Deploy Consul and Vault on Kubernetes with run triggers

11min |

With Terraform Cloud and Terraform Enterprise, you can connect workspaces to each other with a feature called "run triggers". After a successful apply in a source workspace, a run trigger will automatically queue a run in the connected workspace. Run triggers are designed for workspaces that rely on information or infrastructure produced by other workspaces.

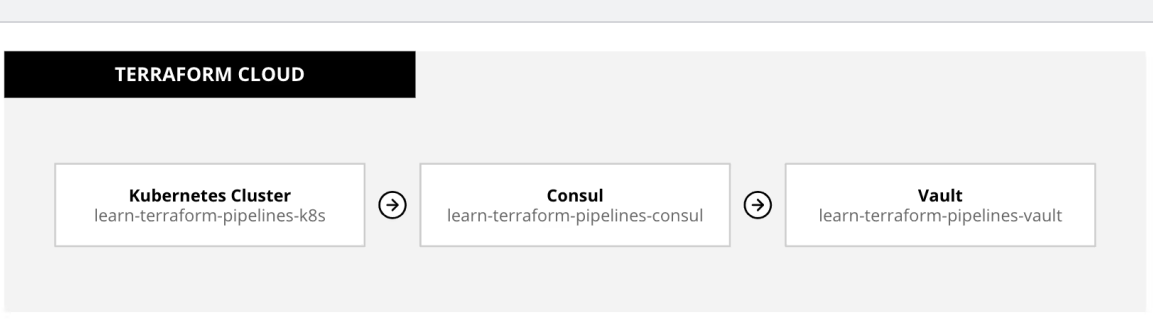
In this tutorial, you will accomplish three things using Terraform Cloud run triggers:

1. Deploy a Kubernetes cluster on Google Cloud.
2. Deploy Consul on the Kubernetes cluster using a Helm chart
3. Deploy Vault (configured to use a Consul backend) on the Kubernetes cluster using a Helm chart.

This tutorial highlights Terraform and Terraform Cloud (TFC) best practices for code management and modules.

The example configuration uses modules and organizes Terraform configuration for each resource (Kubernetes, Consul, and Vault) in separate repositories. First, you will create and configure TFC workspaces for each resource, then link them together using run triggers.

The configuration defines a Kubernetes cluster with 3 nodes and uses the Consul and Vault run triggers to deploy Consul and Vault.



Prerequisites

This tutorial assumes that you are familiar with the standard Terraform workflow, Terraform Cloud, run triggers and provisioning a Kubernetes cluster using Terraform.

If you are unfamiliar with any of these topics, reference their respective tutorials.

- **Terraform Workflow** — All Get Started tutorials
- **Terraform Cloud** — All Get Started with Terraform Cloud tutorials

- **Run Triggers** — Connect Workspaces with Run Triggers
- **Provision GKE cluster using Terraform** — Provision a GKE Cluster (Google Cloud)

For this tutorial, you will need:

1. a Terraform Cloud or Terraform Enterprise account
2. a Google Cloud account with access to **Compute Admin** and **Kubernetes Engine Admin**
3. a GitHub account

If you don't have your GCP credentials as a JSON or your credentials don't have access to **Compute Admin** and **Kubernetes Engine Admin**, reference the GCP Documentation to generate a new service account and with the right permissions.

If you are using a Google Cloud service account, your account must be assigned the **Service Account User** role.

Note

There may be some charges associated with running this configuration. Please reference the Google Cloud pricing guide for more details. Instructions to remove the infrastructure you create can be found at the end of this tutorial.

Create Kubernetes workspace

Fork the Learn Terraform Pipelines K8s repository.

Then, create a VCS-driven Terraform Cloud workspace connected to your forked repository. Terraform Cloud will confirm that the workspace configuration uploaded successfully.

Configure variables

After creating your workspace, Terraform Cloud prompts you to set the **google_project** variable. Set this to the ID of your Google Project, which you can find in your Google Cloud Platform console.

Click **Save variables**.

The configuration also contains two input variables, configured with defaults.

- **region** — GCP region to deploy clusters
  This defaults to `us-central1`. For a full list of GCP regions, refer to Google's Region and Zones documentation.
- **cluster_name** — Name of Kubernetes cluster
  This defaults to `tfc-pipelines`.

Terraform Cloud will use these variable values to configure and deploy your Kubernetes cluster. Terraform will create outputs for the Kubernetes credentials, which other workspaces can then consume. Review the output definitions in [`outputs.tf`]. (https://github.com/hashicorp/learn-terraform-pipelines-k8s/blob/main/outputs.tf)

Create Consul workspace

Fork the Learn Terraform Pipelines Consul repository.

The `main.tf` file contains providers for Kubernetes and Helm as well as the Terraform `tfe_outputs` data source to retrieve values from your Kubernetes workspace.

Create a Terraform Cloud workspace connected to your forked repository. Terraform Cloud will confirm that the configuration uploaded successfully.

Configure variables

Click on **Configure variables** then specify the variables required for this deployment.

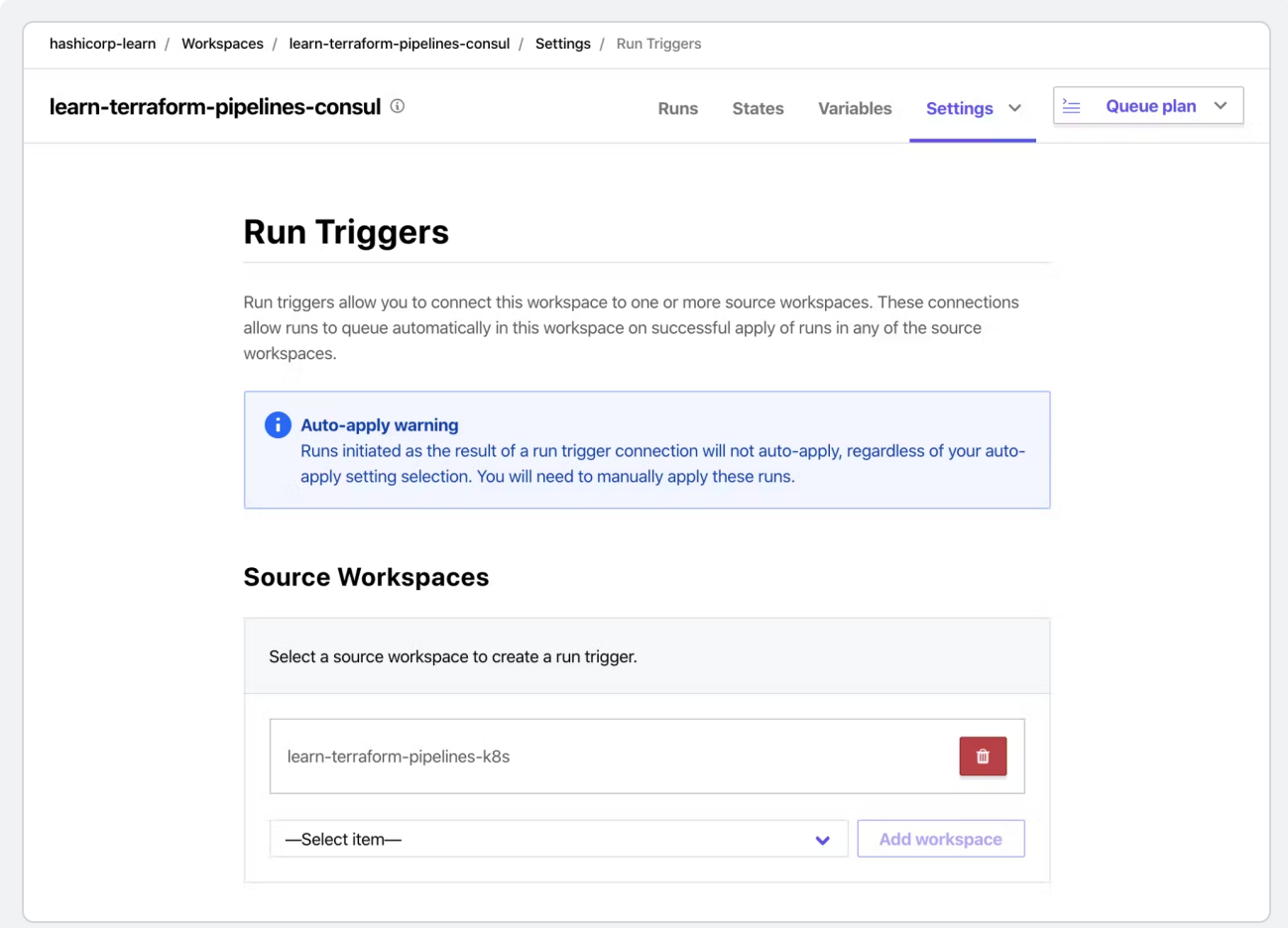
- **cluster_workspace** — Workspace that manages the Kubernetes cluster. If you did not customize the workspace name in the previous step, use `learn-terraform-pipelines-k8s`.
- **organization** - Terraform Cloud organization that contains your Kubernetes workspace.

Click **Save variables**, then click **Go to workspace overview** to view your workspace.

Enable run trigger

Click on **Settings** ,then select **Run Triggers**.

Under **Source Workspaces**, select your Kubernetes workspace (`learn-terraform-pipelines-k8s`) then click **Add Workspace**.



The run trigger will now kick off runs in this workspace after successful runs in the Kubernetes workspace. The configuration in this workspace consumes the Kubernetes credentials outputs from the Kubernetes workspace to authenticate to the Kubernetes and Helm provider.

Create Vault workspace

Fork the Learn Terraform Pipelines Vault repository.

The `main.tf` file contains providers for Kubernetes and Helm as well as the Terraform `tfe_outputs` data source to retrieve values from your Kubernetes and Consul workspaces.

Create a Terraform Cloud workspace connected to your forked repository. Terraform Cloud will confirm that the configuration uploaded successfully.

Configure variables

Click on **Configure variables**, then set the configuration's required variables.

- **consul_workspace** —Terraform Cloud Workspace for the Consul cluster. If you did not customize the workspace name in the previous step, use `learn-terraform-pipelines-consul`.
- **cluster_workspace** — Terraform Cloud Workspace for the Kubernetes cluster. If you did not customize your workspace name, this is `learn-terraform-pipelines-k8s`.
- **organization** — Organization of workspace that created the Kubernetes cluster Set this to your Terraform Cloud Organization.

Click **Save variables**, then click **Go to workspace overview** to view your workspace.

Enable run trigger

Click on **Settings** then select **Run Triggers**.

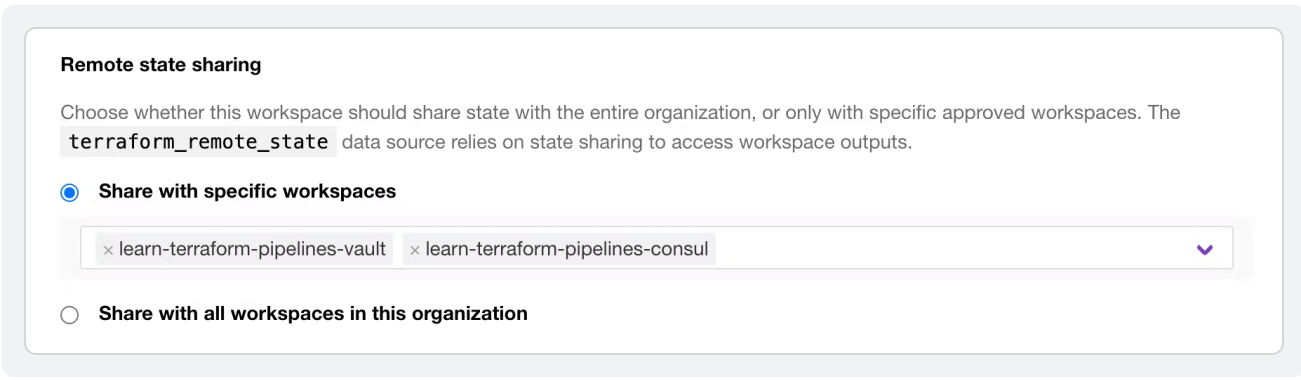
Under **Source Workspaces**, select your Consul workspace (`learn-terraform-pipelines-consul`) then click **Add Workspace**.

The run trigger will now kick off runs in this workspace after successful runs in the Consul workspace. The configuration in this workspace consumes the Kubernetes credentials outputs from the Kubernetes workspace to authenticate to the Kubernetes and Helm provider. It also retrieves the Helm release name and Kubernetes namespace from the Consul workspace.

Allow remote state access

Terraform Cloud protects your state file by encrypting it at rest and automatically restricting access to it from other workspaces. You must explicitly enable state sharing between workspaces.

Navigate to the `learn-terraform-pipelines-k8s` workspace's **Settings**. Under **General Settings**, scroll to the **Remote state sharing** section.



Under **Search for workspaces to share with** select `learn-terraform-pipelines-consul` and `learn-terraform-pipelines-vault` from the drop-down menu. Then click **Save settings**.

Now allow the Vault workspace to access the Consul workspace's state. Navigate to the `learn-terraform-pipelines-consul` workspace's **General Settings**, then scroll to the **Remote state sharing** section. Under **Search for workspaces to share with** select `learn-terraform-pipelines-vault` from the dropdown menu, then click **Save settings**.

Create variable set

The workspaces in this tutorial need your GCP credentials to authenticate the provider. Instead of creating the same variable with your credentials in each workspace, you can use a variable set that applies to multiple workspaces.

Open your Terraform Cloud organization settings, click **Variable sets**, then click **Create variable set**. Name your variable set `Google Credentials`.

Under **Workspaces**, choose the option to **Apply to specific workspaces**. In the dropdown below, add the this tutorial's three workspaces. If you have not changed the names, they are:

- learn-terraform-pipelines-k8s
- learn-terraform-pipelines-consul
- learn-terraform-pipelines-vault

Use `jq` to remove the newlines from your JSON credentials file.

```
$ cat <key file>.json | jq -c
```

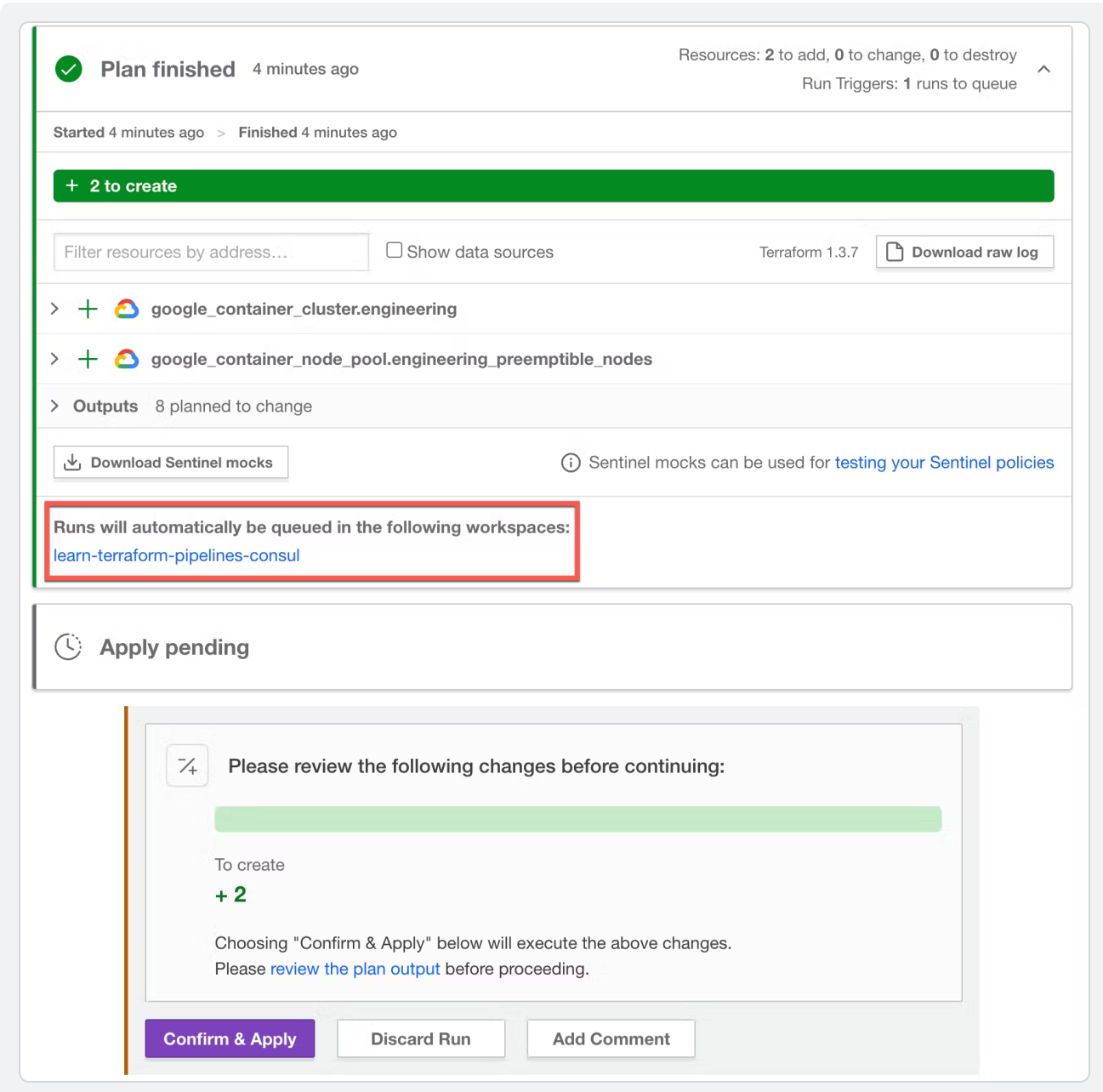
Next, add your credentials to the variable set by clicking **+ Add variable**. Create a variable named **GOOGLE_CREDENTIALS** and set the value to your formatted GCP JSON credentials. Select **Environment variable** as the category, mark the variable as **Sensitive**, and click **Save variable**.

Click **Create variable set**.

Deploy Kubernetes cluster

Now that you have successfully configured all three workspaces (Kubernetes, Consul, and Vault), you can deploy your Kubernetes cluster.

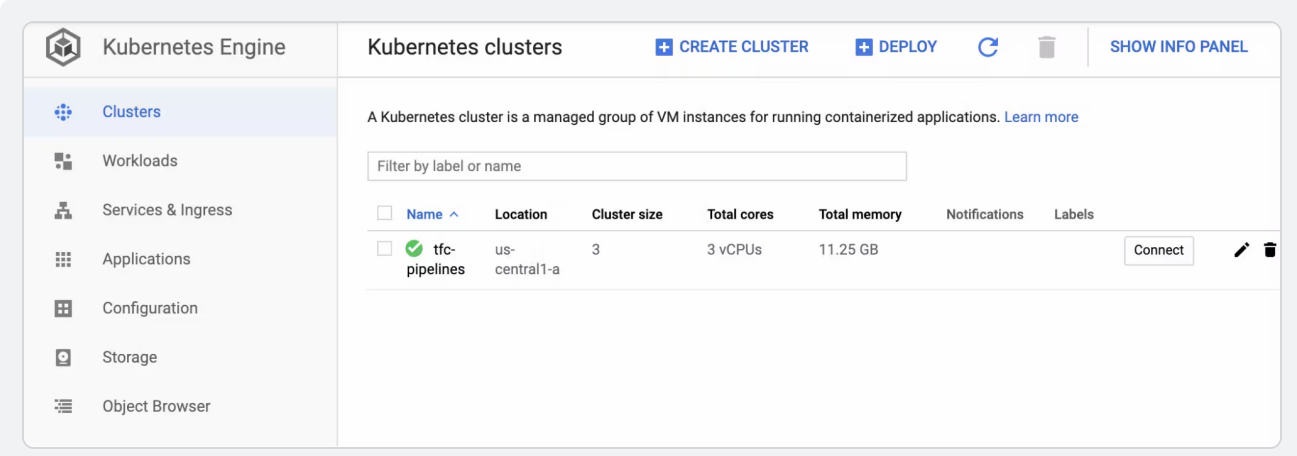
Select your `learn-terraform-pipelines-k8s` workspace, click the **Actions** dropdown, then click **Start new run**.



Notice Terraform Cloud automatically queues a plan for the `learn-terraform-pipelines-consul` workspace after this run completes.

Click **Confirm & Apply** to apply this configuration. This process should take about 10 minutes to complete.

Once the apply completes, verify your Kubernetes cluster by visiting the GKE Console Page. Your Kubernetes cluster should only have three nodes and no workloads.



Deploy Consul

Navigate to the `learn-terraform-pipelines-consul` workspace, view the run plan, and click **Confirm & Apply** to deploy Consul onto your cluster using the Helm provider. The plan retrieves the Kubernetes cluster authentication information from the Kubernetes workspace to configure both the Kubernetes and Helm provider.

This process will take about 2 minutes to complete.

Notice that Terraform Cloud automatically queues a plan for the `learn-terraform-pipelines-vault` workspace after this apply completes.

Deploy Vault

Navigate to the `learn-terraform-pipelines-vault` workspace, view the run plan, then click **Confirm & Apply** to deploy Vault onto your cluster using the Helm provider and configure it to use Consul as the backend. The plan retrieves the Kubernetes namespace from the Consul workspace's remote state and deploys Vault to the same workspace.

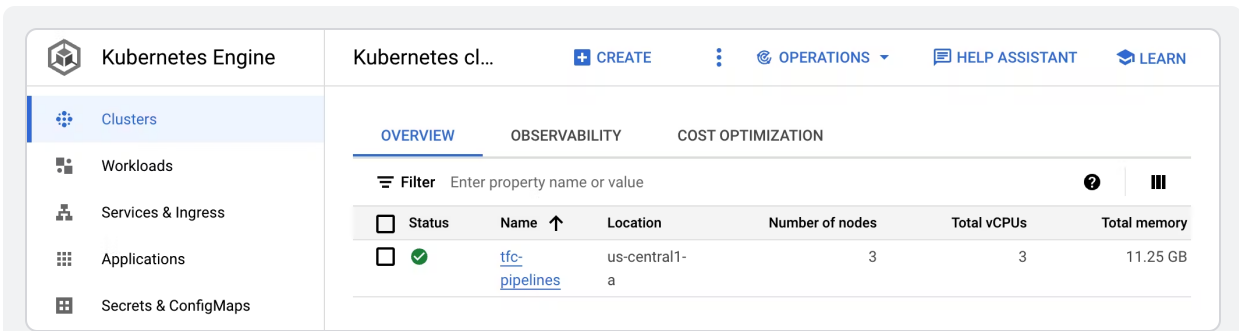
This process will take about 2 minutes to complete.

Verify Consul and Vault deployments

Once the apply completes, verify by visiting the GKE Console. Your Kubernetes cluster should also have 3 nodes. Navigate to **Workloads** to confirm the Consul and Vault deployments.

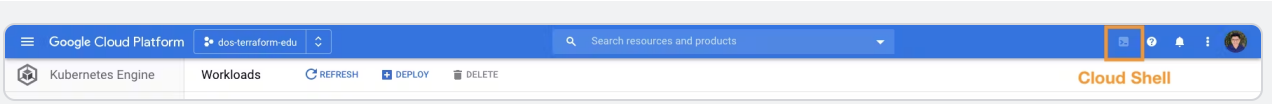
Notice that the Vault pods have warnings because Vault is sealed. You will have the option to unseal Vault and resolve the warnings once you enable port forwarding.

Kubernetes Cluster | Kubernetes Workload



Verify that Consul and Vault have both been deployed by viewing their respective dashboard.

First, activate your Cloud Shell (button on top right).



Run the following command in Cloud Shell to configure access to your Kubernetes cluster. If you did not use the default values, replace `tfc-pipelines` with your Kubernetes cluster name, `us-central1-a` with your zone.

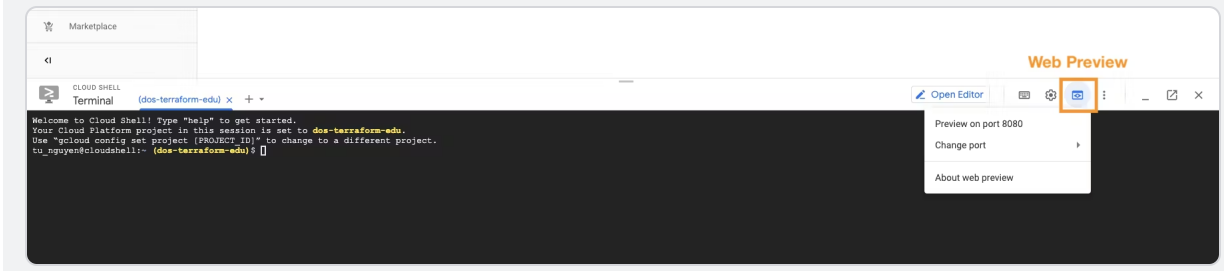
```
$ gcloud container clusters get-credentials tfc-pipelines --zone us-central1-a
```

View Consul UI | View Vault UI

Run the following command — it forwards port `:8500` (Consul UI) to port `:8080`, allowing you to access it in the Web Preview.

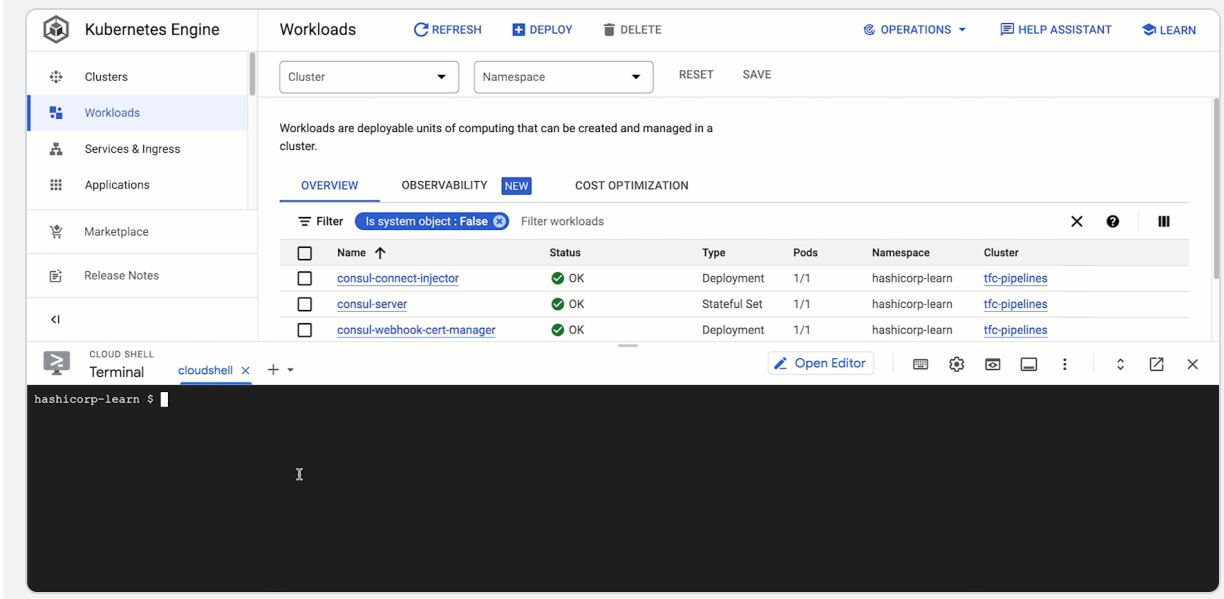
```
$ kubectl port-forward -n hashicorp-learn consul-server-0 8080:8500
```

After you run this command, open your Web Preview to port `:8080` to view the Consul UI.



Note

The Consul UI does not show Vault in the list of services because its `service_registration` stanza in the Helm chart defaults to Kubernetes. However, Vault is still configured to use Consul as a backend.



Congratulations — you have successfully completed the tutorial and applied some Terraform Cloud best practices. By keeping your infrastructure configuration modular and integrating workspaces together using run triggers, your Terraform configuration becomes extensible and easier to understand.

Clean up resources

To clean up the resources and destroy the infrastructure you have provisioned in this tutorial, go to each workspace in the reverse order you created them in, queue a destroy plan, and apply it. Then, delete the workspace from Terraform Cloud. Destroy and delete your workspaces in the following order:

1. Vault workspace
2. Consul workspace
3. Kubernetes workspace

For a more detailed tutorial on destroying resources on Terraform Cloud, reference the Clean up Cloud Resources tutorial.

Next steps

To watch a video of a demo similar to this tutorial, reference the Infrastructure Pipelines with Terraform Cloud webinar.

To learn how to get started with Consul Service Mesh, visit the Getting Started with Consul Service Mesh Learn track.

To learn how to leverage Vault features on Kubernetes, visit the Kubernetes Vault tutorials.

To learn how to use the TFE provider to deploy the Terraform Cloud resources used in this guide, visit the Use the TFE Provider to Manage Terraform Cloud Workspaces tutorial.

Was this tutorial helpful? Yes No

Previous

Kubernetes provider

Next

TFE provider

This tutorial also appears in:

22 tutorials

Vault on Kubernetes

Vault secures, stores, and tightly controls access to passwords, certificates, and other secrets in modern computing. Here...

13 tutorials

HashiCorp product integrations

Vault can manage secrets associated with other HashiCorp products.

16 tutorials

Use Cases for Terraform

Use Terraform to perform common operations with other technologies, including Consul, Vault, Packer, and...

9 tutorials

Interact with Security tools

Use Terraform to interact with security tooling like HashiCorp Vault and Boundary. Manage Vault and Vault Enterprise....

16 tutorials

HashiCorp Products - Better Together

Use Terraform with other Hashicorp products including Vault, Boundary, Consul, Packer, and Hashicorp Cloud Platform.

27 tutorials

Collaborate using Terraform Cloud

Collaborate on infrastructure with Terraform Cloud. Follow these tutorials to migrate state from local storage and take ...

11 tutorials

Automate Terraform

Automate Terraform with Terraform Cloud and integrate it with third-party CI/CD tools such as GitHub Actions and CircleCI.

Give Feedback