Terraform Fundamentals

35 minutes1 Credit

**GSP156**

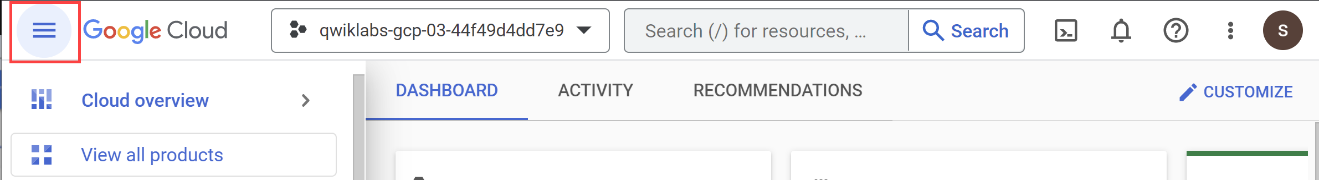
**Overview**

Terraform enables you to safely and predictably create, change, and improve infrastructure. It is an open source tool that codifies APIs into declarative configuration files that can be shared among co-workers, treated as code, edited, reviewed, and versioned.

What you'll learn

In this lab, you will learn how to perform the following tasks:

* Get started with Terraform in Google Cloud.
* Install Terraform from installation binaries.
* Create a VM instance infrastructure using Terraform.

Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

1. Click **Activate Cloud Shell** Activate Cloud Shell icon at the top of the Google Cloud console.

When you are connected, you are already authenticated, and the project is set to your **PROJECT\_ID**. The output contains a line that declares the **PROJECT\_ID** for this session:

Your Cloud Platform project in this session is set to YOUR\_PROJECT\_ID

gcloud is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

1. (Optional) You can list the active account name with this command:

gcloud auth list

1. Click **Authorize**.
2. Your output should now look like this:

**Output:**

ACTIVE: \*

ACCOUNT: student-01-xxxxxxxxxxxx@qwiklabs.net

To set the active account, run:

$ gcloud config set account `ACCOUNT`

1. (Optional) You can list the project ID with this command:

gcloud config list project

**Output:**

[core]

project = <project\_ID>

**Example output:**

[core]

project = qwiklabs-gcp-44776a13dea667a6

**Note:**For full documentation of gcloud, in Google Cloud, refer to [the gcloud CLI overview guide](https://cloud.google.com/sdk/gcloud).

**What is Terraform?**

Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing, popular service providers and custom in-house solutions.

Configuration files describe to Terraform the components needed to run a single application or your entire data center. Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure. As the configuration changes, Terraform can determine what changed and create incremental execution plans that can be applied.

The infrastructure Terraform can manage includes both low-level components such as compute instances, storage, and networking, and high-level components such as DNS entries and SaaS features.

Key features

**Infrastructure as code**

Infrastructure is described using a high-level configuration syntax. This allows a blueprint of your data center to be versioned and treated as you would any other code. Additionally, infrastructure can be shared and re-used.

**Execution plans**

Terraform has a planning step in which it generates an execution plan. The execution plan shows what Terraform will do when you execute the apply command. This lets you avoid any surprises when Terraform manipulates infrastructure.

**Resource graph**

Terraform builds a graph of all your resources and parallelizes the creation and modification of any non-dependent resources. Because of this, Terraform builds infrastructure as efficiently as possible, and operators get insight into dependencies in their infrastructure.

**Change automation**

Complex changesets can be applied to your infrastructure with minimal human interaction. With the previously mentioned execution plan and resource graph, you know exactly what Terraform will change and in what order, which helps you avoid many possible human errors.

**Task 1. Verifying Terraform installation**

Terraform comes pre-installed in Cloud Shell.

* Open a new Cloud Shell tab, and verify that Terraform is available:

terraform

The resulting help output should be similar to this:

Usage: terraform [--version] [--help] [args]

The available commands for execution are listed below.

The most common, useful commands are shown first, followed by

less common or more advanced commands. If you're just getting

started with Terraform, stick with the common commands. For the

other commands, please read the help and docs before usage.

Common commands:

apply Builds or changes infrastructure

console Interactive console for Terraform interpolations

destroy Destroy Terraform-managed infrastructure

env Workspace management

fmt Rewrites config files to canonical format

get Download and install modules for the configuration

graph Create a visual graph of Terraform resources

import Import existing infrastructure into Terraform

init Initialize a Terraform working directory

output Read an output from a state file

plan Generate and show an execution plan

providers Prints a tree of the providers used in the configuration

push Upload this Terraform module to Atlas to run

refresh Update local state file against real resources

show Inspect Terraform state or plan

taint Manually mark a resource for recreation

untaint Manually unmark a resource as tainted

validate Validates the Terraform files

version Prints the Terraform version

workspace Workspace management

All other commands:

debug Debug output management (experimental)

force-unlock Manually unlock the terraform state

state Advanced state management

**Task 2. Build infrastructure**

With Terraform installed, you can immediately start creating some infrastructure.

Configuration

The set of files used to describe infrastructure in Terraform is simply known as a Terraform configuration. In this section, you will write your first configuration to launch a single VM instance. The format of the configuration files can be found in the [Terraform Language Documentation](https://www.terraform.io/docs/configuration/index.html). We recommend using JSON for creating configuration files.

1. In Cloud Shell, create an empty configuration file named instance.tf with the following command:

touch instance.tf

1. Click **Open Editor** on the Cloud Shell toolbar.  
   To switch between Cloud Shell and the code editor, click **Open Editor** or **Open Terminal** as required, or click **Open in a new window** to leave the Editor open in a separate tab.
2. Click the instance.tf file and add the following content in it, replacing <PROJECT\_ID> with your Google Cloud project ID:

resource "google\_compute\_instance" "terraform" {

project = "<PROJECT\_ID>"

name = "terraform"

machine\_type = "n1-standard-1"

zone = "us-west1-c"

boot\_disk {

initialize\_params {

image = "debian-cloud/debian-11"

}

}

network\_interface {

network = "default"

access\_config {

}

}

}

This is a complete configuration that Terraform is ready to apply. The general structure should be intuitive and straightforward.

The "resource" block in the instance.tf file defines a resource that exists within the infrastructure. A resource might be a physical component such as an VM instance.

The resource block has two strings before opening the block: the **resource type** and the **resource name**. For this lab, the resource type is google\_compute\_instance and the name is terraform. The prefix of the type maps to the provider: google\_compute\_instance automatically tells Terraform that it is managed by the Google provider.

Within the resource block itself is the configuration needed for the resource.

1. In Cloud Shell, verify that your new file has been added and that there are no other \*.tf files in your directory, because Terraform loads all of them:

ls

Initialization

The first command to run for a new configuration—or after checking out an existing configuration from version control—is terraform init. This will initialize various local settings and data that will be used by subsequent commands.

Terraform uses a plugin-based architecture to support the numerous infrastructure and service providers available. Each "provider" is its own encapsulated binary that is distributed separately from Terraform itself. The terraform init command will automatically download and install any provider binary for the providers to use within the configuration, which in this case is just the Google provider.

1. Download and install the provider binary:

terraform init

The Google provider plugin is downloaded and installed in a subdirectory of the current working directory, along with various other book keeping files. You will see an "Initializing provider plugins" message. Terraform knows that you're running from a Google project, and it is getting Google resources.

Installing hashicorp/google v3.77.0...

**Note:**Your version number may be higher.

The output specifies which version of the plugin is being installed and suggests that you specify this version in future configuration files to ensure that terraform init will install a compatible version.

1. Create an execution plan:

terraform plan

Terraform performs a refresh, unless explicitly disabled, and then determines what actions are necessary to achieve the desired state specified in the configuration files. This command is a convenient way to check whether the execution plan for a set of changes matches your expectations without making any changes to real resources or to the state. For example, you might run this command before committing a change to version control, to create confidence that it will behave as expected.

**Note:** The optional -out argument can be used to save the generated plan to a file for later execution with terraform apply.

Apply changes

1. In the same directory as the instance.tf file you created, run this command:

terraform apply

This output shows the Execution Plan, which describes the actions Terraform will take in order to change real infrastructure to match the configuration. The output format is similar to the diff format generated by tools like Git.

There is a + next to google\_compute\_instance.terraform, which means that Terraform will create this resource. Following that are the attributes that will be set. When the value displayed is <computed>, it means that the value won't be known until the resource is created.

**Example output:**

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:

# google\_compute\_instance.default will be created

+ resource "google\_compute\_instance" "default" {

+ can\_ip\_forward = false

+ cpu\_platform = (known after apply)

+ deletion\_protection = false

+ guest\_accelerator = (known after apply)

+ id = (known after apply)

+ instance\_id = (known after apply)

+ label\_fingerprint = (known after apply)

+ machine\_type = "n1-standard-1"

+ metadata\_fingerprint = (known after apply)

+ name = "terraform"

+ project = "qwiklabs-gcp-42390cc9da8a4c4b"

+ self\_link = (known after apply)

+ tags\_fingerprint = (known after apply)

+ zone = "us-west1-c"

+ boot\_disk {

+ auto\_delete = true

+ device\_name = (known after apply)

+ disk\_encryption\_key\_sha256 = (known after apply)

+ kms\_key\_self\_link = (known after apply)

+ source = (known after apply)

+ initialize\_params {

+ image = "debian-cloud/debian-11"

+ labels = (known after apply)

+ size = (known after apply)

+ type = (known after apply)

}

}

+ network\_interface {

+ address = (known after apply)

+ name = (known after apply)

+ network = "default"

+ network\_ip = (known after apply)

+ subnetwork = (known after apply)

+ subnetwork\_project = (known after apply)

+ access\_config {

+ assigned\_nat\_ip = (known after apply)

+ nat\_ip = (known after apply)

+ network\_tier = (known after apply)

}

}

+ scheduling {

+ automatic\_restart = (known after apply)

+ on\_host\_maintenance = (known after apply)

+ preemptible = (known after apply)

+ node\_affinities {

+ key = (known after apply)

+ operator = (known after apply)

+ values = (known after apply)

}

}

}

Plan: 1 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:

If the plan was created successfully, Terraform will now pause and wait for approval before proceeding. In a production environment, if anything in the Execution Plan seems incorrect or dangerous, it's safe to cancel here. No changes have been made to your infrastructure.

1. For this case the plan looks acceptable, so type yes at the confirmation prompt to proceed.  
   Executing the plan will take a few minutes because Terraform waits for the VM instance to become available.

After this, Terraform is all done!

**Test completed task**

Click **Check my progress** to verify your performed task. If you have completed the task successfully, you will receive an assessment score.

Create a VM instance in us-west1-c zone with Terraform.

Check my progress

1. In the Google Cloud Console, on the **Navigation menu**, click **Compute Engine** > **VM instances**. The **VM instances** page opens and you'll see the VM instance you just created in the **VM instances** list.

Terraform has written some data into the terraform.tfstate file. This state file is extremely important: it keeps track of the IDs of created resources so that Terraform knows what it is managing.

1. In Cloud Shell, inspect the current state:

terraform show

**Example output:**

# google\_compute\_instance.default:

resource "google\_compute\_instance" "default" {

can\_ip\_forward = false

cpu\_platform = "Intel Haswell"

deletion\_protection = false

guest\_accelerator = []

id = "terraform"

instance\_id = "3408292216444307052"

label\_fingerprint = "42WmSpB8rSM="

machine\_type = "n1-standard-1"

metadata\_fingerprint = "s6I5s2tjfKw="

name = "terraform"

project = "qwiklabs-gcp-42390cc9da8a4c4b"

self\_link = "https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-42390cc9da8a4c4b/zones/us-central1-a/instances/terraform"

tags\_fingerprint = "42WmSpB8rSM="

zone = "us-west1-c"

boot\_disk {

auto\_delete = true

device\_name = "persistent-disk-0"

source = "https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-42390cc9da8a4c4b/zones/us-central1-a/disks/terraform"

....

You can see that by creating this resource, you've also gathered a lot of information about it. These values can be referenced to configure additional resources or outputs.

Congratulations! You've built your first infrastructure with Terraform. You've seen the configuration syntax and an example of a basic execution plan and understand the state file.

**Task 3. Test your understanding**

The following multiple choice questions should reinforce your understanding of this lab's concepts. Answer them to the best of your abilities.

Terraform enables you to safely and predictably create, change, and improve infrastructure.

True

False

With Terraform, you can create your own custom provider plugins.

True

False

**Congratulations**

You installed Terraform from installation binaries and then used it to create a VM infrastructure.

Finish your quest

This self-paced lab is part of the [Managing Cloud Infrastructure with Terraform](https://google.qwiklabs.com/quests/44) and [Automating Infrastructure on Google Cloud with Terraform](https://google.qwiklabs.com/quests/159) quests. A quest is a series of related labs that form a learning path. Completing a quest earns you a badge to recognize your achievement. You can make your badge or badges public and link to them in your online resume or social media account. Enroll in any quest that contains this lab and get immediate completion credit. See the [Google Cloud Skills Boost catalog](http://cloudskillsboost.google/catalog) to see all available quests.

Take your next lab

Continue your quest with [Infrastructure as Code with Terraform](https://www.cloudskillsboost.google/catalog_lab/3362). You can also check out these Google Cloud Skills labs:

* [Custom Providers with Terraform](https://google.qwiklabs.com/catalog_lab/1014)
* [HTTPS Content-Based Load Balancer with Terraform](https://google.qwiklabs.com/catalog_lab/1011)

Next steps/Learn More

* [Hashicorp](https://console.cloud.google.com/marketplace/browse?q=Hashicorp&utm_source=Hashicorp&utm_medium=qwiklabs&utm_campaign=Qwiklabs+to+Marketplace) on the Google Cloud Marketplace!
* [Terraform Community](https://www.terraform.io/community.html)
* [Terraform Google Examples](https://github.com/GoogleCloudPlatform/terraform-google-examples)

Google Cloud training and certification

...helps you make the most of Google Cloud technologies. [Our classes](https://cloud.google.com/training/courses) include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. [Certifications](https://cloud.google.com/certification/) help you validate and prove your skill and expertise in Google Cloud technologies.

**Manual Last Updated July 13, 2022**

**Lab Last Tested July 13, 2022**

Copyright 2023 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are associated.