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.

Verifying Terraform installation

Terraform comes pre-installed in Cloud Shell.

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

```
terraform
content_copy
```

The resulting help output should be similar to this:

```
Usage: terraform [--version] [--help] <command> [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
```
content_copy

Update Terraform

The example code you will use in this lab requires **Terraform** versions **0.13.0** and higher.

First, ensure you are using a sufficient version of **Terraform** by downloading **0.13.0**.

```
wget https://releases.hashicorp.com/terraform/0.13.0/terraform_0.13.0_linux_amd64.zip
```
content_copy

Unzip the downloaded executable:

```
unzip terraform_0.13.0_linux_amd64.zip

content_copy
```

Move the executable into your local **bin** folder:

```
sudo mv terraform /usr/local/bin/

content_copy
```

Run this command to ensure your Terraform version is 0.13.0:

```
terraform -v

content_copy
```

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 is documented here. 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

content_copy
```

2.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.

3.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-central1-a"
  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-9"
    }
  }
  network_interface {
    network = "default"
    access_config {
    }
  }
}
content_copy
```

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.

4.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
content_copy
```

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
content_copy
```

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...
content_copy
```

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.

2.Create an execution plan:

```
terraform plan
content_copy
```

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 be 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
content_copy
```

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-central1-a"

      + 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-9"
              + 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:
content_copy
```

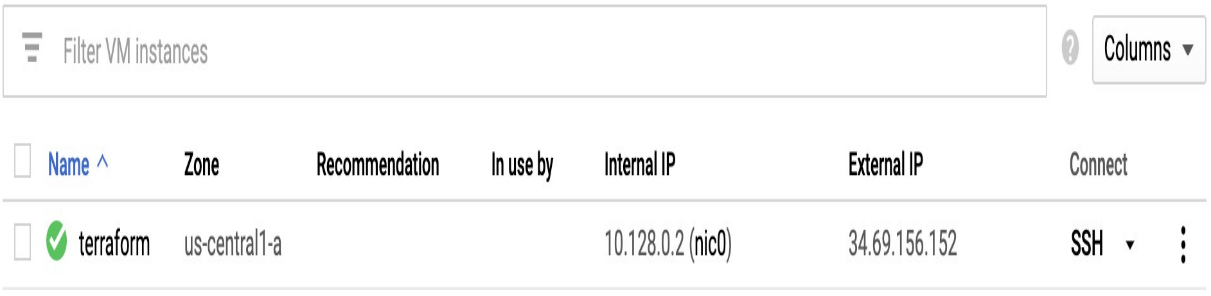
If the plan was created successfully, Terraform will now pause and wait for approval before proceeding. In a production environment, if anything in the Execution Plan seems incorrect or dangerous, it's safe to cancel here. No changes have been made to your infrastructure.

2.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.

3.In the Google Cloud Console, on the **Navigation menu**, click **Compute Engine** > **VM instances** to see the created VM instance.



	Name ^	Zone	Recommendation	In use by	Internal IP	External IP	Connect
	✓ terraform	us-central1-a			10.128.0.2 (nic0)	34.69.156.152	SSH ▾

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.

4.In Cloud Shell, inspect the current state:

```
terraform show
```

content_copy

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-central1-a"
    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"
....content_copy
```

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.

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.

True

ue

Congratulations

Finish your quest

This self-paced lab is part of the Managing Cloud Infrastructure with Terraform and Automating Infrastructure on Google Cloud with Terraform quests. A quest is a series of related labs that form a learning path. Completing this quest earns you the badge above to recognize your achievement. You can make your badge public and link to it in your online resume or social media account. Enroll in the Managing Cloud Infrastructure with Terraform quest or the Automating Infrastructure on Google Cloud with Terraform skill badge quest and get immediate completion credit for taking this lab. See other available Qwiklabs Quests.

Take your next lab

Continue your quest with Using a NAT Gateway with Kubernetes Engine or Infrastructure as Code with Terraform. You can also check out these suggestions:

- Custom Providers with Terraform
- HTTPS Content-Based Load Balancer with Terraform

Next steps/Learn More

•Hashicorp on the Google Cloud Marketplace!
•Terraform Community
•Terraform Google Examples

Google Cloud Training & Certification

...helps you make the most of Google Cloud technologies. Our classes 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 help you validate and prove your skill and expertise in Google Cloud technologies.

Manual Last Updated August 28, 2021
Lab Last Tested July 29, 2021