Provision a GKE cluster (Google Cloud)

- Terraform

The Google Kubernetes Engine (GKE) is a fully managed Kubernetes service for deploying, managing, and scaling containerized applications on Google Cloud.

In this tutorial, you will deploy a 2-node separately managed node pool GKE cluster using Terraform. This GKE cluster will be distributed across multiple zones for high availability. Then, you will configure `kubectl` using Terraform output to deploy a Kubernetes dashboard on the cluster.

Warning! Google Cloud charges about ten cents per hour management fee for each GKE cluster, in addition to the cluster's resource costs. One zonal cluster per billing account is free. 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.

Tip

This example configuration provisions a GKE cluster with 2 nodes so it's under the default `IN_USE_ADDRESSES` quota. This configuration should be used as a learning exercise only — do not run a 2-node cluster in production.

Why deploy with Terraform?

While you could use the built-in GCP provisioning processes (UI, SDK/CLI) for GKE clusters, Terraform provides you with several benefits:

- **Unified Workflow** - If you are already deploying infrastructure to Google Cloud with Terraform, your GKE cluster can fit into that workflow. You can also deploy applications into your GKE 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 you require a separately managed node pool, Terraform won't attempt to create the node pool if the GKE cluster failed to create.

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

- a GCP account
- a configured gcloud SDK
- `kubectl`

Configured gcloud SDKkubectl
In order for Terraform to run operations on your behalf, you must install and configure the `gcloud` SDK tool. To install the `gcloud` SDK, follow these instructions or choose a package manager based on your operating system.
macOS install with HomebrewWindows install with Chocolatey
You can also use the package manager `homebrew` to install the gcloud SDK.

```
$ brew install --cask google-cloud-sdk
```

After you've installed the `gcloud` SDK, initialize it by running the following command.

```
$ gcloud init
```

This will authorize the SDK to access GCP using your user account credentials and add the SDK to your PATH. This steps requires you to login and select the project you want to work in. Finally, add your account to the Application Default Credentials (ADC). This will allow Terraform to access these credentials to provision resources on GCloud.

```
$ gcloud auth application-default login
```

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-gke-cluster
```

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

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

In here, you will find four files used to provision a VPC, subnets and a GKE cluster.

1. `vpc.tf` provisions a VPC and subnet. A new VPC is created for this tutorial so it doesn't impact your existing cloud environment and resources. This file outputs `region`.
2. `gke.tf` provisions a GKE cluster and a separately managed node pool (recommended). Separately managed node pools allows you to customize your Kubernetes cluster profile — this is useful if some Pods require more resources than others. You can learn more here. The number of nodes in the node pool is defined also defined here.
3. `terraform.tfvars` is a template for the `project_id` and `region` variables.
4. `versions.tf` sets the Terraform version to at least 0.14.

Update your `terraform.tfvars` file

Replace the values in your `terraform.tfvars` file with your `project_id` and `region`. Terraform will use these values to target your project when provisioning your resources. Your `terraform.tfvars` file should look like the following.

```
# terraform.tfvars

project_id = "REPLACE_ME"

region     = "us-central1"
```

You can find the project your `gcloud` is configured to with this command.

```
$ gcloud config get-value project
```

The region has been defaulted to `us-central1`; you can find a full list of gcloud regions here.

Initialize Terraform workspace

After you have saved your customized variables file, initialize your Terraform workspace, which will download the provider and initialize it with the values provided in your `terraform.tfvars` file.

```
$ terraform init


Initializing the backend...
Initializing provider plugins...

- Reusing previous version of hashicorp/google from the dependency lock file

- Installing hashicorp/google v4.74.0...

- Installed hashicorp/google v4.74.0 (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 GKE cluster

NOTE Compute Engine API and Kubernetes Engine API are required for `terraform apply` to work on this configuration. Enable both APIs for your Google Cloud project before continuing.

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
```

Terraform will perform the following actions:

...

Plan: 4 to add, 0 to change, 0 to destroy.

...

You can see this terraform apply will provision a VPC, subnet, GKE Cluster and a GKE node pool. Confirm the apply with a `yes`.
This process should take approximately 10 minutes. Upon successful application, your terminal prints the outputs defined in `vpc.tf` and `gke.tf`.
Apply complete! Resources: 4 added, 0 changed, 0 destroyed.

Outputs:

kubernetes_cluster_host = "35.232.196.187"

kubernetes_cluster_name = "dos-terraform-edu-gke"

project_id = "dos-terraform-edu"

region = "us-central1"

Configure kubectl

Now that you've provisioned your GKE cluster, you need to configure `kubectl`.
Run the following command to retrieve the access credentials for your cluster and automatically configure `kubectl`.

```
$ gcloud container clusters get-credentials
                                    --region
```

Fetching cluster endpoint and auth data.

```
kubeconfig entry generated for dos-terraform-edu-gke.
```

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

Troubleshooting

You may see the following warning message when you try to retrieve your cluster credentials. This may be because your Kubernetes cluster is still initializing/updating. If this happens, you can still proceed to the next step.

```
WARNING: cluster dos-terraform-edu-gke is not running. The kubernetes API may not be available.
```

Deploy and access Kubernetes Dashboard

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

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

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

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml
```

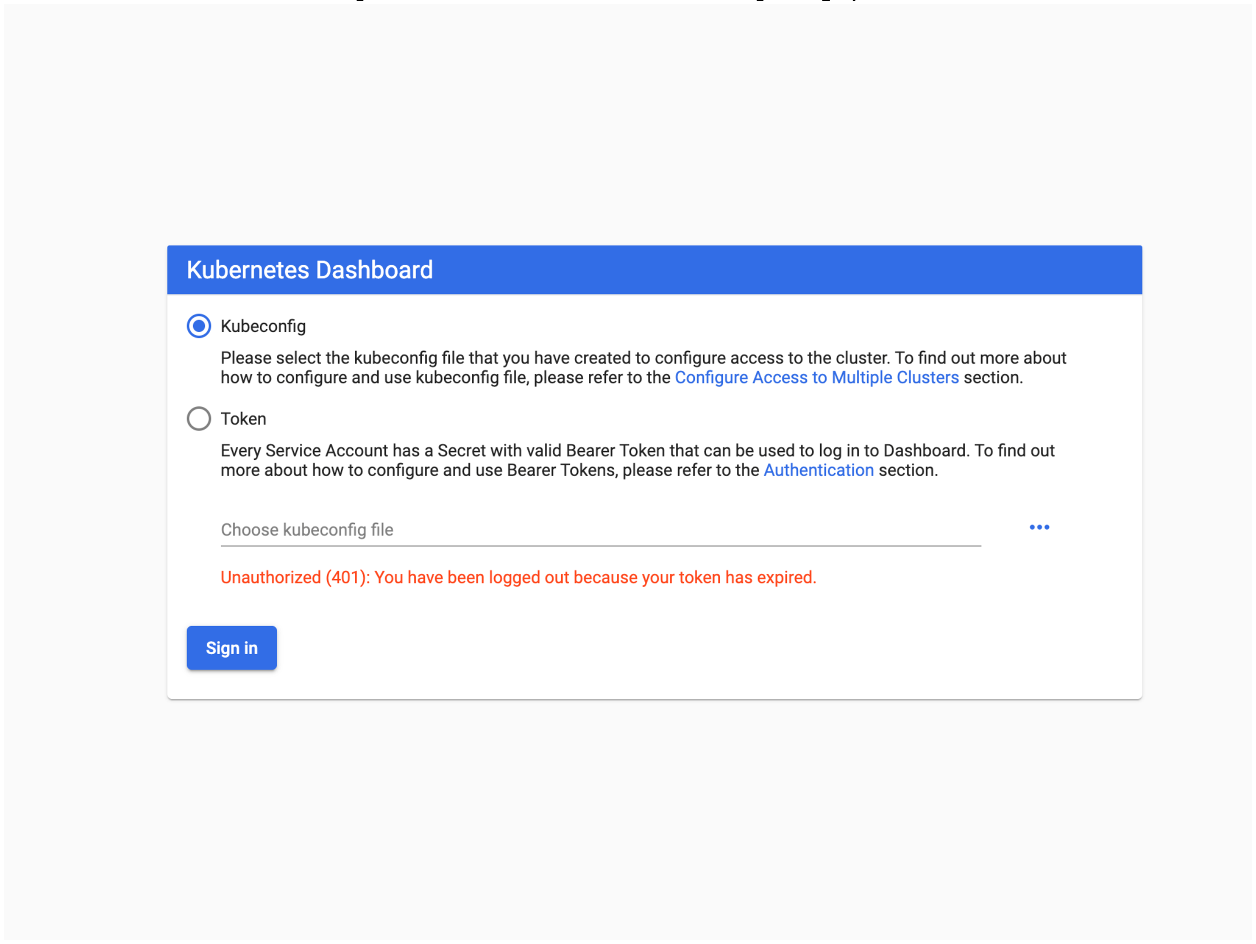
```
namespace/kubernetes-dashboard created

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
```

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/`).



[✓ 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.

Authenticate to Kubernetes 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.

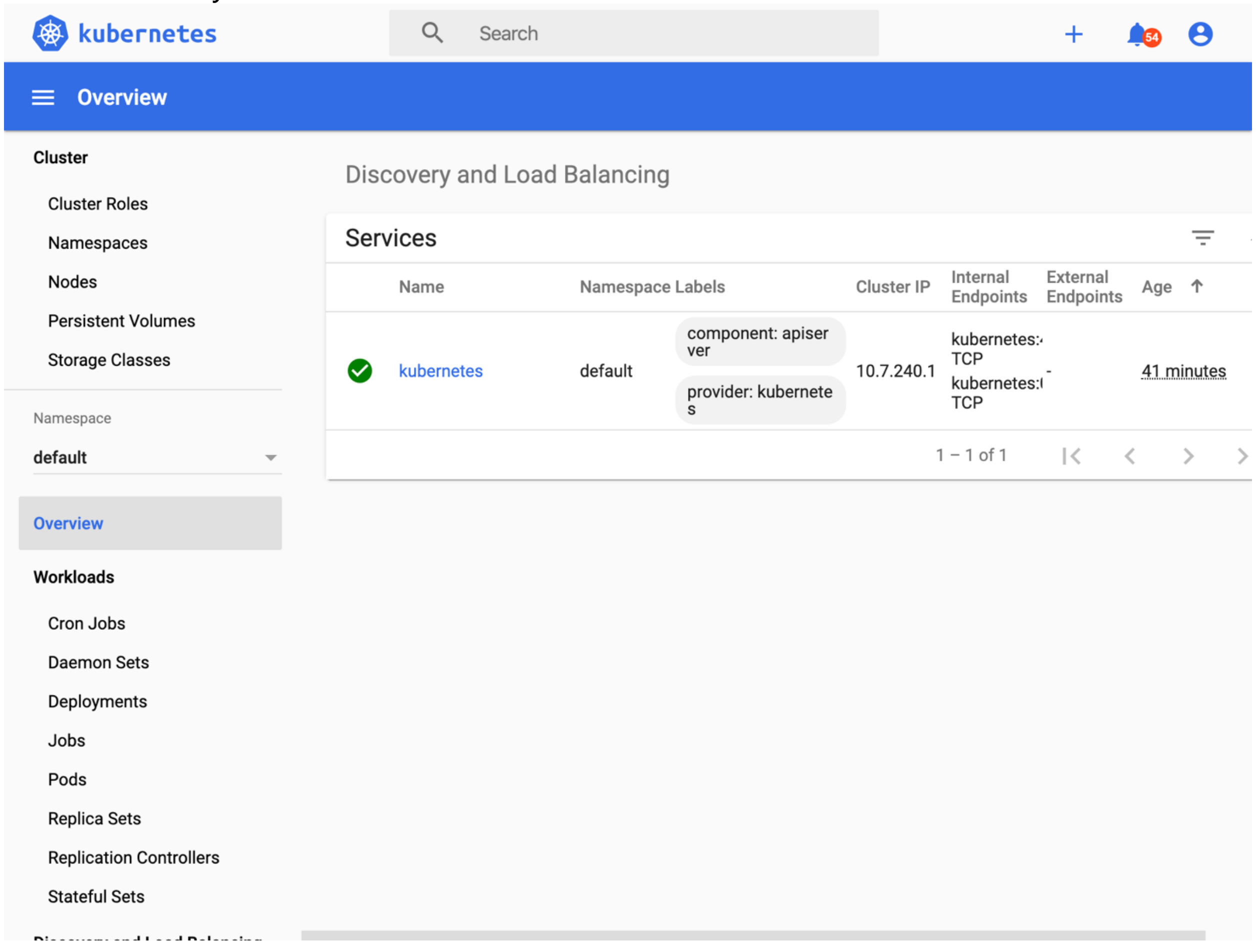
```
$ kubectl apply -f https://raw.githubusercontent.com/hashicorp/learn-terraform-provision-gke-cluster/main/kubernetes-dashboard-admin.rbac.yaml
```

Then, generate the authorization token.

```
$ kubectl -n kube-system describe secret                                   |
                              |     '{print $1}'
```

```
Name:         service-controller-token-m8m7j

Namespace:    kube-system

Labels:       <none>

Annotations:  kubernetes.io/service-account.name: service-controller

              kubernetes.io/service-account.uid: bc99ddad-6be7-11ea-a3c7-42010a800017


Type:  kubernetes.io/service-account-token


Data

====

token:      eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9...

ca.crt:     1119 bytes

namespace:  11 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.



Name	Namespace Labels		Cluster IP	Internal Endpoints	External Endpoints	Age
kubernetes	default	component: apiserver provider: kubernetes	10.7.240.1	kubernetes: TCP kubernetes: TCP	-	41 minutes

(Optional) GKE nodes and node pool

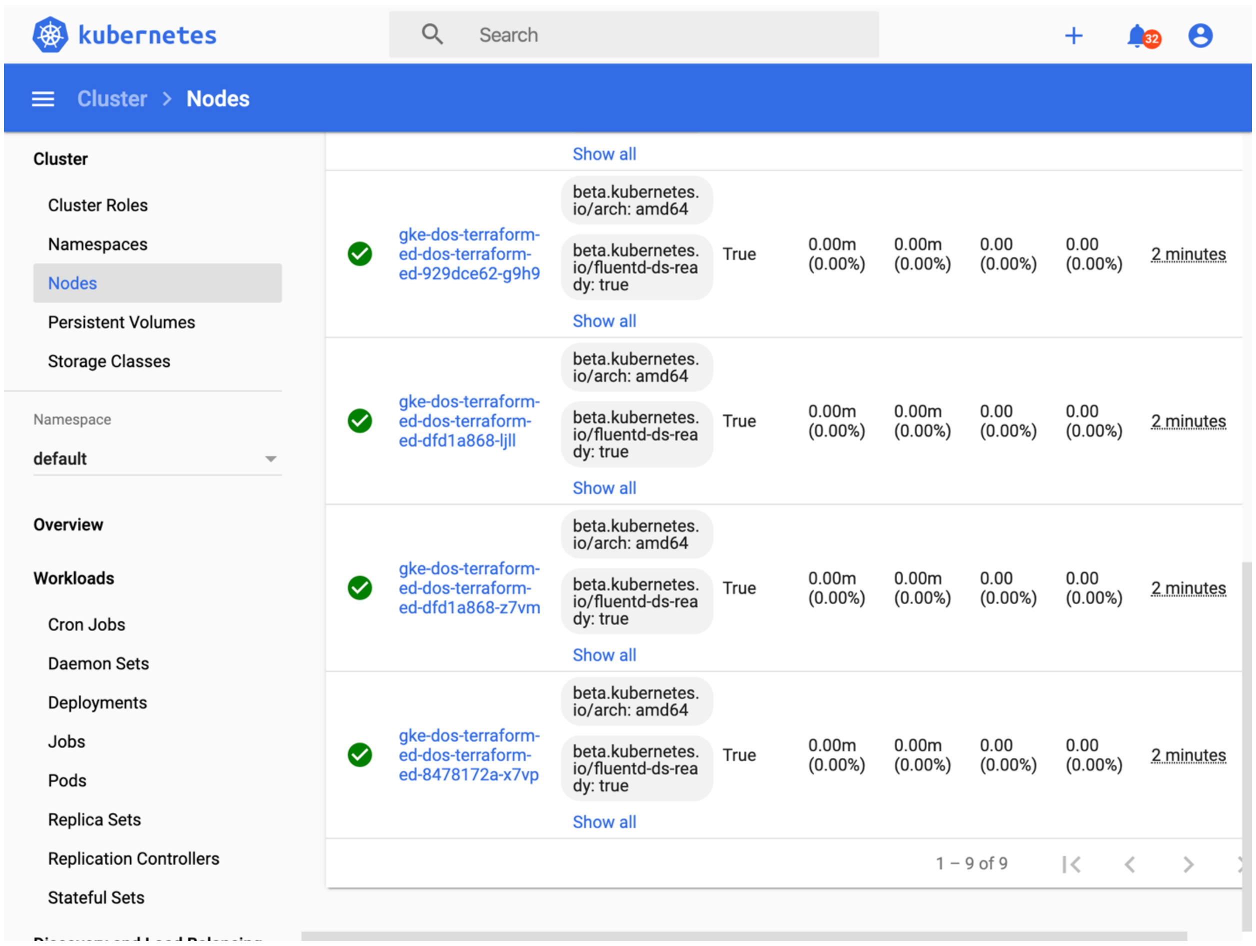
On the Dashboard UI, click *Nodes* on the left hand menu.

Notice there are 6 nodes in your cluster, even though [`gke_num_nodes` in your `gke.tf` file]() was set to 2. This is because a node pool was provisioned in each of the three zones within the region to provide high availability. To see the zones that the cluster deployed each node pool to, run the following in the `learn-terraform-provision-gke-cluster` directory.

```
$ gcloud container clusters describe                                        --
region us-central1 --format='default(locations)'

locations:

- us-central1-b
```

- us-central1-f

- us-central1-c



✓	gke-dos-terraform-ed-dos-terraform-ed-929dce62-g9h9	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/fluentd-ds-ready: true Show all	True	0.00m (0.00%)	0.00m (0.00%)	0.00 (0.00%)	0.00 (0.00%)	2 minutes
✓	gke-dos-terraform-ed-dos-terraform-ed-dfd1a868-ljll	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/fluentd-ds-ready: true Show all	True	0.00m (0.00%)	0.00m (0.00%)	0.00 (0.00%)	0.00 (0.00%)	2 minutes
✓	gke-dos-terraform-ed-dos-terraform-ed-dfd1a868-z7vm	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/fluentd-ds-ready: true Show all	True	0.00m (0.00%)	0.00m (0.00%)	0.00 (0.00%)	0.00 (0.00%)	2 minutes
✓	gke-dos-terraform-ed-dos-terraform-ed-8478172a-x7vp	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/fluentd-ds-ready: true Show all	True	0.00m (0.00%)	0.00m (0.00%)	0.00 (0.00%)	0.00 (0.00%)	2 minutes

1 – 9 of 9

Clean up your workspace

Congratulations, you have provisioned a GKE cluster with a separated node pool, configured `kubectl`, and deployed the Kubernetes dashboard.
If you'd like to learn how to manage your GKE cluster using the Terraform Kubernetes Provider, leave your cluster running and continue to the Kubernetes provider tutorial.
Note
This directory is **only** used to provision a GKE 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
```

Next steps

For more information on the GKE resource, please visit the Google Cloud provider documentation.
For steps on how to manage Kubernetes resources your GKE cluster or any other already created Kubernetes cluster, visit the Kubernetes provider tutorial.
For a more in-depth Kubernetes example, Deploy Consul and Vault on a Kubernetes Cluster using Run Triggers (this tutorial is GKE based