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.
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.

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.

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.

Terraform comes pre-installed in Cloud Shell.

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

```
terraform
```

```
-version       An alias for the "version" subcommand.
student_00_e909601900c8@cloudshell:~ (qwiklabs-gcp-00-b5b16cc0a1dc)$ terraform
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

Main commands:
  init          Prepare your working directory for other commands
  validate      Check whether the configuration is valid
  plan          Show changes required by the current configuration
  apply         Create or update infrastructure
  destroy       Destroy previously-created infrastructure

All other commands:
  console       Try Terraform expressions at an interactive command prompt
  fmt           Reformat your configuration in the standard style
  force-unlock  Release a stuck lock on the current workspace
  get           Install or upgrade remote Terraform modules
  graph         Generate a Graphviz graph of the steps in an operation
  import        Associate existing infrastructure with a Terraform resource
  login         Obtain and save credentials for a remote host
  logout        Remove locally-stored credentials for a remote host
  output        Show output values from your root module
  providers     Show the providers required for this configuration
  refresh       Update the state to match remote systems
  show          Show the current state or a saved plan
  state         Advanced state management
  taint         Mark a resource instance as not fully functional
  test          Experimental support for module integration testing
  untaint       Remove the 'tainted' state from a resource instance
  version       Show the current Terraform version
  workspace     Workspace 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. 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
```

```
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 {
    }
  }
}
```

```
student_00_e909601900c8@cloudshell:~ (qwiklabs-gcp-00-b5b16cc0a1dc)$ cat instance.tf
resource "google_compute_instance" "terraform" {
  project      = "qwiklabs-gcp-00-b5b16cc0a1dc"
  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.

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
```

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
```

```
}
student_00_e909601900c8@cloudshell:~ (qwiklabs-gcp-00-b5b16cc0a1dc)$ ls
instance.tf  README-cloudshell.txt
student_00_e909601900c8@cloudshell:~ (qwiklabs-gcp-00-b5b16cc0a1dc)$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/google...
- Installing hashicorp/google v4.51.0...
- Installed hashicorp/google v4.51.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.
student_00_e909601900c8@cloudshell:~ (qwiklabs-gcp-00-b5b16cc0a1dc)$
```

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.

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
```

```
student_00_e909601900c8@cloudshell:~ (qwiklabs-gcp-00-b5b16cc0a1dc)$ terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # google_compute_instance.terraform will be created
  + resource "google_compute_instance" "terraform" {
      + can_ip_forward       = false
      + cpu_platform         = (known after apply)
      + current_status       = (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)
      + min_cpu_platform     = (known after apply)
      + name                 = "terraform"
      + project              = "<PROJECT_ID>"
      + 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)
          + mode                       = "READ_WRITE"
```

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
```

```
          + specific_reservation {
              + key    = (known after apply)
              + values = (known after apply)
            }
        }

      + scheduling {
          + automatic_restart           = (known after apply)
          + instance_termination_action = (known after apply)
          + min_node_cpus               = (known after apply)
          + on_host_maintenance         = (known after apply)
          + preemptible                 = (known after apply)
          + provisioning_model          = (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: yes

google_compute_instance.terraform: Creating...
google_compute_instance.terraform: Still creating... [10s elapsed]
google_compute_instance.terraform: Creation complete after 18s [id=projects/qwiklabs-gcp-00-b5b16cc0a1dc/zones/us-west1-c/instances/terraform]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
student_00_e909601900c8@cloudshell:~ (qwiklabs-gcp-00-b5b16cc0a1dc)$
```

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.

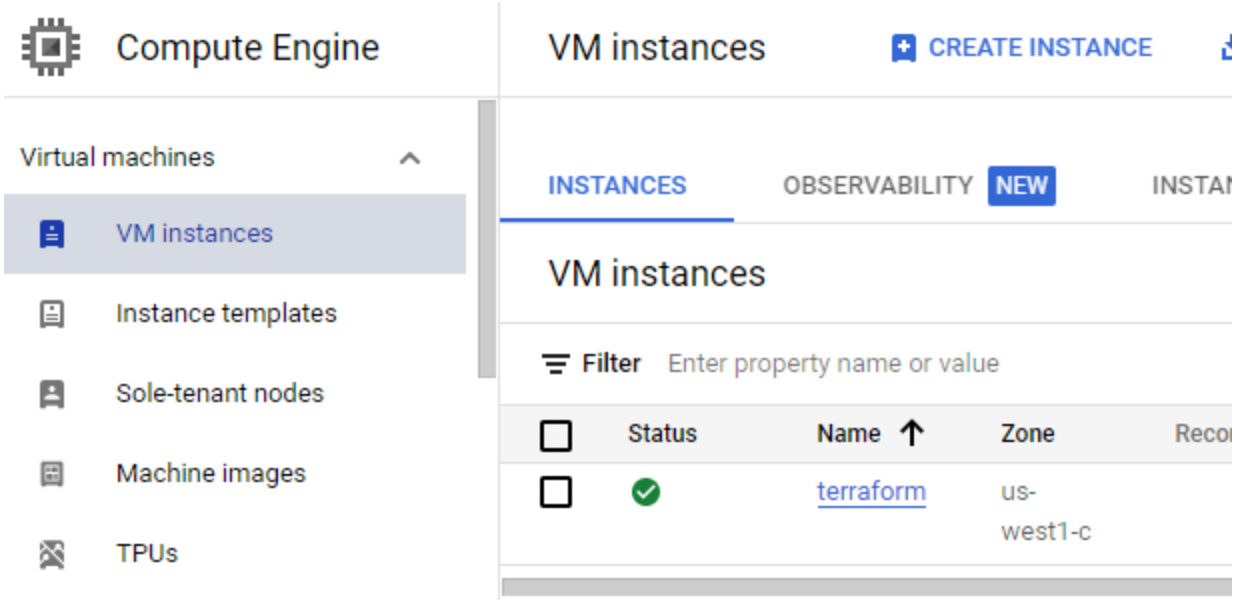
There is a + next to `google_compute_instance.terraform`, which means that Terraform will create this resource. Following that are the attributes that will be set. When the value displayed is `<computed>`, it means that the value won't be known until the resource is created.

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!

3. 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.

```
student_00_e909601900c8@cloudshell:~ (qwiklabs-gcp-00-b5b16cc0a1dc)$ ls
instance.tf  README-cloudshell.txt  terraform.tfstate  terraform.tfstate.backup
student_00_e909601900c8@cloudshell:~ (qwiklabs-gcp-00-b5b16cc0a1dc)$
```

4. In Cloud Shell, inspect the current state:

terraform show

```
student_00_e909601900c8@cloudshell:~ (qwiklabs-gcp-00-b5b16cc0a1dc)$ terraform show
# google_compute_instance.terraform:
resource "google_compute_instance" "terraform" {
    can_ip_forward       = false
    cpu_platform         = "Intel Broadwell"
    current_status       = "RUNNING"
    deletion_protection  = false
    enable_display       = false
    guest_accelerator    = []
    id                   = "projects/qwiklabs-gcp-00-b5b16cc0a1dc/zones/us-west1-c/instances/terraform"
    instance_id          = "5122404727416658715"
    label_fingerprint    = "42WmSpB8rSM="
    machine_type         = "n1-standard-1"
    metadata_fingerprint = "Ik1KdMOBr08="
    name                 = "terraform"
    project              = "qwiklabs-gcp-00-b5b16cc0a1dc"
    self_link            = "https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-00-b5b16cc0a1dc/zones/us-west1-c/instances/terraform"
    tags_fingerprint     = "42WmSpB8rSM="
    zone                 = "us-west1-c"

    boot_disk {
        auto_delete = true
        device_name = "persistent-disk-0"
        mode        = "READ_WRITE"
        source      = "https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-00-b5b16cc0a1dc/zones/us-west1-c/disks/terraform"

        initialize_params {
            image  = "https://www.googleapis.com/compute/v1/projects/debian-cloud/global/images/debian-11-bullseye-v20221206"
            labels = {}
            size   = 10
            type   = "pd-standard"
```

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