Terraform Editions

As our organization adopts infrastructure as code (IaC), we will encounter increasingly complex technical and collaboration challenges. There are 3 Terraform editions designed to help us solve them

* **Terraform Open Source:** Terraform open source is a free, downloadable tool that we interact with on the command line. It lets us provision infrastructure on any cloud provider and manages configuration, plugins, infrastructure and state.
* **Terraform Cloud:** Terraform Cloud is a SaaS application that runs Terraform in a stable, remote environment and securely stores state and secrets. It's user interface that helps us understand our Terraform operations and resources, allows us to define role-based access controls and offers a private registry for sharing modules and providers. Terraform Cloud also integrates with the Terraform CLI and connects to common version control systems (VCS) like GitHub, GitLab, and Bitbucket. When we connect a Terraform Cloud workspace to a VCS repository, new commits and changes can automatically trigger Terraform plans.
* **Terraform Enterprise:** Terraform Enterprise allows us to set up a self-hosted distribution of Terraform Cloud. It offers customizable resource limits and is ideal for organizations with strict security and compliance requirements.

Graphical user interface, application

Description automatically generated

Terraform Enterprise vs Terraform Cloud

* **Deployment Models**: Terraform Enterprise can be deployed on-premises or on a private cloud, while Terraform Cloud is a fully hosted service that runs on HashiCorp's infrastructure.
* **Features**: Terraform Enterprise includes a number of additional features that are not available in Terraform Cloud. These include private module registry, access controls, audit logging, policy enforcement, and collaboration tools.
* **Pricing**: Terraform Enterprise is a licensed product that requires a paid subscription, while Terraform Cloud has a free tier with limited features and a paid tier that includes additional features and support.
* **Support**: Terraform Enterprise customers have access to enterprise-level support, including 24/7 support, response time guarantees, and technical account managers. Terraform Cloud customers have access to community support and email support, with paid plans including additional support options.
* **Security**: Both Terraform Enterprise and Terraform Cloud have built-in security features, but Terraform Enterprise provides more options for customizing security policies and integrating with existing security tools and processes.

Overall, Terraform Enterprise is designed for larger enterprises and organizations with more complex infrastructure needs and security requirements, while Terraform Cloud is a more streamlined and affordable option for smaller organizations or teams who want to get started with Terraform.

Terraform Cloud

[Terraform Cloud](https://cloud.hashicorp.com/products/terraform) provides an environment to run terraform in a consistent and reliable environment instead of on your local machine.

 It securely stores state and secret data, and can connect to version control systems so that you can develop your infrastructure using a workflow similar to application development.

Terraform Cloud also has a private registry for sharing your organization's Terraform modules and providers.

Paid features include access controls for approving changes to infrastructure, detailed policy controls for governing the contents of Terraform configurations, cost estimates for runs, and more.

A picture containing text, diagram, software, screenshot

Description automatically generated

In Terraform Cloud, your resources are organized by workspaces, which contain your resource definitions, environment and input variables, and state files. A Terraform operation occurs within a workspace, and Terraform uses the configuration and state for that workspace to modify your infrastructure.

Terraform Cloud supports three workflows for your Terraform runs:

1. The CLI-driven workflow, which uses Terraform's standard CLI tools to execute runs in Terraform Cloud.
2. The UI/Version Control System(VCS)-driven workflow, in which changes pushed to version control repositories trigger runs in the associated workspace.
3. The API-driven workflow, which allows you to create tooling to interact with the Terraform Cloud API programmatically.

**Create an Account**

1. Visit [https://app.terraform.io/signup/account](https://app.terraform.io/public/signup/account?utm_source=learn) and follow the prompts to create a free Terraform Cloud account.
2. Confirm email address and login.
3. Select from Scratch.

A screenshot of a computer

Description automatically generated with low confidence

1. Create an Organization

A screenshot of a computer

Description automatically generated with medium confidence

1. **Login in to Terraform Cloud from the CLI**
   1. In order to authenticate with Terraform Cloud, run the **terraform login** subcommand. Respond yes to the prompt to confirm that you want to authenticate.
   2. A browser window will automatically open to the Terraform Cloud login screen. Enter a token name in the web UI, or leave the default name, terraform login.
   3. Click **Create API token** to generate the authentication token.
   4. **Save a copy of the token in a secure location**.
   5. Add the token to the CLI (Paste the token)
2. **Create a Credential Variable Set.**
   1. Select Organization 🡪 Settings 🡪 Variable sets 🡪 Create variable set

A screenshot of a computer

Description automatically generated with medium confidence

* 1. Name = AzureCredentials, Variable set scope = **Apply globally**

ARM\_CLIENT\_ID="625a9b41-3942-4da7-81c7-8bc04359c292"

ARM\_CLIENT\_SECRET="T.I8Q~mv7mXtdqtvmsvoiCJ4yWJBVyMMF8oO-ckz"

ARM\_TENANT\_ID="9a2b4fd4-c9d2-4e05-82d5-63405d8e2a1f"

ARM\_SUBSCRIPTION\_ID=**"<SUBSCRIPTION\_ID>"**

A screenshot of a computer

Description automatically generated with medium confidence

1. **Update main.tf on your location machine.**

terraform {

   cloud {

**organization**= "sandeepsoniorg"

      workspaces {

      name = "demo-workspace"

    }

}

   required\_providers {

**azurerm** = {

      source = "hashicorp/azurerm"

      version = "3.11.0"

    }

  }

}

provider "azurerm" {

    features {}

}

#Creating a Resource Group

resource "azurerm\_resource\_group" "rg" {

  name     = "Terraform-rg"

  location = "east us"

}

1. Create a Workspace 🡪 Execute the command.

**terraform init**

Note: As part of the initialization process, Terraform created the new **demo-workspace** workspace in your Terraform Cloud organization, configured for CLI-driven runs.

A screenshot of a computer screen

Description automatically generated with low confidence

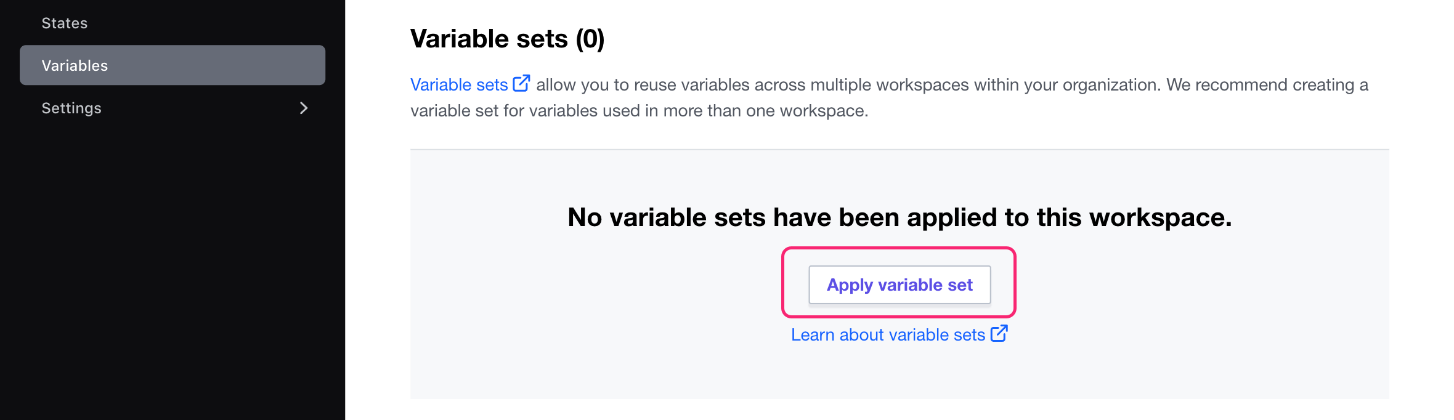
Note:

Every Terraform Cloud workspace belongs to a project, which is a group of workspaces. When you create your workspace using this automated CLI workflow, Terraform Cloud adds the workspace to your organization's **Default** project.

Projects help you organize your workspaces into groups, making it easier to find workspaces in large Terraform Cloud organizations.

1. **Assign variable set to workspace**

Navigate to **demo-workspace** 🡪 **Variables** 🡪 click on **+ Apply variable set**



1. **Add Terraform Variable**
2. Add the following to main.tf

variable "RGName" {

    type = string

}

resource "azurerm\_resource\_group" "rg" {

  name     =  var.RGName

  location = "east us"

}

1. Execute the following commands

* terraform plan
* terraform apply

1. You can visit Runs menu to know the status.

