Topics tested:

- Import existing infrastructure into your Terraform configuration.
- Build and reference your own Terraform modules.
- Add a remote backend to your configuration.
- Use and implement a module from the Terraform Registry.
- Re-provision, destroy, and update infrastructure.
- Test connectivity between the resources you've created.

Challenge scenario

You are a cloud engineer intern for a new startup. For your first project, your new boss has tasked you with creating infrastructure in a quick and efficient manner and generating a mechanism to keep track of it for future reference and changes. You have been directed to use Terraform to complete the project.

For this project, you will use Terraform to create, deploy, and keep track of infrastructure on the startup's preferred provider, Google Cloud. You will also need to import some mismanaged instances into your configuration and fix them.

In this lab, you will use Terraform to import and create multiple VM instances, a VPC network with two subnetworks, and a firewall rule for the VPC to allow connections between the two instances. You will also create a Cloud Storage bucket to host your remote backend.

Note: At the end of every section, `plan` and `apply` your changes to allow your work to be successfully verified.

Task 1. Create the configuration files

1. In Cloud Shell, create your Terraform configuration files and a directory structure that resembles the following:

```
main.tf
variables.tf
modules/
└── instances
    ├── instances.tf
```

```
    ├── outputs.tf
    └── variables.tf
└── storage
    ├── storage.tf
    ├── outputs.tf
    └── variables.tf
```

2. Fill out the `variables.tf` files in the root directory and within the modules. Add three variables to each file: `region`, `zone`, and `project_id`. For their default values, use `us-east1`, `<filled in at lab start>` , and your Google Cloud Project ID.

   ```
   variable "region"{
         description = "region"
         default     = "us-east1"
   }
   variable "project_id"{
         description = "project_id"
         default     = "qwiklabs-gcp-00-6deeb5eac541"
   }
   variable "zone"{
         description = "zone"
         default     = "us-east1-a"
   }
   ```

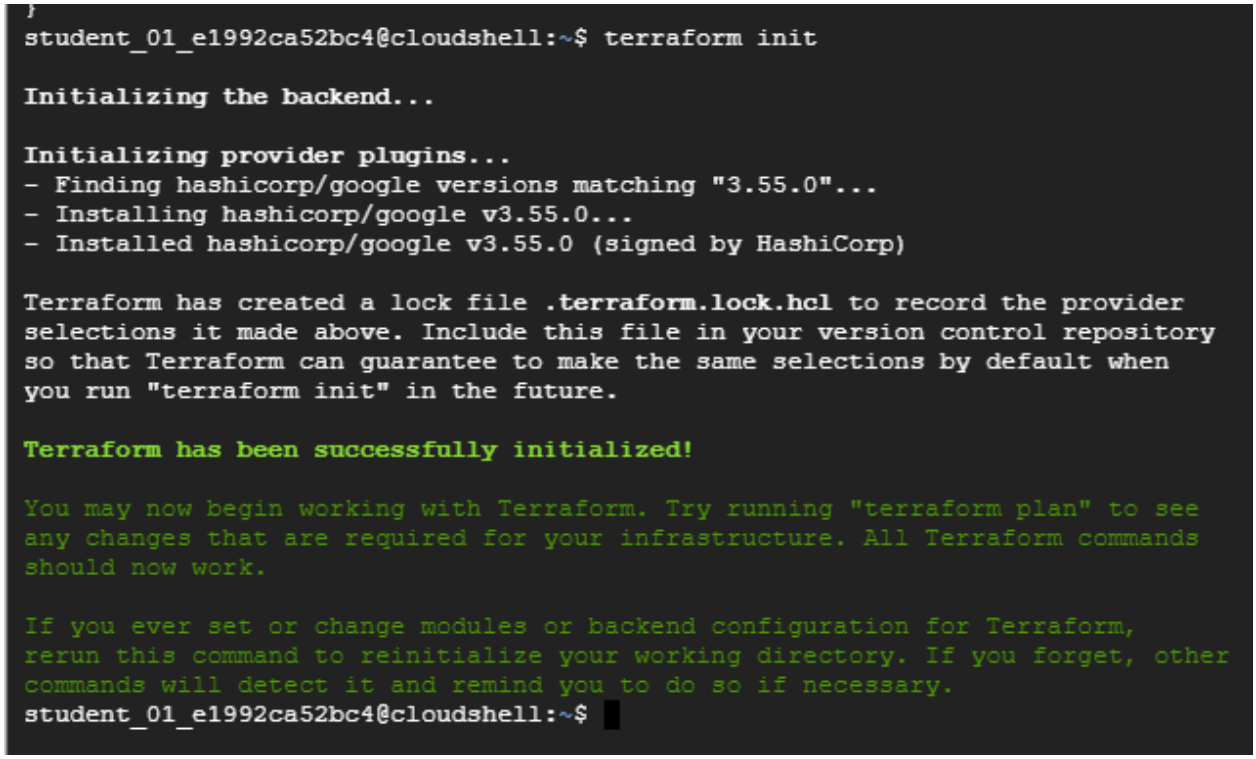
Note: You should use these variables anywhere applicable in your resource configurations.

3. Add the Terraform block and the Google Provider to the `main.tf` file. Verify the **zone** argument is added along with the **project** and **region** arguments in the Google Provider block.

   ```
   student_01_e1992ca52bc4@cloudshell:~$ cat main.tf
   terraform {
    required_providers {
      google = {
        source = "hashicorp/google"
        version = "3.55.0"
      }
   ```

```
  }
}

provider "google" {
  project     = var.project_id
  region      = var.region
  zone        = var.zone
}
```

4. Initialize Terraform.



Task 2. Import infrastructure

1. In the Google Cloud Console, on the **Navigation menu**, click **Compute Engine > VM Instances**. Two instances named `tf-instance-1` and `tf-instance-2` have already been created for you.

Note: by clicking on one of the instances, you can find its **Instance ID**, **boot disk image**, and **machine type**. These are all necessary for writing the configurations correctly and importing them into Terraform.

2. Import the existing instances into the `instances` module. To do this, you will need to follow these steps:

- First, add the module reference into the `main.tf` file then re-initialize Terraform.

```
student_01_e1992ca52bc4@cloudshell:~$ cat main.tf
terraform {
 required_providers {
  google = {
   source = "hashicorp/google"
   version = "3.55.0"
  }
 }
}

provider "google" {
 project     = var.project_id
 region      = var.region
 zone        = var.zone
}
module "instances" {
 source     = "./modules/instances"
}
```

- Next, write the resource configurations in the `instances.tf` file to match the pre-existing instances.
  - Name your instances `tf-instance-1` and `tf-instance-2`.
  - For the purposes of this lab, the resource configuration should be as minimal as possible. To accomplish this, you will only need to include the following additional arguments in your configuration: `machine_type`, `boot_disk`, `network_interface`, `metadata_startup_script`, and `allow_stopping_for_update`. For the last two arguments, use the following configuration as this will ensure you won't need to recreate it:

```
metadata_startup_script = <<-EOT
        #!/bin/bash
    EOT
allow_stopping_for_update = true
—----------instances.tf—----------------
resource "google_compute_instance" "tf-instance-1" {
  name         = "tf-instance-1"
  machine_type = "n1-standard-1"
```

```
  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-10"
    }
  }

  network_interface {
 network = "default"
  }

metadata_startup_script = <<-EOT
#!/bin/bash
EOT

allow_stopping_for_update = true
}

resource "google_compute_instance" "tf-instance-2" {
  name         = "tf-instance-2"
  machine_type = "n1-standard-1"

  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-10"
    }
  }

  network_interface {
 network = "default"
  }

metadata_startup_script = <<-EOT
#!/bin/bash
EOT

allow_stopping_for_update = true
}
```

- Once you have written the resource configurations within the module, use the `terraform import` command to import them into your `instances` module.

  terraform import module.instances.google_compute_instance.tf-instance-1 9764101931419963 35(instance-id)

terraform import module.instances.google_compute_instance.tf-instance-2 5849491116659612 46

```
student_01_e1992ca52bc4@cloudshell:~$ terraform import module.instances.google_compute_instance.tf-instance-1 976410193141996335

Error: Module not installed

  on main.tf line 15:
  15: module "instances" {

This module is not yet installed. Run "terraform init" to install all modules required by this configuration.


student_01_e1992ca52bc4@cloudshell:~$ terraform init
Initializing modules...
- instances in modules/instances

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/google from the dependency lock file
- Using previously-installed hashicorp/google v3.55.0

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_01_e1992ca52bc4@cloudshell:~$
```

```
student_01_e1992ca52bc4@cloudshell:~$ terraform import module.instances.google_compute_instance.tf-instance-1 976410193141996335
module.instances.google_compute_instance.tf-instance-1: Importing from ID "976410193141996335"...
module.instances.google_compute_instance.tf-instance-1: Import prepared!
  Prepared google_compute_instance for import
module.instances.google_compute_instance.tf-instance-1: Refreshing state... [id=projects/qwiklabs-gcp-00-6deeb5eac541/zones/us-east1-a/instances/976410193141996335]

Error: Cannot import non-existent remote object

While attempting to import an existing object to "module.instances.google_compute_instance.tf-instance-1", the provider detected that no object exists with the
given id. Only pre-existing objects can be imported; check that the id is correct and that it is associated with the provider's configured region or endpoint, or
use "terraform apply" to create a new remote object for this resource.

student_01_e1992ca52bc4@cloudshell:~$
```

Getting this error as instance is present in zone b .

```
commands will detect it and remind you to do so if necessary.
student_01_e1992ca52bc4@cloudshell:~$ terraform import module.instances.google_compute_instance.tf-instance-1 976410193141996335
module.instances.google_compute_instance.tf-instance-1: Importing from ID "976410193141996335"...
module.instances.google_compute_instance.tf-instance-1: Import prepared!
  Prepared google_compute_instance for import
module.instances.google_compute_instance.tf-instance-1: Refreshing state... [id=projects/qwiklabs-gcp-00-6deeb5eac541/zones/us-east1-b/instances/976410193141996335]

Import successful!

The resources that were imported are shown above. These resources are now in
your Terraform state and will henceforth be managed by Terraform.

student_01_e1992ca52bc4@cloudshell:~$
```

3. Apply your changes. Note that since you did not fill out all of the arguments in the entire configuration, the `apply` will **update the instances in-place**. This is fine for lab purposes, but in a production environment, you should make sure to fill out all of the arguments correctly before importing.

The two instances have now been imported into your terraform configuration. You can now run the commands to update the state of Terraform. Type *yes* at the dialogue after you run the apply command to accept the state changes.

```
terraform plan
terraform apply
```

```
Plan: 0 to add, 2 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
```

```
Apply complete! Resources: 0 added, 2 changed, 0 destroyed.
student_01_e1992ca52bc4@cloudshell:~$
```

Task 3. Configure a remote backend

1. Create a Cloud Storage bucket resource inside the `storage` module. For the bucket **name**, use `Bucket Name` . For the rest of the arguments, you can simply use:

- `location = "US"`
- `force_destroy = true`
- `uniform_bucket_level_access = true`

Note: You can optionally add output values inside of the `outputs.tf` file.

cat storage.tf
resource "google_storage_bucket" "storage-bucket" {
 name = "tf-bucket-438079"
 location = "US"
 force_destroy = true
 uniform_bucket_level_access = true
}

2. Add the module reference to the `main.tf` file. Initialize the module and `apply` the changes to create the bucket using Terraform.
   —----------
   cat main.tf
   terraform {
    required_providers {
     google = {

```
    source = "hashicorp/google"
    version = "3.55.0"
   }
  }
}
provider "google" {
  project     = var.project_id
  region      = var.region
  zone        = var.zone
}
module "instances" {
  source     = "./modules/instances"
}
module "storage" {
  source     = "./modules/storage"
}
```

```
student_01_e1992ca52bc4@cloudshell:~$ terraform plan
module.instances.google_compute_instance.tf-instance-1: Refreshing
module.instances.google_compute_instance.tf-instance-2: Refreshing

Terraform used the selected providers to generate the following ex
  + create

Terraform will perform the following actions:

  # module.storage.google_storage_bucket.storage-bucket will be cr
  + resource "google_storage_bucket" "storage-bucket" {
      + bucket_policy_only          = (known after apply)
      + force_destroy               = true
      + id                          = (known after apply)
      + location                    = "US"
      + name                        = "tf-bucket-438079"
      + project                     = (known after apply)
      + self_link                   = (known after apply)
      + storage_class               = "STANDARD"
      + uniform_bucket_level_access = true
      + url                         = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

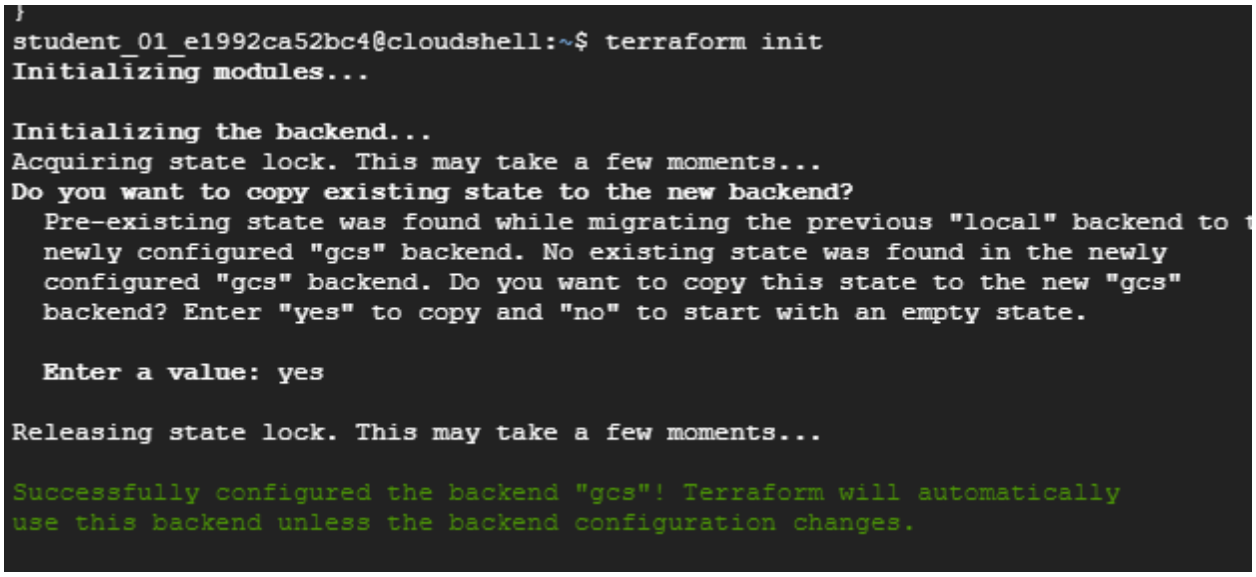
```
Terraform will perform the following actions:

  # module.storage.google_storage_bucket.storage-bucket will be created
  + resource "google_storage_bucket" "storage-bucket" {
      + bucket_policy_only          = (known after apply)
      + force_destroy               = true
      + id                          = (known after apply)
      + location                    = "US"
      + name                        = "tf-bucket-438079"
      + project                     = (known after apply)
      + self_link                   = (known after apply)
      + storage_class               = "STANDARD"
      + uniform_bucket_level_access = true
      + url                         = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

3. Configure this storage bucket as the remote backend inside the `main.tf` file. Be sure to use the **prefix** `terraform/state` so it can be graded successfully.

```
student_01_e1992ca52bc4@cloudshell:~$ cat main.tf
terraform {
 backend "gcs" {
   bucket  = "<FILL IN PROJECT ID>"
 prefix  = "terraform/state"
 }
 required_providers {
   google = {
     source = "hashicorp/google"
     version = "3.55.0"
   }
 }
}

provider "google" {
 project     = var.project_id
 region      = var.region
 zone        = var.zone
}
module "instances" {
 source     = "./modules/instances"
}
module "storage" {
```

```
  source     = "./modules/storage"
}
```

4. If you've written the configuration correctly, upon `init`, Terraform will ask whether you want to copy the existing state data to the new backend. Type `yes` at the prompt.



Task 4. Modify and update infrastructure

1. Navigate to the `instances` module and modify the **tf-instance-1** resource to use an `n1-standard-2` machine type.

```
resource "google_compute_instance" "tf-instance-1" {
  name         = "tf-instance-1"
  machine_type = "n1-standard-2"
  zone         = "us-central1-a"
  allow_stopping_for_update = true

  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-10"
    }
  }

  network_interface {
 network = "default"
  }
}
```

2. Modify the **tf-instance-2** resource to use an `n1-standard-2` machine type.

```
resource "google_compute_instance" "tf-instance-2" {
  name         = "tf-instance-2"
  machine_type = "n1-standard-2"
  zone         = "us-central1-a"
  allow_stopping_for_update = true

  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-10"
    }
  }
  network_interface {
 network = "default"
  }
}
```

3. Add a third instance resource and name it **`Instance Name`** . For this third resource, use an `n1-standard-2` machine type.

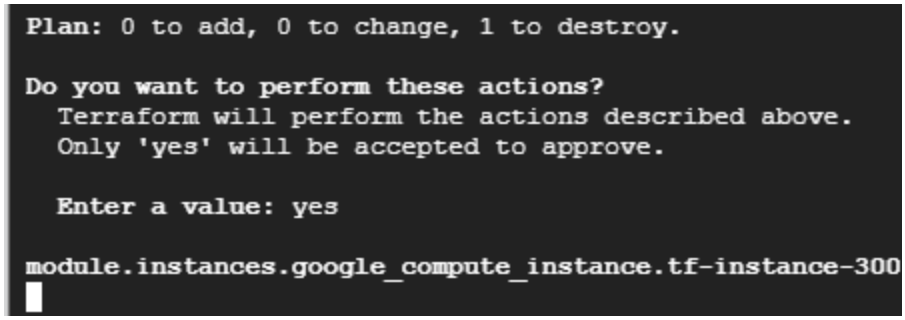
```
resource "google_compute_instance" "<FILL IN INSTANCE 3 NAME>" {
  name         = "<FILL IN INSTANCE 3 NAME>"
  machine_type = "n1-standard-2"
  zone         = "us-central1-a"
  allow_stopping_for_update = true

  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-10"
    }
  }

  network_interface {
 network = "default"
  }
}
```

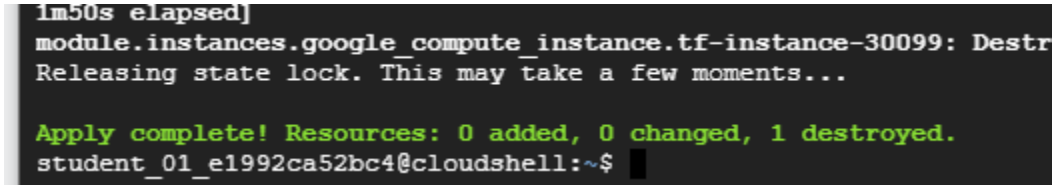
4. Initialize Terraform and `apply` your changes.

Note: Optionally, you can add output values from these resources in the `outputs.tf` file within the module.

Task 5. Destroy resources

1. Destroy the third instance **`Instance Name`** by removing the resource from the configuration file. After removing it, initialize terraform and `apply` the changes.





Task 6. Use a module from the Registry

1. In the Terraform Registry, browse to the Network Module.
2. Add this module to your `main.tf` file. Use the following configurations:

- Use version `6.0.0` (different versions might cause compatibility errors).
- Name the VPC `VPC Name` , and use a **global** routing mode.
- Specify **2** subnets in the `us-east1` region, and name them `subnet-01` and `subnet-02`. For the subnets arguments, you just need the **Name**, **IP**, and **Region**.
- Use the IP `10.10.10.0/24` for `subnet-01`, and `10.10.20.0/24` for `subnet-02`.
- You do **not** need any secondary ranges or routes associated with this VPC, so you can omit them from the configuration.

```
module "vpc" {
  source  = "terraform-google-modules/network/google"
  version = "6.0.0"

  project_id   = "qwiklabs-gcp-00-6deeb5eac541"
  network_name = "tf-vpc-810883"
  routing_mode = "GLOBAL"

  subnets = [
    {
      subnet_name           = "subnet-01"
      subnet_ip             = "10.10.10.0/24"
```

```
        subnet_region        = "us-east1"
    },
    {
        subnet_name          = "subnet-02"
        subnet_ip            = "10.10.20.0/24"
        subnet_region        = "us-east1"
        subnet_private_access = "true"
        subnet_flow_logs     = "true"
        description          = "This subnet has a description"
    }
  ]
}
```

3. Once you've written the module configuration, initialize Terraform and run an `apply` to create the networks.
4. Next, navigate to the `instances.tf` file and update the configuration resources to connect **tf-instance-1** to `subnet-01` and **tf-instance-2** to `subnet-02`.

Note: Within the instance configuration, you will need to update the **network** argument to `VPC Name` , and then add the **subnetwork** argument with the correct subnet for each instance.

Task 7. Configure a firewall

- Create a firewall rule resource in the `main.tf` file, and name it **tf-firewall**.
    - This firewall rule should permit the `VPC Name` network to allow ingress connections on *all* IP ranges (`0.0.0.0/0`) on **TCP port 80**.
    - Make sure you add the `source_ranges` argument with the correct IP range (`0.0.0.0/0`).
    - Initialize Terraform and `apply` your changes.

Note: To retrieve the required `network` argument, you can inspect the state and find the **ID** or **self_link** of the `google_compute_network` resource you created. It will be in the form `projects/PROJECT_ID/global/networks/ VPC Name .`

```
resource "google_compute_firewall" "tf-firewall" {
  name    = "tf-firewall"
 network = "projects/<FILL IN PROJECT_ID>/global/networks/<FILL IN NETWORK NAME>"
```

```
  allow {
    protocol = "tcp"
    ports    = ["80"]
  }

  source_tags = ["web"]
  source_ranges = ["0.0.0.0/0"]
}
```

Connectivity test (Optional)

After you have created a firewall rule to allow internal connections over the VPC, you can optionally run a network connectivity test.

1. Make sure both of your VMs are running.
2. Navigate to **Network Intelligence > Connectivity Tests**. Run a connectivity test on the two VMs to verify that they are reachable. You have now validated the connectivity between the instances!

Note: Ensure that the **Network Management API** is successfully enabled; if it is not, click **Enable**.

Your configuration settings should resemble the following:

Test name *
terrform-network-check

Lowercase letters, numbers, hyphens allowed

Protocol
tcp

Source

Source endpoint
VM instance

Source VM instance *
tf-instance-1 (10.10.10.2)

☐ VM instance is in a project other than "qwiklabs-gcp-04-eff887d14fc7"

Destination

Destination endpoint
VM instance

Destination VM instance *
tf-instance-2 (10.10.20.2)

☐ VM instance is in a project other than "qwiklabs-gcp-04-eff887d14fc7"

Destination port *
80

CREATE CANCEL

Equivalent REST or command line