Provision an EKS Cluster (AWS)

11 MIN | PRODUCTS USED: Terraform

This tutorial also appears in: AWS Services.

The Amazon Elastic Kubernetes Service (EKS) is the AWS service for deploying, managing, and scaling containerized applications with Kubernetes.

In this tutorial, you will deploy an EKS cluster using Terraform. Then, you will configure `kubectl` using Terraform output to deploy a Kubernetes dashboard on the cluster.

> **Warning!** AWS charges `$0.10` per hour for each EKS cluster. As a result, you may be charged to run these examples. The most you should be charged should only be a few dollars, but we're not responsible for any charges that 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 are already deploying infrastructure to AWS with Terraform, your EKS cluster can fit into that workflow. You can also deploy applications into your EKS cluster using Terraform.
- **Full Lifecycle Management** - Terraform doesn't only create resources, it updates, and deletes tracked resources without requiring you to inspect the API to identify those resources.
- **Graph of Relationships** - Terraform understands dependency relationships between resources. For example, if an AWS Kubernetes cluster needs a specific VPC and subnet configurations, Terraform won't attempt to create the cluster if the VPC and subnets failed to create with the proper configuration.

Prerequisites

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

It also assumes that you are familiar with the usual Terraform plan/apply workflow. If you're new to Terraform itself, refer first to the Getting Started tutorial.

For this tutorial, you will need:

- an AWS account with the IAM permissions listed on the EKS module documentation,
- a configured AWS CLI
- AWS IAM Authenticator
- `kubectl`
- wget (required for the `eks` module)

Configured AWS CLI AWS IAM Authenticator kubectl **wget**

To install the wget, follow these instructions or choose a package manager based on your operating system.

macOS install with Homebrew **Windows install with Chocolatey**

Use the package manager `Chocolatey` to install wget.

```
$ choco install wget
```

Copy

Set up and initialize your Terraform workspace

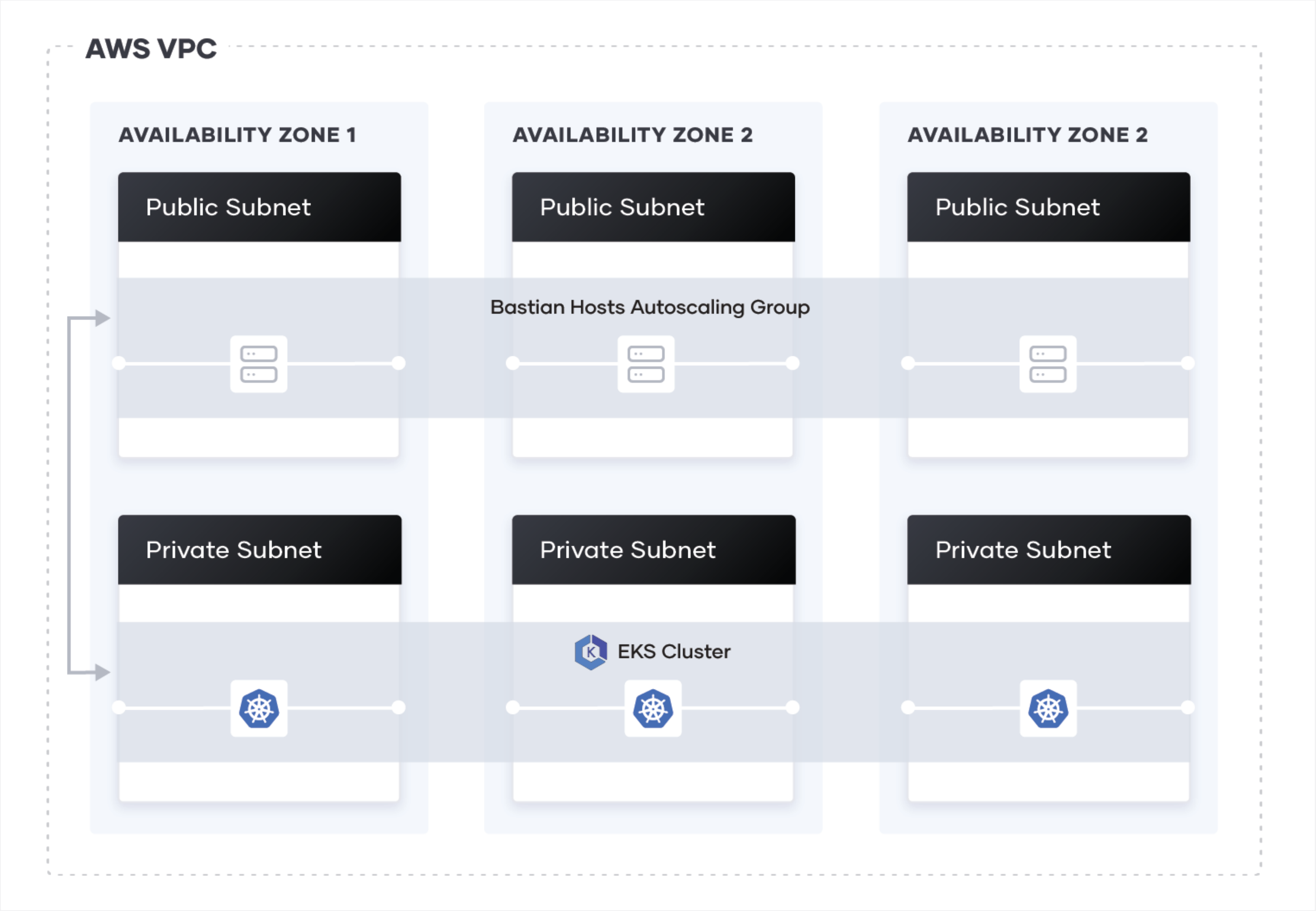
In your terminal, clone the following repository. It contains the example configuration used in this tutorial.

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

You can explore this repository by changing directories or navigating in your UI.

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

In here, you will find six files used to provision a VPC, security groups and an EKS cluster. The final product should be similar to this:


AWS VPC
AVAILABILITY ZONE 1
AVAILABILITY ZONE 2
AVAILABILITY ZONE 2
Public Subnet
Public Subnet
Public Subnet
Bastian Hosts Autoscaling Group
Private Subnet
Private Subnet
Private Subnet
EKS Cluster

1. `vpc.tf` provisions a VPC, subnets and availability zones using the AWS VPC Module. A new VPC is created for this tutorial so it doesn't impact your existing cloud environment and resources.

2. `security-groups.tf` provisions the security groups used by the EKS cluster.

3. `eks-cluster.tf` provisions all the resources (AutoScaling Groups, etc...) required to set up an EKS cluster using the AWS EKS Module.

   On line 14, the AutoScaling group configuration contains three nodes.

```
worker_groups = [
  {
    name                          = "worker-group-1"
    instance_type                 = "t2.small"
    additional_userdata           = "echo foo bar"
    asg_desired_capacity          = 2
    additional_security_group_ids = [aws_security_group.worker_group_mgmt_one.id]
  },
  {
    name                          = "worker-group-2"
    instance_type                 = "t2.medium"
    additional_userdata           = "echo foo bar"
    additional_security_group_ids = [aws_security_group.worker_group_mgmt_two.id]
    asg_desired_capacity          = 1
  },
]
```

4. `outputs.tf` defines the output configuration.

5. `versions.tf` sets the Terraform version to at least 0.14. It also sets versions for the providers used in this sample.

Initialize Terraform workspace

Once you have cloned the repository, initialize your Terraform workspace, which will download and configure the providers.

```
$ terraform init
Initializing modules...
Downloading terraform-aws-modules/eks/aws 13.2.1 for eks...
- eks in .terraform/modules/eks
- eks.fargate in .terraform/modules/eks/modules/fargate
- eks.node_groups in .terraform/modules/eks/modules/node_groups
Downloading terraform-aws-modules/vpc/aws 2.66.0 for vpc...
- vpc in .terraform/modules/vpc

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Reusing previous version of hashicorp/random from the dependency lock file
- Reusing previous version of hashicorp/local from the dependency lock file
- Reusing previous version of hashicorp/null from the dependency lock file
- Reusing previous version of hashicorp/template from the dependency lock file
- Reusing previous version of hashicorp/kubernetes from the dependency lock file
- Installing hashicorp/kubernetes v2.0.1...
- Installed hashicorp/kubernetes v2.0.1 (signed by HashiCorp)
- Installing hashicorp/aws v3.25.0...
- Installed hashicorp/aws v3.25.0 (signed by HashiCorp)
- Installing hashicorp/random v3.0.0...
- Installed hashicorp/random v3.0.0 (signed by HashiCorp)
- Installing hashicorp/local v2.0.0...
- Installed hashicorp/local v2.0.0 (signed by HashiCorp)
- Installing hashicorp/null v3.0.0...
- Installed hashicorp/null v3.0.0 (signed by HashiCorp)
- Installing hashicorp/template v2.2.0...
- Installed hashicorp/template v2.2.0 (signed by HashiCorp)

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, other
commands will detect it and remind you to do so if necessary.
```

Provision the EKS cluster

In your initialized directory, run `terraform apply` and review the planned actions. Your terminal output should indicate the plan is running and what resources will be created.

```
$ terraform apply

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create
 <= read (data resources)

Terraform will perform the following actions:


  ## ...

Plan: 51 to add, 0 to change, 0 to destroy.


  ## ...

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value:
```

This terraform apply will provision a total of 51 resources (VPC, Security Groups, AutoScaling Groups, EKS Cluster, etc…). Confirm the apply with a `yes`.

This process should take approximately 10 minutes. Upon successful application, your terminal prints the outputs defined in `outputs.tf`.

```
Apply complete! Resources: 51 added, 0 changed, 0 destroyed.

Outputs:

cluster_endpoint = "https://B7994AFC945AB5029A4E2BA8DB39B448.gr7.us-east-2.eks.amazonaws.com"
cluster_id = "education-eks-p8Zqwv78"
cluster_name = "education-eks-p8Zqwv78"
cluster_security_group_id = "sg-09378904e5421f22e"
config_map_aws_auth = [
  {
    "binary_data" = tomap(null) /* of string */
    "data" = tomap({
      "mapAccounts" = <<-EOT
      []

      EOT
      "mapRoles" = <<-EOT
      - "groups":
        - "system:bootstrappers"
        - "system:nodes"
        "rolearn": "arn:aws:iam::561656980159:role/education-eks-p8Zqwv782021011204012796010000000c"
        "username": "system:node:{{EC2PrivateDNSName}}"


      EOT
      "mapUsers" = <<-EOT
      []


      EOT
    })
    "id" = "kube-system/aws-auth"
    "metadata" = tolist([
      {
        "annotations" = tomap(null) /* of string */
        "generate_name" = ""
        "generation" = 0
        "labels" = tomap({
          "app.kubernetes.io/managed-by" = "Terraform"
          "terraform.io/module" = "terraform-aws-modules.eks.aws"
        })
        "name" = "aws-auth"
        "namespace" = "kube-system"
        "resource_version" = "942"
        "self_link" = "/api/v1/namespaces/kube-system/configmaps/aws-auth"
        "uid" = "f328b2d0-f099-4e72-a1b1-760781045f10"
      },
    ])
  },
]
```

```
kubectl_config = ...
## ...
```

Configure kubectl

Now that you've provisioned your EKS cluster, you need to configure `kubectl`.

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

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

The Kubernetes cluster name and region correspond to the output variables showed after the successful Terraform run.

Deploy and access Kubernetes Dashboard

To verify that your cluster is configured correctly and running, you will deploy the Kubernetes dashboard and navigate to it in your local browser.

While you can deploy the Kubernetes metrics server and dashboard using Terraform, `kubectl` is used in this tutorial so you don't need to configure your Terraform Kubernetes Provider.

Deploy Kubernetes Metrics Server

The Kubernetes Metrics Server, used to gather metrics such as cluster CPU and memory usage over time, is not deployed by default in EKS clusters.

Download and unzip the metrics server by running the following command.

```
://codeload.github.com/kubernetes-sigs/metrics-server/tar.gz/v0.3.6 && tar -xzf v0.3.6.tar.gz
```

Deploy the metrics server to the cluster by running the following command.

```
$ kubectl apply -f metrics-server-0.3.6/deploy/1.8+/
```

Verify that the metrics server has been deployed. If successful, you should see something like this.

```
$ kubectl get deployment metrics-server -n kube-system
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
metrics-server   1/1     1            1           4s
```

Deploy Kubernetes Dashboard

The following command will schedule the resources necessary for the dashboard.

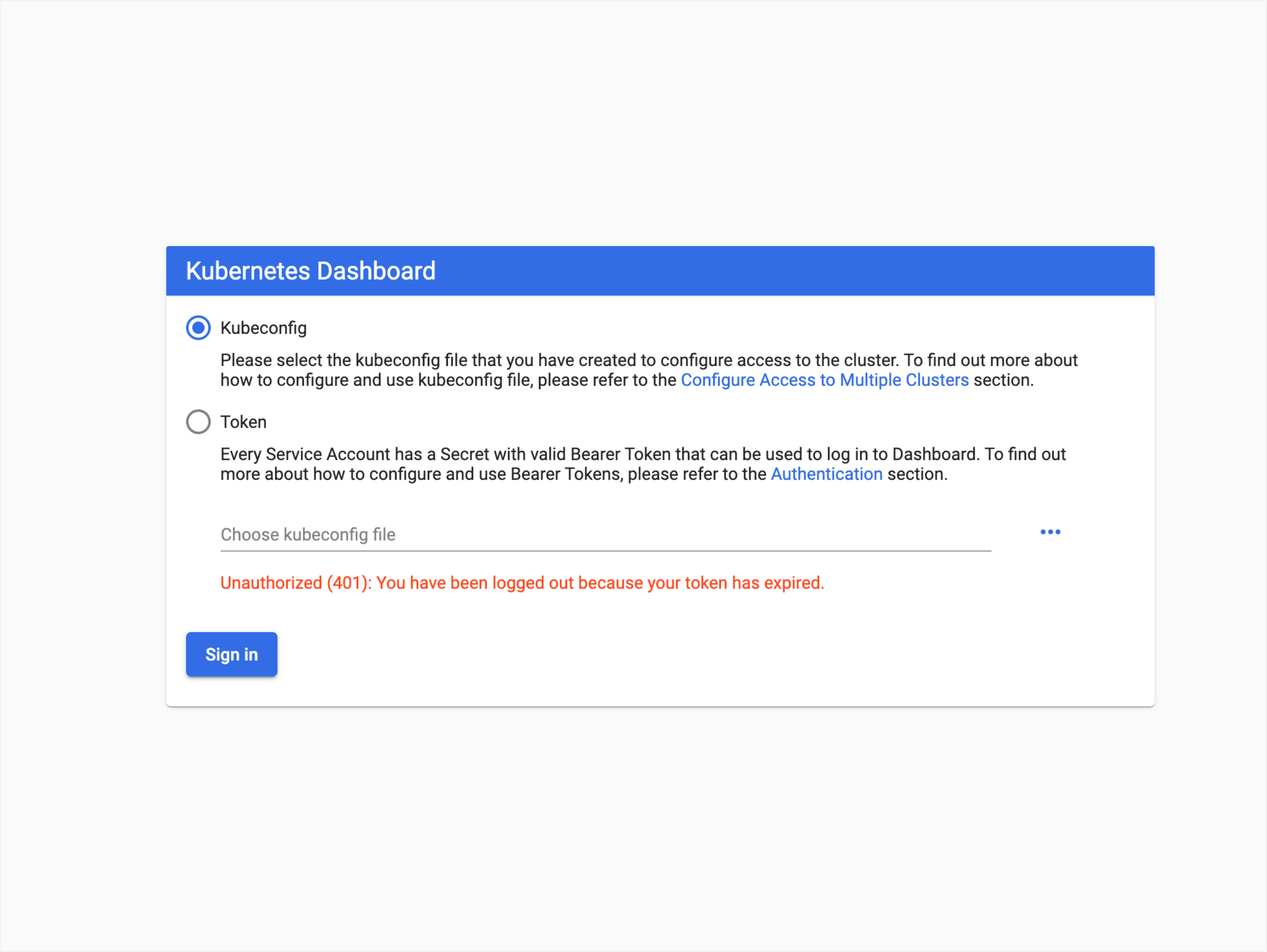
```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-beta8/aio/d

namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created
```

Now, create a proxy server that will allow you to navigate to the dashboard from the browser on your local machine. This will continue running until you stop the process by pressing `CTRL + C`.

```
$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```

You should be able to access the Kubernetes dashboard here (`http://127.0.0.1:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/`).



Kubernetes Dashboard

● Kubeconfig

Please select the kubeconfig file that you have created to configure access to the cluster. To find out more about how to configure and use kubeconfig file, please refer to the Configure Access to Multiple Clusters section.

○ Token

Every Service Account has a Secret with valid Bearer Token that can be used to log in to Dashboard. To find out more about how to configure and use Bearer Tokens, please refer to the Authentication section.

Choose kubeconfig file ...

Unauthorized (401): You have been logged out because your token has expired.

Sign in

Authenticate the dashboard

To use the Kubernetes dashboard, you need to create a `ClusterRoleBinding` and provide an authorization token. This gives the `cluster-admin` permission to access the `kubernetes-dashboard`. Authenticating using `kubeconfig` is **not** an option. You can read more about it in the [Kubernetes documentation](#).

In another terminal (do not close the `kubectl proxy` process), create the `ClusterRoleBinding` resource.

```
$ kubectl apply -f https://raw.githubusercontent.com/hashicorp/learn-terraform-provision-eks
```

Copy

Then, generate the authorization token.

```
$ kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep service-

Name:         service-controller-token-46qlm
Namespace:    kube-system
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: service-controller
```

Copy

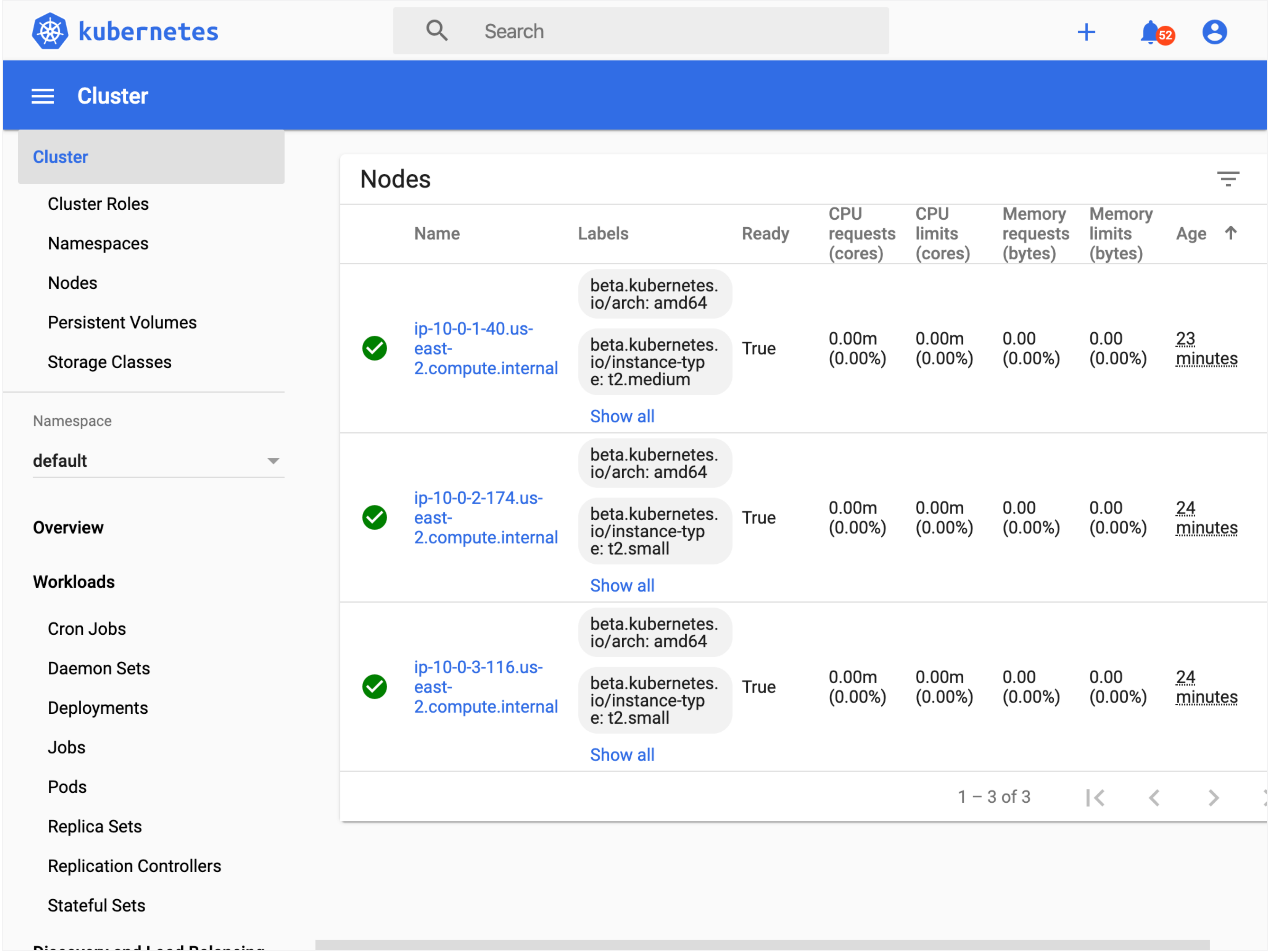
HashiCorp Learn Browse tutorials Sign in

Show sidebar Jump to section Bookmark

```
Data
====
namespace:  11 bytes
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6I...
ca.crt:     1025 bytes
```

Select "Token" on the Dashboard UI then copy and paste the entire token you receive into the dashboard authentication screen to sign in. You are now signed in to the dashboard for your Kubernetes cluster.

Navigate to the "Cluster" page by clicking on "Cluster" in the left navigation bar. You should see a list of nodes in your cluster.



Nodes

	Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	Memory requests (bytes)	Memory limits (bytes)	Age ↑
✓	ip-10-0-1-40.us-east-2.compute.internal	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/instance-type: t2.medium Show all	True	0.00m (0.00%)	0.00m (0.00%)	0.00 (0.00%)	0.00 (0.00%)	23 minutes
✓	ip-10-0-2-174.us-east-2.compute.internal	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/instance-type: t2.small Show all	True	0.00m (0.00%)	0.00m (0.00%)	0.00 (0.00%)	0.00 (0.00%)	24 minutes
✓	ip-10-0-3-116.us-east-2.compute.internal	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/instance-type: t2.small Show all	True	0.00m (0.00%)	0.00m (0.00%)	0.00 (0.00%)	0.00 (0.00%)	24 minutes

1 – 3 of 3

Clean up your workspace

Congratulations, you have provisioned an EKS cluster, configured `kubectl`, and deployed the Kubernetes dashboard.

If you'd like to learn how to manage your EKS cluster using the Terraform Kubernetes Provider, leave your cluster running and continue to the Kubernetes provider Learn tutorial.

> **Note:** This directory is **only** used to provision a EKS cluster with Terraform. By keeping the Terraform configuration for provisioning a Kubernetes cluster and managing a Kubernetes cluster resources separate, changes in one repository don't affect the other. In addition, the modularity makes the configuration more readable and enables you to scope different permissions to each workspace.

If not, remember to destroy any resources you create once you are done with this tutorial. Run the `destroy` command and confirm with `yes` in your terminal.

```
$ terraform destroy
```

Copy

Next steps

For more information on the EKS provider, visit the AWS provider documentation.

For steps on how to manage Kubernetes resources your EKS cluster or any other already created Kubernetes cluster, visit the Kubernetes provider Learn tutorial.

For a more in-depth Kubernetes example, 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 ›

stdin: is not a tty