Provision an EKS Cluster (AWS)

9min |

Reference this often? Create an account to bookmark tutorials.

AWS's Elastic Kubernetes Service (EKS) is a managed service that lets you deploy, manage, and scale containerized applications on Kubernetes.

In this tutorial, you will deploy an EKS cluster using Terraform. Then, you will configure `kubectl` using Terraform output and verify that your cluster is ready to use.

> **Warning**
>
> AWS EKS clusters cost $0.10 per hour, so you may incur charges by running this tutorial. The cost should be a few dollars at most, but be sure to delete your infrastructure promptly to avoid additional charges. We are not responsible for any charges you may incur.

Why deploy with Terraform?

While you could use the built-in AWS provisioning processes (UI, CLI, CloudFormation) for EKS clusters, Terraform provides you with several benefits:

- **Unified Workflow** - If you already use Terraform to deploy AWS infrastructure, you can

without requiring you to inspect an API to identify those resources.

- **Graph of Relationships** - Terraform determines and observes dependencies between resources. For example, if an AWS Kubernetes cluster needs a specific VPC and subnet configurations, Terraform will not attempt to create the cluster if it fails to provision the VPC and subnet first.

Prerequisites

The tutorial assumes some basic familiarity with Kubernetes and `kubectl` but does not assume any pre-existing deployment.

You can complete this tutorial using the same workflow with either Terraform OSS or Terraform Cloud. Terraform Cloud is a platform that you can use to manage and execute your Terraform projects. It includes features like remote state and execution, structured plan output, workspace resource summaries, and more.

Select the **Terraform OSS** tab to complete this tutorial using Terraform OSS.

Terraform Cloud | Terraform OSS

This tutorial assumes that you are familiar with the Terraform workflow. If you are new to Terraform, complete the Get Started collection first.

For this tutorial, you will need:

- Terraform v1.3+ installed locally.
- an AWS account
- the AWS CLI v2.7.0/v1.24.0 or newer, installed and configured
- AWS IAM Authenticator
- kubectl v1.24.0 or newer

Set up and initialize your Terraform workspace

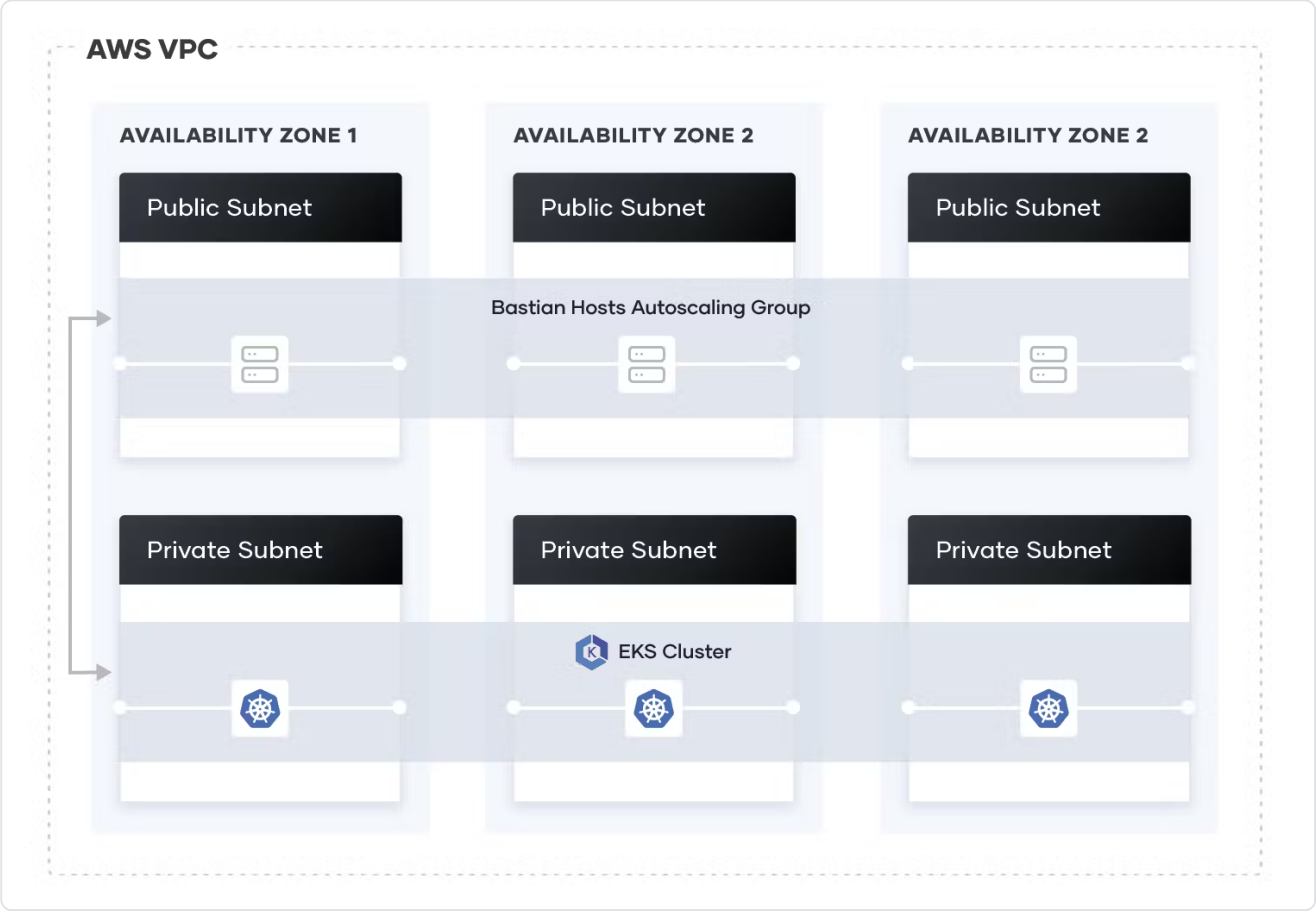
In your terminal, clone the example repository for this tutorial.

```
$ git clone https://github.com/hashicorp/learn-terraform-provision-eks-cluster
```

Change into the repository directory.

```
$ cd learn-terraform-provision-eks-cluster
```

This example repository contains configuration to provision a VPC, security groups, and an EKS cluster with the following architecture:



The configuration defines a new VPC in which to provision the cluster, and uses the public EKS module to create the required resources, including Auto Scaling Groups, security groups, and IAM Roles and Policies.

Open the `main.tf` file to review the module configuration. The `eks_managed_node_groups` parameter configures the cluster with three nodes across two node groups.

main.tf

```
eks_managed_node_groups = {
one = {
    name = "node-group-1"

    instance_types = ["t3.small"]

    min_size     = 1
    max_size     = 3
    desired_size = 2
}

two = {
    name = "node-group-2"

    instance_types = ["t3.small"]

    min_size     = 1
    max_size     = 2
    desired_size = 1
}
}
```

Initialize configuration

Terraform Cloud | Terraform OSS

Open your `terraform.tf` file and comment out the `cloud` block that configures the Terraform Cloud integration.

terraform.tf

```
terraform {
  /*
  cloud {
    workspaces {
      name = "learn-terraform-eks"
    }
  }
  */

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.7.0"
    }
  }
  required_version = ">= 1.3"
  ##...
}
```

Initialize this configuration.

```
$ terraform init
Initializing the backend...
##...
Terraform has been successfully initialized!
You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, othe
commands will detect it and remind you to do so if necessary.
```

Provision the EKS cluster

Run `terraform apply` to create your cluster and other necessary resources. Confirm the operation with a `yes`.

This process can take up to 10 minutes. Upon completion, Terraform will print your configuration's outputs.

```
$ terraform apply

## ...

Terraform used the selected providers to generate the following execution plan. Re
  + create
 <= read (data resources)

Terraform will perform the following actions:

## ...

Plan: 63 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + cluster_endpoint          = (known after apply)
  + cluster_name              = (known after apply)
  + cluster_security_group_id = (known after apply)
  + region                    = "us-east-2"

Do you want to perform these actions in workspace "learn-terraform-eks"?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

## ...

Apply complete! Resources: 63 added, 0 changed, 0 destroyed.

Outputs:

cluster_endpoint = "https://128CA2A0D737317D36E31D0D3A0C366B.gr7.us-east-2.eks.ama
cluster_name = "education-eks-IKQYD53K"
cluster_security_group_id = "sg-0f836e078948afb70"
```

```
region = "us-east-2"
```

Configure kubectl

After provisioning your cluster, you need to configure `kubectl` to interact with it.

First, open the `outputs.tf` file to review the output values. You will use the `region` and `cluster_name` outputs to configure `kubectl`.

outputs.tf

```
output "cluster_endpoint" {
  description = "Endpoint for EKS control plane"
  value       = module.eks.cluster_endpoint
}

output "cluster_security_group_id" {
  description = "Security group ids attached to the cluster control plane"
  value       = module.eks.cluster_security_group_id
}

output "region" {
  description = "AWS region"
  value       = var.region
}

output "cluster_name" {
  description = "Kubernetes Cluster Name"
  value       = module.eks.cluster_name
}
```

Run the following command to retrieve the access credentials for your cluster and configure `kubectl`.

```
$ aws eks --region $(terraform output -raw region) update-kubeconfig \
    --name $(terraform output -raw cluster_name)
```

You can now use `kubectl` to manage your cluster and deploy Kubernetes configurations to it.

Verify the Cluster

Use `kubectl` commands to verify your cluster configuration.

First, get information about the cluster.

```
$ kubectl cluster-info
Kubernetes control plane is running at https://128CA2A0D737317D36E31D0D3A0C366B.gr
CoreDNS is running at https://128CA2A0D737317D36E31D0D3A0C366B.gr7.us-east-2.eks.a

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

Notice that the Kubernetes control plane location matches the `cluster_endpoint` value from the `terraform apply` output above.

Now verify that all three worker nodes are part of the cluster.

```
$ kubectl get nodes
NAME                                          STATUS   ROLES    AGE     VERSION
ip-10-0-1-50.us-east-2.compute.internal    Ready    <none>   6m20s   v1.24.7-eks-f
ip-10-0-3-158.us-east-2.compute.internal   Ready    <none>   6m41s   v1.24.7-eks-f
ip-10-0-3-46.us-east-2.compute.internal    Ready    <none>   6m14s   v1.24.7-eks-f
```

You have verified that you can connect to your cluster using `kubectl` and that all three worker nodes are healthy. Your cluster is ready to use.

Clean up your workspace

You have now provisioned an EKS cluster, configured `kubectl`, and verified that your cluster is ready to use.

To learn how to manage your EKS cluster using the Terraform Kubernetes Provider, leave your cluster running and continue to the Kubernetes provider tutorial.

> **Note**
> This tutorial's example configuration contains **only** the configuration to create an EKS cluster. You can also use Terraform to manage the Kubernetes objects deployed to a cluster, such as Deployments and Services. We recommend that you keep the Terraform configuration for each of those concerns separate, since their workflows and owners often differ.

Destroy the resources you created in this tutorial to avoid incurring extra charges. Respond `yes` to the prompt to confirm the operation.

```
$ terraform destroy

## ...

Plan: 0 to add, 0 to change, 63 to destroy.

Changes to Outputs:
  - cluster_endpoint          = "https://128CA2A0D737317D36E31D0D3A0C366B.gr7.us-e
  - cluster_name              = "education-eks-IKQYD53K" -> null
  - cluster_security_group_id = "sg-0f836e078948afb70" -> null
  - region                    = "us-east-2" -> null

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

## ...

Destroy complete! Resources: 63 destroyed.
```

If you used Terraform Cloud for this tutorial, after destroying your resources, delete the `learn-terraform-eks` workspace from your Terraform Cloud organization.

Next steps

For more information on the EKS module, visit the EKS module page in the Terraform Registry.

To learn how to manage Kubernetes resources, your EKS cluster, or existing Kubernetes clusters, visit the Kubernetes provider tutorial.

You can also use the Kubernetes provider to deploy custom resources.

For a more in-depth Kubernetes example, see Deploy Consul and Vault on a Kubernetes Cluster using Run Triggers (this tutorial is GKE based).

Was this tutorial helpful? Yes No

Back to Collection

Next

This tutorial also appears in:

14 tutorials

Manage AWS Services

Use the AWS provider to manage AWS services with Terraform. Configure IAM policy documents, deploy serverless functions with Lambda, use application load balancers to schedule near-zero downtime releases, manage RDS and more.