such as the node lacking sufficient resources for CPU, memory, or available disk space.

For Amazon EKS optimized Windows AMIs, there's no reservation for compute resources specified by default in the `kubelet` configuration. To help prevent resource issues, you can reserve compute resources for system processes by providing the `kubelet` with configuration values for kube-reserved and/or system-reserved. You do this using the `-KubeletExtraArgs` command-line parameter in the bootstrap script. For more information, see Reserve Compute Resources for System Daemons in the Kubernetes documentation and the section called "Bootstrap script configuration parameters" in this user guide.

CNI log collection tool

The Amazon VPC CNI plugin for Kubernetes has its own troubleshooting script that is available on nodes at `/opt/cni/bin/aws-cni-support.sh`. You can use the script to collect diagnostic logs for support cases and general troubleshooting.

Use the following command to run the script on your node:

```
sudo bash /opt/cni/bin/aws-cni-support.sh
```

Note

If the script is not present at that location, then the CNI container failed to run. You can manually download and run the script with the following command:

```
curl -O https://raw.githubusercontent.com/awslabs/amazon-eks-ami/master/log-
collector-script/linux/eks-log-collector.sh
sudo bash eks-log-collector.sh
```

The script collects the following diagnostic information. The CNI version that you have deployed can be earlier than the script version.

```
      This is version 0.6.1. New versions can be found at https://github.com/awslabs/
amazon-eks-ami

Trying to collect common operating system logs...
Trying to collect kernel logs...
Trying to collect mount points and volume information...
Trying to collect SELinux status...
Trying to collect iptables information...
Trying to collect installed packages...
Trying to collect active system services...
Trying to collect Docker daemon information...
Trying to collect kubelet information...
Trying to collect L-IPAMD information...
Trying to collect sysctls information...
Trying to collect networking information...
Trying to collect CNI configuration information...
Trying to collect running Docker containers and gather container data...
Trying to collect Docker daemon logs...
Trying to archive gathered information...

 Done... your bundled logs are located in /var/log/
eks_i-0717c9d54b6cfaa19_2020-03-24_0103-UTC_0.6.1.tar.gz
```

The diagnostic information is collected and stored at:

```
/var/log/eks_i-0717c9d54b6cfaa19_2020-03-24_0103-UTC_0.6.1.tar.gz
```

Container runtime network not ready

You may receive a `Container runtime network not ready` error and authorization errors similar to the following:

```
4191 kubelet.go:2130] Container runtime network not ready: NetworkReady=false
 reason:NetworkPluginNotReady message:docker: network plugin is not ready: cni config
 uninitialized
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
 *v1.Service: Unauthorized
```

```
4191 kubelet_node_status.go:106] Unable to register node
 "ip-10-40-175-122.ec2.internal" with API server: Unauthorized
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
 *v1.Service: Unauthorized
```

This can happen due to one of the following reasons:

1. You either don't have an `aws-auth` `ConfigMap` on your cluster or it doesn't include entries for the IAM role that you configured your nodes with.

   To resolve the issue, view the existing entries in your `ConfigMap` by replacing *`my-cluster`* in the following command with the name of your cluster and then running the modified command: `eksctl get iamidentitymapping --cluster my-cluster`. If you receive an error message from the command, it might be because your cluster doesn't have an `aws-auth` `ConfigMap`. The following command adds an entry to the `ConfigMap`. If the `ConfigMap` doesn't exist, the command also creates it. Replace *`111122223333`* with the AWS account ID for the IAM role and *`myAmazonEKSNodeRole`* with the name of your node's role.

   ```
   eksctl create iamidentitymapping --cluster my-cluster \
       --arn arn:aws:iam::111122223333:role/myAmazonEKSNodeRole --group
    system:bootstrappers,system:nodes \
       --username system:node:{{EC2PrivateDNSName}}
   ```

   The ARN of the role that you specify can't include a path other than `/`. For example, if the name of your role is `development/apps/my-role`, you'd need to change it to `my-role` when specifying the ARN of the role. Make sure that you specify the node IAM role ARN (not the instance profile ARN).

2. Your self-managed nodes are in a cluster with a platform version at the minimum version listed in the prerequisites in the Grant IAM users access to Kubernetes with EKS access entries topic, but an entry isn't listed in the `aws-auth` `ConfigMap` (see previous item) for the node's IAM role or an access entry doesn't exist for the role. To resolve the issue, view your existing access entries by replacing *`my-cluster`* in the following command with the name of your cluster and then running the modified command: `aws eks list-access-entries --cluster-name my-cluster`. The following command adds an access entry for the node's IAM role. Replace *`111122223333`* with the AWS account ID for the IAM role and *`myAmazonEKSNodeRole`* with the name of your node's role. If you have a Windows node, replace *`EC2_LINUX`* with `EC2_Windows`. Make sure that you specify the node IAM role ARN (not the instance profile ARN).

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/myAmazonEKSNodeRole --type EC2_LINUX
```

TLS handshake timeout

When a node is unable to establish a connection to the public API server endpoint, you may see an error similar to the following error.

```
server.go:233] failed to run Kubelet: could not init cloud provider "aws": error
 finding instance i-1111f2222f333e44c: "error listing AWS instances: \"RequestError:
 send request failed\\ncaused by: Post  net/http: TLS handshake timeout\""
```

The `kubelet` process will continually respawn and test the API server endpoint. The error can also occur temporarily during any procedure that performs a rolling update of the cluster in the control plane, such as a configuration change or version update.

To resolve the issue, check the route table and security groups to ensure that traffic from the nodes can reach the public endpoint.

InvalidClientTokenId

If you're using IAM roles for service accounts for a Pod or DaemonSet deployed to a cluster in a China AWS Region, and haven't set the `AWS_DEFAULT_REGION` environment variable in the spec, the Pod or DaemonSet may receive the following error:

```
An error occurred (InvalidClientTokenId) when calling the GetCallerIdentity operation:
 The security token included in the request is invalid
```

To resolve the issue, you need to add the `AWS_DEFAULT_REGION` environment variable to your Pod or DaemonSet spec, as shown in the following example Pod spec.

```
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
```

```
    purpose: demonstrate-envars
spec:
  containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    env:
    - name: AWS_DEFAULT_REGION
      value: "region-code"
```

Node groups must match Kubernetes version before upgrading control plane

Before you upgrade a control plane to a new Kubernetes version, the minor version of the managed and Fargate nodes in your cluster must be the same as the version of your control plane's current version. The Amazon EKS `update-cluster-version` API rejects requests until you upgrade all Amazon EKS managed nodes to the current cluster version. Amazon EKS provides APIs to upgrade managed nodes. For information on upgrading a managed node group's Kubernetes version, see the section called "Update". To upgrade the version of a Fargate node, delete the pod that's represented by the node and redeploy the pod after you upgrade your control plane. For more information, see the section called "Update Kubernetes version".

When launching many nodes, there are `Too Many Requests` errors

If you launch many nodes simultaneously, you may see an error message in the Amazon EC2 user data execution logs that says `Too Many Requests`. This can occur because the control plane is being overloaded with `describeCluster` calls. The overloading results in throttling, nodes failing to run the bootstrap script, and nodes failing to join the cluster altogether.

Make sure that `--apiserver-endpoint`, `--b64-cluster-ca`, and `--dns-cluster-ip` arguments are being passed to the node's bootstrap script. When including these arguments, there's no need for the bootstrap script to make a `describeCluster` call, which helps prevent the control plane from being overloaded. For more information, see the section called "Provide user data to pass arguments to the `bootstrap.sh` file included with an Amazon EKS optimized Linux/Bottlerocket AMI".

HTTP 401 unauthorized error response on Kubernetes API server requests

You see these errors if a Pod's service account token has expired on a cluster.

Your Amazon EKS cluster's Kubernetes API server rejects requests with tokens older than 90 days. In previous Kubernetes versions, tokens did not have an expiration. This means that clients that rely on these tokens must refresh them within an hour. To prevent the Kubernetes API server from rejecting your request due to an invalid token, the Kubernetes client SDK version used by your workload must be the same, or later than the following versions:

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

You can identify all existing Pods in your cluster that are using stale tokens. For more information, see the section called "Service account tokens".

Amazon EKS platform version is more than two versions behind the current platform version

This can happen when Amazon EKS isn't able to automatically update your cluster's platform version. Though there are many causes for this, some of the common causes follow. If any of these problems apply to your cluster, it may still function, its platform version just won't be updated by Amazon EKS.

Problem

The cluster IAM role was deleted – This role was specified when the cluster was created. You can see which role was specified with the following command. Replace *`my-cluster`* with the name of your cluster.

```
aws eks describe-cluster --name my-cluster --query cluster.roleArn --output text | cut
 -d / -f 2
```

An example output is as follows.

```
eksClusterRole
```

Solution

Create a new cluster IAM role with the same name.

Problem

A subnet specified during cluster creation was deleted – The subnets to use with the cluster were specified during cluster creation. You can see which subnets were specified with the following command. Replace *my-cluster* with the name of your cluster.

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

An example output is as follows.

```
[
"subnet-EXAMPLE1",
"subnet-EXAMPLE2"
]
```

Solution

Confirm whether the subnet IDs exist in your account.

```
vpc_id=$(aws eks describe-cluster --name my-cluster --query
 cluster.resourcesVpcConfig.vpcId --output text)
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$vpc_id" --query
 "Subnets[*].SubnetId"
```

An example output is as follows.

```
[
"subnet-EXAMPLE3",
"subnet-EXAMPLE4"
]
```

If the subnet IDs returned in the output don't match the subnet IDs that were specified when the cluster was created, then if you want Amazon EKS to update the cluster, you need to change the subnets used by the cluster. This is because if you specified more than two subnets when you created your cluster, Amazon EKS randomly selects subnets that you specified to create new elastic network interfaces in. These network interfaces enable the control plane to communicate with your nodes. Amazon EKS won't update the cluster if the subnet it selects doesn't exist. You have no control over which of the subnets that you specified at cluster creation that Amazon EKS chooses to create a new network interface in.

When you initiate a Kubernetes version update for your cluster, the update can fail for the same reason.

Problem

A security group specified during cluster creation was deleted – If you specified security groups during cluster creation, you can see their IDs with the following command. Replace *my-cluster* with the name of your cluster.

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

An example output is as follows.

```
[
    "sg-EXAMPLE1"
]
```

If `[]` is returned, then no security groups were specified when the cluster was created and a missing security group isn't the problem. If security groups are returned, then confirm that the security groups exist in your account.

Solution

Confirm whether these security groups exist in your account.

```
vpc_id=$(aws eks describe-cluster --name my-cluster --query
 cluster.resourcesVpcConfig.vpcId --output text)
aws ec2 describe-security-groups --filters "Name=vpc-id,Values=$vpc_id" --query
 "SecurityGroups[*].GroupId"
```

An example output is as follows.

```
[
"sg-EXAMPLE2"
]
```

If the security group IDs returned in the output don't match the security group IDs that were specified when the cluster was created, then if you want Amazon EKS to update the cluster, you need to change the security groups used by the cluster. Amazon EKS won't update a cluster if the security group IDs specified at cluster creation don't exist.

When you initiate a Kubernetes version update for your cluster, the update can fail for the same reason.

- You don't have at least six (though we recommend 16) available IP addresses in each of the subnets that you specified when you created your cluster. If you don't have enough available IP addresses in the subnet, you either need to free up IP addresses in the subnet or you need to change the subnets used by the cluster to use subnets with enough available IP addresses.
- You enabled secrets encryption when you created your cluster and the AWS KMS key that you specified has been deleted. If you want Amazon EKS to update the cluster, you need to create a new cluster

Cluster health FAQs and error codes with resolution paths

Amazon EKS detects issues with your EKS clusters and the cluster infrastructure and stores it in the *cluster health*. You can detect, troubleshoot, and address cluster issues more rapidly with the aid of cluster health information. This enables you to create application environments that are more secure and up-to-date. Additionally, it may be impossible for you to upgrade to newer versions of Kubernetes or for Amazon EKS to install security updates on a degraded cluster as a result of issues with the necessary infrastructure or cluster configuration. Amazon EKS can take 3 hours to detect issues or detect that an issue is resolved.

The health of an Amazon EKS cluster is a shared responsibility between Amazon EKS and its users. You are responsible for the prerequisite infrastructure of IAM roles and Amazon VPC subnets, as well as other necessary infrastructure, that must be provided in advance. Amazon EKS detects changes in the configuration of this infrastructure and the cluster.

To access your health of your cluster in the Amazon EKS console, look for a section called **Health Issues** in the **Overview** tab of the Amazon EKS cluster detail page. This data will be also be available by calling the `DescribeCluster` action in the EKS API, for example from within the AWS Command Line Interface.

Why should I use this feature?

You will get increased visibility into the health of your Amazon EKS cluster, quickly diagnose and fix any issues, without needing to spend time debugging or opening AWS support cases. For example: you accidentally deleted a subnet for the Amazon EKS cluster, Amazon EKS won't be able to create cross account network interfaces and Kubernetes AWS CLI commands such as `kubectl` exec or `kubectl` logs. These will fail with the error: `Error from server: error dialing backend: remote error: tls: internal error`. Now you will see an Amazon EKS health issue that says: `subnet-da60e280 was deleted: could not create network interface`.

How does this feature relate or work with other AWS services?

IAM roles and Amazon VPC subnets are two examples of prerequisite infrastructure that cluster health detects issues with. This feature will return detailed information if those resources are not configured properly.

Does a cluster with health issues incur charges?

Yes, every Amazon EKS cluster is billed at the standard Amazon EKS pricing. The *cluster health* feature is available at no additional charge.

Does this feature work with Amazon EKS clusters on AWS Outposts?

Yes, cluster issues are detected for EKS clusters in the AWS Cloud including *extended clusters* on AWS Outposts and *local clusters* on AWS Outposts. Cluster health doesn't detect issues with Amazon EKS Anywhere or Amazon EKS Distro (EKS-D).

Can I get notified when new issues are detected?

Yes. AWS sends an email and Personal Health Dashboard notification when new Cluster Health issues are detected.

Does the console give me warnings for health issues?

Yes, any cluster with health issues will include a banner at the top of the console.

The first two columns are what are needed for API response values. The third field of the Health ClusterIssue object is resourceIds, the return of which is dependent on the issue type.

Code	Message	ResourceIds	Cluster Recoverable?
SUBNET_NOT_FOUND	We couldn't find one or more subnets currently associated with your cluster. Call Amazon EKS update-cluster-config API to update subnets.	Subnet Ids	Yes
SECURITY_GROUP_NOT_FOUND	We couldn't find one or more security groups currently associated with your cluster. Call Amazon EKS update-cluster-config API to update security groups	Security group Ids	Yes
IP_NOT_AVAILABLE	One or more of the subnets associated with your cluster does not have enough available IP addresses for Amazon EKS to perform cluster management operations. Free up addresses in the subnet(s), or associate different subnets to your cluster using the Amazon EKS update-cluster-config API.	Subnet Ids	Yes

Code	Message	ResourceIds	Cluster Recoverable?
VPC_NOT_FOUND	We couldn’t find the VPC associated with your cluster. You must delete and recreate your cluster.	VPC id	No
ASSUME_ROLE_ACCESS_DENIED	Your cluster is not using the Amazon EKS service-linked-role. We couldn’t assume the role associated with your cluster to perform required Amazon EKS management operations. Check the role exists and has the required trust policy.	The cluster IAM role	Yes

Code	Message	ResourceIds	Cluster Recoverable?
PERMISSIO N_ACCESS_DENIED	Your cluster is not using the Amazon EKS service-linked-rol e. The role associate d with your cluster does not grant sufficient permissio ns for Amazon EKS to perform required managemen t operations. Check the policies attached to the cluster role and if any separate deny policies are applied.	The cluster IAM role	Yes
ASSUME_RO LE_ACCESS _DENIED_USING_SLR	We couldn't assume the Amazon EKS cluster managemen t service-linked-role. Check the role exists and has the required trust policy.	The Amazon EKS service-linked-role	Yes

Code	Message	ResourceIds	Cluster Recoverable?
PERMISSION_ACCESS_DENIED_USING_SLR	The Amazon EKS cluster management service-linked-role does not grant sufficient permissions for Amazon EKS to perform required management operations. Check the policies attached to the cluster role and if any separate deny policies are applied.	The Amazon EKS service-linked-role	Yes
OPT_IN_REQUIRED	Your account doesn't have an Amazon EC2 service subscription. Update your account subscriptions in your account settings page.	N/A	Yes
STS_REGIONAL_ENDPOINT_DISABLED	The STS regional endpoint is disabled. Enable the endpoint for Amazon EKS to perform required cluster management operations.	N/A	Yes

Code	Message	ResourceIds	Cluster Recoverable?
KMS_KEY_DISABLED	The AWS KMS Key associated with your cluster is disabled. Re-enable the key to recover your cluster.	The KMS Key Arn	Yes
KMS_KEY_N OT_FOUND	We couldn’t find the AWS KMS key associated with your cluster. You must delete and recreate the cluster.	The KMS Key ARN	No
KMS_GRANT _REVOKED	Grants for the AWS KMS Key associated with your cluster are revoked. You must delete and recreate the cluster.	The KMS Key Arn	No

Connect a Kubernetes cluster to an Amazon EKS Management Console with Amazon EKS Connector

You can use Amazon EKS Connector to register and connect any conformant Kubernetes cluster to AWS and visualize it in the Amazon EKS console. After a cluster is connected, you can see the status, configuration, and workloads for that cluster in the Amazon EKS console. You can use this feature to view connected clusters in Amazon EKS console, but you can't manage them. The Amazon EKS Connector requires an agent that is an open source project on Github. For additional technical content, including frequently asked questions and troubleshooting, see the section called "Troubleshoot EKS Connector".

The Amazon EKS Connector can connect the following types of Kubernetes clusters to Amazon EKS.

- On-premises Kubernetes clusters
- Self-managed clusters that are running on Amazon EC2
- Managed clusters from other cloud providers

Amazon EKS Connector considerations

Before you use Amazon EKS Connector, understand the following:

- You must have administrative privileges to the Kubernetes cluster to connect the cluster to Amazon EKS.
- The Kubernetes cluster must have Linux 64-bit (x86) worker nodes present before connecting. ARM worker nodes aren't supported.
- You must have worker nodes in your Kubernetes cluster that have outbound access to the `ssm.` and `ssmmessages.` Systems Manager endpoints. For more information, see Systems Manager endpoints in the *AWS General Reference*.
- By default, you can connect up to 10 clusters in a Region. You can request an increase through the service quota console. See Requesting a quota increase for more information.
- Only the Amazon EKS `RegisterCluster`, `ListClusters`, `DescribeCluster`, and `DeregisterCluster` APIs are supported for external Kubernetes clusters.
- You must have the following permissions to register a cluster:

  - eks:RegisterCluster
  - ssm:CreateActivation
  - ssm:DeleteActivation
  - iam:PassRole
- You must have the following permissions to deregister a cluster:
  - eks:DeregisterCluster
  - ssm:DeleteActivation
  - ssm:DeregisterManagedInstance

Required IAM roles for Amazon EKS Connector

Using the Amazon EKS Connector requires the following two IAM roles:

- The Amazon EKS Connector service-linked role is created when you register a cluster for the first time.
- You must create the Amazon EKS Connector agent IAM role. See the section called "Connector IAM role" for details.

To enable cluster and workload view permission for IAM principals, apply the `eks-connector` and Amazon EKS Connector cluster roles to your cluster. Follow the steps in Grant access to view Kubernetes cluster resources on an Amazon EKS console.

Connect an external Kubernetes cluster to the Amazon EKS Management Console

You can connect an external Kubernetes cluster to Amazon EKS by using multiple methods in the following process. This process involves two steps: Registering the cluster with Amazon EKS and installing the `eks-connector` agent in the cluster.

> **Important**
>
> You must complete the second step within 3 days of completing the first step, before the registration expires.

Considerations

You can use YAML manifests when installing the agent. Alternatively, you can use Helm if you register the cluster with the AWS Management Console or AWS Command Line Interface. However, you cannot use Helm to install the agent if you register the cluster with `eksctl`.

Prerequisites

- Ensure the Amazon EKS Connector agent role was created. Follow the steps in Creating the Amazon EKS connector agent role.
- You must have the following permissions to register a cluster:
  - `eks:RegisterCluster`
  - `ssm:CreateActivation`
  - `ssm:DeleteActivation`
  - `iam:PassRole`

Step 1: Registering the cluster

To register a cluster to Amazon EKS connector, you can use one of these tools:

- the section called "AWS CLI"
- the section called "AWS Management Console"
- the section called "`eksctl`"

AWS CLI

1. AWS CLI must be installed. To install or upgrade it, see Installing the AWS CLI.
2. For the Connector configuration, specify your Amazon EKS Connector agent IAM role. For more information, see the section called "Required IAM roles for Amazon EKS Connector".

```
aws eks register-cluster \
     --name my-first-registered-cluster \
     --connector-config roleArn=arn:aws:iam::111122223333:role/
AmazonEKSConnectorAgentRole,provider="OTHER" \
     --region aws-region
```

An example output is as follows.


```
{
    "cluster": {
        "name": "my-first-registered-cluster",
        "arn": "arn:aws:eks:region:111122223333:cluster/my-first-registered-cluster",
        "createdAt": 1627669203.531,
        "ConnectorConfig": {
            "activationId": "xxxxxxxxACTIVATION_IDxxxxxxxx",
            "activationCode": "xxxxxxxxACTIVATION_CODExxxxxxxx",
            "activationExpiry": 1627672543.0,
            "provider": "OTHER",
            "roleArn": "arn:aws:iam::111122223333:role/AmazonEKSConnectorAgentRole"
        },
        "status": "CREATING"
    }
}
```


You use the `aws-region`, `activationId`, and `activationCode` values in the next step.

AWS Management Console

1. Open the Amazon EKS console.
2. Choose **Add cluster** and select **Register** to bring up the configuration page.
3. On the **Configure cluster** section, fill in the following fields:
   - **Name** – A unique name for your cluster.
   - **Provider** – Choose to display the dropdown list of Kubernetes cluster providers. If you don't know the specific provider, select **Other**.
   - **EKS Connector role** – Select the role to use for connecting the cluster.
4. Select **Register cluster**.
5. The Cluster overview page displays. If you want to use the Helm chart, copy the `helm install` command and continue to the next step. If you want to use the YAML manifest, choose **Download YAML file** to download the manifest file to your local drive.

> **Important**
>
> This is your only opportunity to copy the `helm install` command or download this file. Don't navigate away from this page, as the link will not be accessible and you must deregister the cluster and start the steps from the beginning.

The command or manifest file can be used only once for the registered cluster. If you delete resources from the Kubernetes cluster, you must re-register the cluster and obtain a new manifest file.

Continue to the next step to apply the manifest file to your Kubernetes cluster.

`eksctl`

1. `eksctl` version `0.68` or later must be installed. To install or upgrade it, see the section called "Create cluster (eksctl)".
2. Register the cluster by providing a name, provider, and region.

```
eksctl register cluster --name my-cluster --provider my-provider --region region-code
```

   Example output:

```
2021-08-19 13:47:26 [#]  creating IAM role "eksctl-20210819194112186040"
2021-08-19 13:47:26 [#]  registered cluster "<name>" successfully
2021-08-19 13:47:26 [#]  wrote file eks-connector.yaml to <current directory>
2021-08-19 13:47:26 [#]  wrote file eks-connector-clusterrole.yaml to <current
 directory>
2021-08-19 13:47:26 [#]  wrote file eks-connector-console-dashboard-full-access-
group.yaml to <current directory>
2021-08-19 13:47:26 [!]  note: "eks-connector-clusterrole.yaml" and "eks-connector-
console-dashboard-full-access-group.yaml" give full EKS Console access to IAM
 identity "<aws-arn>", edit if required; read https://eksctl.io/usage/eks-connector
 for more info
2021-08-19 13:47:26 [#]  run `kubectl apply -f eks-connector.yaml,eks-connector-
clusterrole.yaml,eks-connector-console-dashboard-full-access-group.yaml` before
 expiry> to connect the cluster
```

This creates files on your local computer. These files must be applied to the external cluster within 3 days, or the registration expires.

3. In a terminal that can access the cluster, apply the `eks-connector-binding.yaml` file:

```
kubectl apply -f eks-connector-binding.yaml
```

Step 2: Installing the `eks-connector` agent

To install the `eks-connector` agent, use one of the following tools:

- the section called “Helm”
- the section called “yaml”

Helm

> **Note**
>
> If you registered the cluster with `eksctl`, use the YAML manifest method instead of the Helm chart method.

1. If you used the AWS CLI in the previous step, replace the `ACTIVATION_CODE` and `ACTIVATION_ID` in the following command with the `activationId`, and `activationCode` values respectively. Replace the `aws-region` with the AWS Region that you used in the previous step. Then run the command to install the `eks-connector` agent on the registering cluster:

```
$ helm install eks-connector \
  --namespace eks-connector \
  oci://public.ecr.aws/eks-connector/eks-connector-chart \
  --set eks.activationCode=ACTIVATION_CODE \
  --set eks.activationId=ACTIVATION_ID \
  --set eks.agentRegion=aws-region
```

If you used the AWS Management Console in the previous step, use the command that you copied from the previous step that has these values filled in.

2. Check the healthiness of the installed `eks-connector` deployment and wait for the status of the registered cluster in Amazon EKS to be `ACTIVE`.

yaml

Complete the connection by applying the Amazon EKS Connector manifest file to your Kubernetes cluster. To do this, you must use the methods described previously. If the manifest isn't applied within three days, the Amazon EKS Connector registration expires. If the cluster connection expires, the cluster must be deregistered before connecting the cluster again.

1. Download the Amazon EKS Connector YAML file.

```
curl -O https://amazon-eks.s3.us-west-2.amazonaws.com/eks-connector/manifests/eks-connector/latest/eks-connector.yaml
```

2. Edit the Amazon EKS Connector YAML file to replace all references of `%AWS_REGION%`, `%EKS_ACTIVATION_ID%`, `%EKS_ACTIVATION_CODE%` with the `aws-region`, `activationId`, and `activationCode` from the output of the previous step.

   The following example command can replace these values.

```
sed -i "s~%AWS_REGION%~$aws-region~g; s~%EKS_ACTIVATION_ID%~$EKS_ACTIVATION_ID~g; s~%EKS_ACTIVATION_CODE%~$(echo -n $EKS_ACTIVATION_CODE | base64)~g" eks-connector.yaml
```

> **Important**
>
> Ensure that your activation code is in the base64 format.

3. In a terminal that can access the cluster, you can apply the updated manifest file by running the following command:

```
kubectl apply -f eks-connector.yaml
```

4. After the Amazon EKS Connector manifest and role binding YAML files are applied to your Kubernetes cluster, confirm that the cluster is now connected.

```
aws eks describe-cluster \
     --name "my-first-registered-cluster" \
     --region AWS_REGION
```

The output should include `status=ACTIVE`.

5. (Optional) Add tags to your cluster. For more information, see the section called "Tagging your resources".

Next steps

If you have any issues with these steps, see the section called "Troubleshoot EKS Connector".

To grant additional IAM principals access to the Amazon EKS console to view Kubernetes resources in a connected cluster, see the section called "Grant access to clusters".

Grant access to view Kubernetes cluster resources on an Amazon EKS console

Grant IAM principals access to the Amazon EKS console to view information about Kubernetes resources running on your connected cluster.

Prerequisites

The IAM principal that you use to access the AWS Management Console must meet the following requirements:

- It must have the `eks:AccessKubernetesApi` IAM permission.
- The Amazon EKS Connector service account can impersonate the IAM principal in the cluster. This allows the Amazon EKS Connector to map the IAM principal to a Kubernetes user.

To create and apply the Amazon EKS Connector cluster role

1. Download the `eks-connector` cluster role template.

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-clusterrole.yaml
```

2. Edit the cluster role template YAML file. Replace references of `%IAM_ARN%` with the Amazon Resource Name (ARN) of your IAM principal.
3. Apply the Amazon EKS Connector cluster role YAML to your Kubernetes cluster.

```
kubectl apply -f eks-connector-clusterrole.yaml
```

For an IAM principal to view Kubernetes resources in Amazon EKS console, the principal must be associated with a Kubernetes `role` or `clusterrole` with necessary permissions to read the resources. For more information, see [Using RBAC Authorization] in the Kubernetes documentation.

To configure an IAM principal to access the connected cluster

1. You can download either of these example manifest files to create a `clusterrole` and `clusterrolebinding` or a `role` and `rolebinding`, respectively:

   View Kubernetes resources in all namespaces

   - The `eks-connector-console-dashboard-full-access-clusterrole` cluster role gives access to all namespaces and resources that can be visualized in the console. You can change the name of the `role`, `clusterrole` and their corresponding binding before applying it to your cluster. Use the following command to download a sample file.

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/
     eks-connector-console-roles/eks-connector-console-dashboard-full-access-
     group.yaml
     ```

   View Kubernetes resources in a specific namespace

   - The namespace in this file is `default`, so if you want to specify a different namespace, edit the file before applying it to your cluster. Use the following command to download a sample file.

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/
     eks-connector-console-roles/eks-connector-console-dashboard-restricted-access-
     group.yaml
     ```

2. Edit the full access or restricted access YAML file to replace references of `%IAM_ARN%` with the Amazon Resource Name (ARN) of your IAM principal.

3. Apply the full access or restricted access YAML files to your Kubernetes cluster. Replace the YAML file value with your own.

   ```
   kubectl apply -f eks-connector-console-dashboard-full-access-group.yaml
   ```

To view Kubernetes resources in your connected cluster, see the section called “Access cluster resources”. Data for some resource types on the **Resources** tab isn't available for connected clusters.

Deregister a Kubernetes cluster from the Amazon EKS console

If you are finished using a connected cluster, you can deregister it. After it's deregistered, the cluster is no longer visible in the Amazon EKS console.

You must have the following permissions to call the deregisterCluster API:

- `eks:DeregisterCluster`
- `ssm:DeleteActivation`
- `ssm:DeregisterManagedInstance`

This process involves two steps: Deregistering the cluster with Amazon EKS and uninstalling the eks-connector agent in the cluster.

Deregister the Kubernetes cluster

To deregister a cluster from Amazon EKS connector, you can use one of these tools:

- the section called “AWS CLI”
- the section called “AWS Management Console”
- the section called “`eksctl`”

AWS CLI

1. AWS CLI must be installed. To install or upgrade it, see Installing the AWS CLI.
2. Ensure the Amazon EKS Connector agent role was created.
3. Deregister the connected cluster.

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

AWS Management Console

1. Open the Amazon EKS console.
2. Choose **Clusters**.
3. On the **Clusters** page, select the connected cluster and select **Deregister**.
4. Confirm that you want to deregister the cluster.

eksctl

1. Install `eksctl` version `0.68` or later. To install or upgrade it, see the section called "Create cluster (eksctl)".
2. Ensure the Amazon EKS Connector agent role was created.
3. Deregister the connected cluster:

```
eksctl deregister cluster --name my-cluster
```

Clean up the resources in your Kubernetes cluster

To uninstall the `eks-connector` agent, use one of the following tools:

- the section called "helm"
- the section called "yaml"

helm

Run the following command to uninstall the agent.

```
helm -n eks-connector uninstall eks-connector
```

yaml

1. Delete the Amazon EKS Connector YAML file from your Kubernetes cluster.

```
kubectl delete -f eks-connector.yaml
```

2. If you created `clusterrole` or `clusterrolebindings` for additional IAM principals to access the cluster, delete them from your Kubernetes cluster.

Troubleshoot Amazon EKS Connector issues

This topic covers some of the common errors that you might encounter while using the Amazon EKS Connector, including instructions on how to resolve them and workarounds.

Basic troubleshooting

This section describes steps to diagnose Amazon EKS Connector issues.

Check Amazon EKS Connector status

To check the Amazon EKS Connector status, type:

```
kubectl get pods -n eks-connector
```

Inspect Amazon EKS Connector logs

The Amazon EKS Connector Pod consists of three containers. To retrieve full logs for all of these containers so that you can inspect them, run the following commands:

- `connector-init`

```
kubectl logs eks-connector-0 --container connector-init -n eks-connector
kubectl logs eks-connector-1 --container connector-init -n eks-connector
```

- `connector-proxy`

```
kubectl logs eks-connector-0 --container connector-proxy -n eks-connector
kubectl logs eks-connector-1 --container connector-proxy -n eks-connector
```

- `connector-agent`

```
kubectl exec eks-connector-0 --container connector-agent -n eks-connector -- cat /
var/log/amazon/ssm/amazon-ssm-agent.log
kubectl exec eks-connector-1 --container connector-agent -n eks-connector -- cat /
var/log/amazon/ssm/amazon-ssm-agent.log
```

Get the effective cluster name

Amazon EKS clusters are uniquely identified by `clusterName` within a single AWS account and AWS Region. If you have multiple connected clusters in Amazon EKS, you can confirm which Amazon EKS cluster that the current Kubernetes cluster is registered to. To do this, enter the following to find out the `clusterName` of the current cluster.

```
kubectl exec eks-connector-0 --container connector-agent -n eks-connector \
  -- cat /var/log/amazon/ssm/amazon-ssm-agent.log | grep -m1 -oE "eks_c:[a-zA-Z0-9_-]+"
 | sed -E "s/^.*eks_c:([a-zA-Z0-9_-]+)_[a-zA-Z0-9]+.*$/\1/"
kubectl exec eks-connector-1 --container connector-agent -n eks-connector \
  -- cat /var/log/amazon/ssm/amazon-ssm-agent.log | grep -m1 -oE "eks_c:[a-zA-Z0-9_-]+"
 | sed -E "s/^.*eks_c:([a-zA-Z0-9_-]+)_[a-zA-Z0-9]+.*$/\1/"
```

Miscellaneous commands

The following commands are useful to retrieve information that you need to troubleshoot issues.

- Use the following command to gather images that's used by Pods in Amazon EKS Connector.

  ```
  kubectl get pods -n eks-connector -o jsonpath="{.items[*].spec.containers[*].image}"
   | tr -s '[[:space:]]' '\n'
  ```

- Use the following command to determine the node names that Amazon EKS Connector is running on.

  ```
  kubectl get pods -n eks-connector -o jsonpath="{.items[*].spec.nodeName}" | tr -s
   '[[:space:]]' '\n'
  ```

- Run the following command to get your Kubernetes client and server versions.

  ```
  kubectl version
  ```

- Run the following command to get information about your nodes.

  ```
  kubectl get nodes -o wide --show-labels
  ```

Helm issue: 403 Forbidden

If you received the following error when running helm install commands:

```
Error: INSTALLATION FAILED: unexpected status from HEAD request to https://
public.ecr.aws/v2/eks-connector/eks-connector-chart/manifests/0.0.6: 403 Forbidden
```

You can run the following line to fix it:

```
docker logout public.ecr.aws
```

Console error: the cluster is stuck in the Pending state

If the cluster gets stuck in the `Pending` state on the Amazon EKS console after you're registered it, it might be because the Amazon EKS Connector didn't successfully connect the cluster to AWS yet. For a registered cluster, the `Pending` state means that the connection isn't successfully established. To resolve this issue, make sure that you have applied the manifest to the target Kubernetes cluster. If you applied it to the cluster, but the cluster is still in the `Pending` state, then the `eks-connector` statefulset might be unhealthy. To troubleshoot this issue, see the section called "Amazon EKS connector Pods are crash looping"in this topic.

Console error: User system:serviceaccount:eks-connector:eks-connector can't impersonate resource users in API group at cluster scope

The Amazon EKS Connector uses Kubernetes user impersonation to act on behalf of IAM principals from the AWS Management Console. Each principal that accesses the Kubernetes API from the AWS `eks-connector` service account must be granted permission to impersonate the corresponding Kubernetes user with an IAM ARN as its Kubernetes user name. In the following examples, the IAM ARN is mapped to a Kubernetes user.

- IAM user *john* from AWS account *111122223333* is mapped to a Kubernetes user. IAM best practices recommend that you grant permissions to roles instead of users.

  ```
                  arn:aws:iam::111122223333:user/john
  ```

- IAM role *admin* from AWS account *111122223333* is mapped to a Kubernetes user:

  ```
                  arn:aws:iam::111122223333:role/admin
  ```

  The result is an IAM role ARN, instead of the AWS STS session ARN.

For instructions on how to configure the `ClusterRole` and `ClusterRoleBinding` to grant the `eks-connector` service account privilege to impersonate the mapped user, see the section called "Grant access to clusters". Make sure that in the template, `%IAM_ARN%` is replaced with the IAM ARN of the AWS Management Console IAM principal.

Console error: [...] is forbidden: User [...] cannot list resource [...] in API group at the cluster scope

Consider the following problem. The Amazon EKS Connector has successfully impersonated the requesting AWS Management Console IAM principal in the target Kubernetes cluster. However, the impersonated principal doesn't have RBAC permission for Kubernetes API operations.

To resolve this issue, there are two methods to give permissions to additional users. If you previously installed eks-connector via helm chart, you can easily grant users access by running the following command. Replace the `userARN1` and `userARN2` with a list of the ARNs of the IAM roles to give access to view the Kubernetes resources:

```
helm upgrade eks-connector oci://public.ecr.aws/eks-connector/eks-connector-chart \
    --reuse-values \
    --set 'authentication.allowedUserARNs={userARN1,userARN2}'
```

Or, as the cluster administrator, grant the appropriate level of RBAC privileges to individual Kubernetes users. For more information and examples, see the section called "Grant access to clusters".

Console error: Amazon EKS can't communicate with your Kubernetes cluster API server. The cluster must be in an ACTIVE state for successful connection. Try again in few minutes.

If the Amazon EKS service can't communicate with the Amazon EKS connector in the target cluster, it might be because of one of the following reasons:

- The Amazon EKS Connector in the target cluster is unhealthy.
- Poor connectivity or an interrupted connection between the target cluster and the AWS Region.

To resolve this problem, check the Amazon EKS Connector logs. If you don't see an error for the Amazon EKS Connector, retry the connection after a few minutes. If you regularly experience high latency or intermittent connectivity for the target cluster, consider re-registering the cluster to an AWS Region that's located closer to you.

Amazon EKS connector Pods are crash looping

There are many reasons that can cause an Amazon EKS connector Pod to enter the `CrashLoopBackOff` status. This issue likely involves the `connector-init` container. Check the status of the Amazon EKS connector Pod.

```
kubectl get pods -n eks-connector
```

An example output is as follows.

```
NAME              READY   STATUS                  RESTARTS   AGE
eks-connector-0   0/2     Init:CrashLoopBackOff   1          7s
```

If your output is similar to the previous output, see the section called "Inspect Amazon EKS Connector logs" to troubleshoot the issue.

Failed to initiate eks-connector: InvalidActivation

When you start the Amazon EKS Connector for the first time, it registers an `activationId` and `activationCode` with Amazon Web Services. The registration might fail, which can cause the `connector-init` container to crash with an error similar to the following error.

```
F1116 20:30:47.261469       1 init.go:43] failed to initiate eks-connector:
 InvalidActivation:
```

To troubleshoot this issue, consider the following causes and recommended fixes:

- Registration might have failed because the `activationId` and `activationCode` aren't in your manifest file. If this is the case, make sure that they are the correct values that were returned from the `RegisterCluster` API operation, and that the `activationCode` is in the manifest file. The `activationCode` is added to Kubernetes secrets, so it must be `base64` encoded. For more information, see the section called "Step 1: Registering the cluster".
- Registration might have failed because your activation expired. This is because, for security reasons, you must activate the Amazon EKS Connector within three days after registering the cluster. To resolve this issue, make sure that the Amazon EKS Connector manifest is applied to

the target Kubernetes cluster before the expiry date and time. To confirm your activation expiry date, call the `DescribeCluster` API operation.

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

In the following example response, the expiry date and time is recorded as `2021-11-12T22:28:51.101000-08:00`.

```
{
    "cluster": {
        "name": "my-cluster",
        "arn": "arn:aws:eks:region:111122223333:cluster/my-cluster",
        "createdAt": "2021-11-09T22:28:51.449000-08:00",
        "status": "FAILED",
        "tags": {
        },
        "connectorConfig": {
            "activationId": "00000000-0000-0000-0000-000000000000",
            "activationExpiry": "2021-11-12T22:28:51.101000-08:00",
            "provider": "OTHER",
            "roleArn": "arn:aws:iam::111122223333:role/my-connector-role"
        }
    }
}
```

If the `activationExpiry` passed, deregister the cluster and register it again. Doing this generates a new activation.

Cluster node is missing outbound connectivity

To work properly, the Amazon EKS Connector requires outbound connectivity to several AWS endpoints. You can't connect a private cluster without outbound connectivity to a target AWS Region. To resolve this issue, you must add the necessary outbound connectivity. For information about connector requirements, see the section called "Amazon EKS Connector considerations".

Amazon EKS connector Pods are in ImagePullBackOff state

If you run the `get pods` command and Pods are in the `ImagePullBackOff` state, they can't work properly. If the Amazon EKS Connector Pods are in the `ImagePullBackOff` state, they can't work properly. Check the status of your Amazon EKS Connector Pods.

```
kubectl get pods -n eks-connector
```

An example output is as follows.

```
NAME              READY   STATUS                  RESTARTS   AGE
eks-connector-0   0/2     Init:ImagePullBackOff   0          4s
```

The default Amazon EKS Connector manifest file references images from the [Amazon ECR Public Gallery](). It's possible that the target Kubernetes cluster can't pull images from the Amazon ECR Public Gallery. Either resolve the Amazon ECR Public Gallery image pull issue, or consider mirroring the images in the private container registry of your choice.

AWS Connector frequently asked questions

Q: How does the underlying technology behind the Amazon EKS Connector work?

A: The Amazon EKS Connector is based on the AWS Systems Manager (Systems Manager) agent. The Amazon EKS Connector runs as a `StatefulSet` on your Kubernetes cluster. It establishes a connection and proxies the communication between the API server of your cluster and Amazon Web Services. It does this to display cluster data in the Amazon EKS console until you disconnect the cluster from AWS. The Systems Manager agent is an open source project. For more information about this project, see the [GitHub project page]().

Q: I have an on-premises Kubernetes cluster that I want to connect. Do I need to open firewall ports to connect it?

A: No, you don't need to open any firewall ports. The Kubernetes cluster only requires outbound connection to AWS Regions. AWS services never access resources in your on-premises network. The Amazon EKS Connector runs on your cluster and initiates the connection to AWS. When the cluster registration completes, AWS only issues commands to the Amazon EKS Connector after you start an action from the Amazon EKS console that requires information from the Kubernetes API server on your cluster.

Q: What data is sent from my cluster to AWS by the Amazon EKS Connector?

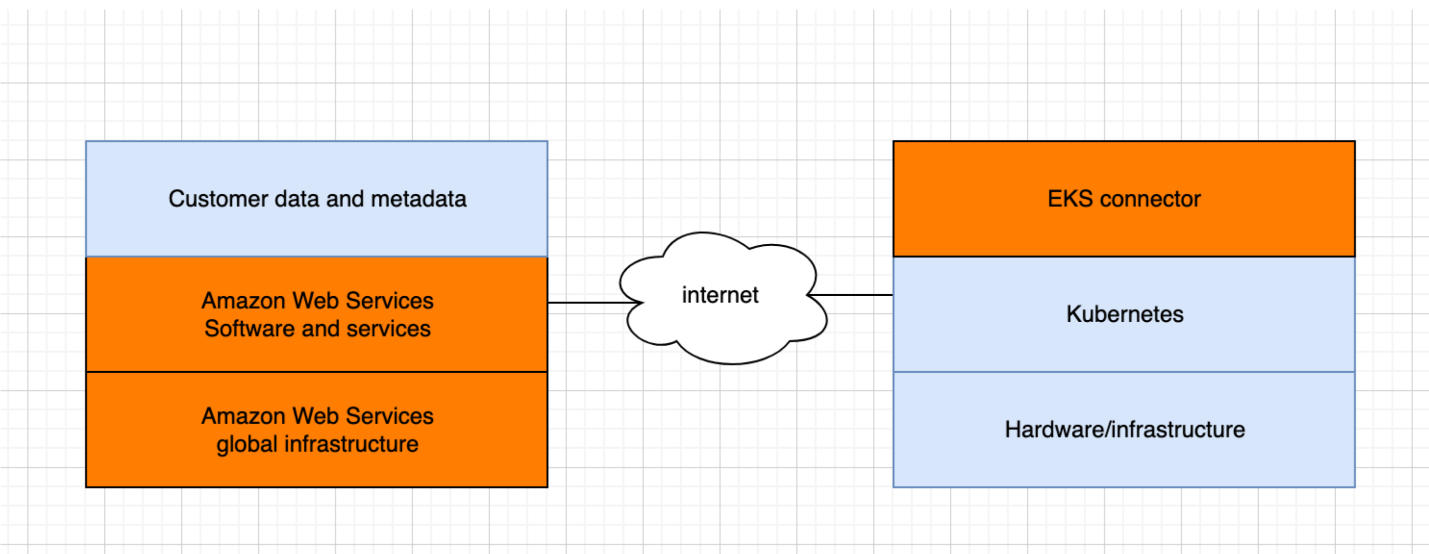
A: The Amazon EKS Connector sends technical information that's necessary for your cluster to be registered on AWS. It also sends cluster and workload metadata for the Amazon EKS console features that customers request. The Amazon EKS Connector only gathers or sends this data if you start an action from the Amazon EKS console or the Amazon EKS API that necessitates the data to be sent to AWS. Other than the Kubernetes version number, AWS doesn't store any data by default. It stores data only if you authorize it to.

Q: Can I connect a cluster outside of an AWS Region?

A: Yes, you can connect a cluster from any location to Amazon EKS. Moreover, your Amazon EKS service can be located in any AWS public commercial AWS Region. This works with a valid network connection from your cluster to the target AWS Region. We recommend that you pick an AWS Region that is closest to your cluster location for UI performance optimization. For example, if you have a cluster running in Tokyo, connect your cluster to the AWS Region in Tokyo (that is, the `ap-northeast-1` AWS Region) for low latency. You can connect a cluster from any location to Amazon EKS in any of the public commercial AWS Regions, except the China or GovCloud AWS Regions.

Understand security in Amazon EKS Connector

The Amazon EKS Connector is an open source component that runs on your Kubernetes cluster. This cluster can be located outside of the AWS environment. This creates additional considerations for security responsibilities. This configuration can be illustrated by the following diagram. Orange represents AWS responsibilities, and blue represents customer responsibilities:



This topic describes the differences in the responsibility model if the connected cluster is outside of AWS.

AWS responsibilities

- Maintaining, building, and delivering Amazon EKS Connector, which is an open source component that runs on a customer's Kubernetes cluster and communicates with AWS.
- Maintaining transport and application layer communication security between the connected Kubernetes cluster and AWS services.

Customer responsibilities

- Kubernetes cluster specific security, specifically along the following lines:
  - Kubernetes secrets must be properly encrypted and protected.
  - Lock down access to the `eks-connector` namespace.
- Configuring role-based access control (RBAC) permissions to manage IAM principal access from AWS. For instructions, see the section called "Grant access to clusters".
- Installing and upgrading Amazon EKS Connector.
- Maintaining the hardware, software, and infrastructure that supports the connected Kubernetes cluster.
- Securing their AWS accounts (for example, through safeguarding your root user credentials).

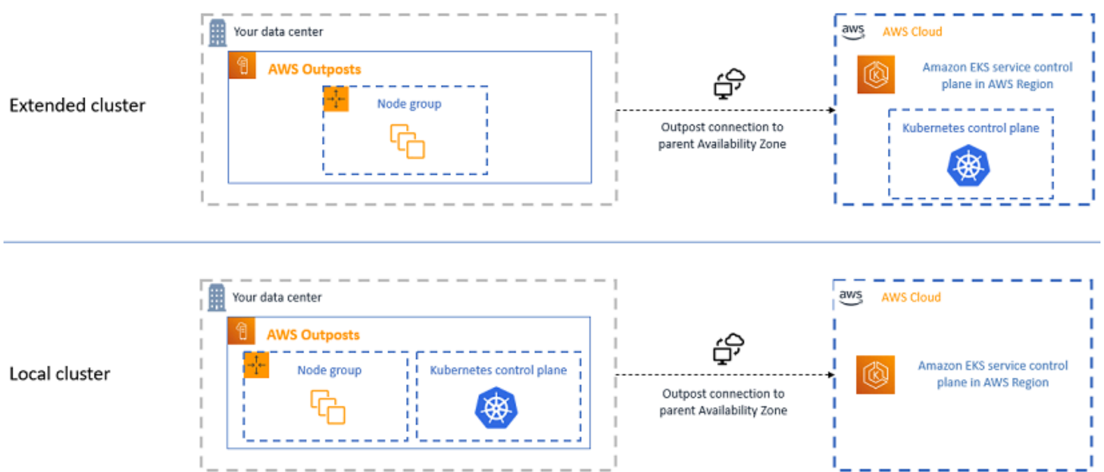
Deploy Amazon EKS on-premises with AWS Outposts

You can use Amazon EKS to run on-premises Kubernetes applications on AWS Outposts. You can deploy Amazon EKS on Outposts in the following ways:

- **Extended clusters** – Run the Kubernetes control plane in an AWS Region and nodes on your Outpost.
- **Local clusters** – Run the Kubernetes control plane and nodes on your Outpost.

For both deployment options, the Kubernetes control plane is fully managed by AWS. You can use the same Amazon EKS APIs, tools, and console that you use in the cloud to create and run Amazon EKS on Outposts.

The following diagram shows these deployment options.



When to use each deployment option

Both local and extended clusters are general-purpose deployment options and can be used for a range of applications.

With local clusters, you can run the entire Amazon EKS cluster locally on Outposts. This option can mitigate the risk of application downtime that might result from temporary network disconnects to the cloud. These network disconnects can be caused by fiber cuts or weather events. Because the entire Amazon EKS cluster runs locally on Outposts, applications remain available. You can perform cluster operations during network disconnects to the cloud. For more information, see the section called "Prepare for disconnects". If you're concerned about the quality of the network connection from your Outposts to the parent AWS Region and require high availability through network disconnects, use the local cluster deployment option.

With extended clusters, you can conserve capacity on your Outpost because the Kubernetes control plane runs in the parent AWS Region. This option is suitable if you can invest in reliable, redundant network connectivity from your Outpost to the AWS Region. The quality of the network connection is critical for this option. The way that Kubernetes handles network disconnects between the Kubernetes control plane and nodes might lead to application downtime. For more information on the behavior of Kubernetes, see Scheduling, Preemption, and Eviction in the Kubernetes documentation.

Comparing the deployment options

The following table compares the differences between the two options.

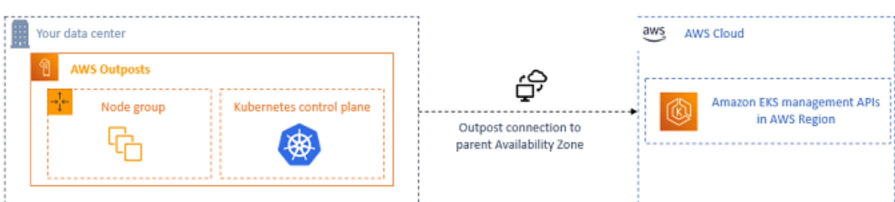
Feature	Extended cluster	Local cluster
Kubernetes control plane location	AWS Region	Outpost
Kubernetes control plane account	AWS account	Your account
Regional availability	see Service endpoints	US East (Ohio), US East (N. Virginia), US West (N. California), US West (Oregon), Asia Pacific (Seoul), Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Tokyo), Canada (Central), Europe (Frankfurt), Europe (Ireland), Europe (London), Middle East (Bahrain), and South America (São Paulo)
Kubernetes minor versions	Supported Amazon EKS versions.	Supported Amazon EKS versions.

Feature	Extended cluster	Local cluster
Platform versions	See the section called "Platform versions"	See the section called "EKS platform versions"
Outpost form factors	Outpost racks	Outpost racks
User interfaces	AWS Management Console, AWS CLI, Amazon EKS API, `eksctl`, AWS CloudFormation, and Terraform	AWS Management Console, AWS CLI, Amazon EKS API, `eksctl`, AWS CloudFormation, and Terraform
Managed policies	AmazonEKSClusterPolicy and the section called "AWS managed policy: AmazonEKSServiceRolePolicy"	AmazonEKSLocalOutpostClusterPolicy and the section called "AWS managed policy: AmazonEKSLocalOutpostServiceRolePolicy"
Cluster VPC and subnets	See the section called "VPC and subnet requirements"	See the section called "Create a VPC and subnets"
Cluster endpoint access	Public or private or both	Private only
Kubernetes API server authentication	AWS Identity and Access Management (IAM) and OIDC	IAM and `x.509` certificates
Node types	Self-managed only	Self-managed only
Node compute types	Amazon EC2 on-demand	Amazon EC2 on-demand
Node storage types	Amazon EBS gp2 and local NVMe SSD	Amazon EBS gp2 and local NVMe SSD
Amazon EKS optimized AMIs	Amazon Linux, Windows, and Bottlerocket	Amazon Linux only
IP versions	`IPv4` only	`IPv4` only
Add-ons	Amazon EKS add-ons or self-managed add-ons	Self-managed add-ons only

Feature	Extended cluster	Local cluster
Default Container Network Interface	Amazon VPC CNI plugin for Kubernetes	Amazon VPC CNI plugin for Kubernetes
Kubernetes control plane logs	Amazon CloudWatch Logs	Amazon CloudWatch Logs
Load balancing	Use the AWS Load Balancer Controller to provision Application Load Balancers only (no Network Load Balancers)	Use the AWS Load Balancer Controller to provision Application Load Balancers only (no Network Load Balancers)
Secrets envelope encryption	See the section called "Enable secret encryption"	Not supported
IAM roles for service accounts	See the section called "Credentials with IRSA"	Not supported
Troubleshooting	See *Troubleshooting*	See the section called "Troubleshoot clusters"

Create local Amazon EKS clusters on AWS Outposts for high availability

You can use local clusters to run your entire Amazon EKS cluster locally on AWS Outposts. This helps mitigate the risk of application downtime that might result from temporary network disconnects to the cloud. These disconnects can be caused by fiber cuts or weather events. Because the entire Kubernetes cluster runs locally on Outposts, applications remain available. You can perform cluster operations during network disconnects to the cloud. For more information, see the section called "Prepare for disconnects". The following diagram shows a local cluster deployment.



Local clusters are generally available for use with Outposts racks.

Supported AWS Regions

You can create local clusters in the following AWS Regions: US East (Ohio), US East (N. Virginia), US West (N. California), US West (Oregon), Asia Pacific (Seoul), Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Tokyo), Canada (Central), Europe (Frankfurt), Europe (Ireland), Europe (London), Middle East (Bahrain), and South America (São Paulo). For detailed information about supported features, see the section called “Comparing the deployment options”.

Deploy an Amazon EKS cluster on AWS Outposts

This topic provides an overview of what to consider when running a local cluster on an Outpost. The topic also provides instructions for how to deploy a local cluster on an Outpost.

> **Important**
>
> - These considerations aren’t replicated in related Amazon EKS documentation. If other Amazon EKS documentation topics conflict with the considerations here, follow the considerations here.
> - These considerations are subject to change and might change frequently. So, we recommend that you regularly review this topic.
> - Many of the considerations are different than the considerations for creating a cluster on the AWS Cloud.

- Local clusters support Outpost racks only. A single local cluster can run across multiple physical Outpost racks that comprise a single logical Outpost. A single local cluster can’t run across multiple logical Outposts. Each logical Outpost has a single Outpost ARN.
- Local clusters run and manage the Kubernetes control plane in your account on the Outpost. You can’t run workloads on the Kubernetes control plane instances or modify the Kubernetes control plane components. These nodes are managed by the Amazon EKS service. Changes to the Kubernetes control plane don’t persist through automatic Amazon EKS management actions, such as patching.
- Local clusters support self-managed add-ons and self-managed Amazon Linux node groups. The Amazon VPC CNI plugin for Kubernetes, kube-proxy, and CoreDNS add-ons are automatically installed on local clusters.

- Local clusters require the use of Amazon EBS on Outposts. Your Outpost must have Amazon EBS available for the Kubernetes control plane storage.
- Local clusters use Amazon EBS on Outposts. Your Outpost must have Amazon EBS available for the Kubernetes control plane storage. Outposts support Amazon EBS `gp2` volumes only.
- Amazon EBS backed Kubernetes `PersistentVolumes` are supported using the Amazon EBS CSI driver.
- The control plane instances of local clusters are set up in stacked highly available topology. Two out of the three control plane instances must be healthy at all times to maintain quorum. If quorum is lost, contact AWS support, as some service-side actions will be required to enable the new managed instances.

Prerequisites

- Familiarity with the Outposts deployment options, Select instance types and placement groups for Amazon EKS clusters on AWS Outposts based on capacity considerations, and VPC requirements and considerations.
- An existing Outpost. For more information, see What is AWS Outposts.
- The `kubectl` command line tool is installed on your computer or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. For example, if your cluster version is `1.29`, you can use `kubectl` version `1.28`, `1.29`, or `1.30` with it. To install or upgrade `kubectl`, see the section called "Set up kubectl and eksctl".
- Version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version, use `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*. The AWS CLI version that is installed in AWS CloudShell might also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the *AWS CloudShell User Guide*.
- An IAM principal (user or role) with permissions to `create` and `describe` an Amazon EKS cluster. For more information, see the section called "Create a local Kubernetes cluster on an Outpost" and the section called "List or describe all clusters".

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

Create an Amazon EKS local cluster

You can create a local cluster with the following tools described in this page:

- the section called "`eksctl`"
- the section called "AWS Management Console"

You could also use the AWS CLI, the Amazon EKS API, the AWS SDKs, AWS CloudFormation or Terraform to create clusters on Outposts.

`eksctl`

To create a local cluster with `eksctl`

1. Install version `0.204.0` or later of the `eksctl` command line tool on your device or AWS CloudShell. To install or update `eksctl`, see Installation in the `eksctl` documentation.
2. Copy the contents that follow to your device. Replace the following values and then run the modified command to create the `outpost-control-plane.yaml` file:
   - Replace *`region-code`* with the supported AWS Region that you want to create your cluster in.
   - Replace *`my-cluster`* with a name for your cluster. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.

- Replace *vpc-ExampleID1* and *subnet-ExampleID1* with the IDs of your existing VPC and subnet. The VPC and subnet must meet the requirements in [Create a VPC and subnets for Amazon EKS clusters on AWS Outposts](#).
- Replace *uniqueid* with the ID of your Outpost.
- Replace *m5.large* with an instance type available on your Outpost. Before choosing an instance type, see [the section called “Capacity considerations”](#). Three control plane instances are deployed. You can’t change this number.

```
cat >outpost-control-plane.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: "1.24"

vpc:
  clusterEndpoints:
    privateAccess: true
  id: "vpc-vpc-ExampleID1"
  subnets:
    private:
      outpost-subnet-1:
        id: "subnet-subnet-ExampleID1"

outpost:
  controlPlaneOutpostARN: arn:aws:outposts:region-code:111122223333:outpost/op-
uniqueid
  controlPlaneInstanceType: m5.large
EOF
```

For a complete list of all available options and defaults, see [AWS Outposts Support](#) and [Config file schema](#) in the `eksctl` documentation.

3. Create the cluster using the configuration file that you created in the previous step. `eksctl` creates a VPC and one subnet on your Outpost to deploy the cluster in.

```
eksctl create cluster -f outpost-control-plane.yaml
```

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

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

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

The `eksctl` command automatically created an access entry for the IAM principal (user or role) that created the cluster and granted the IAM principal administrator permissions to Kubernetes objects on the cluster. If you don't want the cluster creator to have administrator access to Kubernetes objects on the cluster, add the following text to the previous configuration file: `bootstrapClusterCreatorAdminPermissions: false` (at the same level as `metadata`, `vpc`, and `outpost`). If you added the option, then after cluster creation, you need to create an access entry for at least one IAM principal, or no IAM principals will have access to Kubernetes objects on the cluster.

AWS Management Console

To create your cluster with the AWS Management Console

1. You need an existing VPC and subnet that meet Amazon EKS requirements. For more information, see the section called "Create a VPC and subnets".
2. If you already have a local cluster IAM role, or you're going to create your cluster with `eksctl`, then you can skip this step. By default, `eksctl` creates a role for you.

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

```
cat >eks-local-cluster-role-trust-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

b. Create the Amazon EKS cluster IAM role. To create an IAM role, the IAM principal that is creating the role must be assigned the `iam:CreateRole` action (permission).

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

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

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonEKSLocalOutpostClusterPolicy --role-name myAmazonEKSLocalClusterRole
```

3. Open the Amazon EKS console.
4. At the top of the console screen, make sure that you have selected a supported AWS Region.
5. Choose **Add cluster** and then choose **Create**.
6. On the **Configure cluster** page, enter or select values for the following fields:
   - **Kubernetes control plane location** – Choose AWS Outposts.
   - **Outpost ID** – Choose the ID of the Outpost that you want to create your control plane on.
   - **Instance type** – Select an instance type. Only the instance types available in your Outpost are displayed. In the dropdown list, each instance type describes how many nodes the instance type is recommended for. Before choosing an instance type, see the section called “Capacity considerations”. All replicas are deployed using the same instance type. You can’t change the instance type after your cluster is created. Three control plane instances are deployed. You can’t change this number.
   - **Name** – A name for your cluster. It must be unique in your AWS account. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an

alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.

- **Kubernetes version –** Choose the Kubernetes version that you want to use for your cluster. We recommend selecting the latest version, unless you need to use an earlier version.
- **Cluster service role –** Choose the Amazon EKS cluster IAM role that you created in a previous step to allow the Kubernetes control plane to manage AWS resources.
- **Kubernetes cluster administrator access –** If you want the IAM principal (role or user) that's creating the cluster to have administrator access to the Kubernetes objects on the cluster, accept the default (allow). Amazon EKS creates an access entry for the IAM principal and grants cluster administrator permissions to the access entry. For more information about access entries, see the section called "Grant permissions".

  If you want a different IAM principal than the principal creating the cluster to have administrator access to Kubernetes cluster objects, choose the disallow option. After cluster creation, any IAM principal that has IAM permissions to create access entries can add an access entries for any IAM principals that need access to Kubernetes cluster objects. For more information about the required IAM permissions, see Actions defined by Amazon Elastic Kubernetes Service in the Service Authorization Reference. If you choose the disallow option and don't create any access entries, then no IAM principals will have access to the Kubernetes objects on the cluster.
- **Tags –** (Optional) Add any tags to your cluster. For more information, see the section called "Tagging your resources". When you're done with this page, choose **Next**.

7. On the **Specify networking** page, select values for the following fields:

   - **VPC –** Choose an existing VPC. The VPC must have a sufficient number of IP addresses available for the cluster, any nodes, and other Kubernetes resources that you want to create. Your VPC must meet the requirements in VPC requirements and considerations.
   - **Subnets –** By default, all available subnets in the VPC specified in the previous field are preselected. The subnets that you choose must meet the requirements in Subnet requirements and considerations.
   - **Security groups –** (Optional) Specify one or more security groups that you want Amazon EKS to associate to the network interfaces that it creates. Amazon EKS automatically creates a security group that enables communication between your cluster and your VPC. Amazon EKS associates this security group, and any that you choose, to the network interfaces that it creates. For more information about the cluster security group that Amazon EKS creates,

see the section called "Security group requirements". You can modify the rules in the cluster security group that Amazon EKS creates. If you choose to add your own security groups, you can't change the ones that you choose after cluster creation. For on-premises hosts to communicate with the cluster endpoint, you must allow inbound traffic from the cluster security group. For clusters that don't have an ingress and egress internet connection (also knows as private clusters), you must do one of the following:

- Add the security group associated with required VPC endpoints. For more information about the required endpoints, see the section called "Using interface VPC endpoints" in Subnet access to AWS services.
- Modify the security group that Amazon EKS created to allow traffic from the security group associated with the VPC endpoints. When you're done with this page, choose **Next**.

8. On the **Configure observability** page, you can optionally choose which **Metrics** and **Control plane logging** options that you want to turn on. By default, each log type is turned off.
   - For more information on the Prometheus metrics option, see the section called "Step 1: Turn on Prometheus metrics".
   - For more information on the **Control plane logging** options, see the section called "Control plane logs". When you're done with this page, choose **Next**.
9. On the **Review and create** page, review the information that you entered or selected on the previous pages. If you need to make changes, choose **Edit**. When you're satisfied, choose **Create**. The **Status** field shows **CREATING** while the cluster is provisioned.

   Cluster provisioning takes several minutes.

View your Amazon EKS local cluster

1. After your cluster is created, you can view the Amazon EC2 control plane instances that were created.

```
aws ec2 describe-instances --query 'Reservations[*].Instances[*].{Name:Tags[?
Key==`Name`]|[0].Value}' | grep my-cluster-control-plane
```

   An example output is as follows.

```
"Name": "my-cluster-control-plane-id1"
"Name": "my-cluster-control-plane-id2"
"Name": "my-cluster-control-plane-id3"
```

Each instance is tainted with `node-role.eks-local.amazonaws.com/control-plane` so that no workloads are ever scheduled on the control plane instances. For more information about taints, see Taints and Tolerations in the Kubernetes documentation. Amazon EKS continuously monitors the state of local clusters. We perform automatic management actions, such as security patches and repairing unhealthy instances. When local clusters are disconnected from the cloud, we complete actions to ensure that the cluster is repaired to a healthy state upon reconnect.

2. If you created your cluster using `eksctl`, then you can skip this step. `eksctl` completes this step for you. Enable `kubectl` to communicate with your cluster by adding a new context to the `kubectl config` file. For instructions on how to create and update the file, see the section called "Access cluster with kubectl".

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

An example output is as follows.

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

3. To connect to your local cluster's Kubernetes API server, have access to the local gateway for the subnet, or connect from within the VPC. For more information about connecting an Outpost rack to your on-premises network, see How local gateways for racks work in the AWS Outposts User Guide. If you use Direct VPC Routing and the Outpost subnet has a route to your local gateway, the private IP addresses of the Kubernetes control plane instances are automatically broadcasted over your local network. The local cluster's Kubernetes API server endpoint is hosted in Amazon Route 53 (Route 53). The API service endpoint can be resolved by public DNS servers to the Kubernetes API servers' private IP addresses.

Local clusters' Kubernetes control plane instances are configured with static elastic network interfaces with fixed private IP addresses that don't change throughout the cluster lifecycle. Machines that interact with the Kubernetes API server might not have connectivity to Route 53 during network disconnects. If this is the case, we recommend configuring `/etc/hosts` with the static private IP addresses for continued operations. We also recommend setting up local DNS servers and connecting them to your Outpost. For more information, see the AWS Outposts documentation. Run the following command to confirm that communication's established with your cluster.

```
kubectl get svc
```

An example output is as follows.

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

4. (Optional) Test authentication to your local cluster when it's in a disconnected state from the AWS Cloud. For instructions, see the section called "Prepare for disconnects".

Internal resources

Amazon EKS creates the following resources on your cluster. The resources are for Amazon EKS internal use. For proper functioning of your cluster, don't edit or modify these resources.

- The following mirror Pods:
  - `aws-iam-authenticator-`*`node-hostname`*
  - `eks-certificates-controller-`*`node-hostname`*
  - `etcd-`*`node-hostname`*
  - `kube-apiserver-`*`node-hostname`*
  - `kube-controller-manager-`*`node-hostname`*
  - `kube-scheduler-`*`node-hostname`*
- The following self-managed add-ons:
  - `kube-system/coredns`
  - `kube-system/ kube-proxy` (not created until you add your first node)
  - `kube-system/aws-node` (not created until you add your first node). Local clusters use the Amazon VPC CNI plugin for Kubernetes plugin for cluster networking. Do not change the configuration for control plane instances (Pods named `aws-node-controlplane-*`). There are configuration variables that you can use to change the default value for when the plugin creates new network interfaces. For more information, see the documentation on GitHub.
- The following services:
  - `default/kubernetes`
  - `kube-system/kube-dns`
- A `PodSecurityPolicy` named `eks.system`

- A `ClusterRole` named `eks:system:podsecuritypolicy`
- A `ClusterRoleBinding` named `eks:system`
- A default PodSecurityPolicy
- In addition to the cluster security group, Amazon EKS creates a security group in your AWS account that's named `eks-local-internal-do-not-use-or-edit-cluster-name-uniqueid`. This security group allows traffic to flow freely between Kubernetes components running on the control plane instances.

Recommended next steps:

- Grant the IAM principal that created the cluster the required permissions to view Kubernetes resources in the AWS Management Console
- Grant IAM entities access to your cluster. If you want the entities to view Kubernetes resources in the Amazon EKS console, grant the Required permissions to the entities.
- Configure logging for your cluster
- Familiarize yourself with what happens during network disconnects.
- Add nodes to your cluster
- Consider setting up a backup plan for your `etcd`. Amazon EKS doesn't support automated backup and restore of `etcd` for local clusters. For more information, see Backing up an etcd cluster in the Kubernetes documentation. The two main options are using `etcdctl` to automate taking snapshots or using Amazon EBS storage volume backup.

Learn Kubernetes and Amazon EKS platform versions for AWS Outposts

Local cluster platform versions represent the capabilities of the Amazon EKS cluster on AWS Outposts. The versions include the components that run on the Kubernetes control plane, which Kubernetes API server flags are enabled. They also include the current Kubernetes patch version. Each Kubernetes minor version has one or more associated platform versions. The platform versions for different Kubernetes minor versions are independent. The platform versions for local clusters and Amazon EKS clusters in the cloud are independent.

When a new Kubernetes minor version is available for local clusters, such as `1.30`, the initial platform version for that Kubernetes minor version starts at `eks-local-outposts.1`. However, Amazon EKS releases new platform versions periodically to enable new Kubernetes control plane settings and to provide security fixes.

When new local cluster platform versions become available for a minor version:

- The platform version number is incremented (`eks-local-outposts.n+1`).
- Amazon EKS automatically updates all existing local clusters to the latest platform version for their corresponding Kubernetes minor version. Automatic updates of existing platform versions are rolled out incrementally. The roll-out process consists of the replacement of the managed Kubernetes control-plane instances running on the Outpost, one at a time, until all 3 instances get replaced by new ones.
- The Kubernetes control-plane instance replacement process will stop progressing if there is risk of service interruption. Amazon EKS will only attempt to replace an instance in case the other 2 Kubernetes control-plane instances are healthy and passing all readiness conditions as a cluster node.
- A platform version rollout will typically take less than 30 minutes to complete. If a cluster remains on `UPDATING` state for an extended amount of time, see the the section called "Troubleshoot clusters" and seek help from AWS Support. Never manually terminate Kubernetes control-plane instances unless instructed by AWS Support.
- Amazon EKS might publish a new node AMI with a corresponding patch version. All patch versions are compatible between the Kubernetes control plane and node AMIs for a single Kubernetes minor version.

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

Local clusters are always created with the latest available platform version (`eks-local-outposts.n`) for the specified Kubernetes version.

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

Kubernetes version 1.30

The following admission controllers are enabled for all `1.30` platform versions: `CertificateApproval`, `CertificateSigning`, `CertificateSubjectRestriction`, `DefaultIngressClass`, `DefaultStorageClass`, `DefaultTolerationSeconds`, `ExtendedResourceToleration`, `LimitRanger`, `MutatingAdmissionWebhook`, `NamespaceLifecycle`, `NodeRestriction`, `PersistentVolumeClaimResize`, `Priority`, `PodSecurity`, `ResourceQuota`, `RuntimeClass`, `ServiceAccount`, `StorageObjectInUseProtection`, `TaintNodesByCondition`, `ValidatingAdmissionPolicy`, and `ValidatingAdmissionWebhook`.

Kubernetes version	Amazon EKS platform version	Release notes	Release date
1.30.7	eks-local-outposts.2	New platform version with security fixes and enhancements. kube-proxy updated to v1.30.7. AWS IAM Authenticator updated to v0.6.28. Amazon VPC CNI plugin for Kubernetes updated to v1.19.0. Updated Bottlerocket version to v1.29.0.	January 10, 2025
1.30.5	eks-local-outposts.1	Initial release of Kubernetes version v1.30 for local Amazon EKS clusters on Outposts.	November 13, 2024

Kubernetes version 1.29

The following admission controllers are enabled for all 1.29 platform versions: CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, DefaultStorageClass, DefaultTolerationSeconds, ExtendedResourceToleration, LimitRanger, MutatingAdmissionWebhook, NamespaceLifecycle, NodeRestriction, PersistentVolumeClaimResize, Priority, PodSecurity, ResourceQuota, RuntimeClass, ServiceAccount, StorageObjectInUseProtection, TaintNodesByCondition, ValidatingAdmissionPolicy, and ValidatingAdmissionWebhook.

Kubernetes version	Amazon EKS platform version	Release notes	Release date
v1.29.11	eks-local-outposts.12	New platform version with security fixes and enhancements. kube-proxy updated to v1.29.11. Amazon VPC CNI plugin for Kubernetes updated to v1.19.0. Updated CoreDNS image to v1.11.3. Updated Bottlerocket version to v1.29.0.	January 10, 2025
1.29.9	eks-local-outposts.4	New platform version with security fixes and enhancements. kube-proxy updated to v1.29.9. AWS IAM Authenticator updated to v0.6.26. Updated Bottlerocket version to v1.26.0.	November 8, 2024
1.29.6	eks-local-outposts.3	New platform version with security fixes and enhancements. Updated Bottlerocket version to v1.22.0.	October 22, 2024
1.29.6	eks-local-outposts.2	New platform version with security fixes and enhancements. Updated Bottlerocket version to v1.21.0.	August 27, 2024

Kubernetes version	Amazon EKS platform version	Release notes	Release date
1.29.6	eks-local-outposts.1	Initial release of Kubernetes version v1.29 for local Amazon EKS clusters on Outposts.	August 20, 2024

Kubernetes version 1.28

The following admission controllers are enabled for all 1.28 platform versions: CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, DefaultStorageClass, DefaultTolerationSeconds, ExtendedResourceToleration, LimitRanger, MutatingAdmissionWebhook, NamespaceLifecycle, NodeRestriction, PersistentVolumeClaimResize, Priority, PodSecurity, ResourceQuota, RuntimeClass, ServiceAccount, StorageObjectInUseProtection, TaintNodesByCondition, ValidatingAdmissionPolicy, and ValidatingAdmissionWebhook.

Kubernetes version	Amazon EKS platform version	Release notes	Release date
1.28.15	eks-local-outposts.12	New platform version with security fixes and enhancements. kube-proxy updated to v1.28.15. Amazon VPC CNI plugin for Kubernetes updated to v1.19.0. Updated Bottlerocket version to v1.29.0.	January 10, 2025
1.28.14	eks-local-outposts.11	New platform version with security fixes	November 8, 2024

Kubernetes version	Amazon EKS platform version	Release notes	Release date
		and enhancements. kube-proxy updated to v1.28.14. AWS IAM Authenticator updated to v0.6.26. Updated CoreDNS image to v1.11.1. Updated Bottlerocket version to v1.26.0.	
1.28.10	eks-local-outposts.10	New platform version with security fixes and enhancements. Updated Bottlerocket version to v1.22.0.	October 22, 2024
1.28.10	eks-local-outposts.9	New platform version with security fixes and enhancements. Updated Bottlerocket version to v1.21.0.	August 27, 2024
1.28.10	eks-local-outposts.8	New platform version with security fixes and enhancements.	July 30, 2024
1.28.10	eks-local-outposts.6	New platform version with security fixes and enhancements. kube-proxy updated to v1.28.10. AWS IAM Authenticator updated to v0.6.20. Updated Bottlerocket version to v1.20.2.	June 19, 2024

Kubernetes version	Amazon EKS platform version	Release notes	Release date
`1.28.6`	`eks-local-outposts.5`	Updated Bottlerocket version to v1.19.3 containing newest bugfixes to support local boot in Outposts.	April 18, 2024
`1.28.6`	`eks-local-outposts.4`	New platform version with security fixes and enhancements. Restored support or local boot in Outposts. Downgraded Bottlerocket version to `v1.15.1` for compatibility.	April 2, 2024
`1.28.6`	`eks-local-outposts.3`	New platform version with security fixes and enhancements.	March 22, 2024

Kubernetes version	Amazon EKS platform version	Release notes	Release date
1.28.6	eks-local-outposts.2	New platform version with security fixes and enhancements. kube-proxy updated to v1.28.6. AWS IAM Authenticator updated to v0.6.17. Amazon VPC CNI plugin for Kubernetes downgraded to v1.13.2 for compatibility reasons. Updated Bottlerocket version to v1.19.2.	March 8, 2024
1.28.1	eks-local-outposts.1	Initial release of Kubernetes version v1.28 for local Amazon EKS clusters on Outposts.	October 4, 2023

Kubernetes version 1.27

The following admission controllers are enabled for all 1.27 platform versions: CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, DefaultStorageClass, DefaultTolerationSeconds, ExtendedResourceToleration, LimitRanger, MutatingAdmissionWebhook, NamespaceLifecycle, NodeRestriction, PersistentVolumeClaimResize, Priority, PodSecurity, ResourceQuota, RuntimeClass, ServiceAccount, StorageObjectInUseProtection, TaintNodesByCondition, ValidatingAdmissionPolicy, and ValidatingAdmissionWebhook.

Kubernetes version	Amazon EKS platform version	Release notes	Release date
`1.27.16`	`eks-local-outposts.12`	New platform version with security fixes and enhancements. Amazon VPC CNI plugin for Kubernetes updated to `v1.19.0`. Updated Bottlerocket version to `v1.29.0`.	January 10, 2025
`1.27.16`	`eks-local-outposts.11`	New platform version with security fixes and enhancements. kube-proxy updated to `v1.27.16`. AWS IAM Authenticator updated to `v0.6.26`. Updated CoreDNS image to `v1.11.1`. Updated Bottlerocket version to `v1.26.0`.	November 8, 2024
`1.27.14`	`eks-local-outposts.10`	New platform version with security fixes and enhancements. Updated Bottlerocket version to `v1.22.0`.	October 22, 2024
`1.27.14`	`eks-local-outposts.9`	New platform version with security fixes and enhancements. Updated Bottlerocket version to `v1.21.0`.	August 27, 2024

Kubernetes version	Amazon EKS platform version	Release notes	Release date
`1.27.14`	`eks-local-outposts.8`	New platform version with security fixes and enhancements.	July 30, 2024
`1.27.14`	`eks-local-outposts.6`	New platform version with security fixes and enhancements. kube-proxy updated to `v1.27.14`. AWS IAM Authenticator updated to `v0.6.20`. Updated Bottlerocket version to `v1.20.2`.	June 19, 2024
`1.27.10`	`eks-local-outposts.5`	New platform version with security fixes and enhancements.	April 2, 2024
`1.27.10`	`eks-local-outposts.4`	New platform version with security fixes and enhancements. kube-proxy updated to `v1.27.10`. AWS IAM Authenticator updated to `v0.6.17`. Updated Bottlerocket version to `v1.19.2`.	March 22, 2024

Kubernetes version	Amazon EKS platform version	Release notes	Release date
1.27.3	eks-local-outposts.3	New platform version with security fixes and enhancements. kube-proxy updated to v1.27.3. Amazon VPC CNI plugin for Kubernetes updated to v1.13.2.	July 14, 2023
1.27.1	eks-local-outposts.2	Updated CoreDNS image to v1.10.1.	June 22, 2023
1.27.1	eks-local-outposts.1	Initial release of Kubernetes version 1.27 for local Amazon EKS clusters on Outposts.	May 30, 2023

Create a VPC and subnets for Amazon EKS clusters on AWS Outposts

When you create a local cluster, you specify a VPC and at least one private subnet that runs on Outposts. This topic provides an overview of the VPC and subnets requirements and considerations for your local cluster.

VPC requirements and considerations

When you create a local cluster, the VPC that you specify must meet the following requirements and considerations:

- Make sure that the VPC has enough IP addresses for the local cluster, any nodes, and other Kubernetes resources that you want to create. If the VPC that you want to use doesn't have enough IP addresses, increase the number of available IP addresses. You can do this by associating additional Classless Inter-Domain Routing (CIDR) blocks with your VPC. You can associate private (RFC 1918) and public (non-RFC 1918) CIDR blocks to your VPC either before or after you create your cluster. It can take a cluster up to 5 hours for a CIDR block that you associated with a VPC to be recognized.

- The VPC can't have assigned IP prefixes or IPv6 CIDR blocks. Because of these constraints, the information that's covered in Assign more IP addresses to Amazon EKS nodes with prefixes and the section called "IPv6" isn't applicable to your VPC.
- The VPC has a DNS hostname and DNS resolution enabled. Without these features, the local cluster fails to create, and you need to enable the features and recreate your cluster. For more information, see DNS attributes for your VPC in the Amazon VPC User Guide.
- To access your local cluster over your local network, the VPC must be associated with your Outpost's local gateway route table. For more information, see VPC associations in the AWS Outposts User Guide.

Subnet requirements and considerations

When you create the cluster, specify at least one private subnet. If you specify more than one subnet, the Kubernetes control plane instances are evenly distributed across the subnets. If more than one subnet is specified, the subnets must exist on the same Outpost. Moreover, the subnets must also have proper routes and security group permissions to communicate with each other. When you create a local cluster, the subnets that you specify must meet the following requirements:

- The subnets are all on the same logical Outpost.
- The subnets together have at least three available IP addresses for the Kubernetes control plane instances. If three subnets are specified, each subnet must have at least one available IP address. If two subnets are specified, each subnet must have at least two available IP addresses. If one subnet is specified, the subnet must have at least three available IP addresses.
- The subnets have a route to the Outpost rack's local gateway to access the Kubernetes API server over your local network. If the subnets don't have a route to the Outpost rack's local gateway, you must communicate with your Kubernetes API server from within the VPC.
- The subnets must use IP address-based naming. Amazon EC2 resource-based naming isn't supported by Amazon EKS.

Subnet access to AWS services

The local cluster's private subnets on Outposts must be able to communicate with Regional AWS services. You can achieve this by using a NAT gateway for outbound internet access or, if you want to keep all traffic private within your VPC, using interface VPC endpoints.

Using a NAT gateway

The local cluster's private subnets on Outposts must have an associated route table that has a route to a NAT gateway in a public subnet that is in the Outpost's parent Availability Zone. The public subnet must have a route to an internet gateway. The NAT gateway enables outbound internet access and prevents unsolicited inbound connections from the internet to instances on the Outpost.

Using interface VPC endpoints

If the local cluster's private subnets on Outposts don't have an outbound internet connection, or if you want to keep all traffic private within your VPC, then you must create the following interface VPC endpoints and gateway endpoint in a Regional subnet before creating your cluster.

Endpoint	Endpoint type
`com.amazonaws.` *`region-code`* `.ssm`	Interface
`com.amazonaws.` *`region-code`* `.ssmmessages`	Interface
`com.amazonaws.` *`region-code`* `.ec2messages`	Interface
`com.amazonaws.` *`region-code`* `.ec2`	Interface
`com.amazonaws.` *`region-code`* `.secretsmanager`	Interface
`com.amazonaws.` *`region-code`* `.logs`	Interface
`com.amazonaws.` *`region-code`* `.sts`	Interface
`com.amazonaws.` *`region-code`* `.ecr.api`	Interface

Endpoint	Endpoint type
com.amazonaws. *region-code* .ecr.dkr	Interface
com.amazonaws. *region-code* .s3	Gateway

The endpoints must meet the following requirements:

- Created in a private subnet located in your Outpost's parent Availability Zone
- Have private DNS names enabled
- Have an attached security group that permits inbound HTTPS traffic from the CIDR range of the private outpost subnet.

Creating endpoints incurs charges. For more information, see AWS PrivateLink pricing. If your Pods need access to other AWS services, then you need to create additional endpoints. For a comprehensive list of endpoints, see AWS services that integrate with AWS PrivateLink.

Create a VPC

You can create a VPC that meets the previous requirements using one of the following AWS CloudFormation templates:

- **Template 1** – This template creates a VPC with one private subnet on the Outpost and one public subnet in the AWS Region. The private subnet has a route to an internet through a NAT Gateway that resides in the public subnet in the AWS Region. This template can be used to create a local cluster in a subnet with egress internet access.
- **Template 2** – This template creates a VPC with one private subnet on the Outpost and the minimum set of VPC Endpoints required to create a local cluster in a subnet that doesn't have ingress or egress internet access (also referred to as a private subnet).

Prepare local Amazon EKS clusters on AWS Outposts for network disconnects

If your local network has lost connectivity with the AWS Cloud, you can continue to use your local Amazon EKS cluster on an Outpost. This topic covers how you can prepare your local cluster for network disconnects and related considerations.

- Local clusters enable stability and continued operations during temporary, unplanned network disconnects. AWS Outposts remains a fully connected offering that acts as an extension of the AWS Cloud in your data center. In the event of network disconnects between your Outpost and AWS Cloud, we recommend attempting to restore your connection. For instruction, see AWS Outposts rack network troubleshooting checklist in the *AWS Outposts User Guide*. For more information about how to troubleshoot issues with local clusters, see the section called "Troubleshoot clusters".
- Outposts emit a `ConnectedStatus` metric that you can use to monitor the connectivity state of your Outpost. For more information, see Outposts Metrics in the *AWS Outposts User Guide*.
- Local clusters use IAM as the default authentication mechanism using the AWS Identity and Access Management authenticator for Kubernetes. IAM isn't available during network disconnects. So, local clusters support an alternative authentication mechanism using `x.509` certificates that you can use to connect to your cluster during network disconnects. For information about how to obtain and use an `x.509` certificate for your cluster, see the section called "Authenticating to your local cluster during a network disconnect".
- If you can't access Route 53 during network disconnects, consider using local DNS servers in your on-premises environment. The Kubernetes control plane instances use static IP addresses. You can configure the hosts that you use to connect to your cluster with the endpoint hostname and IP addresses as an alternative to using local DNS servers. For more information, see DNS in the *AWS Outposts User Guide*.
- If you expect increases in application traffic during network disconnects, you can provision spare compute capacity in your cluster when connected to the cloud. Amazon EC2 instances are included in the price of AWS Outposts. So, running spare instances doesn't impact your AWS usage cost.
- During network disconnects to enable create, update, and scale operations for workloads, your application's container images must be accessible over the local network and your cluster must have enough capacity. Local clusters don't host a container registry for you. If the Pods have previously run on those nodes, container images are cached on the nodes. If you typically pull your application's container images from Amazon ECR in the cloud, consider running a local cache or registry. A local cache or registry is helpful if you require create, update, and scale operations for workload resources during network disconnects.
- Local clusters use Amazon EBS as the default storage class for persistent volumes and the Amazon EBS CSI driver to manage the lifecycle of Amazon EBS persistent volumes. During network disconnects, Pods that are backed by Amazon EBS can't be created, updated, or scaled. This is because these operations require calls to the Amazon EBS API in the cloud. If you're

deploying stateful workloads on local clusters and require create, update, or scale operations during network disconnects, consider using an alternative storage mechanism.

- Amazon EBS snapshots can't be created or deleted if AWS Outposts can't access the relevant AWS in-region APIs (such as the APIs for Amazon EBS or Amazon S3).
- When integrating ALB (Ingress) with AWS Certificate Manager (ACM), certificates are pushed and stored in memory of the AWS Outposts ALB Compute instance. Current TLS termination will continue to operate in the event of a disconnect from the AWS Region. Mutating operations in this context will fail (such as new ingress definitions, new ACM based certificates API operations, ALB compute scale, or certificate rotation). For more information, see Troubleshooting managed certificate renewal in the *AWS Certificate Manager User Guide*.
- The Amazon EKS control plane logs are cached locally on the Kubernetes control plane instances during network disconnects. Upon reconnect, the logs are sent to CloudWatch Logs in the parent AWS Region. You can use Prometheus, Grafana, or Amazon EKS partner solutions to monitor the cluster locally using the Kubernetes API server's metrics endpoint or using Fluent Bit for logs.
- If you're using the AWS Load Balancer Controller on Outposts for application traffic, existing Pods fronted by the AWS Load Balancer Controller continue to receive traffic during network disconnects. New Pods created during network disconnects don't receive traffic until the Outpost is reconnected to the AWS Cloud. Consider setting the replica count for your applications while connected to the AWS Cloud to accommodate your scaling needs during network disconnects.
- The Amazon VPC CNI plugin for Kubernetes defaults to secondary IP mode. It's configured with `WARM_ENI_TARGET=1`, which allows the plugin to keep "a full elastic network interface" of available IP addresses available. Consider changing `WARM_ENI_TARGET`, `WARM_IP_TARGET`, and `MINIMUM_IP_TARGET` values according to your scaling needs during a disconnected state. For more information, see the readme file for the plugin on GitHub. For a list of the maximum number of Pods that's supported by each instance type, see the eni-max-pods.txt file on GitHub.

Authenticating to your local cluster during a network disconnect

AWS Identity and Access Management (IAM) isn't available during network disconnects. You can't authenticate to your local cluster using IAM credentials while disconnected. However, you can connect to your cluster over your local network using `x509` certificates when disconnected. You need to download and store a client `X509` certificate to use during disconnects. In this topic, you learn how to create and use the certificate to authenticate to your cluster when it's in a disconnected state.

1. Create a certificate signing request.

a. Generate a certificate signing request.

```
openssl req -new -newkey rsa:4096 -nodes -days 365 \
    -keyout admin.key -out admin.csr -subj "/CN=admin"
```

b. Create a certificate signing request in Kubernetes.

```
BASE64_CSR=$(cat admin.csr | base64 -w 0)
cat << EOF > admin-csr.yaml
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: admin-csr
spec:
  signerName: kubernetes.io/kube-apiserver-client
  request: ${BASE64_CSR}
  usages:
  - client auth
EOF
```

2. Create a certificate signing request using `kubectl`.

```
kubectl create -f admin-csr.yaml
```

3. Check the status of the certificate signing request.

```
kubectl get csr admin-csr
```

An example output is as follows.

```
NAME        AGE   REQUESTOR                              CONDITION
admin-csr   11m   kubernetes-admin                       Pending
```

Kubernetes created the certificate signing request.

4. Approve the certificate signing request.

```
kubectl certificate approve admin-csr
```

5. Recheck the certificate signing request status for approval.

```
kubectl get csr admin-csr
```

An example output is as follows.

```
NAME        AGE   REQUESTOR                        CONDITION
admin-csr   11m   kubernetes-admin                 Approved
```

6. Retrieve and verify the certificate.

   a. Retrieve the certificate.

   ```
   kubectl get csr admin-csr -o jsonpath='{.status.certificate}' | base64 --decode >
    admin.crt
   ```

   b. Verify the certificate.

   ```
   cat admin.crt
   ```

7. Create a cluster role binding for an `admin` user.

```
kubectl create clusterrolebinding admin --clusterrole=cluster-admin \
    --user=admin --group=system:masters
```

8. Generate a user-scoped kubeconfig for a disconnected state.

   You can generate a `kubeconfig` file using the downloaded `admin` certificates. Replace *`my-cluster`* and *`apiserver-endpoint`* in the following commands.

```
aws eks describe-cluster --name my-cluster \
    --query "cluster.certificateAuthority" \
    --output text | base64 --decode > ca.crt
```

```
kubectl config --kubeconfig admin.kubeconfig set-cluster my-cluster \
    --certificate-authority=ca.crt --server apiserver-endpoint --embed-certs
```

```
kubectl config --kubeconfig admin.kubeconfig set-credentials admin \
    --client-certificate=admin.crt --client-key=admin.key --embed-certs
```

```
kubectl config --kubeconfig admin.kubeconfig set-context admin@my-cluster \
```

```
    --cluster my-cluster --user admin
```

```
kubectl config --kubeconfig admin.kubeconfig use-context admin@my-cluster
```

9. View your `kubeconfig` file.

```
kubectl get nodes --kubeconfig admin.kubeconfig
```

10. If you have services already in production on your Outpost, skip this step. If Amazon EKS is the only service running on your Outpost and the Outpost isn't currently in production, you can simulate a network disconnect. Before you go into production with your local cluster, simulate a disconnect to make sure that you can access your cluster when it's in a disconnected state.

    a. Apply firewall rules on the networking devices that connect your Outpost to the AWS Region. This disconnects the service link of the Outpost. You can't create any new instances. Currently running instances lose connectivity to the AWS Region and the internet.

    b. You can test the connection to your local cluster while disconnected using the `x509` certificate. Make sure to change your `kubeconfig` to the `admin.kubeconfig` that you created in a previous step. Replace *`my-cluster`* with the name of your local cluster.

    ```
    kubectl config use-context admin@my-cluster --kubeconfig admin.kubeconfig
    ```

    If you notice any issues with your local clusters while they're in a disconnected state, we recommend opening a support ticket.

Select instance types and placement groups for Amazon EKS clusters on AWS Outposts based on capacity considerations

This topic provides guidance for selecting the Kubernetes control plane instance type and (optionally) using placement groups to meet high-availability requirements for your local Amazon EKS cluster on an Outpost.

Before you select an instance type (such as `m5`, `c5`, or `r5`) to use for your local cluster's Kubernetes control plane on Outposts, confirm the instance types that are available on your Outpost configuration. After you identify the available instance types, select the instance size (such as `large`, `xlarge`, or `2xlarge`) based on the number of nodes that your workloads require. The following table provides recommendations for choosing an instance size.

> **Note**
>
> The instance sizes must be slotted on your Outposts. Make sure that you have enough capacity for three instances of the size available on your Outposts for the lifetime of your local cluster. For a list of the available Amazon EC2 instance types, see the Compute and storage sections in AWS Outposts rack features.

Number of nodes	Kubernetes control plane instance size
1–20	`large`
21–100	`xlarge`
101–250	`2xlarge`
251–500	`4xlarge`

The storage for the Kubernetes control plane requires 246 GB of Amazon EBS storage for each local cluster to meet the required IOPS for `etcd`. When the local cluster is created, the Amazon EBS volumes are provisioned automatically for you.

Control plane placement

When you don't specify a placement group with the `OutpostConfig.ControlPlanePlacement.GroupName` property, the Amazon EC2 instances provisioned for your Kubernetes control plane don't receive any specific hardware placement enforcement across the underlying capacity available on your Outpost.

You can use placement groups to meet the high-availability requirements for your local Amazon EKS cluster on an Outpost. By specifying a placement group during cluster creation, you influence the placement of the Kubernetes control plane instances. The instances are spread across independent underlying hardware (racks or hosts), minimizing correlated instance impact on the event of hardware failures.

The type of spread that you can configure depends on the number of Outpost racks you have in your deployment.

- **Deployments with one or two physical racks in a single logical Outpost –** You must have at least three hosts that are configured with the instance type that you choose for your Kubernetes control plane instances. A *spread* placement group using *host-level spread* ensures that all Kubernetes control plane instances run on distinct hosts within the underlying racks available in your Outpost deployment.
- **Deployments with three or more physical racks in a single logical Outpost –** You must have at least three hosts configured with the instance type you choose for your Kubernetes control plane instances. A *spread* placement group using *rack-level spread* ensures that all Kubernetes control plane instances run on distinct racks in your Outpost deployment. You can alternatively use the *host-level spread* placement group as described in the previous option.

You are responsible for creating the desired placement group. You specify the placement group when calling the `CreateCluster` API. For more information about placement groups and how to create them, see Placement Groups in the Amazon EC2 User Guide.

- When a placement group is specified, there must be available slotted capacity on your Outpost to successfully create a local Amazon EKS cluster. The capacity varies based on whether you use the host or rack spread type. If there isn't enough capacity, the cluster remains in the `Creating` state. You are able to check the `Insufficient Capacity Error` on the health field of the DescribeCluster API response. You must free capacity for the creation process to progress.
- During Amazon EKS local cluster platform and version updates, the Kubernetes control plane instances from your cluster are replaced by new instances using a rolling update strategy. During this replacement process, each control plane instance is terminated, freeing up its respective slot. A new updated instance is provisioned in its place. The updated instance might be placed in the slot that was released. If the slot is consumed by another unrelated instance and there is no more capacity left that respects the required spread topology requirement, then the cluster remains in the `Updating` state. You are able to see the respective `Insufficient Capacity Error` on the health field of the DescribeCluster API response. You must free capacity so the update process can progress and reestablish prior high availability levels.
- You can create a maximum of 500 placement groups per account in each AWS Region. For more information, see General rules and limitations in the Amazon EC2 User Guide.

Troubleshoot local Amazon EKS clusters on AWS Outposts

This topic covers some common errors that you might see while using local clusters and how to troubleshoot them. Local clusters are similar to Amazon EKS clusters in the cloud, but there are some differences in how they're managed by Amazon EKS.

> **Important**
>
> Never terminate any managed EKS local cluster `Kubernetes` control-plane instance running on Outpost unless explicitly instructed by AWS Support. Terminating these instances impose a risk to local cluster service availability, including loss of the local cluster in case multiple instances are simultaneously terminated. EKS local cluster `Kubernetes` control-plane instances are identified by the tag `eks-local:controlplane-name` on the EC2 instance console.

API behavior

Local clusters are created through the Amazon EKS API, but are run in an asynchronous manner. This means that requests to the Amazon EKS API return immediately for local clusters. However, these requests might succeed, fail fast because of input validation errors, or fail and have descriptive validation errors. This behavior is similar to the Kubernetes API.

Local clusters don't transition to a `FAILED` status. Amazon EKS attempts to reconcile the cluster state with the user-requested desired state in a continuous manner. As a result, a local cluster might remain in the `CREATING` state for an extended period of time until the underlying issue is resolved.

Describe cluster health field

Local cluster issues can be discovered using the describe-cluster Amazon EKS AWS CLI command. Local cluster issues are surfaced by the `cluster.health` field of the `describe-cluster` command's response. The message contained in this field includes an error code, descriptive message, and related resource IDs. This information is available through the Amazon EKS API and AWS CLI only. In the following example, replace *`my-cluster`* with the name of your local cluster.

```
aws eks describe-cluster --name my-cluster --query 'cluster.health'
```

An example output is as follows.

```
{
    "issues": [
        {
            "code": "ConfigurationConflict",
            "message": "The instance type 'm5.large' is not supported in Outpost 'my-
outpost-arn'.",
            "resourceIds": [
                "my-cluster-arn"
            ]
        }
    ]
}
```

If the problem is beyond repair, you might need to delete the local cluster and create a new one. For example, trying to provision a cluster with an instance type that's not available on your Outpost. The following table includes common health related errors.

Error scenario	Code	Message	ResourceIds
Provided subnets couldn't be found.	`ResourceNotFound`	`The subnet ID` *`subnet-id`* `does not exist`	All provided subnet IDs
Provided subnets don't belong to the same VPC.	`ConfigurationConflict`	`Subnets specified must belong to the same VPC`	All provided subnet IDs
Some provided subnets don't belong to the specified Outpost.	`ConfigurationConflict`	`Subnet` *`subnet-id`* `expected to be in` *`outpost-arn`* `, but is in` *`other-outpost-arn`*	Problematic subnet ID
Some provided subnets don't belong to any Outpost.	`ConfigurationConflict`	`Subnet` *`subnet-id`* `is not part of any Outpost`	Problematic subnet ID

Error scenario	Code	Message	ResourceIds
Some provided subnets don't have enough free addresses to create elastic network interfaces for control plane instances.	`ResourceLimitExceeded`	`The specified subnet does not have enough free addresses to satisfy the request.`	Problematic subnet ID
The specified control plane instance type isn't supported on your Outpost.	`ConfigurationConflict`	`The instance type` *`type`* `is not supported in Outpost` *`outpost-arn`*	Cluster ARN
You terminated a control plane Amazon EC2 instance or `run-instance` succeeded, but the state observed changes to `Terminated`. This can happen for a period of time after your Outpost reconnects and Amazon EBS internal errors cause an Amazon EC2 internal work flow to fail.	`InternalFailure`	`EC2 instance state "Terminated" is unexpected`	Cluster ARN

Error scenario	Code	Message	ResourceIds
You have insufficient capacity on your Outpost. This can also happen when a cluster is being created if an Outpost is disconnected from the AWS Region.	`ResourceLimitExceeded`	`There is not enough capacity on the Outpost to launch or start the instance.`	Cluster ARN
Your account exceeded your security group quota.	`ResourceLimitExceeded`	Error message returned by Amazon EC2 API	Target VPC ID
Your account exceeded your elastic network interface quota.	`ResourceLimitExceeded`	Error message returned by Amazon EC2 API	Target subnet ID
Control plane instances weren't reachable through AWS Systems Manager. For resolution, see the section called "Control plane instances aren't reachable through AWS Systems Manager".	`ClusterUnreachable`	Amazon EKS control plane instances are not reachable through SSM. Please verify your SSM and network configuration, and reference the EKS on Outposts troubleshooting documentation.	Amazon EC2 instance IDs

Error scenario	Code	Message	ResourceIds
An error occurred while getting details for a managed security group or elastic network interface.	Based on Amazon EC2 client error code.	Error message returned by Amazon EC2 API	All managed security group IDs
An error occurred while authorizing or revoking security group ingress rules. This applies to both the cluster and control plane security groups.	Based on Amazon EC2 client error code.	Error message returned by Amazon EC2 API	Problematic security group ID
An error occurred while deleting an elastic network interface for a control plane instance.	Based on Amazon EC2 client error code.	Error message returned by Amazon EC2 API	Problematic elastic network interface ID

The following table lists errors from other AWS services that are presented in the health field of the `describe-cluster` response.

Amazon EC2 error code	Cluster health issue code	Description
`AuthFailure`	`AccessDenied`	This error can occur for a variety of reasons. The most common reason is that you accidentally removed a tag that the service uses to scope down the service linked role policy from the control plane. If this occurs, Amazon EKS

Amazon EC2 error code	Cluster health issue code	Description
		can no longer manage and monitor these AWS resources.
`UnauthorizedOperation`	`AccessDenied`	This error can occur for a variety of reasons. The most common reason is that you accidentally removed a tag that the service uses to scope down the service linked role policy from the control plane. If this occurs, Amazon EKS can no longer manage and monitor these AWS resources.
`InvalidSubnetID.NotFound`	`ResourceNotFound`	This error occurs when subnet ID for the ingress rules of a security group can't be found.
`InvalidPermission.NotFound`	`ResourceNotFound`	This error occurs when the permissions for the ingress rules of a security group aren't correct.
`InvalidGroup.NotFound`	`ResourceNotFound`	This error occurs when the group of the ingress rules of a security group can't be found.
`InvalidNetworkInterfaceID.NotFound`	`ResourceNotFound`	This error occurs when the network interface ID for the ingress rules of a security group can't be found.
`InsufficientFreeAddressesInSubnet`	`ResourceLimitExceeded`	This error occurs when the subnet resource quota is exceeded.

Amazon EC2 error code	Cluster health issue code	Description
`InsufficientCapacityOnOutpost`	`ResourceLimitExceeded`	This error occurs when the outpost capacity quota is exceeded.
`NetworkInterfaceLimitExceeded`	`ResourceLimitExceeded`	This error occurs when the elastic network interface quota is exceeded.
`SecurityGroupLimitExceeded`	`ResourceLimitExceeded`	This error occurs when the security group quota is exceeded.
`VcpuLimitExceeded`	`ResourceLimitExceeded`	This is observed when creating an Amazon EC2 instance in a new account. The error might be similar to the following: "`You have requested more vCPU capacity than your current vCPU limit of 32 allows for the instance bucket that the specified instance type belongs to. Please visit` http://aws.amazon.com/contact-us/ec2-request `to request an adjustment to this limit.`"
`InvalidParameterValue`	`ConfigurationConflict`	Amazon EC2 returns this error code if the specified instance type isn't supported on the Outpost.

Amazon EC2 error code	Cluster health issue code	Description
All other failures	`InternalFailure`	None

Unable to create or modify clusters

Local clusters require different permissions and policies than Amazon EKS clusters that are hosted in the cloud. When a cluster fails to create and produces an `InvalidPermissions` error, double check that the cluster role that you're using has the AmazonEKSLocalOutpostClusterPolicy managed policy attached to it. All other API calls require the same set of permissions as Amazon EKS clusters in the cloud.

Cluster is stuck in `CREATING` state

The amount of time it takes to create a local cluster varies depending on several factors. These factors include your network configuration, Outpost configuration, and the cluster's configuration. In general, a local cluster is created and changes to the `ACTIVE` status within 15–20 minutes. If a local cluster remains in the `CREATING` state, you can call `describe-cluster` for information about the cause in the `cluster.health` output field.

The most common issues are the following:

- Your cluster can't connect to the control plane instance from the AWS Region that Systems Manager is in. You can verify this by calling `aws ssm start-session --target instance-id` from an in-Region bastion host. If that command doesn't work, check if Systems Manager is running on the control plane instance. Or, another work around is to delete the cluster and then recreate it.
- The control plane instances fail to create due to KMS key permissions for EBS volumes. With user managed KMS keys for encrypted EBS volumes, the control plane instances will terminate if the key is not accessible. If the instances are terminated, either switch to an AWS managed KMS key or ensure that your user managed key policy grants the necessary permissions to the cluster role.
- Systems Manager control plane instances might not have internet access. Check if the subnet that you provided when you created the cluster has a NAT gateway and a VPC with an internet gateway. Use VPC reachability analyzer to verify that the control plane instance can reach the internet gateway. For more information, see Getting started with VPC Reachability Analyzer.

- The role ARN that you provided is missing policies. Check if the AWS managed policy: AmazonEKSLocalOutpostClusterPolicy was removed from the role. This can also occur if an AWS CloudFormation stack is misconfigured.
- All the provided subnets must be associated with the same Outpost and must reach each other. When multiple subnets are specified when a cluster is created, Amazon EKS attempts to spread the control plane instances across multiple subnets.
- The Amazon EKS managed security groups are applied at the elastic network interface. However, other configuration elements such as NACL firewall rules might conflict with the rules for the elastic network interface.

VPC and subnet DNS configuration is misconfigured or missing

Review Create a VPC and subnets for Amazon EKS clusters on AWS Outposts.

Cluster is stuck in `UPDATING` state

Amazon EKS automatically updates all existing local clusters to the latest platform versions for their corresponding Kubernetes minor version. For more information about platform versions, please refer to the section called “EKS platform versions”.

During an automatic platform-version rollout a cluster status changes to `UPDATING`. The update process consists of the replacement of all Kubernetes control-plane instances with new ones containing the latest security pathces and bugfixes released for the respective Kubernetes minor version. In general, a local cluster platform update process completes within less than 30 minutes and the cluster changes back to `ACTIVE` status. If a local cluster remains in the `UPDATING` state for an extended period of time, you may call `describe-cluster` to check for information about the cause in the `cluster.health` output field.

Amazon EKS ensures at least 2 out of 3 Kubernetes control-plane instances are healthy and operational cluster nodes in order to maintain the local cluster availability and prevent service interruption. If a local cluster is stalled in `UPDATING` state it is usually because there is some infrastructure or configuration issue preventing the two-instances minimum availability to be guaranteed in case the process continues. So the update process stops progressing to protect the local cluster service interruption.

It is important to troubleshoot a local cluster stuck in `UPDATING` status and address the root-cause so that the update process can complete and restore the local cluster back to `ACTIVE` with the high-availability of 3 Kubernetes control-plane instances.

Do not terminate any managed EKS local cluster `Kubernetes` instances on Outposts unless explicitly instructed by AWS Support. This is specially important for local clusters stuck in `UPDATING` state because there's a high probability that another control-plane nodes is not completely healthy and terminating the wrong instance could cause service interruption and risk local-cluster data loss.

The most common issues are the following:

- One or more control-plane instances are unable to connect to System Manager because of a networking configuration change since the local cluster was first created. You can verify this by calling `aws ssm start-session --target instance-id` from an in-Region bastion host. If that command doesn't work, check if Systems Manager is running on the control plane instance.
- New control plane instances fail to be created due to KMS key permissions for EBS volumes. With user managed KMS keys for encrypted EBS volumes, the control plane instances will terminate if the key is not accessible. If the instances are terminated, either switch to an AWS managed KMS key or ensure that your user managed key policy grants the necessary permissions to the cluster role.
- Systems Manager control plane instances might have lost internet access. Check if the subnet that was provided when you created the cluster has a NAT gateway and a VPC with an internet gateway. Use VPC reachability analyzer to verify that the control plane instance can reach the internet gateway. For more information, see Getting started with VPC Reachability Analyzer. If your private networks don't have outbound internet connection, ensure that all the required VPC endpoints and gateway endpoint are still present in the Regional subnet from your cluster (see the section called "Subnet access to AWS services").
- The role ARN that you provided is missing policies. Check if the AWS managed policy: AmazonEKSLocalOutpostClusterPolicy was not removed from the role.
- One of the new Kubernetes control-plane instances may have experienced an unexpected bootstrapping failure. Please file a ticket with AWS Support Center for further guidance on troubleshooting and log-collection in this exceptional case.

Can't join nodes to a cluster

- AMI issues:
  - You're using an unsupported AMI. You must use v20220620 or later for the Create nodes with optimized Amazon Linux AMIs Amazon EKS optimized Amazon Linux.

  - If you used an AWS CloudFormation template to create your nodes, make sure it wasn't using an unsupported AMI.
- Missing the AWS IAM Authenticator `ConfigMap` – If it's missing, you must create it. For more information, see the section called "Apply the `aws-auth` `ConfigMap` to your cluster" .
- The wrong security group is used – Make sure to use `eks-cluster-sg-cluster-name-uniqueid` for your worker nodes' security group. The selected security group is changed by AWS CloudFormation to allow a new security group each time the stack is used.
- Following unexpected private link VPC steps – Wrong CA data (`--b64-cluster-ca`) or API Endpoint (`--apiserver-endpoint`) are passed.
- Misconfigured Pod security policy:
  - The CoreDNS and Amazon VPC CNI plugin for Kubernetes Daemonsets must run on nodes for nodes to join and communicate with the cluster.
  - The Amazon VPC CNI plugin for Kubernetes requires some privileged networking features to work properly. You can view the privileged networking features with the following command: `kubectl describe psp eks.privileged`.

  We don't recommend modifying the default pod security policy. For more information, see the section called "Legacy default PSP".

Collecting logs

When an Outpost gets disconnected from the AWS Region that it's associated with, the Kubernetes cluster likely will continue working normally. However, if the cluster doesn't work properly, follow the troubleshooting steps in Prepare local Amazon EKS clusters on AWS Outposts for network disconnects. If you encounter other issues, contact AWS Support. AWS Support can guide you on downloading and running a log collection tool. That way, you can collect logs from your Kubernetes cluster control plane instances and send them to AWS Support support for further investigation.

Control plane instances aren't reachable through AWS Systems Manager

When the Amazon EKS control plane instances aren't reachable through AWS Systems Manager (Systems Manager), Amazon EKS displays the following error for your cluster.

```
Amazon EKS control plane instances are not reachable through SSM. Please verify your
 SSM and network configuration, and reference the EKS on Outposts troubleshooting
 documentation.
```

To resolve this issue, make sure that your VPC and subnets meet the requirements in Create a VPC and subnets for Amazon EKS clusters on AWS Outposts and that you completed the steps in Setting up Session Manager in the AWS Systems Manager User Guide.

Create Amazon Linux nodes on AWS Outposts

This topic describes how you can launch Auto Scaling groups of Amazon Linux nodes on an Outpost that register with your Amazon EKS cluster. The cluster can be on the AWS Cloud or on an Outpost.

- An existing Outpost. For more information, see What is AWS Outposts.
- An existing Amazon EKS cluster. To deploy a cluster on the AWS Cloud, see the section called "Create a cluster". To deploy a cluster on an Outpost, see the section called "Run local clusters".
- Suppose that you're creating your nodes in a cluster on the AWS Cloud and you have subnets in the AWS Region where you have AWS Outposts, AWS Wavelength, or AWS Local Zones enabled. Then, those subnets must not have been passed in when you created your cluster. If you're creating your nodes in a cluster on an Outpost, you must have passed in an Outpost subnet when creating your cluster.
- (Recommended for clusters on the AWS Cloud) The Amazon VPC CNI plugin for Kubernetes add-on configured with its own IAM role that has the necessary IAM policy attached to it. For more information, see the section called "Configure for IRSA". Local clusters do not support IAM roles for service accounts.

You can create a self-managed Amazon Linux node group with `eksctl` or the AWS Management Console (with an AWS CloudFormation template). You can also use Terraform.

You can create a self-managed node group for local cluster with the following tools described in this page:

- the section called "`eksctl`"
- the section called "AWS Management Console"

Important

- Self-managed node group includes Amazon EC2 instances in your account. These instances aren't automatically upgraded when you or Amazon EKS update the control plane version on your behalf. A self-managed node group doesn't have any indication in the console that it needs updating. You can view the `kubelet` version installed on a node by selecting the node in the **Nodes** list on the **Overview** tab of your cluster to determine which nodes need updating. You must manually update the nodes. For more information, see the section called "Update methods".

- The certificates used by kubelet on your self-managed nodes are issued with one year expiration. By default certificate rotation is **not** enabled (see: https://kubernetes.io/docs/reference/config-api/kubelet-config.v1beta1/#kubelet-config-k8s-io-v1beta1-KubeletConfiguration), this means if you have a self-managed node running for more than one year, it will no longer be able to authenticate to the Kubernetes API.
- As a best practice we recommend customers to regularly update their self-managed node groups to receive CVEs and security patches from latest Amazon EKS optimized AMI. Updating AMI used in self-managed node groups also triggers re-creation of nodes and make sure they do not run into issue due to expired kubelet certificates.
- Alternatively you can also enable client certificate rotation (see: https://kubernetes.io/docs/tasks/tls/certificate-rotation/) when creating the self-managed node groups to make sure kubelet certificates are renewed as the current certificate approaches expiration.

`eksctl`

To launch self-managed Linux nodes using `eksctl`

1. Install version `0.204.0` or later of the `eksctl` command line tool installed on your device or AWS CloudShell. To install or update `eksctl`, see Installation in the `eksctl` documentation.
2. If your cluster is on the AWS Cloud and the **AmazonEKS_CNI_Policy** managed IAM policy is attached to your Amazon EKS node IAM role, we recommend assigning it to an IAM role that you associate to the Kubernetes `aws-node` service account instead. For more information, see the section called "Configure for IRSA". If your cluster in on your Outpost, the policy must be attached to your node role.
3. The following command creates a node group in an existing cluster. The cluster must have been created using `eksctl`. Replace *`al-nodes`* with a name for your node group. The node group name can't be longer than 63 characters. It must start with letter or digit, but can also include hyphens and underscores for the remaining characters. Replace *`my-cluster`* with the name of your cluster. The name can contain only alphanumeric characters (case-sensitive) and hyphens.

It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in. If your cluster exists on an Outpost, replace *id* with the ID of an Outpost subnet. If your cluster exists on the AWS Cloud, replace *id* with the ID of a subnet that you didn't specify when you created your cluster. Replace *instance-type* with an instance type supported by your Outpost. Replace the remaining *example values* with your own values. The nodes are created with the same Kubernetes version as the control plane, by default.

Replace *instance-type* with an instance type available on your Outpost.

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

Create your node group with the following command.

```
eksctl create nodegroup --cluster my-cluster --name al-nodes --node-type instance-type \
    --nodes 3 --nodes-min 1 --nodes-max 4 --managed=false --node-volume-type gp2 --subnet-ids subnet-id
```

If your cluster is deployed on the AWS Cloud:

- The node group that you deploy can assign `IPv4` addresses to Pods from a different CIDR block than that of the instance. For more information, see the section called "Custom networking".
- The node group that you deploy doesn't require outbound internet access. For more information, see the section called "Private clusters".

For a complete list of all available options and defaults, see AWS Outposts Support in the `eksctl` documentation.

- If nodes fail to join the cluster, then see the section called "Nodes fail to join cluster" in Troubleshoot problems with Amazon EKS clusters and nodes and the section called "Can't join nodes to a cluster" in Troubleshoot local Amazon EKS clusters on AWS Outposts.
- An example output is as follows. Several lines are output while the nodes are created. One of the last lines of output is the following example line.

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

4. (Optional) Deploy a sample application to test your cluster and Linux nodes.

AWS Management Console

Step 1: Launch self-managed Linux nodes using AWS Management Console

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

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2022-12-23/
amazon-eks-nodegroup.yaml
```

2. Open the AWS CloudFormation console.
3. Choose **Create stack** and then select **With new resources (standard)**.
4. For **Specify template**, select **Upload a template file** and then select **Choose file**. Select the `amazon-eks-nodegroup.yaml` file that you downloaded in a previous step and then select **Next**.
5. On the **Specify stack details** page, enter the following parameters accordingly, and then choose **Next**:
   - **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it *al-nodes*. The name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphanumeric character and can't be longer than 100 characters. The name must be unique within the AWS Region and AWS account that you're creating the cluster in.
   - **ClusterName**: Enter the name of your cluster. If this name doesn't match your cluster name, your nodes can't join the cluster.
   - **ClusterControlPlaneSecurityGroup**: Choose the **SecurityGroups** value from the AWS CloudFormation output that you generated when you created your VPC.

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

     i. Open the Amazon EKS console.

     ii. Choose the name of the cluster.

     iii. Choose the **Networking** tab.

iv. Use the **Additional security groups** value as a reference when selecting from the **ClusterControlPlaneSecurityGroup** dropdown list.

- **NodeGroupName**: Enter a name for your node group. This name can be used later to identify the Auto Scaling node group that's created for your nodes.
- **NodeAutoScalingGroupMinSize**: Enter the minimum number of nodes that your node Auto Scaling group can scale in to.
- **NodeAutoScalingGroupDesiredCapacity**: Enter the desired number of nodes to scale to when your stack is created.
- **NodeAutoScalingGroupMaxSize**: Enter the maximum number of nodes that your node Auto Scaling group can scale out to.
- **NodeInstanceType**: Choose an instance type for your nodes. If your cluster is running on the AWS Cloud, then for more information, see the section called "Amazon EC2 instance types". If your cluster is running on an Outpost, then you can only select an instance type that is available on your Outpost.
- **NodeImageIdSSMParam**: Pre-populated with the Amazon EC2 Systems Manager parameter of a recent Amazon EKS optimized AMI for a variable Kubernetes version. To use a different Kubernetes minor version supported with Amazon EKS, replace *`1.XX`* with a different supported version. We recommend specifying the same Kubernetes version as your cluster.

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

  > **Note**
  >
  > The Amazon EKS node AMIs are based on Amazon Linux. You can track security or privacy events for Amazon Linux at the Amazon Linux security center by choosing the tab for your desired version. You can also subscribe to the applicable RSS feed. Security and privacy events include an overview of the issue, what packages are affected, and how to update your instances to correct the issue.

- **NodeImageId**: (Optional) If you're using your own custom AMI (instead of an Amazon EKS optimized AMI), enter a node AMI ID for your AWS Region. If you specify a value here, it overrides any values in the **NodeImageIdSSMParam** field.
- **NodeVolumeSize**: Specify a root volume size for your nodes, in GiB.

- **NodeVolumeType**: Specify a root volume type for your nodes.
- **KeyName**: Enter the name of an Amazon EC2 SSH key pair that you can use to connect using SSH into your nodes with after they launch. If you don't already have an Amazon EC2 key pair, you can create one in the AWS Management Console. For more information, see Amazon EC2 key pairs in the *Amazon EC2 User Guide*.

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

- **BootstrapArguments**: There are several optional arguments that you can pass to your nodes. For more information, view the bootstrap script usage information on GitHub. If you're adding nodes to an Amazon EKS Local Cluster on AWS Outposts (where the Kubernetes control plane instances run on AWS Outposts) and the cluster doesn't have ingress and egress internet connection (also known as private clusters), then you must provide the following bootstrap arguments (as a single line).

```
--b64-cluster-ca ${CLUSTER_CA} --apiserver-endpoint https://${APISERVER_ENDPOINT}
 --enable-local-outpost true --cluster-id ${CLUSTER_ID}
```

  To retrieve the values for `CLUSTER_CA`, `APISERVER_ENDPOINT`, and `CLUSTER_ID` of your Amazon EKS local cluster, run the following AWS CLI commands. Replace cluster-name with the name of your cluster and region (for example, us-east-1) with your cluster's AWS Region.

```
echo "CLUSTER_CA=$(aws eks describe-cluster --name cluster-name --region region --
query cluster.certificateAuthority.data --output text)"

echo "APISERVER_ENDPOINT=$(aws eks describe-cluster --name cluster-name --region
 region --query cluster.endpoint --output text)"

echo "CLUSTER_ID=$(aws eks describe-cluster --name cluster-name --region region --
query cluster.id --output text)"
```

- **DisableIMDSv1**: By default, each node supports the Instance Metadata Service Version 1 (IMDSv1) and IMDSv2. You can disable IMDSv1. To prevent future nodes and Pods in the node group from using IMDSv1, set **DisableIMDSv1** to **true**. For more information about IMDS, see Configuring the instance metadata service. For more information about restricting access to it on your nodes, see Restrict access to the instance profile assigned to the worker node.

- **VpcId**: Enter the ID for the VPC that you created. Before choosing a VPC, review VPC requirements and considerations.
- **Subnets**: If your cluster is on an Outpost, then choose at least one private subnet in your VPC. Before choosing subnets, review Subnet requirements and considerations. You can see which subnets are private by opening each subnet link from the **Networking** tab of your cluster.

6. Select your desired choices on the **Configure stack options** page, and then choose **Next**.
7. Select the check box to the left of **I acknowledge that AWS CloudFormation might create IAM resources.**, and then choose **Create stack**.
8. When your stack has finished creating, select it in the console and choose **Outputs**.
9. Record the **NodeInstanceRole** for the node group that was created. You need this when you configure your Amazon EKS nodes.

Step 2: Enable nodes to join your cluster

1. Check to see if you already have an `aws-auth` `ConfigMap`.

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

2. If you are shown an `aws-auth` `ConfigMap`, then update it as needed.

   a. Open the `ConfigMap` for editing.

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

   b. Add a new `mapRoles` entry as needed. Set the `rolearn` value to the **NodeInstanceRole** value that you recorded in the previous procedure.

   ```
   [...]
   data:
     mapRoles: |
       - rolearn: <ARN of instance role (not instance profile)>
         username: system:node:{{EC2PrivateDNSName}}
         groups:
           - system:bootstrappers
           - system:nodes
   [...]
   ```

   c. Save the file and exit your text editor.

3. If you received an error stating "`Error from server (NotFound): configmaps "aws-auth" not found`, then apply the stock `ConfigMap`.

   a. Download the configuration map.

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

   b. In the `aws-auth-cm.yaml` file, set the `rolearn` to the **NodeInstanceRole** value that you recorded in the previous procedure. You can do this with a text editor, or by replacing *`my-node-instance-role`* and running the following command:

   ```
   sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-role|' aws-auth-cm.yaml
   ```

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

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

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

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

   Enter `Ctrl+C` to return to a shell prompt.

   Note

   If you receive any authorization or resource type errors, see the section called "Unauthorized or access denied (`kubectl`)" in the troubleshooting topic.

   If nodes fail to join the cluster, then see the section called "Nodes fail to join cluster" in Troubleshoot problems with Amazon EKS clusters and nodes and the section called "Can't join nodes to a cluster" in Troubleshoot local Amazon EKS clusters on AWS Outposts.

5. Install the Amazon EBS CSI driver. For more information, see Installation on GitHub. In the **Set up driver permission** section, make sure to follow the instruction for the **Using IAM instance profile** option. You must use the gp2 storage class. The gp3 storage class isn't supported.

   To create a gp2 storage class on your cluster, complete the following steps.

a. Run the following command to create the `gp2-storage-class.yaml` file.

```
cat >gp2-storage-class.yaml <<EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp2
  encrypted: "true"
allowVolumeExpansion: true
EOF
```

b. Apply the manifest to your cluster.

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

6. (GPU nodes only) If you chose a GPU instance type and an Amazon EKS optimized accelerated AMI, you must apply the NVIDIA device plugin for Kubernetes as a DaemonSet on your cluster. Replace *vX.X.X* with your desired NVIDIA/k8s-device-plugin version before running the following command.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
deployments/static/nvidia-device-plugin.yml
```

Step3: Additional actions

1. (Optional) Deploy a sample application to test your cluster and Linux nodes.
2. If your cluster is deployed on an Outpost, then skip this step. If your cluster is deployed on the AWS Cloud, the following information is optional. If the **AmazonEKS_CNI_Policy** managed IAM policy is attached to your Amazon EKS node IAM role, we recommend assigning it to an IAM role that you associate to the Kubernetes `aws-node` service account instead. For more information, see the section called “Configure for IRSA”.

Overview of Machine Learning on Amazon EKS

Machine Learning (ML) is an area of Artificial Intelligence (AI) where machines process large amounts of data to look for patterns and make connections between the data. This can expose new relationships and help predict outcomes that might not have been apparent otherwise.

For large-scale ML projects, data centers must be able to store large amounts of data, process data quickly, and integrate data from many sources. The platforms running ML applications must be reliable and secure, but also offer resiliency to recover from data center outages and application failures. AWS Elastic Kubernetes Service (EKS), running in the AWS cloud, is particularly suited for ML workloads.

The primary goal of this section of the EKS User Guide is to help you put together the hardware and software component to build platforms to run Machine Learning workloads in an EKS cluster. We start by explaining the features and services available to you in EKS and the AWS cloud, then provide you with tutorials to help you work with ML platforms, frameworks, and models.

Advantages of Machine Learning on EKS and the AWS cloud

Amazon Elastic Kubernetes Service (EKS) is a powerful, managed Kubernetes platform that has become a cornerstone for deploying and managing AI/ML workloads in the cloud. With its ability to handle complex, resource-intensive tasks, Amazon EKS provides a scalable and flexible foundation for running AI/ML models, making it an ideal choice for organizations aiming to harness the full potential of machine learning.

Key Advantages of AI/ML Platforms on Amazon EKS include:

- **Scalability and Flexibility** Amazon EKS enables organizations to scale AI/ML workloads seamlessly. Whether you’re training large language models that require vast amounts of compute power or deploying inference pipelines that need to handle unpredictable traffic patterns, EKS scales up and down efficiently, optimizing resource use and cost.
- **High Performance with GPUs and Neuron Instances** Amazon EKS supports a wide range of compute options, including GPUs and AWS} Neuron instances, which are essential for accelerating AI/ML workloads. This support allows for high-performance training and low-latency inference, ensuring that models run efficiently in production environments.
- **Integration with AI/ML Tools** Amazon EKS integrates seamlessly with popular AI/ML tools and frameworks like TensorFlow, PyTorch, and Ray, providing a familiar and robust ecosystem for

data scientists and engineers. These integrations enable users to leverage existing tools while benefiting from the scalability and management capabilities of Kubernetes.

- **Automation and Management** Kubernetes on Amazon EKS automates many of the operational tasks associated with managing AI/ML workloads. Features like automatic scaling, rolling updates, and self-healing ensure that your applications remain highly available and resilient, reducing the overhead of manual intervention.
- **Security and Compliance** Running AI/ML workloads on Amazon EKS provides robust security features, including fine-grained IAM roles, encryption, and network policies, ensuring that sensitive data and models are protected. EKS also adheres to various compliance standards, making it suitable for enterprises with strict regulatory requirements.

Why Choose Amazon EKS for AI/ML?

Amazon EKS offers a comprehensive, managed environment that simplifies the deployment of AI/ML models while providing the performance, scalability, and security needed for production workloads. With its ability to integrate with a variety of AI/ML tools and its support for advanced compute resources, EKS empowers organizations to accelerate their AI/ML initiatives and deliver innovative solutions at scale.

By choosing Amazon EKS, you gain access to a robust infrastructure that can handle the complexities of modern AI/ML workloads, allowing you to focus on innovation and value creation rather than managing underlying systems. Whether you are deploying simple models or complex AI systems, Amazon EKS provides the tools and capabilities needed to succeed in a competitive and rapidly evolving field.

Start using Machine Learning on EKS

To begin planning for and using Machine Learning platforms and workloads on EKS on the AWS cloud, proceed to the the section called "Get started with ML" section.

Get started with ML

To jump into Machine Learning on EKS, start by choosing from these prescriptive patterns to quickly get an EKS cluster and ML software and hardware ready to begin running ML workloads. Most of these patterns are based on Terraform blueprints that are available from the Data on Amazon EKS site. Before you begin, here are few things to keep in mind:

- GPUs or Neuron instances are required to run these procedures. Lack of availability of these resources can cause these procedures to fail during cluster creation or node autoscaling.
- Neuron SDK (Tranium and Inferentia-based instances) can save money and are more available than NVIDIA GPUs. So, when your workloads permit it, we recommend that you consider using Neuron for your Machine Learning workloads (see Welcome to AWS Neuron).
- Some of the getting started experiences here require that you get data via your own Hugging Face account.

To get started, choose from the following selection of patterns that are designed to get you started setting up infrastructure to run your Machine Learning workloads:

- **JupyterHub on EKS** : Explore the JupyterHub blueprint, which showcases Time Slicing and MIG features, as well as multi-tenant configurations with profiles. This is ideal for deploying large-scale JupyterHub platforms on EKS.
- **Large Language Models on AWS Neuron and RayServe** : Use AWS Neuron to run large language models (LLMs) on Amazon EKS and AWS Trainium and AWS Inferentia accelerators. See Serving LLMs with RayServe and vLLM on AWS Neuron for instructions on setting up a platform for making inference requests, with components that include:
  - AWS Neuron SDK toolkit for deep learning
  - AWS Inferentia and Trainium accelerators
  - vLLM - variable-length language model (see the vLLM documentation site)
  - RayServe scalable model serving library (see the Ray Serve: Scalable and Programmable Serving site)
  - Llama-3 language model, using your own Hugging Face account.
  - Observability with AWS CloudWatch and Neuron Monitor
  - Open WebUI
- **Large Language Models on NVIDIA and Triton** : Deploy multiple large language models (LLMs) on Amazon EKS and NVIDIA GPUs. See Deploying Multiple Large Language Models with NVIDIA Triton Server and vLLM for instructions for setting up a platform for making inference requests, with components that include:
  - NVIDIA Triton Inference Server (see the Triton Inference Server GitHub site)
  - vLLM - variable-length language model (see the vLLM documentation site)
  - Two language models: mistralai/Mistral-7B-Instruct-v0.2 and meta-llama/Llama-2-7b-chat-hf, using your own Hugging Face account.

Continuing with ML on EKS

Along with choosing from the blueprints described on this page, there are other ways you can proceed through the ML on EKS documentation if you prefer. For example, you can:

- **Try tutorials for ML on EKS** – Run other end-to-end tutorials for building and running your own Machine Learning models on EKS. See the section called "Try tutorials for ML on EKS".

To improve your work with ML on EKS, refer to the following:

- **Prepare for ML** – Learn how to prepare for ML on EKS with features like custom AMIs and GPU reservations. See the section called "Prepare for ML".

Prepare for ML clusters

There are ways that you can enhance your Machine Learning on EKS experience. Following pages in this section will help you:

- Understand your choices for using ML on EKS and
- Help in preparation of your EKS and ML environment.

In particular, this will help you:

- **Choose AMIs**: AWS offers multiple customized AMIs for running ML workloads on EKS. See the section called "Run Linux GPU AMIs" and the section called "Run Windows GPU AMIs".
- **Customize AMIs**: You can further modify AWS custom AMIs to add other software and drivers needed for your particular use cases. See the section called "Reserve GPUs for SMN".
- **Reserve GPUs**: Because of the demand for GPUs, to ensure that the GPUs you need are available when you need them, you can reserve the GPUs you need in advance. See the section called "Taint GPU nodes".
- **Add EFA**: Add the Elastic Fabric Adapter to improve network performance for inter-node cluster communications. See the section called "Training clusters with EFA".
- **Use AWSInferentia workloads**: Create an EKS cluster with Amazon EC2 Inf1 instances. See the section called "Prepare Inferentia clusters".

Topics

- Run GPU-accelerated containers (Linux on EC2)
- Run GPU-accelerated containers (Windows on EC2 G-Series)
- Create a managed node group with Capacity Blocks for ML
- Create self-managed nodes with Capacity Blocks for ML
- Prevent Pods from being scheduled on specific nodes
- Run machine learning training on Amazon EKS with Elastic Fabric Adapter
- Use AWS Inferentia instances with Amazon EKS for Machine Learning

Run GPU-accelerated containers (Linux on EC2)

The Amazon EKS optimized accelerated Amazon Linux AMIs are built on top of the standard Amazon EKS optimized Amazon Linux AMIs. For details on these AMIs, see the section called "Amazon EKS optimized accelerated Amazon Linux AMIs". The following text describes how to enable AWS Neuron-based workloads.

To enable AWS Neuron (ML accelerator) based workloads

For details on training and inference workloads using Neuron in Amazon EKS, see the following references:

- Containers - Kubernetes - Getting Started in the *AWS Neuron Documentation*
- Training in AWS Neuron EKS Samples on GitHub
- Deploy ML inference workloads with AWSInferentia on Amazon EKS

The following procedure describes how to run a workload on a GPU based instance with the Amazon EKS optimized accelerated AMIs.

1. After your GPU nodes join your cluster, you must apply the NVIDIA device plugin for Kubernetes as a DaemonSet on your cluster. Replace *vX.X.X* with your desired NVIDIA/k8s-device-plugin version before running the following command.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
deployments/static/nvidia-device-plugin.yml
```

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

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

3. Create a file named `nvidia-smi.yaml` with the following contents. Replace *`tag`* with your desired tag for nvidia/cuda. This manifest launches an NVIDIA CUDA container that runs `nvidia-smi` on a node.

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

4. Apply the manifest with the following command.

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

5. After the Pod has finished running, view its logs with the following command.

```
kubectl logs nvidia-smi
```

An example output is as follows.

```
Mon Aug  6 20:23:31 20XX
+-----------------------------------------------------------------------------+
| NVIDIA-SMI XXX.XX                 Driver Version: XXX.XX                    |
|-------------------------------+----------------------+----------------------+
| 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                                                 |
+-----------------------------------------------------------------------------+
```

Run GPU-accelerated containers (Windows on EC2 G-Series)

> **Important**
>
> The Kubernetes Device Plugin for DirectX by TensorWorks is a third-party tool that is not endorsed, supported, or maintained by AWS. AWS assumes no responsibility for the security, reliability, or performance of this plugin.

Learn how to run GPU-accelerated Windows container workloads on Amazon EKS (Elastic Kubernetes Service) using NVIDIA GPUs with the Kubernetes Device Plugin for DirectX by TensorWorks. For more information, see Kubernetes Device Plugin for DirectX.

There are two main approaches to setting up GPU-acceleration for your Windows containers:

- **Option 1**: Build a custom EKS Windows Optimized AMI with the required GPU drivers pre-installed.
  - Use this approach when you need a consistent, pre-configured environment ready to run GPU-accelerated Windows containers, and you're able to invest the additional effort to build and maintain the custom AMI.
- **Option 2**: Install the necessary GPU drivers on your EKS worker nodes after launching your instance.
  - Use this approach when you want a simpler setup process and don't mind installing the GPU drivers on each new worker node. More suited to a development environment when you are evaluating or prototyping GPU-accelerated workloads.

Both approaches can be leveraged using the steps detailed in this guide.

Considerations

This guide provides steps to install and set up GPU-acceleration for your Windows containers using NVIDIA GPUs, NVIDIA GRID drivers, and the Kubernetes Device Plugin for DirectX by TensorWorks. The steps have been tested and verified to provide GPU-acceleration for your Windows container workloads on Amazon EKS. See the section called "Known limitations" for more information on compatible drivers and device plugins. Before proceeding, note the following:

- Only G-family instance types with NVIDIA GRID drivers have been tested and verified to work with this guide. While other instance types and driver combinations may also be capable of running GPU-accelerated Windows containers, they may require additional configuration steps not covered in this guide.
- Only DirectX-based workloads have been tested and verified to work with this guide. While other GPU APIs like OpenGL, Vulkan, and OpenCL may potentially be compatible to run GPU-accelerated Windows containers, they may require additional configuration steps not covered in this guide.
- There are some known limitations to be aware of before running GPU-accelerated Windows containers. Please see the the section called "Known limitations" section for more information.

Prerequisites

To enable GPU acceleration for your Windows containers on Amazon EKS, you'll need to prepare the following requirements before proceeding:

- Launch an Amazon EKS cluster with Kubernetes v1.27 or newer.
- Provision Windows nodes with Windows Server 2022 or newer.
- Provision Windows nodes in the G-family of instance types, such as G4 or G5.
- Provision Windows nodes with a container runtime with containerd `1.7.x` or `2.x.x`. (See the section called "Get version information" to verify the containerd version in your Amazon EKS Optimized AMI.)

Install the GPU driver on each Windows Windows node

To install the NVIDIA GRID drivers on your EKS worker nodes, follow the steps outlined in NVIDIA drivers for your Amazon EC2 instance. Navigate to Installation options - Option 3: GRID drivers and follow the installation steps.

Install for Windows Server Core

For Windows Server Core, which doesn't have a desktop experience, install NVIDIA GRID drivers silently by using the following commands:

```
$nvidiaInstallerFilePath = nvidia-driver-installer.exe # Replace with path to installer
$installerArguments = "-s -clean -noreboot -noeula"
Start-Process -FilePath $nvidiaInstallerFilePath -ArgumentList $installerArguments -
Wait -NoNewWindow -PassThru
```

Verify your installation

Run the following PowerShell command to show diagnostic information about the GPUs on the instance:

```
nvidia-smi
```

This command displays the NVIDIA driver version, as well as information about the GPU hardware. Ensure that the output of this command matches the NVIDIA GRID driver version you expected to be installed.

Deploy the GPU device plugin on each node

To enable discovery and exposure of the GPU resources to containers on your Windows nodes, you will need a device plugin. Deploy the DirectX Device Plugin by Tensorworks on each worker node by running it as a DaemonSet in your EKS cluster. Follow the installation guide specified in the README.md, which will entail the following steps. It is recommended to:

- Deploy the device plugin in the `kube-system` namespace.
- Set appropriate resource limits for the DaemonSet to ensure it does not consume excessive resources on your nodes.

> **Note**
>
> The device plugin DaemonSet will run on every node as a host process container with elevated privileges. It is recommended to implement RBAC controls to restrict access to this DaemonSet so only authorized users can execute privileged commands.

When running GPU-accelerated containers, the device plugin supports two modes:

- **Single-tenancy mode**: This mode dedicates all GPU resources to a single container on the instance. Install the device plugins with single-tenancy support using the following command. See README.md for more information.

```
kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-
plugins/main/deployments/default-daemonsets.yml"
```

- **Multi-tenancy mode**: This mode allows sharing GPU resources among multiple containers on the instance. Install the device plugins with multi-tenancy support using the following command. See README.md for more information.

```
kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-
plugins/main/deployments/multitenancy-inline.yml"
```

  Alternatively, use a ConfigMap to specify the multi-tenancy.

```
kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-
plugins/main/deployments/multitenancy-configmap.yml"
```

Verifying the device plugin deployment

After you have deployed the device plugin, replace `<namespace>` and run the following command to verify the DirectX Device Plugin is running correctly on your all your Windows nodes.

```
kubectl get ds device-plugin-wddm -n <namespace>
```

Verifying containers are ready for deployment

Once the device plugin DaemonSet is running on the GPU-powered Windows worker nodes, use the following command to verify that each node has allocatable GPUs. The corresponding number should match the number of DirectX devices on each node.

```
kubectl get nodes "-o=custom-
columns=NAME:.metadata.name,DirectX:.status.allocatable.directx\.microsoft\.com/
display"
```

Running Windows containers with GPU-acceleration

Before launching your pods, specify the resource name `directx.microsoft.com/display` in `.spec.containers[].resources`. This will indicate that your containers require GPU-enabled capabilities, and the `kube-scheduler` will attempt to place your pods on your pre-configured Windows node with available GPU resources.

As an example, see the sample command below which launches a `Job` to run Monte Carlo simulation to estimate the value of pi. This example is from the [Kubernetes Device Plugins for DirectX] GitHub repository, which has [multiple examples] to choose from that you can run to test your Windows node GPU capabilities.

```
cat <<EOF | kubectl apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: example-cuda-montecarlo-wddm
spec:
  template:
    spec:
      containers:
      - name: example-cuda-montecarlo-wddm
        image: "index.docker.io/tensorworks/example-cuda-montecarlo:0.0.1"
        resources:
          limits:
            directx.microsoft.com/display: 1
      nodeSelector:
        "kubernetes.io/os": windows
      restartPolicy: Never
  backoffLimit: 0
EOF
```

Known limitations

All GPUs are usable

All the GPUs on the instance will be usable by each running container on the host, even when you request a specific number of GPUs for a given container. Additionally, the default behavior is that all containers running on the host will use the GPU with index 0, even if there are multiple GPUs available on the node. Thus, for multi-GPU tasks to operate correctly, you must explicitly designate the specific GPU device to be utilized within your application's code.

The exact implementation to allocate a device to use for the application will depend on the programming language or framework you are using. For example, if you're using CUDA programming, to select a specific GPU, you can explicitly specify the device to use in your application code by using the function cudaSetDevice().

The need to explicitly specify the device is due to a known issue affecting Windows containers. You can track the progress on resolving this issue in the microsoft/Windows-Containers issue #333. The following table represents a visual representation and practical example of this GPU allocation behavior.

Consider a scenario whereby there is a single Windows node of EC2 instance type `g4dn.12xlarge`, which comes with four GPUs. Consider a scenario where three pods are launched on this instance. The table shows that regardless of the number of GPUs requested by each container, all three pods have access to all four GPUs on the instance, and by default will utilize the GPU with device index 0.

Pod	Requested GPUs	Actual GPU Access	Default GPU Usage	Available GPU Indices	Total Instance GPUs
Pod 1	1 GPU	All 4 GPUs	GPU with index 0	0, 1, 2, 3	4
Pod 2	2 GPUs	All 4 GPUs	GPU with index 0	0, 1, 2, 3	4
Pod 3	1 GPU	All 4 GPUs	GPU with index 0	0, 1, 2, 3	4

Kubernetes device plugin support

NVIDIA's official implementation of the Kubernetes device plugin does not support Windows. You can track the progress on adding official Windows support in the NVIDIA/k8s-device-plugin issue #419.

GPU compute instance limitations

Depending on your AWS account configuration, you may have service limits on the number and types of Amazon EC2 GPU compute instances that you can launch. If you require additional capacity, you can Request a quota increase.

Must build a Windows GPU Optimized AMI

There is no EKS Windows GPU Optimized AMI or EC2 Image Builder managed component provided by Amazon EKS. You will need to follow the steps in this guide to build a custom EKS Windows Optimized AMI with the required GPU drivers pre-installed, or install the necessary GPU drivers on your EKS worker nodes after launching your instances.

Inferentia and Trainium not supported

AWS Inferentia and AWS Trainium based workloads are not supported on Windows.

Create a managed node group with Capacity Blocks for ML

Capacity Blocks for machine learning (ML) allow you to reserve GPU instances on a future date to support your short duration ML workloads. For more information, see Capacity Blocks for ML in the *Amazon EC2 User Guide for Linux Instances*.

Considerations

Important

- Capacity Blocks are only available for certain Amazon EC2 instance types and AWS Regions. For compatibility information, see Work with Capacity Blocks Prerequisites in the *Amazon EC2 User Guide for Linux Instances*.
- For more information, see Use Capacity Blocks for machine learning workloads in the *Amazon EC2 Auto Scaling User Guide*.
- Managed node groups with Capacity Blocks can only be created with custom launch templates.
- When upgrading managed node groups with Capacity Blocks, make sure that the desired size of the node group is set to `0`.

Create a managed node group with Amazon EC2 Capacity Blocks

You can use Capacity Blocks with Amazon EKS managed node groups for provisioning and scaling GPU-accelerated worker nodes. The AWS CloudFormation template examples that follow don't cover every aspect needed in a production clusters. Typically, you'd also want a bootstrapping script to join the node to the cluster and specify an Amazon EKS accelerated AMI. For more information, see the section called "Create".

1. Create a launch template that's appropriate for your workloads and works with Amazon EKS managed node groups. For more information, see the section called "Launch templates".

   In addition to the requirements in the above procedures, make sure that the `LaunchTemplateData` includes the following:

   - `InstanceMarketOptions` with `MarketType` set to `"capacity-block"`
   - `CapacityReservationSpecification: CapacityReservationTarget` with `CapacityReservationId` set to the Capacity Block (for example: `cr-`*`02168da1478b509e0`*)
   - `InstanceType` set to an instance type that supports Capacity Blocks (for example: *`p5.48xlarge`*)

     The following is an excerpt of a CloudFormation template that creates a launch template targeting a Capacity Block. To create a custom AMI managed node group, you can also add `ImageId` and `UserData` parameters.

     ```
     NodeLaunchTemplate:
       Type: "AWS::EC2::LaunchTemplate"
       Properties:
         LaunchTemplateData:
           InstanceMarketOptions:
             MarketType: "capacity-block"
           CapacityReservationSpecification:
             CapacityReservationTarget:
               CapacityReservationId: "cr-02168da1478b509e0"
           InstanceType: p5.48xlarge
     ```

2. Use the launch template to create a managed node group.

   The following is an example create node group command for Capacity Blocks. Replace *`example-values`* with ones applicable to your cluster.

When creating the Capacity Block managed node group, do the following:

- Set the `capacity-type` to `"CAPACITY_BLOCK"`. If the capacity type isn't set to `"CAPACITY_BLOCK"` or any of the other above required launch template values are missing, then the create request will be rejected.
- When specifying `subnets` in the create request, make sure to only specify the subnet in the same Availability Zone as the capacity reservation.
- If you specify a non-zero `desiredSize` in the create request, Amazon EKS will honor that when creating the Auto Scaling group (ASG). However, if the create request is made before the capacity reservation is active, then the ASG won't be able to launch Amazon EC2 instances until it becomes active. As a result, ASG scaling activities will have launch errors. Whenever the reservation becomes active, then the launch of instances will succeed and the ASG will be scaled up to the `desiredSize` mentioned at create time.

```
aws eks create-nodegroup \
    --cluster-name my-cluster \
    --nodegroup-name my-mng \
    --node-role node-role-arn \
    --region region-code \
    --subnets subnet-id \
    --scaling-config minSize=node-group-min-size,maxSize=node-group-max-
size,desiredSize=node-group-desired-size \
    --ami-type "AL2023_x86_64_NVIDIA" \
    --capacity-type "CAPACITY_BLOCK" \
    --launch-template id="lt-id",version=1
```

3. Make sure that the nodes join after scale up. Amazon EKS clusters using managed node groups with Capacity Blocks don't perform any validations that instances launched actually join and register with the cluster.
4. If you set `desiredSize` to `0` at create time, then you have different options to scale up the node group when the capacity reservation becomes active:
   - Create a scheduled scaling policy for the ASG that aligns to the Capacity Block reservation start time. For more information, see [Scheduled scaling for Amazon EC2 Auto Scaling] in the *Amazon EC2 Auto Scaling User Guide*.
   - Use the Amazon EKS console or `eks update-nodegroup-config` to update the scaling config and set the desired size of the node group.
   - Use the Kubernetes Cluster Autoscaler. For more information, see [Cluster Autoscaler on AWS].

5. The node group is now ready for workloads and Pods to be scheduled.
6. In order for your Pods to be gracefully drained before reservation ends, Amazon EKS uses a scheduled scaling policy to scale down the node group size to `0` . This scheduled scaling will be set with name titled `Amazon EKS Node Group Capacity Scaledown Before Reservation End` . We recommend not editing or deleting this action.

   Amazon EC2 starts shutting down the instances 30 minutes before reservation end time. As a result, Amazon EKS will setup a scheduled scale down on the node group 40 minutes prior to their reservation end in order to safely and gracefully evict Pods.

Create self-managed nodes with Capacity Blocks for ML

Capacity Blocks for machine learning (ML) allow you to reserve GPU instances on a future date to support your short duration ML workloads. For more information, see Capacity Blocks for ML in the *Amazon EC2 User Guide for Linux Instances*.

Considerations

Important

- Capacity Blocks are only available for certain Amazon EC2 instance types and AWS Regions. For compatibility information, see Work with Capacity Blocks Prerequisites in the *Amazon EC2 User Guide for Linux Instances*.
- Capacity Blocks currently cannot be used with Karpenter.
- If you create a self-managed node group prior to the capacity reservation becoming active, then set the desired capacity to `0`.
- To allow sufficient time to gracefully drain the node(s), we suggest that you schedule scaling to scale to zero more than 30 minutes before the Capacity Block reservation end time.
- In order for your Pods to be gracefully drained, we recommend that you set up AWS Node Termination Handler as explained in the example steps.

Use Capacity Blocks with self-managed nodes

You can use Capacity Blocks with Amazon EKS for provisioning and scaling your self-managed nodes. The following steps give a general example overview. The AWS CloudFormation template examples don't cover every aspect needed in a production workload. Typically you'd also want a bootstrapping script to join the node to the cluster, specify an Amazon EKS accelerated AMI, and an appropriate instance profile for joining the cluster. For more information, see the section called "Amazon Linux".

1. Create a launch template that's applicable to your workload. For more information, see Use Capacity Blocks for machine learning workloads in the *Amazon EC2 Auto Scaling User Guide*.

   Make sure the `LaunchTemplateData` includes the following:

   - `InstanceMarketOptions` with `MarketType` set to `"capacity-block"`
   - `CapacityReservationSpecification: CapacityReservationTarget` with `CapacityReservationId` set to the Capacity Block (for example: `cr-`*`02168da1478b509e0`*)
   - `IamInstanceProfile` with the `Arn` set to the applicable *`iam-instance-profile-arn`*
   - `ImageId` set to the applicable *`image-id`*
   - `InstanceType` set to an instance type that supports Capacity Blocks (for example: *`p5.48xlarge`*)
   - `SecurityGroupIds` set to the applicable IDs (for example: *`sg-05b1d815d1EXAMPLE`*)
   - `UserData` set to the applicable *`user-data`* for your self-managed node group

     The following is an excerpt of a CloudFormation template that creates a launch template targeting a Capacity Block.

```
NodeLaunchTemplate:
  Type: "aws::EC2::LaunchTemplate"
  Properties:
    LaunchTemplateData:
      InstanceMarketOptions:
        MarketType: "capacity-block"
      CapacityReservationSpecification:
        CapacityReservationTarget:
          CapacityReservationId: "cr-02168da1478b509e0"
      IamInstanceProfile:
        Arn: iam-instance-profile-arn
```

```
        ImageId: image-id
        InstanceType: p5.48xlarge
        KeyName: key-name
        SecurityGroupIds:
        - sg-05b1d815d1EXAMPLE
        UserData: user-data
```

You must pass the subnet in the Availability Zone in which the reservation is made because Capacity Blocks are zonal.

2. Use the launch template to create a self-managed node group. If you're doing this prior to the capacity reservation becoming active, then set the desired capacity to `0`. When creating the node group, make sure that you are only specifying the respective subnet for the Availability Zone in which the capacity is reserved.

   The following is a sample CloudFormation template that you can reference when creating one that is applicable to your workload. This example gets the `LaunchTemplateId` and `Version` of the `AWS::Amazon EC2::LaunchTemplate` resource shown in the previous step. It also gets the values for `DesiredCapacity`, `MaxSize`, `MinSize`, and `VPCZoneIdentifier` that are declared elsewhere in the same template.

```
NodeGroup:
  Type: "AWS::AutoScaling::AutoScalingGroup"
  Properties:
    DesiredCapacity: !Ref NodeAutoScalingGroupDesiredCapacity
    LaunchTemplate:
      LaunchTemplateId: !Ref NodeLaunchTemplate
      Version: !GetAtt NodeLaunchTemplate.LatestVersionNumber
    MaxSize: !Ref NodeAutoScalingGroupMaxSize
    MinSize: !Ref NodeAutoScalingGroupMinSize
    VPCZoneIdentifier: !Ref Subnets
    Tags:
      - Key: Name
        PropagateAtLaunch: true
        Value: !Sub ${ClusterName}-${NodeGroupName}-Node
      - Key: !Sub kubernetes.io/cluster/${ClusterName}
        PropagateAtLaunch: true
        Value: owned
```

3. Once the node group is created successfully, make sure to record the `NodeInstanceRole` for the node group that was created. You need this in order to make sure that when node group is scaled, the new nodes join the cluster and Kubernetes is able to recognize the nodes. For more

information, see the AWS Management Console instructions in Create self-managed Amazon Linux nodes.

4. We recommend that you create a scheduled scaling policy for the Auto Scaling group that aligns to the Capacity Block reservation times. For more information, see Scheduled scaling for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.

   You can use all of the instances you reserved until 30 minutes before the end time of the Capacity Block. Instances that are still running at that time will start terminating. To allow sufficient time to gracefully drain the node(s), we suggest that you schedule scaling to scale to zero more than 30 minutes before the Capacity Block reservation end time.

   If you want to instead scale up manually whenever the capacity reservation becomes `Active`, then you need to update the Auto Scaling group's desired capacity at the start time of the Capacity Block reservation. Then you would need to also scale down manually more than 30 minutes before the Capacity Block reservation end time.

5. The node group is now ready for workloads and Pods to be scheduled.
6. In order for your Pods to be gracefully drained, we recommend that you set up AWS Node Termination Handler. This handler will be able to watch for "ASG Scale-in" lifecycle events from Amazon EC2 Auto Scaling using EventBridge and allow the Kubernetes control plane to take required action before the instance becomes unavailable. Otherwise, your Pods and Kubernetes objects will get stuck in a pending state. For more information, see AWS Node Termination Handler on GitHub.

   If you don't setup a Node Termination Handler, we recommend that you start draining your Pods manually before hitting the 30 minute window so that they have enough time to be gracefully drained.

Prevent Pods from being scheduled on specific nodes

Nodes with specialized processors, such as GPUs, can be more expensive to run than nodes running on more standard machines. For that reason, you may want to protect those nodes from having workloads that don't require special hardware from being deployed to those nodes. One way to do that is with taints.

Amazon EKS supports configuring Kubernetes taints through managed node groups. Taints and tolerations work together to ensure that Pods aren't scheduled onto inappropriate nodes. One or more taints can be applied to a node. This marks that the node shouldn't accept any Pods that don't tolerate the taints. Tolerations are applied to Pods and allow, but don't require, the Pods to schedule onto nodes with matching taints. For more information, see Taints and Tolerations in the Kubernetes documentation.

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

- For information on creating a node group with a taint using the AWS Management Console, see the section called "Create".
- The following is an example of creating a node group with a taint using the AWS CLI:

```
aws eks create-nodegroup \
 --cli-input-json '
{
  "clusterName": "my-cluster",
  "nodegroupName": "node-taints-example",
  "subnets": [
     "subnet-1234567890abcdef0",
     "subnet-abcdef01234567890",
     "subnet-021345abcdef67890"
   ],
  "nodeRole": "arn:aws:iam::111122223333:role/AmazonEKSNodeRole",
  "taints": [
     {
         "key": "dedicated",
         "value": "gpuGroup",
         "effect": "NO_SCHEDULE"
     }
   ]
}'
```

For more information and examples of usage, see taint in the Kubernetes reference documentation.

Note

- Taints can be updated after you create the node group using the `UpdateNodegroupConfig` API.
- The taint key must begin with a letter or number. It can contain letters, numbers, hyphens (`-`), periods (`.`), and underscores (`_`). It can be up to 63 characters long.

- Optionally, the taint key can begin with a DNS subdomain prefix and a single `/`. If it begins with a DNS subdomain prefix, it can be 253 characters long.
- The value is optional and must begin with a letter or number. It can contain letters, numbers, hyphens (`-`), periods (`.`), and underscores (`_`). It can be up to 63 characters long.
- When using Kubernetes directly or the AWS Management Console, the taint effect must be `NoSchedule`, `PreferNoSchedule`, or `NoExecute`. However, when using the AWS CLI or API, the taint effect must be `NO_SCHEDULE`, `PREFER_NO_SCHEDULE`, or `NO_EXECUTE`.
- A maximum of 50 taints are allowed per node group.
- If taints that were created using a managed node group are removed manually from a node, then Amazon EKS doesn't add the taints back to the node. This is true even if the taints are specified in the managed node group configuration.

You can use the aws eks update-nodegroup-config AWS CLI command to add, remove, or replace taints for managed node groups.

Run machine learning training on Amazon EKS with Elastic Fabric Adapter

This topic describes how to integrate Elastic Fabric Adapter (EFA) with Pods deployed in your Amazon EKS cluster. Elastic Fabric Adapter (EFA) is a network interface for Amazon EC2 instances that enables you to run applications requiring high levels of inter-node communications at scale on AWS. Its custom-built operating system bypass hardware interface enhances the performance of inter-instance communications, which is critical to scaling these applications. With EFA, High Performance Computing (HPC) applications using the Message Passing Interface (MPI) and Machine Learning (ML) applications using NVIDIA Collective Communications Library (NCCL) can scale to thousands of CPUs or GPUs. As a result, you get the application performance of on-premises HPC clusters with the on-demand elasticity and flexibility of the AWS cloud. Integrating EFA with applications running on Amazon EKS clusters can reduce the time to complete large scale distributed training workloads without having to add additional instances to your cluster. For more information about EFA, Elastic Fabric Adapter.

Instance types with EFA

The *AWS EFA Kubernetes Device Plugin* supports all Amazon EC2 instance types that have EFA. To see a list of all instance types that have EFA, see Supported instance types in the *Amazon EC2*

User Guide. However, to run ML applications quickly, we recommend that an instance has hardware acceleration chips such as nVidia GPUs, AWS Inferentia chips, or AWS Trainium chips, in addition to the EFA. To see a list of instance types that have hardware acceleration chips and EFA, see Accelerated computing in the *Amazon EC2 User Guide*.

As you compare instance types to choose between them, consider the number of EFA network cards available for that instance type as well as the number of accelerator cards, amount of CPU, and amount of memory. You can assign up to one EFA per network card. An EFA counts as a network interface.. To see how many EFA are available for each instance types that have EFA, see the Network cards list in the *Amazon EC2 User Guide*.

EFA and EFA-only interfaces

An *Elastic Fabric Adapter (EFA)* is a network interface that combines the capabilities of an Elastic Network Adapter (ENA) and an OS-bypass interface, powered by the AWS Scalable Reliable Datagram (SRD) protocol. The EFA functionalities allow applications to communicate directly with the hardware for low-latency transport. You can choose to access only the EFA capabilities using *EFA-only* interfaces, limiting communication to interfaces within the same Availability Zone.

To create nodes that can have EFA-only interfaces, you must use a custom EC2 Launch Template and set the `InterfaceType` to `efa-only`. In your custom Launch Template, you can't set the network card `0` to an EFA-only interface, as that is the primary network card and network interface of the EC2 instance. You must have VPC CNI version `1.18.5` or later for EFA-only interfaces. If you are using Amazon Linux 2, ami version has to be `v20240928` or later for EfA-only interfaces.

The following procedure guides you to create an EKS cluster with `eksctl` with nodes that have nVidia GPUs and EFA interfaces. You can't use `eksctl` to create nodes and node groups that use EFA-only interfaces.

Prerequisites

- An existing Amazon EKS cluster. If you don't have an existing cluster, create one using *Get started*.. Your cluster must be deployed in a VPC that has at least one private subnet with enough available IP addresses to deploy nodes in. The private subnet must have outbound internet access provided by an external device, such as a NAT gateway.

  If you plan to use `eksctl` to create your node group, `eksctl` can also create a cluster for you.
- Version `2.12.3` or later or version `1.27.160` or later of the AWS Command Line Interface (AWS CLI) installed and configured on your device or AWS CloudShell. To check your current version,

use `aws --version | cut -d / -f2 | cut -d ' ' -f1`. Package managers such `yum`, `apt-get`, or Homebrew for macOS are often several versions behind the latest version of the AWS CLI. To install the latest version, see Installing and Quick configuration with aws configure in the *AWS Command Line Interface User Guide*. The AWS CLI version that is installed in AWS CloudShell might also be several versions behind the latest version. To update it, see Installing AWS CLI to your home directory in the *AWS CloudShell User Guide*.

- The `kubectl` command line tool is installed on your device or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. For example, if your cluster version is `1.29`, you can use `kubectl` version `1.28`, `1.29`, or `1.30` with it. To install or upgrade `kubectl`, see the section called "Set up kubectl and eksctl".
- You must have the Amazon VPC CNI plugin for Kubernetes version `1.7.10` or later installed before launching worker nodes that support multiple Elastic Fabric Adapters, such as the `p4d` or `p5`. For more information about updating your Amazon VPC CNI plugin for Kubernetes version, see the section called "Amazon VPC CNI".

Important

An important consideration required for adopting EFA with Kubernetes is configuring and managing Huge Pages as a resource in the cluster. For more information, see Manage Huge Pages in the Kubernetes documentation. Amazon EC2 instances with the EFA driver installed pre-allocate 5128 2MiB Huge Pages, which you can request as resources to consume in your job specifications.

Create node group

The following procedure helps you create a node group with a `p4d.24xlarge` backed node group with EFA interfaces and GPUDirect RDMA, and run an example NVIDIA Collective Communications Library (NCCL) test for multi-node NCCL Performance using EFAs. The example can be used a template for distributed deep learning training on Amazon EKS using EFAs.

1. Determine which Amazon EC2 instance types that support EFA are available in the AWS Region that you want to deploy nodes in. Replace *`region-code`* with the AWS Region that you want to deploy your node group in.

```
aws ec2 describe-instance-types --region region-code \
    --filters Name=network-info.efa-supported,Values=true \
```

```
    --query "InstanceTypes[*].[InstanceType]" --output text
```

When you deploy nodes, the instance type that you want to deploy must be available in the AWS Region that your cluster is in.

2. Determine which Availability Zones that the instance type that you want to deploy is available in. In this tutorial, the `p5.48xlarge` instance type is used and must be returned in the output for the AWS Region that you specified in the previous step. When you deploy nodes in a production cluster, replace *`p5.48xlarge`* with any instance type returned in the previous step.

```
aws ec2 describe-instance-type-offerings --region region-code \
    --location-type availability-zone --filters Name=instance-
type,Values=p4d.24xlarge,p5.48xlarge \
    --query 'InstanceTypeOfferings[*].Location' --output text
```

An example output is as follows.

```
us-west-2a    us-west-2c    us-west-2b
```

Note the Availability Zones returned for use in later steps. When you deploy nodes to a cluster, your VPC must have subnets with available IP addresses in one of the Availability Zones returned in the output.

3. Create a node group using `eksctl`. You need version `0.204.0` or later of the `eksctl` command line tool installed on your device or AWS CloudShell. To install or update `eksctl`, see [Installation] in the `eksctl` documentation.

   a. Copy the following contents to a file named *`efa-cluster.yaml`*. Replace the *`example values`* with your own. You can replace *`p5.48xlarge`* with a different instance, but if you do, make sure that the values for `availabilityZones` are Availability Zones that were returned for the instance type in step 1.

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

metadata:
  name: my-efa-cluster
  region: region-code
  version: "1.XX"

iam:
```

```
  withOIDC: true

availabilityZones: ["us-west-2a", "us-west-2c"]

managedNodeGroups:
  - name: my-efa-ng
    instanceType: p5.48xlarge
    minSize: 1
    desiredCapacity: 2
    maxSize: 3
    availabilityZones: ["us-west-2a"]
    volumeSize: 300
    privateNetworking: true
    efaEnabled: true
```

b. Create a managed node group in an existing cluster.

```
eksctl create nodegroup -f efa-cluster.yaml
```

If you don't have an existing cluster, you can run the following command to create a cluster and the node group.

```
eksctl create cluster -f efa-cluster.yaml
```

> **Note**
>
> Because the instance type used in this example has GPUs, `eksctl` automatically installs the NVIDIA Kubernetes device plugin on each instance for you.

4. Deploy the EFA Kubernetes device plugin.

The EFA Kubernetes device plugin detects and advertises EFA interfaces as allocatable resources to Kubernetes. An application can consume the extended resource type `vpc.amazonaws.com/efa` in a Pod request spec just like CPU and memory. For more information, see [Consuming extended resources](#) in the Kubernetes documentation. Once requested, the plugin automatically assigns and mounts an EFA interface to the Pod. Using the device plugin simplifies EFA setup and does not require a Pod to run in privileged mode.

```
helm repo add eks https://aws.github.io/eks-charts
```

```
helm install aws-efa-k8s-device-plugin --namespace kube-system eks/aws-efa-k8s-
device-plugin
```

(Optional) Test the performance of the EFA

We recommend that you test the EFA setup. You can use the NCCL Tests in the `aws-samples/awsome-distributed-training` repository on GitHub. NCCL Tests evaluate the performance of the network using the Nvidia Collective Communication Library. The following steps submit NCCL tests on Amazon EKS.

1. Deploy the Kubeflow MPI Operator:

   For the NCCL tests you can apply the Kubeflow MPI Operator. The MPI Operator makes it easy to run Allreduce-style distributed training on Kubernetes. For more information, see MPI Operator on GitHub.
2. Run the multi-node NCCL Performance Test to verify GPUDirectRDMA/EFA:

   To verify NCCL performance with GPUDirectRDMA over EFA, run the standard NCCL Performance test. For more information, see the official NCCL-Tests repo on GitHub.

   Complete the following steps to run a two node NCCL Performance Test. In the example NCCL test job, each worker requests eight GPUs, 5210Mi of `hugepages-2Mi`, four EFAs, and 8000Mi of memory, which effectively means each worker consumes all the resources of a `p5.48xlarge` instance.

   a. Create the MPIJob manifest:

      Copy the following to a file named `nccl-tests.yaml`:

```
apiVersion: kubeflow.org/v2beta1
kind: MPIJob
metadata:
  name: nccl-tests
spec:
  runPolicy:
    cleanPodPolicy: Running
    backoffLimit: 20
  slotsPerWorker: 8
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
```

```
      template:
         spec:
          restartPolicy: OnFailure
          containers:
          - image: public.ecr.aws/hpc-cloud/nccl-tests:latest
            imagePullPolicy: IfNotPresent
            name: test-nccl-launcher
            env:
             - name: PATH
               value: $PATH:/opt/amazon/efa/bin:/usr/bin
             - name: LD_LIBRARY_PATH
               value: /opt/amazon/openmpi/lib:/opt/nccl/build/lib:/opt/amazon/efa/
lib:/opt/aws-ofi-nccl/install/lib:/usr/local/nvidia/lib:$LD_LIBRARY_PATH
             - name: NCCL_DEBUG
               value: INFO
             - name: NCCL_BUFFSIZE
               value: '8388608'
             - name: NCCL_P2P_NET_CHUNKSIZE
               value: '524288'
             - name: NCCL_TUNER_PLUGIN
               value: /opt/aws-ofi-nccl/install/lib/libnccl-ofi-tuner.so
            command:
            - /opt/amazon/openmpi/bin/mpirun
            - --allow-run-as-root
            - --tag-output
            - -np
            - "16"
            - -N
            - "8"
            - --bind-to
            - none
            - -x
            - PATH
            - -x
            - LD_LIBRARY_PATH
            - -x
            - NCCL_DEBUG=INFO
            - -x
            - NCCL_BUFFSIZE
            - -x
            - NCCL_P2P_NET_CHUNKSIZE
            - -x
            - NCCL_TUNER_PLUGIN
            - --mca
```

```
            - pml
            - ^cm,ucx
            - --mca
            - btl
            - tcp,self
            - --mca
            - btl_tcp_if_exclude
            - lo,docker0,veth_def_agent
            - /opt/nccl-tests/build/all_reduce_perf
            - -b
            - "8"
            - -e
            - "16G"
            - -f
            - "2"
            - -g
            - "1"
            - -c
            - "1"
            - -n
            - "100"
    Worker:
      replicas: 2
      template:
        spec:
          nodeSelector:
            node.kubernetes.io/instance-type: "p5.48xlarge"
          containers:
          - image: public.ecr.aws/hpc-cloud/nccl-tests:latest
            imagePullPolicy: IfNotPresent
            name: nccl-tests-worker
            volumeMounts:
            - name: shmem
              mountPath: /dev/shm
            resources:
              limits:
                nvidia.com/gpu: 8
                hugepages-2Mi: 5120Mi
                vpc.amazonaws.com/efa: 32
                memory: 32000Mi
              requests:
                nvidia.com/gpu: 8
                hugepages-2Mi: 5120Mi
                vpc.amazonaws.com/efa: 32
```

```
                memory: 32000Mi
          volumes:
          - name: shmem
            hostPath:
              path: /dev/shm
```

b. Apply the NCCL-tests MPIJob:

   Submit the `MPIJob` by applying the manifest. This will create two `p5.48xlarge` Amazon EC2 instances.

```
kubectl apply -f nccl-tests.yaml
```

   An example output is as follows.

```
mpijob.kubeflow.org/nccl-tests created
```

c. Verify that the job started pods:

   View your running Pods.

```
kubectl get pods
```

   An example output is as follows.

```
NAME                             READY   STATUS     RESTARTS   AGE
nccl-tests-launcher-nbql9    0/1     Init:0/1   0          2m49s
nccl-tests-worker-0          1/1     Running    0          2m49s
nccl-tests-worker-1          1/1     Running    0          2m49s
```

   The MPI Operator creates a launcher Pod and 2 worker Pods (one on each node).

d. Verify that the job is running successfully with the logs:

   View the log for the `nccl-tests-launcher` Pod. Replace *`nbql9`* with the value from your output.

```
kubectl logs -f nccl-tests-launcher-nbql9
```

If the test completed successfully, you can deploy your applications that use the Nvidia Collective Communication Library.

Use AWS Inferentia instances with Amazon EKS for Machine Learning

This topic describes how to create an Amazon EKS cluster with nodes running Amazon EC2 Inf1 instances and (optionally) deploy a sample application. Amazon EC2 Inf1 instances are powered by AWS Inferentia chips, which are custom built by AWS to provide high performance and lowest cost inference in the cloud. Machine learning models are deployed to containers using AWS Neuron, a specialized software development kit (SDK) consisting of a compiler, runtime, and profiling tools that optimize the machine learning inference performance of Inferentia chips. AWS Neuron supports popular machine learning frameworks such as TensorFlow, PyTorch, and MXNet.

> **Note**
>
> Neuron device logical IDs must be contiguous. If a Pod requesting multiple Neuron devices is scheduled on an `inf1.6xlarge` or `inf1.24xlarge` instance type (which have more than one Neuron device), that Pod will fail to start if the Kubernetes scheduler selects non-contiguous device IDs. For more information, see Device logical IDs must be contiguous on GitHub.

Prerequisites

- Have `eksctl` installed on your computer. If you don't have it installed, see Installation in the `eksctl` documentation.
- Have `kubectl` installed on your computer. For more information, see the section called "Set up kubectl and eksctl".
- (Optional) Have `python3` installed on your computer. If you don't have it installed, then see Python downloads for installation instructions.

Create a cluster

1. Create a cluster with Inf1 Amazon EC2 instance nodes. You can replace *`inf1.2xlarge`* with any Inf1 instance type. The `eksctl` utility detects that you are launching a node group with an `Inf1` instance type and will start your nodes using one of the Amazon EKS optimized accelerated Amazon Linux AMIs.

> **Note**
>
> You can't use IAM roles for service accounts with TensorFlow Serving.

```
eksctl create cluster \
    --name inferentia \
    --region region-code \
    --nodegroup-name ng-inf1 \
    --node-type inf1.2xlarge \
    --nodes 2 \
    --nodes-min 1 \
    --nodes-max 4 \
    --ssh-access \
    --ssh-public-key your-key \
    --with-oidc
```

> **Note**
>
> Note the value of the following line of the output. It's used in a later (optional) step.

```
[9]  adding identity "arn:aws:iam::111122223333:role/eksctl-inferentia-nodegroup-ng-in-NodeInstanceRole-FI7HIYS3BS09" to auth ConfigMap
```

When launching a node group with `Inf1` instances, `eksctl` automatically installs the AWS Neuron Kubernetes device plugin. This plugin advertises Neuron devices as a system resource to the Kubernetes scheduler, which can be requested by a container. In addition to the default Amazon EKS node IAM policies, the Amazon S3 read only access policy is added so that the sample application, covered in a later step, can load a trained model from Amazon S3.

2. Make sure that all Pods have started correctly.

```
kubectl get pods -n kube-system
```

Abbreviated output:

```
NAME                                       READY    STATUS     RESTARTS    AGE
```

```
[...]
neuron-device-plugin-daemonset-6djhp   1/1     Running    0          5m
neuron-device-plugin-daemonset-hwjsj   1/1     Running    0          5m
```

(Optional) Deploy a TensorFlow Serving application image

A trained model must be compiled to an Inferentia target before it can be deployed on Inferentia instances. To continue, you will need a Neuron optimized TensorFlow model saved in Amazon S3. If you don't already have a SavedModel, please follow the tutorial for creating a Neuron compatible ResNet50 model and upload the resulting SavedModel to S3. ResNet-50 is a popular machine learning model used for image recognition tasks. For more information about compiling Neuron models, see The AWS Inferentia Chip With DLAMI in the AWS Deep Learning AMIs Developer Guide.

The sample deployment manifest manages a pre-built inference serving container for TensorFlow provided by AWS Deep Learning Containers. Inside the container is the AWS Neuron Runtime and the TensorFlow Serving application. A complete list of pre-built Deep Learning Containers optimized for Neuron is maintained on GitHub under Available Images. At start-up, the DLC will fetch your model from Amazon S3, launch Neuron TensorFlow Serving with the saved model, and wait for prediction requests.

The number of Neuron devices allocated to your serving application can be adjusted by changing the `aws.amazon.com/neuron` resource in the deployment yaml. Please note that communication between TensorFlow Serving and the Neuron runtime happens over GRPC, which requires passing the `IPC_LOCK` capability to the container.

1. Add the `AmazonS3ReadOnlyAccess` IAM policy to the node instance role that was created in step 1 of Create a cluster. This is necessary so that the sample application can load a trained model from Amazon S3.

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess \
    --role-name eksctl-inferentia-nodegroup-ng-in-NodeInstanceRole-FI7HIYS3BS09
```

2. Create a file named `rn50_deployment.yaml` with the following contents. Update the region-code and model path to match your desired settings. The model name is for identification purposes when a client makes a request to the TensorFlow server. This example uses a model name to match a sample ResNet50 client script that will be used in a later step for sending prediction requests.

```
aws ecr list-images --repository-name neuron-rtd --registry-id 790709498068 --region
 us-west-2
```

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: eks-neuron-test
  labels:
    app: eks-neuron-test
    role: master
spec:
  replicas: 2
  selector:
    matchLabels:
      app: eks-neuron-test
      role: master
  template:
    metadata:
      labels:
        app: eks-neuron-test
        role: master
    spec:
      containers:
        - name: eks-neuron-test
          image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-
neuron:1.15.4-neuron-py37-ubuntu18.04
          command:
            - /usr/local/bin/entrypoint.sh
          args:
            - --port=8500
            - --rest_api_port=9000
            - --model_name=resnet50_neuron
            - --model_base_path=s3://${your-bucket-of-models}/resnet50_neuron/
          ports:
            - containerPort: 8500
            - containerPort: 9000
          imagePullPolicy: IfNotPresent
          env:
            - name: AWS_REGION
              value: "us-east-1"
            - name: S3_USE_HTTPS
              value: "1"
```

```
            - name: S3_VERIFY_SSL
              value: "0"
            - name: S3_ENDPOINT
              value: s3.us-east-1.amazonaws.com
            - name: AWS_LOG_LEVEL
              value: "3"
          resources:
            limits:
              cpu: 4
              memory: 4Gi
              aws.amazon.com/neuron: 1
            requests:
              cpu: "1"
              memory: 1Gi
          securityContext:
            capabilities:
              add:
                - IPC_LOCK
```

3. Deploy the model.

```
kubectl apply -f rn50_deployment.yaml
```

4. Create a file named `rn50_service.yaml` with the following contents. The HTTP and gRPC ports are opened for accepting prediction requests.

```
kind: Service
apiVersion: v1
metadata:
  name: eks-neuron-test
  labels:
    app: eks-neuron-test
spec:
  type: ClusterIP
  ports:
    - name: http-tf-serving
      port: 8500
      targetPort: 8500
    - name: grpc-tf-serving
      port: 9000
      targetPort: 9000
  selector:
    app: eks-neuron-test
```

```
    role: master
```

5. Create a Kubernetes service for your TensorFlow model Serving application.

```
kubectl apply -f rn50_service.yaml
```

(Optional) Make predictions against your TensorFlow Serving service

1. To test locally, forward the gRPC port to the `eks-neuron-test` service.

```
kubectl port-forward service/eks-neuron-test 8500:8500 &
```

2. Create a Python script called `tensorflow-model-server-infer.py` with the following content. This script runs inference via gRPC, which is service framework.

```
import numpy as np
   import grpc
   import tensorflow as tf
   from tensorflow.keras.preprocessing import image
   from tensorflow.keras.applications.resnet50 import preprocess_input
   from tensorflow_serving.apis import predict_pb2
   from tensorflow_serving.apis import prediction_service_pb2_grpc
   from tensorflow.keras.applications.resnet50 import decode_predictions

   if __name__ == '__main__':
       channel = grpc.insecure_channel('localhost:8500')
       stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)
       img_file = tf.keras.utils.get_file(
           "./kitten_small.jpg",
           "https://raw.githubusercontent.com/awslabs/mxnet-model-server/master/docs/
images/kitten_small.jpg")
       img = image.load_img(img_file, target_size=(224, 224))
       img_array = preprocess_input(image.img_to_array(img)[None, ...])
       request = predict_pb2.PredictRequest()
       request.model_spec.name = 'resnet50_inf1'
       request.inputs['input'].CopyFrom(
           tf.make_tensor_proto(img_array, shape=img_array.shape))
       result = stub.Predict(request)
       prediction = tf.make_ndarray(result.outputs['output'])
       print(decode_predictions(prediction))
```

3. Run the script to submit predictions to your service.

```
python3 tensorflow-model-server-infer.py
```

An example output is as follows.

```
[[(u'n02123045', u'tabby', 0.68817204), (u'n02127052', u'lynx', 0.12701613),
 (u'n02123159', u'tiger_cat', 0.08736559), (u'n02124075', u'Egyptian_cat',
 0.063844085), (u'n02128757', u'snow_leopard', 0.009240591)]]
```

Try tutorials for deploying Machine Learning workloads on EKS

If you are interested in setting up Machine Learning platforms and frameworks in EKS, explore the tutorials described in this page. These tutorials cover everything from patterns for making the best use of GPU processors to choosing modeling tools to building frameworks for specialized industries.

Build generative AI platforms on EKS

- Deploy Generative AI Models on Amazon EKS
- Building multi-tenant JupyterHub Platforms on Amazon EKS
- Run Spark-RAPIDS ML workloads with GPUs on Amazon EMR on EKS

Run specialized generative AI frameworks on EKS

- Accelerate drug discovery with NVIDIA BioNeMo Framework on Amazon EKS
- Host the Whisper Model with Streaming Mode on Amazon EKS and Ray Serve
- Accelerate your generative AI distributed training workloads with the NVIDIA NeMo Framework on Amazon EKS
- Virtualizing satellite communication operations with AWS
- Running TorchServe on Amazon Elastic Kubernetes Service

Maximize NVIDIA GPU performance for ML on EKS

- Implement GPU sharing to efficiently use NVIDIA GPUs for your EKS clusters:

GPU sharing on Amazon EKS with NVIDIA time-slicing and accelerated EC2 instances

- Use Multi-Instance GPUs (MIGs) and NIM microservices to run more pods per GPU on your EKS clusters:

  Maximizing GPU utilization with NVIDIA's Multi-Instance GPU (MIG) on Amazon EKS: Running more pods per GPU for enhanced performance
- Leverage NVIDIA NIM microservices to optimize inference workloads using optimized microservices to deploy AI models at scale:

  Part 1: Deploying generative AI applications with NVIDIA NIMs on Amazon EKS

  Part 2: Deploying Generative AI Applications with NVIDIA NIM Microservices on Amazon Elastic Kubernetes Service (Amazon EKS)
- Scaling a Large Language Model with NVIDIA NIM on Amazon EKS with Karpenter
- Build and deploy a scalable machine learning system on Kubernetes with Kubeflow on AWS

Run video encoding workloads on EKS

- Delivering video content with fractional GPUs in containers on Amazon EKS

Accelerate image loading for inference workloads

- How H2O.ai optimized and secured their AI/ML infrastructure with Karpenter and Bottlerocket

Testimonials for ML on EKS

- Quora achieved 3x lower latency and 25% lower Costs by modernizing model serving with Nvidia Triton on Amazon EKS

Monitoring ML workloads

- Monitoring GPU workloads on Amazon EKS using AWS managed open-source services
- Enable pod-based GPU metrics in Amazon CloudWatch

Announcements for ML on EKS

- Bottlerocket support for NVIDIA GPUs
- New – EC2 Instances (G5) with NVIDIA A10G Tensor Core GPUs
- Utilizing NVIDIA Multi-Instance GPU (MIG) in Amazon EC2 P4d Instances on Amazon Elastic Kubernetes Service
- New – GPU-Equipped EC2 P4 Instances for Machine Learning & HPC
- Amazon EC2 P5e instances are generally available
- Deploying managed P4d Instances in Amazon Elastic Kubernetes Service with NVIDIA GPUDirectRDMA
- Establishing an AI/ML center of excellence

Extend Amazon EKS capabilities with open source projects

These open-source projects extend the functionality of Kubernetes clusters running on or outside of AWS, including clusters managed by Amazon EKS.

Support for software deployed to EKS

When reviewing the Amazon EKS docs, you'll encounter references to various open-source tools and software throughout our procedures and examples. These tools include the Kubernetes Metrics Server and Cert Manager.

Please note that any third-party or open-source software you choose to deploy falls outside the scope of your AWS Support Agreements. A benefit of using Kubernetes is the active open source community. We recommend working directly with the relevant open-source communities and project maintainers to establish appropriate support channels for such components. For more information, see the graduated and incubating projects associated with the Cloud Native Computing Foundation (CNCF).

The Kubernetes ecosystem includes numerous projects and components that come with different levels of community support, response times, and intended use cases. When implementing these technologies alongside EKS, ensure you understand the support matrix for each component.

AWS maintains the open-source components we integrate into the EKS control plane. This includes our comprehensive security pipeline covering build verification, vulnerability scanning, validation testing, and patch management for all container images and binaries we distribute. For example, AWS is responsible for the Kubernetes API Server. The Kubernetes API server is covered by Amazon EKS Service Level Agreement. You can use your Amazon Web Services Support Plan to resolve issues with the Kubernetes API server, or get general guidance.

You need to carefully review the support offered for various Amazon EKS Add-ons. AWS add-ons are the only type of Amazon EKS add-on that are fully supported by AWS. AWS Marketplace add-ons are primarily supported by AWS Partners. Community add-ons receive basic lifecycle support from AWS. For more information, see add-on Support.

Every EKS add-ons, irrespective of the type, receives basic lifecycle support from EKS including Marketplace add-ons. Basic lifecycle support includes installing and uninstalling the add-on. For more information on the types of Amazon EKS Add-ons available and the associated levels of support, see Scope of Support for Amazon EKS add-ons. To view add-ons fully supported by AWS, see Amazon Web Services add-ons.

- For more information about our security practices and support boundaries, see Security in Amazon EKS.
- For more information about community and AWS marketplace add-ons available through Amazon EKS Add-ons, see EKS Add-ons Support.

Management tools

Related management tools for Amazon EKS and Kubernetes clusters.

eksctl

`eksctl` is a simple CLI tool for creating clusters on Amazon EKS.

- Project URL
- Project documentation
- AWS open source blog: eksctl: Amazon EKS cluster with one command

AWS controllers for Kubernetes

With AWS Controllers for Kubernetes, you can create and manage AWS resources directly from your Kubernetes cluster.

- Project URL
- AWS open source blog: AWS service operator for Kubernetes now available

Flux CD

Flux is a tool that you can use to manage your cluster configuration using Git. It uses an operator in the cluster to trigger deployments inside of Kubernetes. For more information about operators, see OperatorHub.io on GitHub.

- Project URL
- Project documentation

CDK for Kubernetes

With the CDK for Kubernetes (cdk8s), you can define Kubernetes apps and components using familiar programming languages. cdk8s apps synthesize into standard Kubernetes manifests, which can be applied to any Kubernetes cluster.

- Project URL
- Project documentation
- AWS containers blog: Introducing cdk8s+: Intent-driven APIs for Kubernetes objects

Networking

Related networking projects for Amazon EKS and Kubernetes clusters.

Amazon VPC CNI plugin for Kubernetes

Amazon EKS supports native VPC networking through the Amazon VPC CNI plugin for Kubernetes. The plugin assigns an IP address from your VPC to each Pod.

- Project URL
- Project documentation

AWS Load Balancer Controller for Kubernetes

The AWS Load Balancer Controller helps manage AWS Elastic Load Balancers for a Kubernetes cluster. It satisfies Kubernetes Ingress resources by provisioning AWS Application Load Balancers. It satisfies Kubernetes service resources by provisioning AWS Network Load Balancers.

- Project URL
- Project documentation

ExternalDNS

ExternalDNS synchronizes exposed Kubernetes services and ingresses with DNS providers including Amazon Route 53 and AWS Service Discovery.

- Project URL
- Project documentation

Machine learning

Related machine learning projects for Amazon EKS and Kubernetes clusters.

Kubeflow

A machine learning toolkit for Kubernetes.

- Project URL
- Project documentation
- AWS open source blog: Kubeflow on Amazon EKS

Auto Scaling

Related auto scaling projects for Amazon EKS and Kubernetes clusters.

Cluster autoscaler

Cluster Autoscaler is a tool that automatically adjusts the size of the Kubernetes cluster based on CPU and memory pressure.

- Project URL
- Project documentation
- Amazon EKS workshop: Cluster Autoscaler

Karpenter

Karpenter is a Kubernetes Node Autoscaler built for flexibility, performance, and simplicity.

- Project URL
- Project documentation
- Amazon EKS workshop: Karpenter

Escalator

Escalator is a batch or job optimized horizontal autoscaler for Kubernetes.

- Project URL
- Project documentation

Monitoring

Related monitoring projects for Amazon EKS and Kubernetes clusters.

Prometheus

Prometheus is an open-source systems monitoring and alerting toolkit.

- Project URL
- Project documentation
- Amazon EKS workshop: https://eksworkshop.com/intermediate/240_monitoring/

Continuous integration / continuous deployment

Related CI/CD projects for Amazon EKS and Kubernetes clusters.

Jenkins X

CI/CD solution for modern cloud applications on Amazon EKS and Kubernetes clusters.

- Project URL
- Project documentation

Learn about Amazon EKS new features and roadmap

You can learn about new Amazon EKS features by scrolling to the What's New feed on the What's New with AWS page. You can also review the roadmap on GitHub, which lets you know about upcoming features and priorities so that you can plan how you want to use Amazon EKS in the future. You can provide direct feedback to us about the roadmap priorities.

Document history

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.

Change	Description	Date
Update strategies for managed node groups	You can now use update strategies to configure the version update process for managed node groups. This introduces the *minimal* update strategy to terminate nodes before making new ones, which is useful in capacity constrained environments. The *default* update strategy continues the existing behavior.	January 27, 2025
Kubernetes version `1.32`	Added Kubernetes version `1.32` support for new clusters and version upgrades.	January 23, 2025
Amazon EKS AWS Region expansion	Amazon EKS is now available in the Asia Pacific (Thailand) Region (`ap-southeast-7`) and Mexico (Central) (`mx-central-1`) AWS Regions. EKS Auto Mode and VPC Endpoints for the EKS API aren't available in either Region.	January 14, 2025
AWS managed policy updates	Added multiple permissions to `AmazonEBSCSIDriverPolicy` to allow the	January 13, 2025

	Amazon EBS CSI Driver restore all snapshots, enable Fast Snapshot Restore (FSR) on EBS volumes, and modify tags on volumes.	
AWS managed policy updates	Added permissions to `AmazonEKSLoadBalancingPolicy` .	December 26, 2024
Updated cluster insights	Amazon EKS upgrade insights will now warn about more cluster health and version compatibility issues. It can detect issues between different Kubernetes and Amazon EKS components such as `kubelet`, `kube-proxy` , and Amazon EKS add-ons.	December 20, 2024
Node monitoring agent and auto repair	You can use the new `eks-node-monitoring-agent` as an Amazon EKS add-on to detect and show health issues. You can also enable node auto repair to automatically replace nodes when issues are detected.	December 16, 2024

Amazon EKS Hybrid Nodes	You can now run node on-premises connected to Amazon EKS clusters. With Amazon EKS Hybrid Nodes, you can use your on-premis es and edge infrastructure as nodes in Amazon EKS clusters. AWS manages the AWS-hosted Kubernetes control plane of the Amazon EKS cluster, and you manage the hybrid nodes that run in your on-premises or edge environments.	December 1, 2024
Amazon EKS Auto Mode	Amazon EKS Auto Mode fully automates Kubernete s cluster infrastructure management for compute, storage, and networkin g on AWS. It simplifies Kubernetes management by automatically provision ing infrastructure, selecting optimal compute instances, dynamically scaling resources, continuously optimizing costs, patching operating systems, and integrating with AWS security services.	December 1, 2024

Amazon EKS platform version update	This is a new platform version with security fixes and enhancements. This includes new patch versions of Kubernetes `1.31.2`, `1.30.6`, `1.29.10`, and `1.28.15`.	November 22, 2024
AWS managed policy updates	Updated `AWSServiceRoleForAmazonEKSNodegroup` for compatibility with China regions.	November 22, 2024
Kubernetes version 1.30 is now available for local clusters on AWS Outposts	You can now create an Amazon EKS local cluster on an AWS Outposts using Kubernetes version 1.30.	November 21, 2024
AWS managed policy updates	EKS updated AWS managed policy `AmazonEKSLocalOutpostClusterPolicy`. Added `ec2:DescribeAvailabilityZones` permission so the AWS Cloud Controller Manager on the cluster control plane can identify the Availability Zone that each node is in.	November 21, 2024

Bottlerocket AMIs that use FIPS 140-3	Bottlerocket AMIs are available that are preconfig ured to use FIPS 140-3 validated cryptographic modules. This includes the Amazon Linux 2023 Kernel Crypto API Cryptographic Module and the AWS-LC Cryptographic Module.	November 20, 2024
AWS managed policy updates	Updated `AWSServiceRoleForAmazonEKSNodegroup` policy to allow `ec2:RebootInstances` for instances created by Amazon EKS managed node groups. Restricted the `ec2:CreateTags` permissions for Amazon EC2 resources.	November 20, 2024
Observability dashboard	The observability dashboard helps you to quickly detect, troubleshoot, and remediate issues. There are also new CloudWatch vended metrics available in the `AWS/EKS` namespace.	November 18, 2024
AWS managed policy updates	EKS updated AWS managed policy `AmazonEKSServiceRolePolicy`. Added permissions for EKS access policies, load balancer management, and automated cluster resource cleanup.	November 16, 2024

New role creation in console for add-ons that support EKS Pod Identities	There are new steps when using the console to create or update add-ons that support EKS Pod Identities where you can automatically generate IAM roles with the appropriate name, role policy, and trust policy for the add-on.	November 15, 2024
Managed node groups in AWS Local Zones	Managed node groups can now be created in AWS Local Zones.	November 15, 2024
New metrics are available	There are new metrics available under the API group `metrics.eks.amazonaws.com` .	November 11, 2024
AWS managed policy updates	EKS updated AWS managed policy `AmazonEKSComputePolicy` . Updated resource permissions for the `iam:AddRoleToInstanceProfile` action.	November 7, 2024
AWS managed policy updates	EKS added a new AWS managed policy: `AmazonEKSComputePolicy`	November 1, 2024
AWS managed policy updates	Added permissions to `AmazonEKSClusterPolicy` . Added `ec2:DescribeInstanceTopology` permission to allow Amazon EKS to attach topology information to the node as labels.	November 1, 2024

AWS managed policy updates	EKS added a new AWS managed policy: `AmazonEKSBlockStoragePolicy`	October 30, 2024
AWS managed policy updates	EKS added a new AWS managed policy: `AmazonEKSLoadBalancingPolicy`	October 30, 2024
AWS managed policy updates	Added `cloudwatch:PutMetricData` permissions to `AmazonEKSServiceRolePolicy` to allow Amazon EKS to publish metrics to Amazon CloudWatch.	October 29, 2024
AWS managed policy updates	EKS added a new AWS managed policy: `AmazonEKSNetworkingPolicy`	October 28, 2024
Dual-stack endpoints for new IPv6 clusters	Connect to new `IPv6` clusters with a `eks-cluster. *region*.api.aws` endpoint that is dual-stack. This endpoint is returned when you describe these clusters. `kubectl` and other Kubernetes API clients in `IPv4`, `IPv6`, or dual-stack environments can resolve and connect to these endpoints for public or private clusters.	October 21, 2024

AWS managed policy updates	Added `autoscaling:ResumeProcesses` , `autoscaling:SuspendProcesses` , and associated permissions to `AWSServiceRoleForAmazonEKSNodegroup` in China regions to integrate with Amazon Application Recovery Controller for EKS. No changes to other regions.	October 21, 2024
AL2023 accelerated AMIs	You can now use accelerated `NVIDIA` and AWS Neuron instances for AMIs based on AL2023.	October 11, 2024
New source format	We have switched over to a new source format with some layout changes. There are temporary minor formatting issues that we are addressing.	October 10, 2024
AWS managed policy updates	Added permissions to `AmazonEKSServicePolicy` and `AmazonEKSServiceRolePolicy` . Added `ec2:GetSecurityGroupsForVpc` and associated tag permissions to allow EKS to read security group information and update related tags.	October 10, 2024
AWS managed policy updates - New policy	EKS added a new AWS managed policy.	October 3, 2024

Kubernetes version `1.31`	Added Kubernetes version `1.31` support for new clusters and version upgrades.	September 24, 2024
AWS managed policy updates - Update to an existing policy	Amazon EKS updated an existing AWS managed policy.	August 21, 2024
Kubernetes version 1.29 is now available for local clusters on AWS Outposts	You can now create an Amazon EKS local cluster on an AWS Outposts using Kubernetes version 1.29.	August 20, 2024
EKS Pod Identity in AWS GovCloud (US)	Amazon EKS Pod Identities associate an IAM role with a Kubernetes service account. With this feature, you no longer need to provide extended permissions to the node IAM role. This way, Pods on that node can call AWS APIs. Unlike IAM roles for service accounts, EKS Pod Identities are completely inside EKS; you don't need an OIDC identity provider.	August 14, 2024
Scenario-driven content updates	We renamed and updated topics to be more scenario-driven throughout the entire guide.	August 9, 2024
Dual-stack VPC interface endpoints for Amazon EKS	You can now create dual-stack VPC interface endpoints for Amazon EKS with both `IPv4` and `IPv6` IP addresses and DNS names.	August 7, 2024

New dual-stack endpoints for the Amazon EKS APIs with IPv6 addresses	The EKS API for creating and managing clusters, and the OIDC issuer URLs for clusters have new dual-stack endpoints. The new DNS name for the Amazon EKS API is `eks.`*`region`*`.api.aws` which resolves to `IPv4` addresses and `IPv6` addresses. New clusters have a new dual-stack OIDC issuer URL (`oidc-eks.` *`region`*`.api.aws`).	August 1, 2024
Capacity Blocks for managed node groups	You can now use Capacity Blocks for managed node groups.	July 1, 2024
Auto Scaling Group metrics collection enabled by default	Amazon EKS managed node groups now have Amazon EC2 Auto Scaling group metrics enabled by default with no additional charge. Previously, you had to do several steps to enable this feature.	June 28, 2024
AWS managed policy updates - Update to an existing policy	Amazon EKS updated an existing AWS managed policy.	June 27, 2024
Kubernetes version `1.26`	Kubernetes version `1.26` is now in extended support.	June 12, 2024
Improvements to AMI information references	We made improvements to the AMI information references, in particular for Bottlerocket.	June 12, 2024

Kubernetes version 1.30	Added Kubernetes version 1.30 support for new clusters and version upgrades.	May 23, 2024
CoreDNS Autoscaling	CoreDNS autoscaler will dynamically adapt the number of replicas of the CoreDNS deployment in an EKS cluster based on the number of nodes and CPU cores. This feature works for CoreDNS v1.9 and the latest platform version of EKS release version 1.25 and later.	May 14, 2024
Amazon EKS platform version update	This is a new platform version with security fixes and enhancements. This includes new patch versions of Kubernetes 1.29.4, 1.28.9, and 1.27.13.	May 14, 2024
CloudWatch Container Insights support for Windows	The Amazon CloudWatch Observability Operator add-on now also allows Container Insights on Windows worker nodes in the cluster.	April 10, 2024
Kubernetes concepts	Added new Kubernetes concepts topic.	April 5, 2024

Restructure Access and IAM Content	Move existing pages related to access and IAM topics, such as auth config map, access entries, Pod ID, and IRSA into new section. Revise overview content.	April 2, 2024
Bottlerocket OS support for Amazon S3 CSI driver	The Mountpoint for Amazon S3 CSI driver is now compatibl e with Bottlerocket.	March 13, 2024
AWS managed policy updates - Update to an existing policy	Amazon EKS updated an existing AWS managed policy.	March 4, 2024
Amazon Linux 2023	Amazon Linux 2023 (AL2023) is a new Linux-based operating system designed to provide a secure, stable, and high-performance environme nt for your cloud applications.	February 29, 2024
EKS Pod Identity and IRSA support sidecars in Kubernete s `1.29`	In Kubernetes `1.29`, sidecar containers are available in Amazon EKS clusters. Sidecar containers are supported with IAM roles for service accounts or EKS Pod Identity. For more information about sidecars, see Sidecar Containers in the Kubernetes documentation.	February 26, 2024
Kubernetes version `1.29`	Added Kubernetes version `1.29` support for new clusters and version upgrades.	January 23, 2024

Full release: Amazon EKS Extended Support for Kubernetes versions	Extended Kubernetes version support allows you to stay at a specific Kubernetes version for longer than 14 months.	January 16, 2024
Amazon EKS cluster health detection in the AWS Cloud	Amazon EKS detects issues with your Amazon EKS clusters and the infrastructure of the cluster prerequisites in *cluster health*. You can view the issues with your EKS clusters in the AWS Management Console and in the `health` of the cluster in the EKS API. These issues are in addition to the issues that are detected by and displayed by the console. Previously, cluster health was only available for local clusters on AWS Outposts.	December 28, 2023
Cluster insights	You can now get recommendations on your cluster based on recurring checks.	December 20, 2023
Amazon EKS AWS Region expansion	Amazon EKS is now available in the Canada West (Calgary) (`ca-west-1`) AWS Region.	December 20, 2023

You can now grant IAM roles and users access to your cluster using access entries	Before the introduction of access entries, you granted IAM roles and users access to your cluster by adding entries to the `aws-auth` `ConfigMap` . Now each cluster has an access mode, and you can switch to using access entries on your schedule. After you switch modes, you can add users by adding access entries in the AWS CLI, AWS CloudFormation, and the AWS SDKs.	December 18, 2023
Amazon EKS platform version update	This is a new platform version with security fixes and enhancements. This includes new patch versions of Kubernetes `1.28.4`, `1.27.8`, `1.26.11`, and `1.25.16`.	December 12, 2023
Mountpoint for Amazon S3 CSI driver	You can now install the Mountpoint for Amazon S3 CSI driver on Amazon EKS clusters.	November 27, 2023
Turn on Prometheus metrics when creating a cluster	In the AWS Management Console, you can now turn on Prometheus metrics when creating a cluster. You can also view Prometheus scraper details in the **Observability** tab.	November 26, 2023

Amazon EKS Pod Identities	Amazon EKS Pod Identities associate an IAM role with a Kubernetes service account. With this feature, you no longer need to provide extended permissions to the node IAM role. This way, Pods on that node can call AWS APIs. Unlike IAM roles for service accounts, EKS Pod Identities are completely inside EKS; you don't need an OIDC identity provider.	November 26, 2023
AWS managed policy updates - Update to an existing policy	Amazon EKS updated an existing AWS managed policy.	November 26, 2023
CSI snapshot controller	You can now install the CSI snapshot controller for use with compatible CSI drivers, such as the Amazon EBS CSI driver.	November 17, 2023
ADOT Operator topic rewrite	The Amazon EKS add-on support for ADOT Operator section was redundant with the AWS Distro for OpenTelemetry documentation. We migrated remaining essential information to that resource to reduce outdated and inconsistent information.	November 14, 2023

CoreDNS EKS add-on support for Prometheus metrics	The `v1.10.1-eksbuild.5` , `v1.9.3-eksbuild.9` , and `v1.8.7-eksbuild.8` versions of the EKS add-on for CoreDNS expose the port that CoreDNS published metrics to, in the `kube-dns` service. This makes it easier to include the CoreDNS metrics in your monitoring systems.	November 10, 2023
Amazon EKS CloudWatch Observability Operator add-on	Added Amazon EKS CloudWatch Observability Operator page.	November 6, 2023
Capacity Blocks for self-managed P5 instances in US East (Ohio)	In US East (Ohio), you can now use Capacity Blocks for self-managed P5 instances.	October 31, 2023
Clusters support modifying subnets and security groups	You can update the cluster to change which subnets and security groups the cluster uses. You can update from the AWS Management Console, the latest version of the AWS CLI, AWS CloudFormation, and `eksctl` version `v0.164.0-rc.0` or later. You might need to do this to provide subnets with more available IP addresses to successfully upgrade a cluster version.	October 24, 2023

Cluster role and managed node group role supports customer managed AWS Identity and Access Management policies	You can use a custom IAM policy on the cluster role, instead of the AmazonEKS ClusterPolicy AWS managed policy. Also, you can use a custom IAM policy on the node role in a managed node group, instead of the AmazonEKSWorkerNod ePolicy AWS managed policy. Do this to create a policy with the least privilege to meet strict compliance requireme nts.	October 23, 2023
Fix link to eksctl installation	Fix install link for eksctl after the page was moved.	October 6, 2023
Preview release: Amazon EKS Extended Support for Kubernetes versions	Extended Kubernetes version support allows you to stay at a specific Kubernetes version for longer than 14 months.	October 4, 2023
Remove references to AWS App Mesh integration	Amazon EKS integrations with AWS App Mesh remain for existing customers of App Mesh only.	September 29, 2023
Kubernetes version `1.28`	Added Kubernetes version `1.28` support for new clusters and version upgrades.	September 26, 2023

Existing clusters support Kubernetes network policy enforcement in the Amazon VPC CNI plugin for Kubernetes	You can use Kubernetes *network policy* in existing clusters with the Amazon VPC CNI plugin for Kubernetes, instead of requiring a third party solution. You can use Kubernetes *network policy* in existing clusters with the Amazon VPC CNI plugin for Kubernetes, instead of requiring a third party solution.	September 15, 2023
CoreDNS Amazon EKS add-on supports modifying PDB	You can modify the `PodDisruptionBudget` of the EKS add-on for CoreDNS in versions `v1.9.3-eksbuild.7` and later and `v1.10.1-eksbuild.4` and later.	September 15, 2023
Amazon EKS support for shared subnets	New Shared subnet requirements and considerations for making Amazon EKS clusters in shared subnets.	September 7, 2023
Updates to What is Amazon EKS?	Added new Common use cases and Architecture topics. Refreshed other topics.	September 6, 2023

Kubernetes network policy enforcement in the Amazon VPC CNI plugin for Kubernetes	You can use Kubernetes *network policy* with the Amazon VPC CNI plugin for Kubernetes, instead of requiring a third party solution. You can use Kubernetes *network policy* with the Amazon VPC CNI plugin for Kubernetes, instead of requiring a third party solution.	August 29, 2023
Amazon EKS AWS Region expansion	Amazon EKS is now available in the Israel (Tel Aviv) (`il-central-1`) AWS Region.	August 1, 2023
Configurable ephemeral storage for Fargate	You can increase the total amount of ephemeral storage for each Pod running on Amazon EKS Fargate.	July 31, 2023
Add-on support for Amazon EFS CSI driver	You can now use the AWS Management Console, AWS CLI, and API to manage the Amazon EFS CSI driver.	July 26, 2023
AWS managed policy updates - New policy	Amazon EKS added a new AWS managed policy.	July 26, 2023
Kubernetes version updates for 1.27, 1.26, 1.25, and 1.24 are now available for local clusters on AWS Outposts	Kubernetes version updates to 1.27.3, 1.26.6, 1.25.11, and 1.24.15 are now available for local clusters on AWS Outposts	July 20, 2023

IP prefixes support for Windows nodes	Assigning IP prefixes to your nodes can enable you to host a significantly higher number of Pods on your nodes than you can when assigning individual secondary IP addresses to your nodes.	July 6, 2023
Amazon FSx for OpenZFS CSI driver	You can now install the Amazon FSx for OpenZFS CSI driver on Amazon EKS clusters.	June 30, 2023
Pods on Linux nodes in `IPv4` clusters can now communicate with `IPv6` endpoints.	After assigning an IPv6 address to your node, your Pods' `IPv4` address is network address translated to the `IPv6` address of the node that it's running on.	June 19, 2023
Windows managed node groups in AWS GovCloud (US) Regions	In the AWS GovCloud (US) Regions, Amazon EKS managed node groups can now run Windows containers.	May 30, 2023
Kubernetes version `1.27`	Added Kubernetes version `1.27` support for new clusters and version upgrades.	May 24, 2023
Kubernetes version `1.26`	Added Kubernetes version `1.26` support for new clusters and version upgrades.	April 11, 2023
Domainless gMSA	You can now use domainless gMSA with Windows Pods.	March 27, 2023

Amazon EKS AWS Region expansion	Amazon EKS is now available in the Asia Pacific (Melbourne) (`ap-southeast-4`) AWS Region.	March 10, 2023
Amazon File Cache CSI driver	You can now install the Amazon File Cache CSI driver on Amazon EKS clusters.	March 3, 2023
Kubernetes version 1.25 is now available for local clusters on AWS Outposts	You can now create an Amazon EKS local cluster on an Outpost using Kubernetes versions `1.22` – `1.25`.	March 1, 2023
Kubernetes version `1.25`	Added Kubernetes version `1.25` support for new clusters and version upgrades.	February 22, 2023
AWS managed policy updates - Update to an existing policy	Amazon EKS updated an existing AWS managed policy.	February 7, 2023
Amazon EKS AWS Region expansion	Amazon EKS is now available in the Asia Pacific (Hyderabad) (`ap-south-2`), Europe (Zurich) (`eu-central-2`), and Europe (Spain) (`eu-south-2`) AWS Regions.	February 6, 2023
Kubernetes versions `1.21` – `1.24` are now available for local clusters on AWS Outposts.	You can now create an Amazon EKS local cluster on an Outpost using Kubernetes versions `1.21` – `1.24`. Previously, only version `1.21` was available.	January 17, 2023

Amazon EKS now supports AWS PrivateLink	You can use an AWS PrivateLink to create a private connection between your VPC and Amazon EKS.	December 16, 2022
Managed node group Windows support	You can now use Windows for Amazon EKS managed node groups.	December 15, 2022
Amazon EKS add-ons from independent software vendors are now available in the AWS Marketplace	You can now browse and subscribe to Amazon EKS add-ons from independent software vendors through the AWS Marketplace.	November 28, 2022
AWS managed policy updates - Update to an existing policy	Amazon EKS updated an existing AWS managed policy.	November 17, 2022
Kubernetes version `1.24`	Added Kubernetes version `1.24` support for new clusters and version upgrades.	November 15, 2022
Amazon EKS AWS Region expansion	Amazon EKS is now available in the Middle East (UAE) (`me-central-1`) AWS Region.	November 3, 2022
AWS managed policy updates - Update to an existing policy	Amazon EKS updated an existing AWS managed policy.	October 24, 2022
AWS managed policy updates - Update to an existing policy	Amazon EKS updated an existing AWS managed policy.	October 20, 2022
Local clusters on AWS Outposts are now available	You can now create an Amazon EKS local cluster on an Outpost.	September 19, 2022
Fargate vCPU based quotas	Fargate is transitioning from Pod based quotas to vCPU based quotas.	September 8, 2022

AWS managed policy updates - Update to an existing policy	Amazon EKS updated an existing AWS managed policy.	August 31, 2022
Cost monitoring	Amazon EKS now supports Kubecost, which enables you to monitor costs broken down by Kubernetes resources including Pods, nodes, namespaces, and labels.	August 24, 2022
AWS managed policy updates - New policy	Amazon EKS added a new AWS managed policy.	August 24, 2022
AWS managed policy updates - New policy	Amazon EKS added a new AWS managed policy.	August 23, 2022
Tag resources for billing	Added `aws:eks:cluster-name` generated cost allocation tag support for all clusters.	August 16, 2022
Fargate profile wildcards	Added support for Fargate profile wildcards in the selector criteria for namespaces, label keys, and label values.	August 16, 2022
Kubernetes version `1.23`	Added Kubernetes version `1.23` support for new clusters and version upgrades.	August 11, 2022
View Kubernetes resources in the AWS Management Console	You can now view informati on about the Kubernete s resources deployed to your cluster using the AWS Management Console.	May 3, 2022

Amazon EKS AWS Region expansion	Amazon EKS is now available in the Asia Pacific (Jakarta) (`ap-southeast-3`) AWS Region.	May 2, 2022
Observability page and ADOT add-on support	Added Observability page and AWS Distro for OpenTelemetry (ADOT).	April 21, 2022
Kubernetes version `1.22`	Added Kubernetes version `1.22` support for new clusters and version upgrades.	April 4, 2022
AWS managed policy updates - New policy	Amazon EKS added a new AWS managed policy.	April 4, 2022
Added Fargate Pod patching details	When upgrading Fargate Pods, Amazon EKS first tries to evict Pods based on your Pod disruption budgets. You can create event rules to react to failed evictions before the Pods are deleted.	April 1, 2022
Full release: Add-on support for Amazon EBS CSI driver	You can now use the AWS Management Console, AWS CLI, and API to manage the Amazon EBS CSI driver.	March 31, 2022
AWS Outposts content update	Instructions to deploy an Amazon EKS cluster on AWS Outposts.	March 22, 2022
AWS managed policy updates - Update to an existing policy	Amazon EKS updated an existing AWS managed policy.	March 21, 2022

Windows containerd support	You can now select the containerd runtime for Windows nodes.	March 14, 2022
Added Amazon EKS Connector considerations to security documentation	Describes the shared responsibility model as it relates to connected clusters.	February 25, 2022
Assign IPv6 addresses to your Pods and services	You can now create a 1.21 or later cluster that assigns IPv6 addresses to your Pods and services.	January 6, 2022
AWS managed policy updates - Update to an existing policy	Amazon EKS updated an existing AWS managed policy.	December 13, 2021
Preview release: Add-on support for Amazon EBS CSI driver	You can now preview using the AWS Management Console, AWS CLI, and API to manage the Amazon EBS CSI driver.	December 9, 2021
Karpenter autoscaler support	You can now use the Karpenter open-source project to autoscale your nodes.	November 29, 2021
Fluent Bit Kubernetes filter support in Fargate logging	You can now use the Fluent Bit Kubernetes filter with Fargate logging.	November 10, 2021
Windows support available in the control plane	Windows support is now available in your control plane. You no longer need to enable it in your data plane.	November 9, 2021

Bottlerocket added as an AMI type for managed node groups	Previously, Bottlerocket was only available as a self-mana ged node option. Now it can be configured as a managed node group, reducing the effort that's required to meet node compliance requireme nts.	October 28, 2021
DL1 driver support	Custom Amazon Linux AMIs now support deep learning workloads for Amazon Linux 2. This enablement allows a generic on-premises or cloud baseline configuration.	October 25, 2021
VT1 video support	Custom Amazon Linux AMIs now support VT1 for some distributions. This enablemen t advertises Xilinx U30 devices on your Amazon EKS cluster.	September 13, 2021
Amazon EKS Connector is now available	You can use Amazon EKS Connector to register and connect any conformant Kubernetes cluster to AWS and visualize it in the Amazon EKS console.	September 8, 2021
Amazon EKS Anywhere is now available	Amazon EKS Anywhere is a new deployment option for Amazon EKS that you can use to create and operate Kubernetes clusters on-premises.	September 8, 2021

Amazon FSx for NetApp ONTAP CSI driver	Added topic that summarize s the Amazon FSx for NetApp ONTAP CSI driver and gives links to other references.	September 2, 2021
Managed node groups now auto-calculates the Amazon EKS recommended maximum Pods for nodes	Managed node groups now auto-calculate the Amazon EKS maximum Pods for nodes that you deploy without a launch template, or with a launch template that you haven't specified an AMI ID in.	August 30, 2021
Remove Amazon EKS management of add-on settings without removing the Amazon EKS add-on software	You can now remove an Amazon EKS add-on without removing the add-on software from your cluster.	August 20, 2021
Create multi-homed Pods using Multus	You can now add multiple network interfaces to a Pod using Multus.	August 2, 2021
Add more IP addresses to your Linux Amazon EC2 nodes	You can now add significa ntly more IP addresses to your Linux Amazon EC2 nodes. This means that you can run a higher density of Pods on each node. You can now add significantly more IP addresses to your Linux Amazon EC2 nodes. This means that you can run a higher density of Pods on each node.	July 27, 2021

`containerd` runtime bootstrap	The Amazon EKS optimized accelerated Amazon Linux Amazon Machine Image (AMI) now contains a bootstrap flag that you can use to enable the `containerd` runtime in Amazon EKS optimized and Bottlerocket AMIs. This flag is available in all supported Kubernetes versions of the AMI.	July 19, 2021
Kubernetes version `1.21`	Added Kubernetes version `1.21` support.	July 19, 2021
Added managed policies topic	A list of all Amazon EKS IAM managed policies and changes that were made to them since June 17, 2021.	June 17, 2021
Use security groups for Pods with Fargate	You can now use security groups for Pods with Fargate, in addition to using them with Amazon EC2 nodes.	June 1, 2021
Added CoreDNS and `kube-proxy` Amazon EKS add-ons	Amazon EKS can now help you manage the CoreDNS and `kube-proxy` Amazon EKS add-ons for your cluster.	May 19, 2021
Kubernetes version `1.20`	Added Kubernetes version `1.20` support for new clusters and version upgrades.	May 18, 2021
AWS Load Balancer Controller `2.2.0` released	You can now use the AWS Load Balancer Controller to create Elastic Load Balancers using instance or IP targets.	May 14, 2021

Node taints for managed node groups	Amazon EKS now supports adding note taints to managed node groups.	May 11, 2021
Secrets encryption for existing clusters	Amazon EKS now supports adding secrets encryption to existing clusters.	February 26, 2021
Kubernetes version 1.19	Added Kubernetes version 1.19 support for new clusters and version upgrades.	February 16, 2021
Amazon EKS now supports OpenID Connect (OIDC) identity providers as a method to authenticate users to a version 1.16 or later cluster.	OIDC identity providers can be used with, or as an alternative to AWS Identity and Access Management (IAM).	February 12, 2021
View node and workload resources in the AWS Management Console	You can now view details about your managed, self-managed, and Fargate nodes and your deployed Kubernete s workloads in the AWS Management Console.	December 1, 2020
Deploy Spot Instance types in a managed node group	You can now deploy multiple Spot or On-Demand Instance types to a managed node group.	December 1, 2020
Amazon EKS can now manage specific add-ons for your cluster	You can manage add-ons yourself, or allow Amazon EKS to control the launch and version of an add-on through the Amazon EKS API.	December 1, 2020

Share an ALB across multiple Ingresses	You can now share an AWS Application Load Balancer (ALB) across multiple Kubernetes Ingresses. In the past, you had to deploy a separate ALB for each Ingress.	October 23, 2020
NLB IP target support	You can now deploy a Network Load Balancer with IP targets. This means that you can use an NLB to load balance network traffic to Fargate Pods and directly to Pods that are running on Amazon EC2 nodes.	October 23, 2020
Kubernetes version `1.18`	Added Kubernetes version `1.18` support for new clusters and version upgrades.	October 13, 2020
Specify a custom CIDR block for Kubernetes service IP address assignment.	You can now specify a custom CIDR block that Kubernetes assigns service IP addresses from.	September 29, 2020
Assign security groups to individual Pods	You can now associate different security groups to some of the individual Pods that are running on many Amazon EC2 instance types.	September 9, 2020
Deploy Bottlerocket on your nodes	You can now deploy nodes that are running Bottlerocket.	August 31, 2020
The ability to launch Arm nodes is generally available	You can now launch Arm nodes in managed and self-managed node groups.	August 17, 2020

Managed node group launch templates and custom AMI	You can now deploy a managed node group that uses an Amazon EC2 launch template. The launch template can specify a custom AMI, if you choose.	August 17, 2020
EFS support for AWS Fargate	You can now use Amazon EFS with AWS Fargate.	August 17, 2020
Amazon EKS platform version update	This is a new platform version with security fixes and enhancements. This includes UDP support for services of type `LoadBalancer` when using Network Load Balancers with Kubernetes version `1.15` or later. For more information, see the Allow UDP for AWS Network Load Balancer issue on GitHub.	August 12, 2020
Amazon EKS AWS Region expansion	Amazon EKS is now available in the Africa (Cape Town) (`af-south-1`) and Europe (Milan) (`eu-south-1`) AWS Regions.	August 6, 2020
Fargate usage metrics	AWS Fargate provides CloudWatch usage metrics that provide visibility into your account's usage of Fargate On-Demand resources.	August 3, 2020

Kubernetes version 1.17	Added Kubernetes version 1.17 support for new clusters and version upgrades.	July 10, 2020
Create and manage App Mesh resources from within Kubernetes with the App Mesh controller for Kubernetes	You can create and manage App Mesh resources from within Kubernetes. The controller also automatically injects the Envoy proxy and init containers into Pods that you deploy.	June 18, 2020
Amazon EKS now supports Amazon EC2 Inf1 nodes	You can add Amazon EC2 Inf1 nodes to your cluster.	June 4, 2020
Amazon EKS AWS Region expansion	Amazon EKS is now available in the AWS GovCloud (US-East) (`us-gov-east-1`) and AWS GovCloud (US-West) (`us-gov-west-1`) AWS Regions.	May 13, 2020
Kubernetes 1.12 is no longer supported on Amazon EKS	Kubernetes version 1.12 is no longer supported on Amazon EKS. Update any 1.12 clusters to version 1.13 or later to avoid service interruption.	May 12, 2020
Kubernetes version 1.16	Added Kubernetes version 1.16 support for new clusters and version upgrades.	April 30, 2020
Added the **AWSServiceRoleForAmazonEKS** service-linked role	Added the **AWSServiceRoleForAmazonEKS** service-linked role.	April 16, 2020

Kubernetes version `1.15`	Added Kubernetes version `1.15` support for new clusters and version upgrades.	March 10, 2020
Amazon EKS AWS Region expansion	Amazon EKS is now available in the Beijing (`cn-north-1`) and Ningxia (`cn-northwest-1`) AWS Regions.	February 26, 2020
FSx for Lustre CSI driver	Added topic for installing the FSx for Lustre CSI driver on Kubernetes `1.14` Amazon EKS clusters.	December 23, 2019
Restrict network access to the public access endpoint of a cluster	With this update, you can use Amazon EKS to restrict the CIDR ranges that can communicate to the public access endpoint of the Kubernetes API server.	December 20, 2019
Resolve the private access endpoint address for a cluster from outside of a VPC	With this update, you can use Amazon EKS to resolve the private access endpoint of the Kubernetes API server from outside of a VPC.	December 13, 2019
(Beta) Amazon EC2 A1 Amazon EC2 instance nodes	Launch Amazon EC2 A1 Amazon EC2 instance nodes that register with your Amazon EKS cluster.	December 4, 2019
Creating a cluster on AWS Outposts	Amazon EKS now supports creating clusters on AWS Outposts.	December 3, 2019

AWS Fargate on Amazon EKS	Amazon EKS Kubernetes clusters now support running Pods on Fargate.	December 3, 2019
Amazon EKS AWS Region expansion	Amazon EKS is now available in the Canada (Central) (`ca-central-1`) AWS Region.	November 21, 2019
Managed node groups	Amazon EKS managed node groups automate the provisioning and lifecycle management of nodes (Amazon EC2 instances) for Amazon EKS Kubernetes clusters.	November 18, 2019
Amazon EKS platform version update	New platform versions to address CVE-2019-11253.	November 6, 2019
Kubernetes `1.11` is no longer supported on Amazon EKS	Kubernetes version `1.11` is no longer supported on Amazon EKS. Please update any `1.11` clusters to version `1.12` or higher to avoid service interruption.	November 4, 2019
Amazon EKS AWS Region expansion	Amazon EKS is now available in the South America (São Paulo) (`sa-east-1`) AWS Region.	October 16, 2019
Windows support	Amazon EKS clusters running Kubernetes version `1.14` now support Windows workloads.	October 7, 2019

Autoscaling	Added a chapter to cover some of the different types of Kubernetes autoscaling that are supported on Amazon EKS clusters.	September 30, 2019
Kubernetes Dashboard update	Updated topic for installing the Kubernetes Dashboard on Amazon EKS clusters to use the beta `2.0` version.	September 28, 2019
Amazon EFS CSI driver	Added topic for installing the Amazon EFS CSI driver on Kubernetes `1.14` Amazon EKS clusters.	September 19, 2019
Amazon EC2 Systems Manager parameter for Amazon EKS optimized AMI ID	Added topic for retrieving the Amazon EKS optimized AMI ID using an Amazon EC2 Systems Manager parameter. The parameter eliminates the need for you to look up AMI IDs.	September 18, 2019
Amazon EKS resource tagging	You can manage the tagging of your Amazon EKS clusters.	September 16, 2019
Amazon EBS CSI driver	Added topic for installing the Amazon EBS CSI driver on Kubernetes `1.14` Amazon EKS clusters.	September 9, 2019
New Amazon EKS optimized AMI patched for `CVE-2019-9512` and `CVE-2019-9514`	Amazon EKS has updated the Amazon EKS optimized AMI to address CVE-2019-9512 and CVE-2019-9514.	September 6, 2019

Announcing deprecation of Kubernetes `1.11` in Amazon EKS	Amazon EKS discontinued support for Kubernetes version `1.11` on November 4, 2019.	September 4, 2019
Kubernetes version `1.14`	Added Kubernetes version `1.14` support for new clusters and version upgrades.	September 3, 2019
IAM roles for service accounts	With IAM roles for service accounts on Amazon EKS clusters, you can associate an IAM role with a Kubernete s service account. With this feature, you no longer need to provide extended permissio ns to the node IAM role. This way, Pods on that node can call AWS APIs.	September 3, 2019
Amazon EKS AWS Region expansion	Amazon EKS is now available in the Middle East (Bahrain) (`me-south-1`) AWS Region.	August 29, 2019
Amazon EKS platform version update	New platform versions to address CVE-2019-9512 and CVE-2019-9514.	August 28, 2019
Amazon EKS platform version update	New platform versions to address CVE-2019-11247 and CVE-2019-11249.	August 5, 2019
Amazon EKS Region expansion	Amazon EKS is now available in the Asia Pacific (Hong Kong) (`ap-east-1`) AWS Region.	July 31, 2019

Kubernetes `1.10` no longer supported on Amazon EKS	Kubernetes version `1.10` is no longer supported on Amazon EKS. Update any `1.10` clusters to version `1.11` or higher to avoid service interruption.	July 30, 2019
Added topic on ALB ingress controller	The AWS ALB Ingress Controller for Kubernetes is a controller that causes an ALB to be created when ingress resources are created.	July 11, 2019
New Amazon EKS optimized AMI	Removing unnecessary `kubectl` binary from AMIs.	July 3, 2019
Kubernetes version `1.13`	Added Kubernetes version `1.13` support for new clusters and version upgrades.	June 18, 2019
New Amazon EKS optimized AMI patched for `AWS-2019-005`	Amazon EKS has updated the Amazon EKS optimized AMI to address the vulnerabilities that are described in link:security/security-bulletins/AWS-2019-005/[AWS-2019-005,type="marketing"].	June 17, 2019
Announcing discontinuation of support of Kubernetes `1.10` in Amazon EKS	Amazon EKS stopped supporting Kubernetes version `1.10` on July 22, 2019.	May 21, 2019

Amazon EKS platform version update	New platform version for Kubernetes `1.11` and `1.10` clusters to support custom DNS names in the `kubelet` certificate and improve `etcd` performance.	May 21, 2019
Getting started with `eksctl`	This getting started guide describes how you can install all of the required resources to get started with Amazon EKS using `eksctl`. This is a simple command line utility for creating and managing Kubernetes clusters on Amazon EKS.	May 10, 2019
AWS CLI `get-token` command	The `aws eks get-token` command was added to the AWS CLI. You no longer need to install the AWS IAM Authenticator for Kubernetes to create client security tokens for cluster API server communication. Upgrade your AWS CLI installation to the latest version to use this new functionality. For more information, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.	May 10, 2019

Amazon EKS platform version update	New platform version for Kubernetes `1.12` clusters to support custom DNS names in the `kubelet` certificate and improve `etcd` performance. This fixes a bug that caused node `kubelet` daemons to request a new certificate every few seconds.	May 8, 2019
Prometheus tutorial	Added topic for deploying Prometheus to your Amazon EKS cluster.	April 5, 2019
Amazon EKS control plane logging	With this update, you can get audit and diagnostic logs directly from the Amazon EKS control pane. You can use these CloudWatch logs in your account as reference for securing and running clusters.	April 4, 2019
Kubernetes version `1.12`	Added Kubernetes version `1.12` support for new clusters and version upgrades.	March 28, 2019
Added App Mesh getting started guide	Added documentation for getting started with App Mesh and Kubernetes.	March 27, 2019
Amazon EKS API server endpoint private access	Added documentation for disabling public access for your Amazon EKS cluster's Kubernetes API server endpoint.	March 19, 2019

Added topic for installing the Kubernetes Metrics Server	The Kubernetes Metrics Server is an aggregator of resource usage data in your cluster.	March 18, 2019
Added list of related open source projects	These open source projects extend the functionality of Kubernetes clusters running on AWS, including clusters that are managed by Amazon EKS.	March 15, 2019
Added topic for installing Helm locally	The `helm` package manager for Kubernetes helps you install and manage applicati ons on your Kubernetes cluster. This topic shows how to install and run the `helm` and `tiller` binaries locally. That way, you can install and manage charts using the Helm CLI on your local system.	March 11, 2019
Amazon EKS platform version update	New platform version that updates Amazon EKS Kubernetes `1.11` clusters to patch level `1.11.8` to address CVE-2019-1002100.	March 8, 2019
Increased cluster limit	Amazon EKS has increased the number of clusters that you can create in an AWS Region from 3 to 50.	February 13, 2019

Amazon EKS AWS Region expansion	Amazon EKS is now available in the Europe (London) (`eu-west-2`), Europe (Paris) (`eu-west-3`), and Asia Pacific (Mumbai) (`ap-south-1`) AWS Regions.	February 13, 2019
New Amazon EKS optimized AMI patched for ALAS-2019-1156	Amazon EKS has updated the Amazon EKS optimized AMI to address the vulnerability that's described in ALAS-2019-1156.	February 11, 2019
New Amazon EKS optimized AMI patched for ALAS2-2019-1141	Amazon EKS has updated the Amazon EKS optimized AMI to address the CVEs that are referenced in ALAS2-2019-1141.	January 9, 2019
Amazon EKS AWS Region expansion	Amazon EKS is now available in the Asia Pacific (Seoul) (`ap-northeast-2`) AWS Region.	January 9, 2019
Amazon EKS region expansion	Amazon EKS is now available in the following additional AWS Regions: Europe (Frankfurt) (`eu-central-1`), Asia Pacific (Tokyo) (`ap-northeast-1`), Asia Pacific (Singapore) (`ap-southeast-1`), and Asia Pacific (Sydney) (`ap-southeast-2`).	December 19, 2018

Amazon EKS cluster updates	Added documentation for Amazon EKS cluster Kubernetes version updates and node replacement.	December 12, 2018
Amazon EKS AWS Region expansion	Amazon EKS is now available in the Europe (Stockholm) (`eu-north-1`) AWS Region.	December 11, 2018
Amazon EKS platform version update	New platform version updating Kubernetes to patch level `1.10.11` to address link:security/security-bulletins/AWS-2018-020/[CVE-2018-1002105,type="marketing"].	December 4, 2018
Added version `1.0.0` support for the ALB ingress controller	The ALB ingress controller releases version `1.0.0` with formal support from AWS.	November 20, 2018
Added support for CNI network configuration	The Amazon VPC CNI plugin for Kubernetes version `1.2.1` now supports custom network configuration for secondary Pod network interfaces.	October 16, 2018
Added support for `MutatingAdmissionWebhook` and `ValidatingAdmissionWebhook`	Amazon EKS platform version `1.10-eks.2` now supports `MutatingAdmissionWebhook` and `ValidatingAdmissionWebhook` admission controllers.	October 10, 2018

Added partner AMI information	Canonical has partnered with Amazon EKS to create node AMIs that you can use in your clusters.	October 3, 2018
Added instructions for AWS CLI `update-kubeconfig` command	Amazon EKS has added the `update-kubeconfig` to the AWS CLI to simplify the process of creating a `kubeconfig` file for accessing your cluster.	September 21, 2018
New Amazon EKS optimized AMIs	Amazon EKS has updated the Amazon EKS optimized AMIs (with and without GPU support) to provide various security fixes and AMI optimizations.	September 13, 2018
Amazon EKS AWS Region expansion	Amazon EKS is now available in the Europe (Ireland) (`eu-west-1`) Region.	September 5, 2018
Amazon EKS platform version update	New platform version with support for Kubernetes aggregation layer and the Horizontal Pod Autoscaler(HPA).	August 31, 2018
New Amazon EKS optimized AMIs and GPU support	Amazon EKS has updated the Amazon EKS optimized AMI to use a new AWS CloudFormation node template and bootstrap script. In addition, a new Amazon EKS optimized AMI with GPU support is available.	August 22, 2018

New Amazon EKS optimized AMI patched for `ALAS2-2018-1058`	Amazon EKS has updated the Amazon EKS optimized AMI to address the CVEs that are referenced in ALAS2-2018-1058.	August 14, 2018
Amazon EKS optimized AMI build scripts	Amazon EKS has open-sourced the build scripts that are used to build the Amazon EKS optimized AMI. These build scripts are now available on GitHub.	July 10, 2018
Amazon EKS initial release	Initial documentation for service launch	June 5, 2018

Contribute to the EKS User Guide

AWS has launched an improved contribution experience for the EKS User Guide.

You can now edit the EKS User Guide source directly on GitHub.

The docs now use AsciiDoc, a powerful authoring language similar to markdown. AsciiDoc combines simple syntax with enterprise documentation features like advanced formatting, cross-referencing, and security controls.

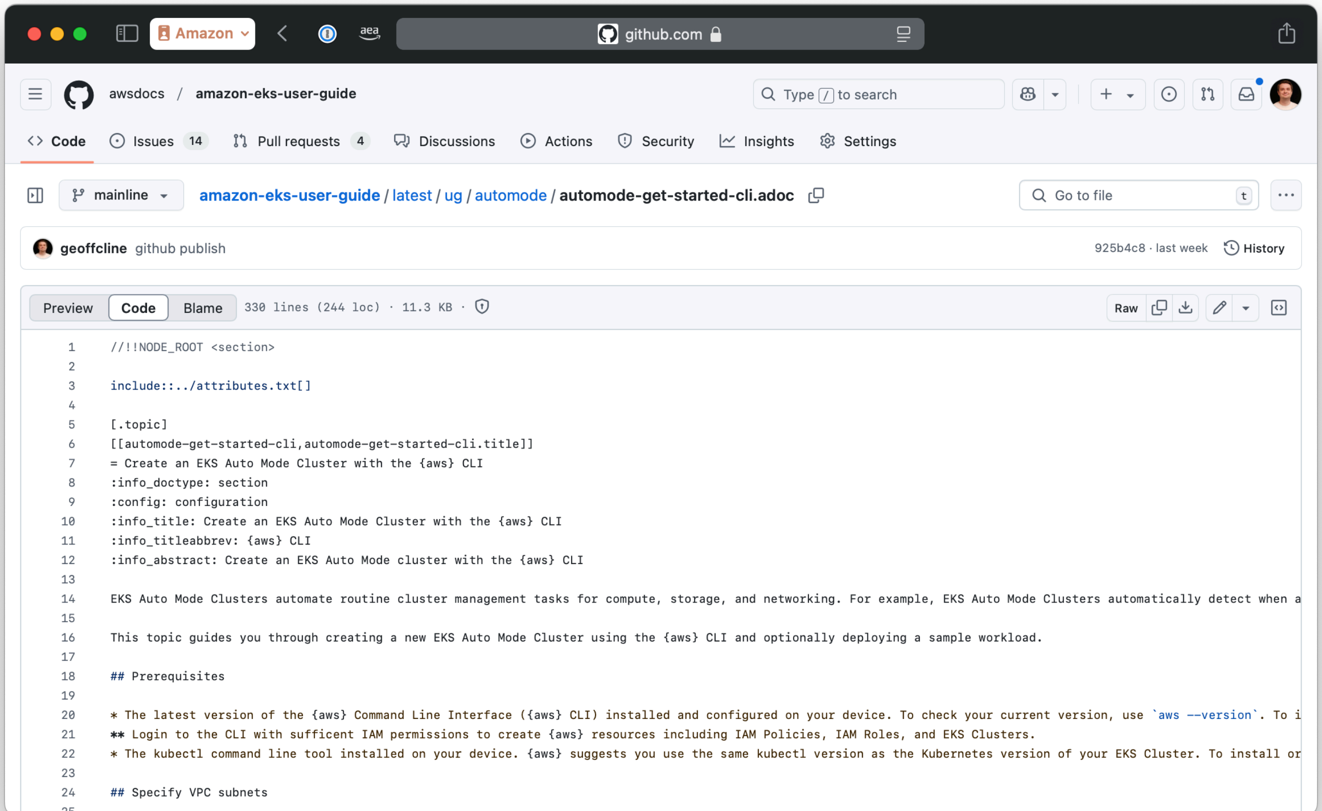
You can now edit the EKS Docs directly on GitHub. Our streamlined process includes:

- Faster pull request processing
- Reduced manual steps
- Automated content quality checks

We look forward to your contributions.

Edit a single page from a web browser

You can easily edit a single page in the EKS User Guide directly through your web browser.



If you want to edit multiple pages from your web browser, see the section called “Edit files with GitHub”.

Prerequisites

- Docs page to change opened in web browser.
- Signed into GitHub.

Procedure

1. Navigate to the page you want to edit in the Amazon EKS User Guide.
2. In the right pane, choose the **Edit this page on GitHub** link.
3. Once on GitHub, open the editor by either:
    - Pressing the e key on your keyboard.
    - Clicking the pencil icon and selecting **Edit in Place** from the dropdown menu.

- If you don't have the option to edit, you need to login to GitHub. Your GitHub account does not need any special permissions to suggest changes. However, internal Amazon contributors should link their GitHub profile.

4. Make your required changes to the content in the GitHub editor.
    - The editor provides syntax highlighting and preview capabilities.
    - You can use AsciiDoc markup to format your changes.
    - You can use `ctrl-f` to open a find/replace interface.
5. (Optional) Preview your changes.
    - Use the `preview` tab to preview your changes with rich formatting.
    - Use the `show diff` option to highlight changed sections. Removed sections have a red indicator in the left margin. New sections have a green indicator in the left margin.
6. When finished editing, click the **Commit changes...** button at the top of the editor
7. In the commit dialog:
    - Verify your email address is correct.
    - Add a brief but descriptive commit message explaining your changes.
    - Optionally add a longer description if needed.
    - Select to create a new branch and pull request.

You have created a pull request including the proposed changes.

Pull request overview

When you create a PR:

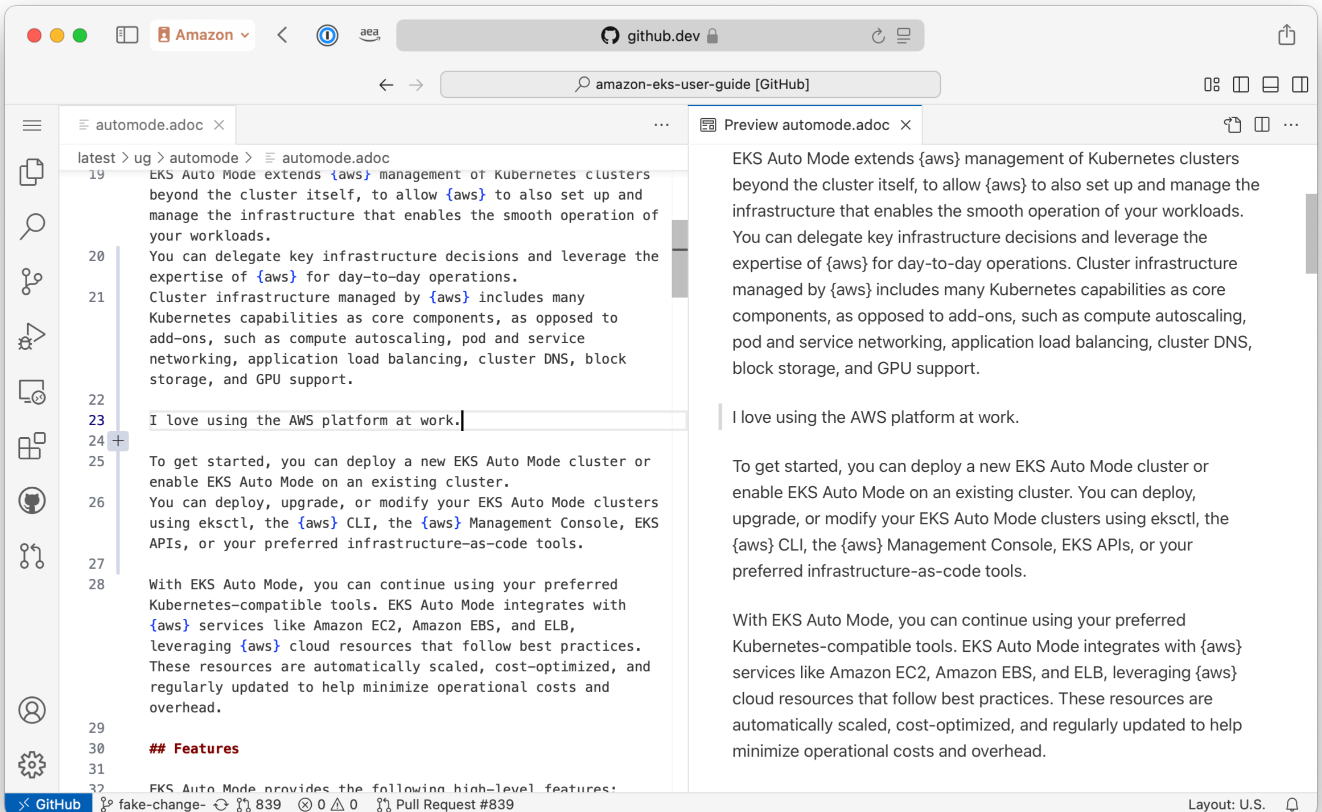
- Your changes are submitted for review by repository maintainers.
- Reviewers can comment on your changes and request modifications.
- Automated tests may run to validate your changes.
- Once approved, your changes can be merged into the main repository.

Pull requests help ensure quality and provide a way to discuss changes before they are integrated.

Learn how pull requests are reviewed and approved in the GitHub Docs.

Edit multiple files from a web browser with the GitHub Web Editor

If you want to propose change to multiple pages, or create a new docs page, use the GitHub.dev web editor. This web editor is based on the popular Visual Studio Code text editor.



Prerequisites

- Logged in to GitHub
- Familiarity with Visual Studio Code editor
- Familiarity with Git

Procedure

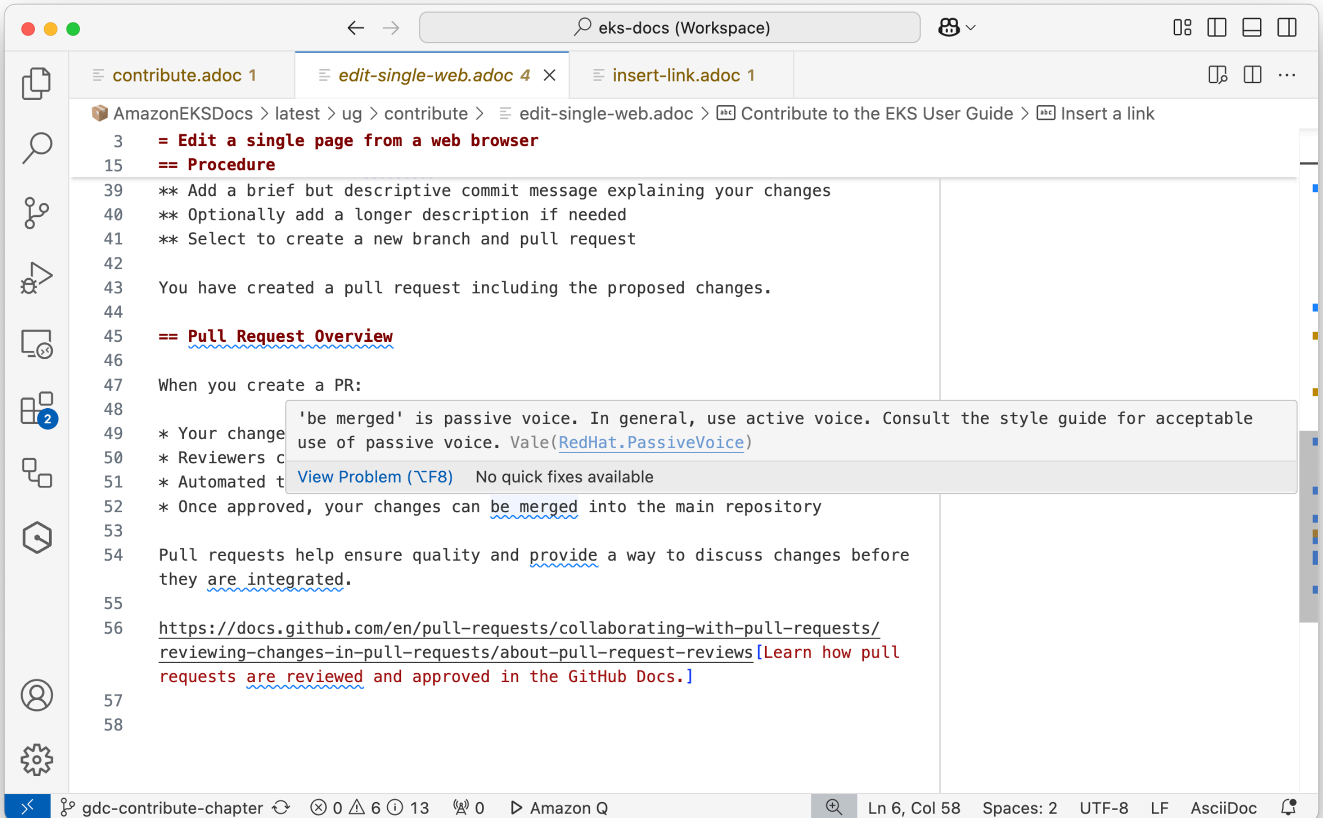
> **Note**
>
> The EKS Docs team has created a workspace file that includes suggested configurations for the editor, such as text wrapping and AsciiDoc syntax highlighting. We suggest you load this workspace file.

1. Open the workspace on GitHub.dev.
   - You can bookmark the URL `https://github.dev/awsdocs/amazon-eks-user-guide/blob/mainline/eks-docs.code-workspace?workspace=true`
2. (First time setup only) You may be prompted to create a fork of the repo in your own GitHub account. Accept this prompt. For more information, see About forks in the GitHub docs.
3. (First time setup only) Accept the prompt in the bottom right to install the AsciiDoc extension.
4. Navigate to the docs content at `latest/ug`.
   - Docs files are organized by their top level section. For example, pages in the "Security" chapter have source files under the "security/" directory.
5. To view a preview of a docs page, use the **Open preview to the Side** button in the top right. The icon includes a small magnifying glass.
6. Use the **Source Control** tab in the left to commit your changes. For more information, see the Visual Studio Code docs:
   - Commit changes
   - Create a pull request

After you create a pull request, it will be reviewed by the docs team.

View style feedback as you type by installing Vale locally

You can see style feedback as you type. This helps identify awkward writing and typos.



Overview:

- The Vale CLI loads style guides and runs them against source files.
- The EKS Docs repo includes a vale configuration file that loads style guides and local rules.
- The Vale extension for Visual Studio (VS) Code displays vale feedback inside the editor.

Install Vale

Follow the instructions in the Vale CLI docs to [Install Vale with a Package Manager](https://vale.sh/docs/install).

Install VS Code Vale extension

1. Open VS Code.
2. Click the Extensions icon in the Activity Bar (or press Ctrl+Shift+X).
3. Search for "Vale".

4. Click Install on the "Vale VSCode" extension by Chris Chinchilla.
5. Reload VS Code when prompted.

Sync Vale

Vale uses the `.vale.ini` configuration file in your project root to determine which style rules to apply.

1. Open VS Code.
2. Click **View** > **Terminal** (or press Ctrl+`).
3. Navigate to your project root directory if needed.
4. Run the command:

```
vale sync
```

5. Wait for Vale to finish downloading and syncing style rules

View style feedback in VS Code

1. Open a Markdown or AsciiDoc file in VS Code.
2. The Vale extension will automatically check your text against the style rules.
3. Style issues will be underlined in the editor.
4. Hover over underlined text to see the specific style suggestion.
5. Fix issues by following the suggestions or consulting the style guide.

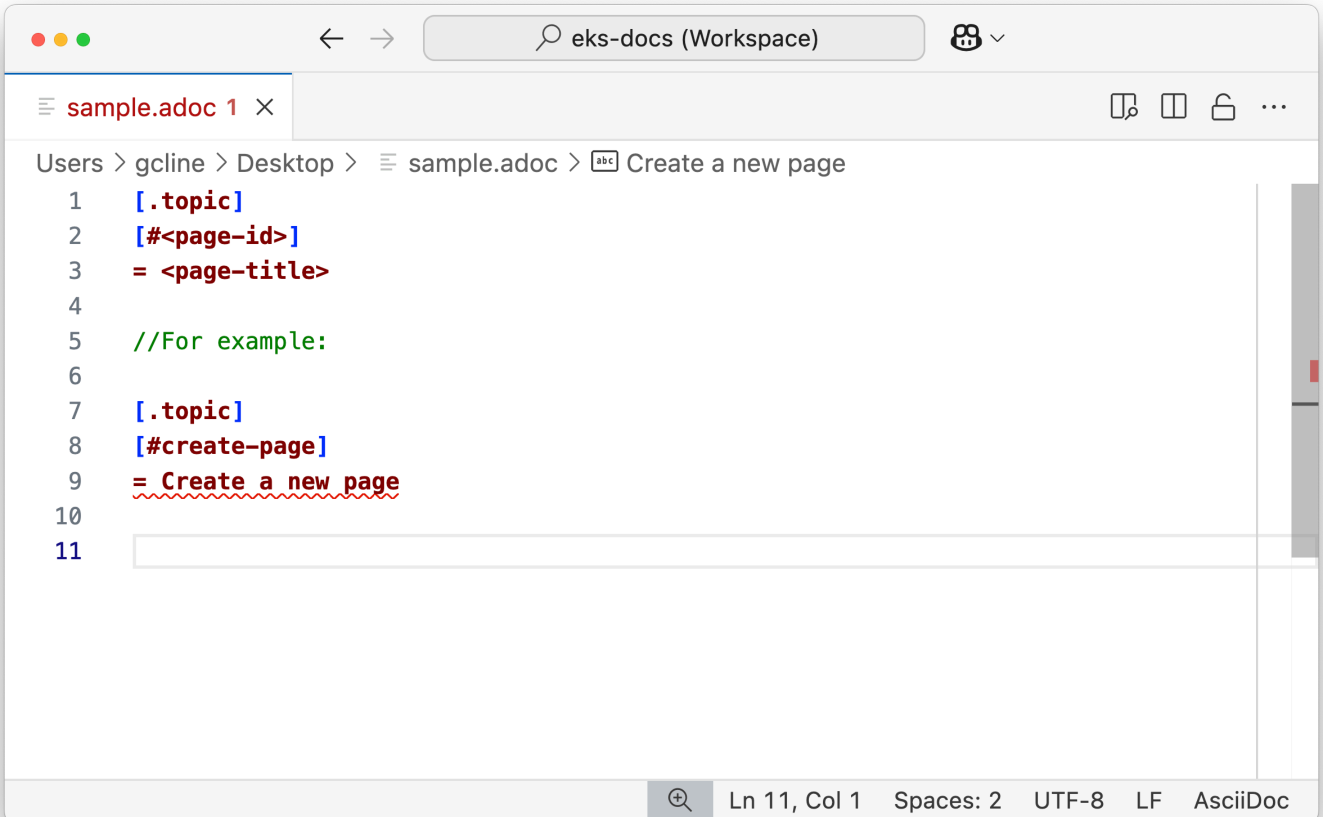
Create a new page

Learn how to create a new docs page. This topic includes instructions for creating the initial page metadata, and adding the page to the guide table of contents.

Create page

1. Navigate to the chapter directory. For example, if you want to create a new page in the "Security" section, navigate to the `latest/ug/security` directory.

2. Determine the page ID. By convention, the page ID is all lowercase and segmented with `-`. The ID of this page is `create-page`.
3. Create a new file with the page ID and the `adoc` extension. For example, `create-page.adoc`.
4. Insert the page metadata using this template:



Add page to navigation

1. Navigate to the parent page. The parent page of top level sections is `book.adoc`.
2. At the bottom of the parent page, include the child page.

   ```
   include::${filename}[leveloffset=+1]
   ```

   For example:

   ```
   include::create-page.adoc[leveloffset=+1]
   ```

Insert a link

AsciiDoc supports multiple types of links. Using the right link type is important so the link works properly in different environments.

Link to a page or section in the EKS User Guide

Use cross references (xref) to link between pages or sections within the same documentation site, such as the EKS User Guide. They automatically update if the target section moves or is renamed.

Define custom link text

```
xref :section-id[${link-text}]
```

Use page title as link text

Use the section ID surrounded by angle brackets. For example << `page-id`>>.

Link to another page in the AWS Docs

1. Find the link to the AWS Docs page
2. Remove the `https://docs.aws.amazon.com/` prefix, keeping only the path. The path should start with an a-z character.
3. Create a link as shown below

```
link :AmazonS3/latest/userguide/create-bucket-overview.html["Create a
bucket", type="documentation"]
```

Link out to the internet

This format creates a standard link out to the internet. Use this for Non-Amazon content or content on GitHub.

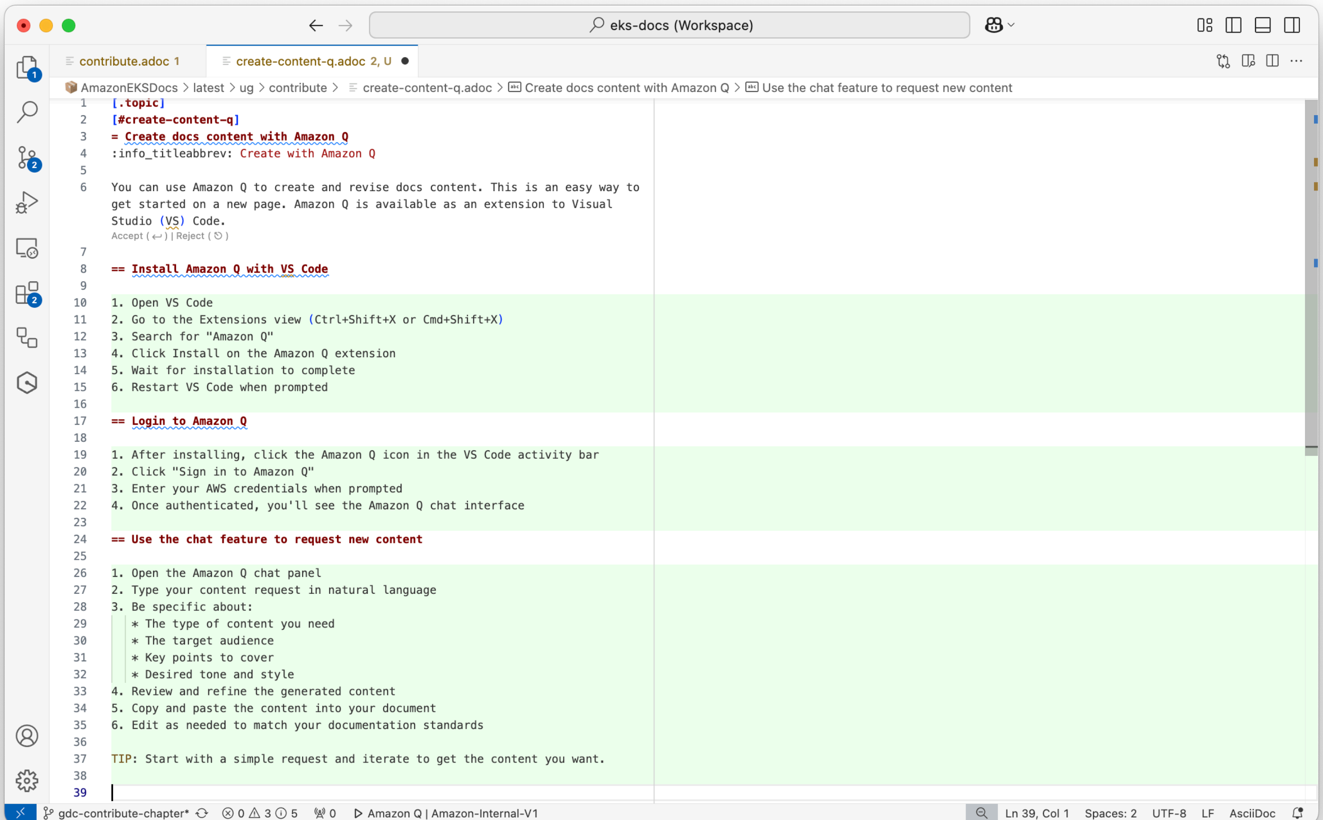
```
link :https://example.com[Visit Example Site]
```

> **Note**
>
> We allowlist external domains. The allowlist is at `vale/styles/EksDocs/ExternalDomains.yml`

Create docs content with Amazon Q

You can use Amazon Q to create and revise docs content. This is an easy way to get started on a new page. Amazon Q is available as an extension to Visual Studio (VS) Code.

In the following image, Amazon Q generated the lines marked with green.



Install Amazon Q with VS Code

1. Open VS Code
2. Go to the Extensions view (Ctrl+Shift+X or Cmd+Shift+X)
3. Search for "Amazon Q"
4. Click Install on the Amazon Q extension
5. Wait for installation to complete
6. Restart VS Code when prompted

Login to Amazon Q

1. After installing, choose the Amazon Q icon in the VS Code activity bar.
2. Choose **Sign in to Amazon Q**.
3. Enter your AWS credentials when prompted.
4. Once authenticated, you'll see the Amazon Q chat interface.

Use Amazon Q to create content

1. Open the file you want to edit in VS Code.
2. Select the text you want to revise or the location for new content.
3. Press **Ctrl+I** or **Cmd+I**.
4. In the prompt, be specific about:
   - The type of content you need.
   - The target audience.
   - Key points to cover.
   - Desired tone and style.
5. Review the generated content in the inline preview.
6. Use **enter** to accept the changes, or **esc** to reject them.
7. Edit further as needed.

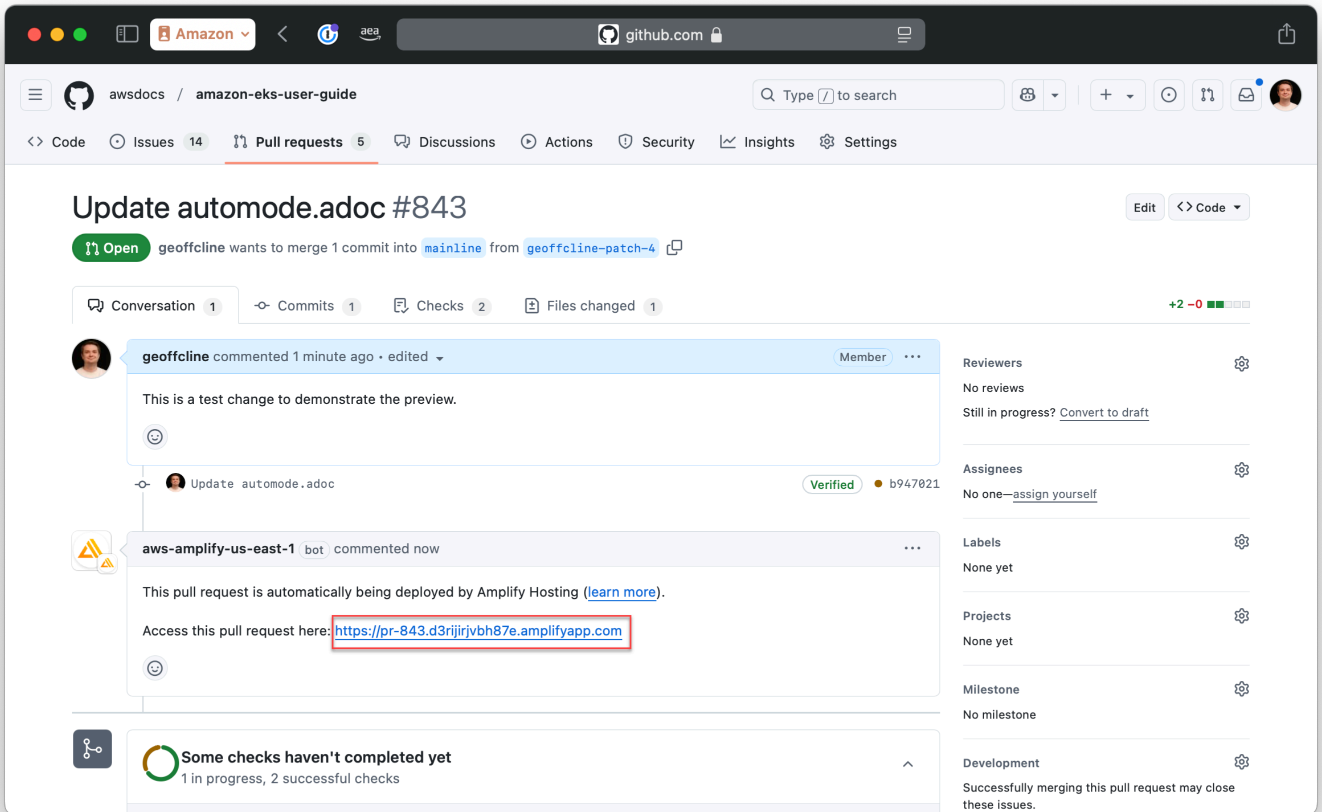
Tips

- Start with a simple request and iterate to get the content you want.
- Create a first draft of the page headings, then ask Q to fill them in.
- Amazon Q might output Markdown. This is fine. The AsciiDoc tooling can understand most markdown syntax.

To learn more about Amazon Q Developer, see Using Amazon Q Developer in the IDE.

View a preview of pull request content

The Amazon EKS User Guide GitHub is configured to build and generate a preview of the docs site. This preview doesn't have the full AWS theme, but it does check the content builds properly and links work.



This preview is hosted at a temporary URL by AWS Amplify.

View preview

When you submit a pull request, AWS Amplify attempts to build and deploy a preview of the content.

If the build succeeds, **aws-amplify-us-east-1** adds a comment to the pull request that has a link to the preview. Choose the link to the right of "Access this pull request here" (as called out in the screenshot with a red outline).

If the build fails, the repo admins can see the logs and provide feedback.

> **Note**
>
> If you haven't contributed before, a project maintainer may need to approve running the build.

Preview limitations

The preview is built as a single large HTML file. It will be displayed as multiple pages when published.

What works:

- Cross references (`xref`)
- Links to the internet
- Images
- Content hosted from `samples/`

What doesn't work:

- Links to other AWS content, using `type="documentation"`. This is because this content doesn't exist in the preview environment.
- The attribute `{aws}` will not display properly. The value of this changes based on the environment.

AsciiDoc Syntax Reference

This page is a quick overview of AsciiDoc syntax.

We suggest using AsciiDoc syntax. The AsciiDoc tooling has partial support for Markdown syntax, including headings and lists.

For advanced syntax, review the AsciiDoc Website.

It's important to get the link syntax right for the AWS docs. Review how to the section called "Insert link". You cannot use Markdown syntax for links.

Basic formatting

```
*bold text*
_italic text_
`monospace text`
```

Headers

```
= Document Title (Header 1)
== Header 2
=== Header 3
==== Header 4
===== Header 5
====== Header 6
```

Lists

Unordered Lists:

```
- Item 1
- Item 2
-- Subitem 2.1
-- Subitem 2.2
- Item 3
```

Ordered Lists:

```
. First item
. Second item
.. Subitem 2.1
.. Subitem 2.2
. Third item
```

Links

```
External link: https://example.com[Link text]
Internal link: <<page-id>>
Internal link: <<page-id,Link text>>
```

Images

```
image::image-file.jpg[Alt text]
```

Code Blocks

```
[source,python]
----
def hello_world():
    print("Hello, World!")
----
```

Tables

[Learn how to build a basic table.]

```
[cols="1,1"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3
|===
```

Admonitions

```
NOTE: This is a note admonition.

WARNING: This is a warning admonition.

TIP: This is a tip admonition.

IMPORTANT: This is an important admonition.

CAUTION: This is a caution admonition.
```

Preview:

> **Note**
>
> This is a note admonition.

Includes

```
include::filename.adoc[]
```