Terraform GCP lab

Getting Started

It is best to open a new incognito window and perform all tasks in it.

Before you click the Start Lab button, read these instructions. This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.
- When ready click Start Lab.

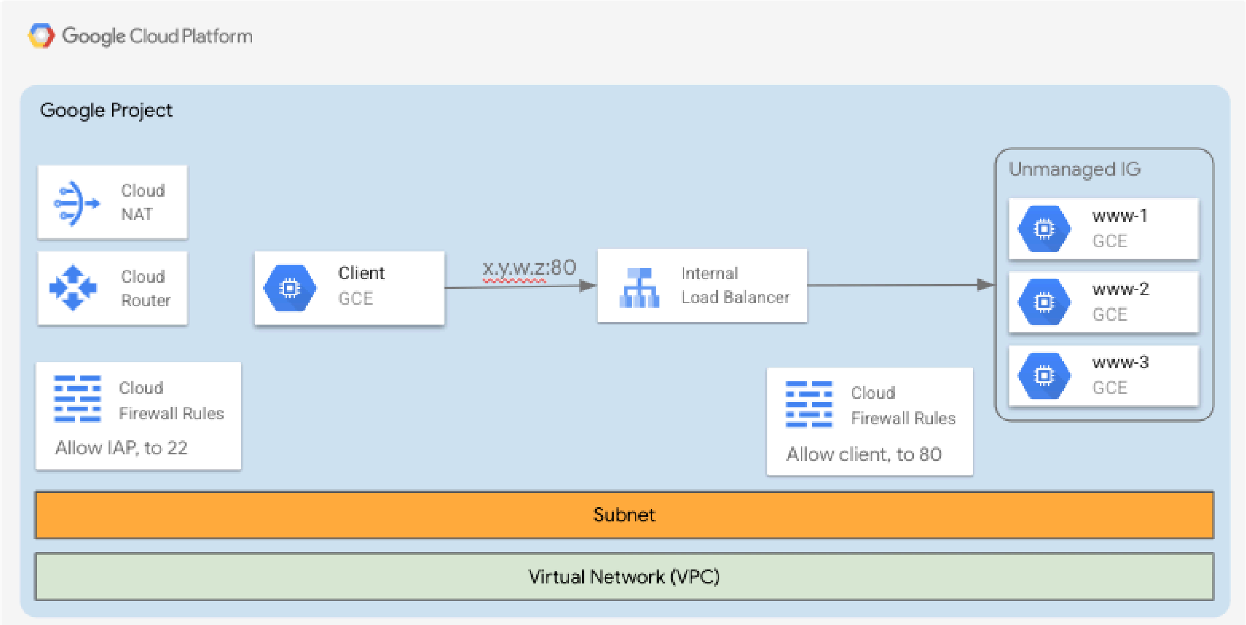
Objective

The goal of this lab is for you to learn on how to use Terraform to deploy GCP resources.

Scenario

For this lab, you will set up a private network, deploy a simple web application configured on a collection of 3 VMs which are part of a single unmanaged Instance Group and allow connectivity from a client machine.

During this lab, you will learn how to initialize terraform with Google cloud providers, configure network components, create multiple vm instances using loops in Terraform, create network load balancer consuming Cloud Fabric network module and finally configure firewall rules to allow traffic flow between client VM and web application.

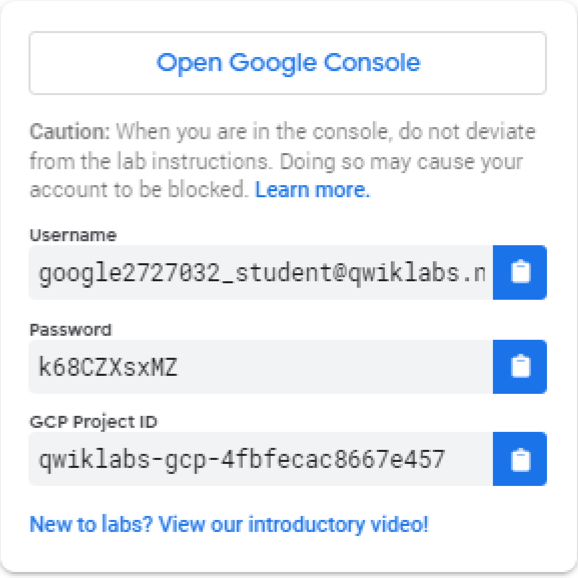


Starting the lab

Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you. If you already have your own personal Google Cloud account or project, do not use it for this lab. If you are using a Pixelbook, open an Incognito window to run this lab.

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary

credentials that you must use for this lab.



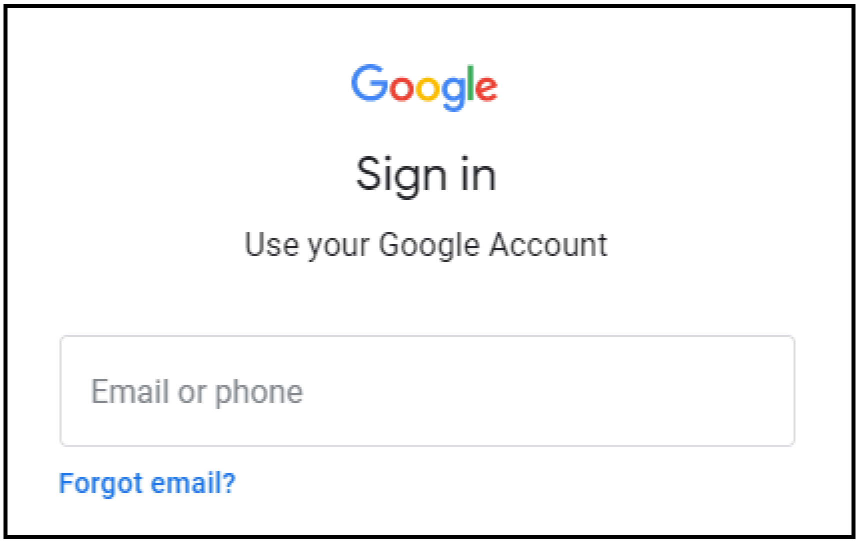
Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

Username
google2727032_student@qwiklabs.n

Password
k68CZXsxMZ

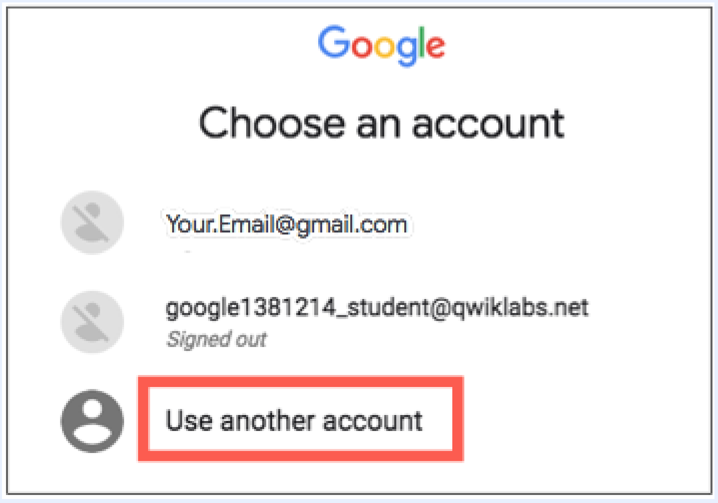
GCP Project ID
qwiklabs-gcp-4fbfecac8667e457

2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



Tip: Open the tabs in separate windows, side by side.

3. **Note:** If you see the *Choose an account* page, click **Use another account**.



Google

Choose an account

Your.Email@gmail.com

google1381214_student@qwiklabs.net
Signed out

Use another account

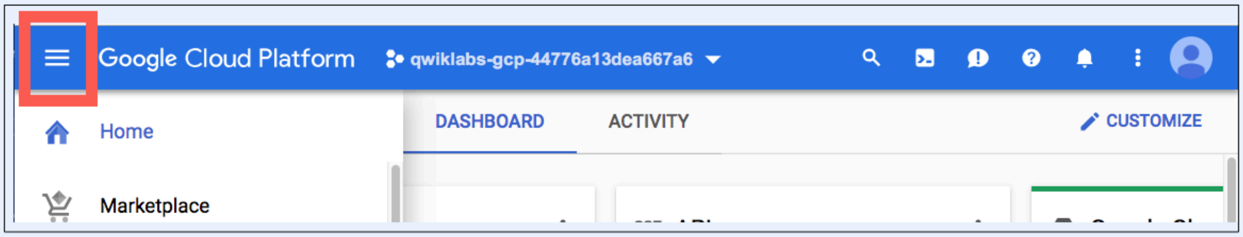
4. On the **Sign in** page, paste the username that you copied from the Lab details pane. Then copy and paste the password.

> **Important:** You must use the credentials from the Lab details pane. Do not use your own Qwiklabs credentials. If you have a personal Google Cloud account, do not use it for this lab (to avoid incurring charges).

5. Click through the subsequent pages:
   - Accept the terms and conditions.
   - Do not add recovery options or two-factor authentication (because this is a temporary account).
   - Do not sign up for free trials.

After a few moments, the Cloud Console opens in this tab.

Note: You can view the menu with a list of Google Cloud products and services by clicking the **Navigation menu** in the top left.

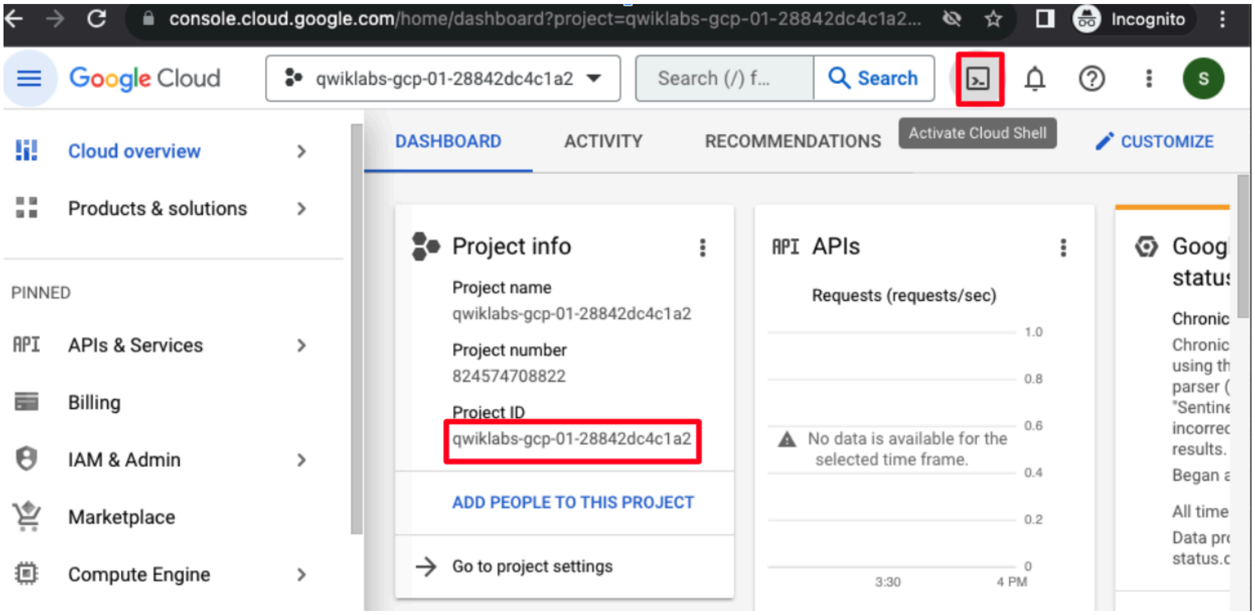


Lab 1

Task 1.1 : Define Terraform Root Module Structure & Initialize Terraform

Step 1.1.1 : Activate Cloud Shell

Open Cloud shell by clicking on Activate Cloud Shell button



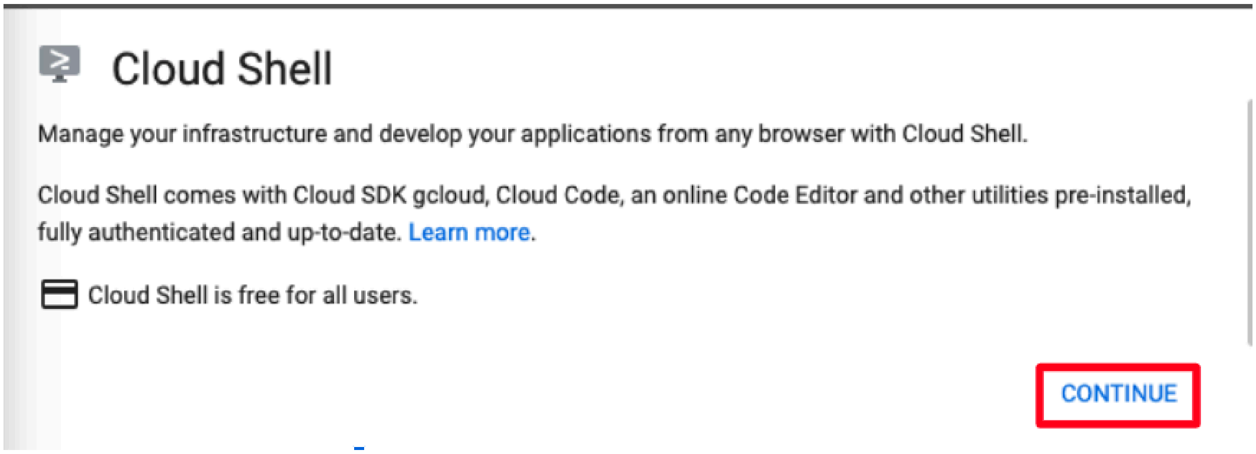
Project info

Project name
qwiklabs-gcp-01-28842dc4c1a2

Project number
824574708822

Project ID
qwiklabs-gcp-01-28842dc4c1a2

If prompted, Click on continue & wait for the cloud shell machine to be provisioned



Cloud Shell

Manage your infrastructure and develop your applications from any browser with Cloud Shell.

Cloud Shell comes with Cloud SDK gcloud, Cloud Code, an online Code Editor and other utilities pre-installed, fully authenticated and up-to-date. Learn more.

Cloud Shell is free for all users.

Step 1.1.2 : Create & download Terraform root module structure

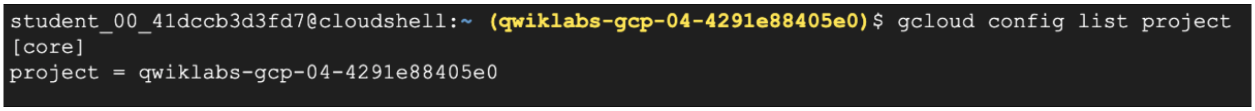
Terraform root module structure is available in the Cloud storage bucket in the lab. You will copy this folder to cloud shell by following the below steps.

Query & Copy your project ID

Command:

```
gcloud config list project
```

Example Output:



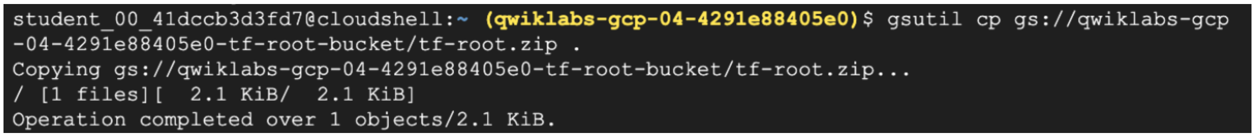
```
student_00_41dccb3d3fd7@cloudshell:~ (qwiklabs-gcp-04-4291e88405e0)$ gcloud config list project
[core]
project = qwiklabs-gcp-04-4291e88405e0
```

Execute following command replacing the project ID with actual value copied from above step

Command:

```
gsutil cp gs://<project-ID>-tf-root-bucket/tf-root.zip .
```

Example Output:



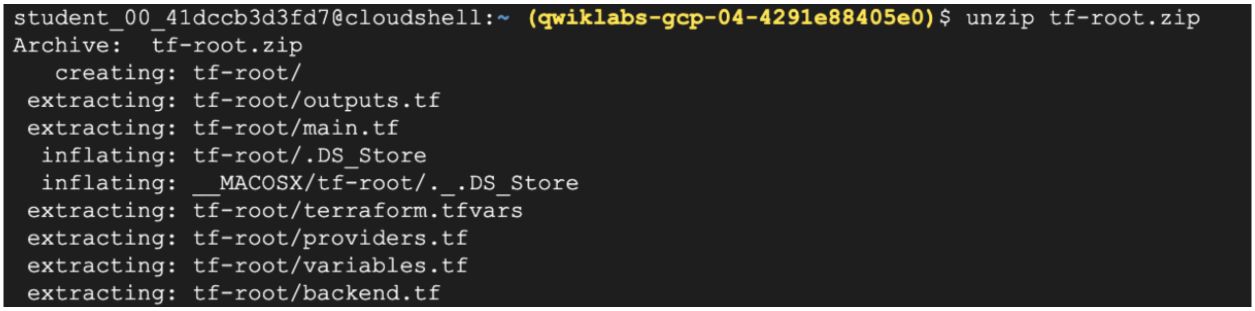
```
student_00_41dccb3d3fd7@cloudshell:~ (qwiklabs-gcp-04-4291e88405e0)$ gsutil cp gs://qwiklabs-gcp
-04-4291e88405e0-tf-root-bucket/tf-root.zip .
Copying gs://qwiklabs-gcp-04-4291e88405e0-tf-root-bucket/tf-root.zip...
/ [1 files][  2.1 KiB/  2.1 KiB]
Operation completed over 1 objects/2.1 KiB.
```

Unzip downloaded content by executing

Command:

```
unzip tf-root.zip
```

Example Output:



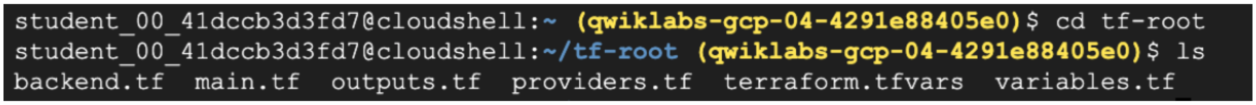
```
student_00_41dccb3d3fd7@cloudshell:~ (qwiklabs-gcp-04-4291e88405e0)$ unzip tf-root.zip
Archive:  tf-root.zip
   creating: tf-root/
  extracting: tf-root/outputs.tf
  extracting: tf-root/main.tf
   inflating: tf-root/.DS_Store
   inflating: __MACOSX/tf-root/._.DS_Store
  extracting: tf-root/terraform.tfvars
  extracting: tf-root/providers.tf
  extracting: tf-root/variables.tf
  extracting: tf-root/backend.tf
```

Switch to directory tf-root

Command:

```
cd tf-root
ls
```

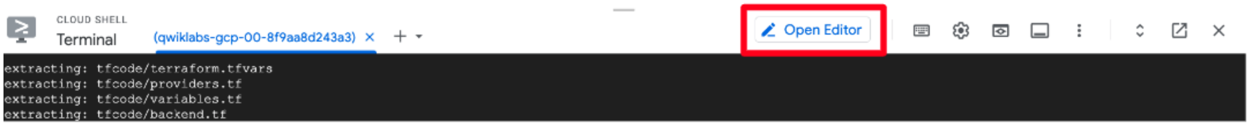
Example output:



```
student_00_41dccb3d3fd7@cloudshell:~ (qwiklabs-gcp-04-4291e88405e0)$ cd tf-root
student_00_41dccb3d3fd7@cloudshell:~/tf-root (qwiklabs-gcp-04-4291e88405e0)$ ls
backend.tf  main.tf  outputs.tf  providers.tf  terraform.tfvars  variables.tf
```

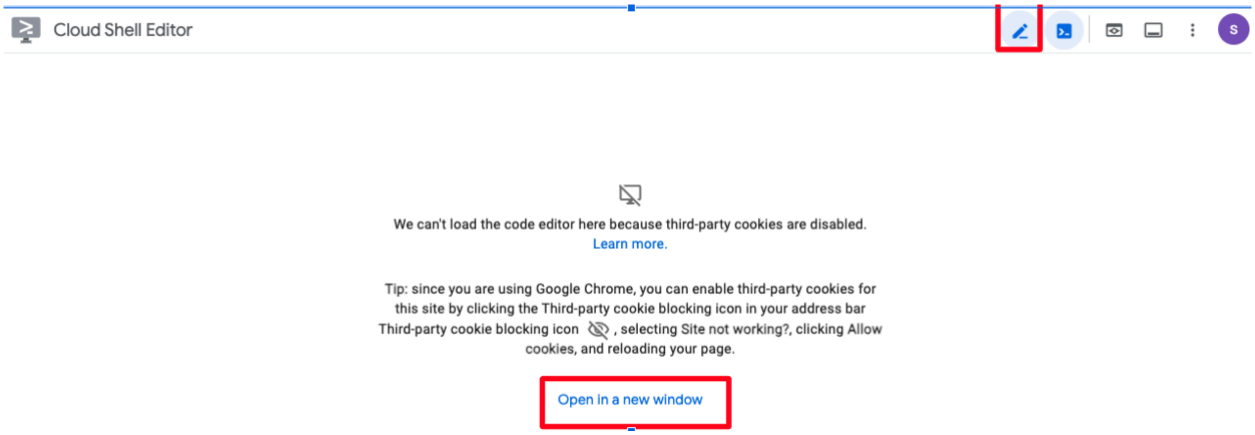
Step 1.1.3 : Open Cloud Shell editor in a new window

- Open Cloud Shell editor by clicking on Open Editor button as shown below



```
extracting: tfcode/terraform.tfvars
extracting: tfcode/providers.tf
extracting: tfcode/variables.tf
extracting: tfcode/backend.tf
```

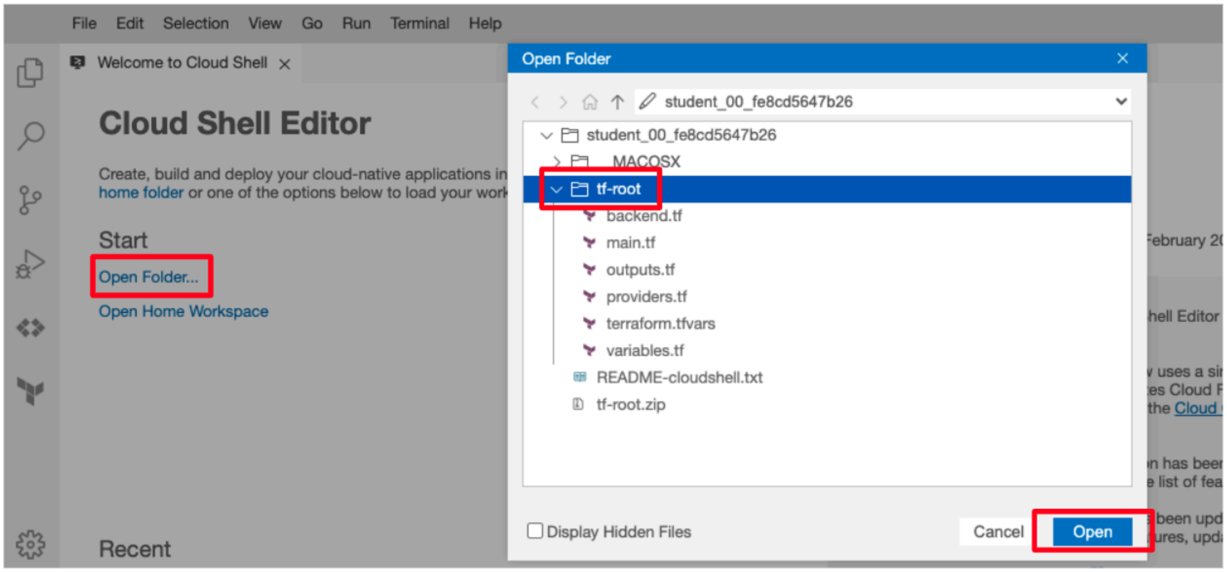
- Click on Open in New Window



We can't load the code editor here because third-party cookies are disabled. Learn more.

Tip: since you are using Google Chrome, you can enable third-party cookies for this site by clicking the Third-party cookie blocking icon in your address bar Third-party cookie blocking icon , selecting Site not working?, clicking Allow cookies, and reloading your page.

- Open tf-root folder in Cloud Shell editor



Create, build and deploy your cloud-native applications in home folder or one of the options below to load your work

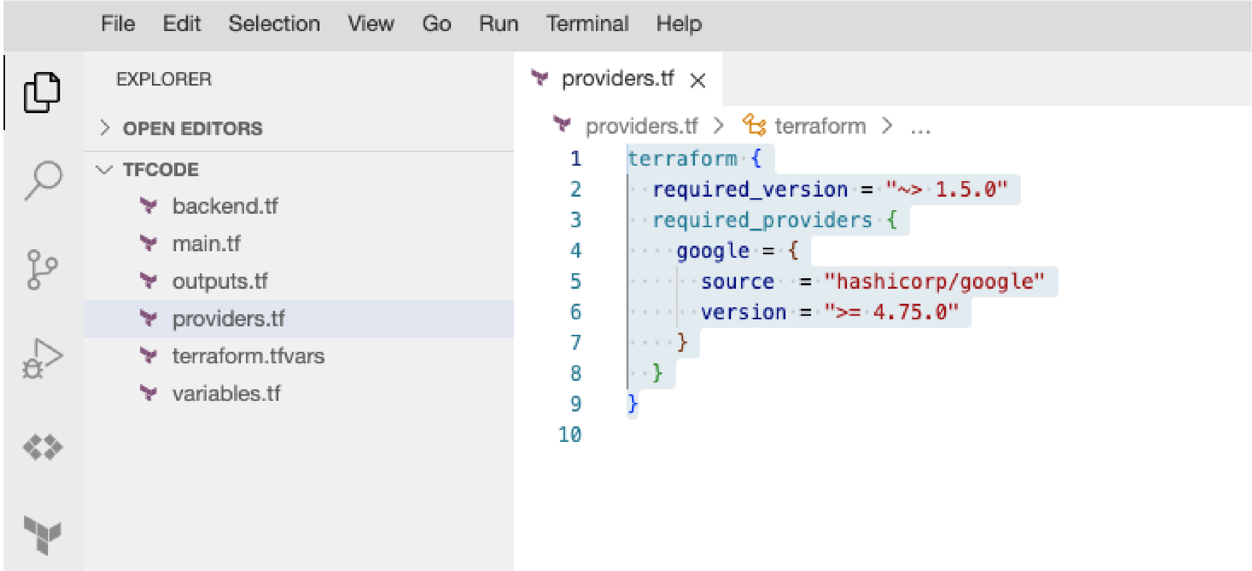
Step 1.1.4 : Define Google provider

Copy & Paste the following code to providers.tf and save

File: Providers.tf

```
terraform {
  required_version = "~> 1.5"
  required_providers {
    google = {
      source  = "hashicorp/google"
      version = ">= 4.75.0"
    }
  }
}
```

Example providers.tf file:



```
terraform {
  required_version = "~> 1.5.0"
  required_providers {
    google = {
      source  = "hashicorp/google"
      version = ">= 4.75.0"
    }
  }
}
```

Step 1.1.5 : Initialize Terraform Environment

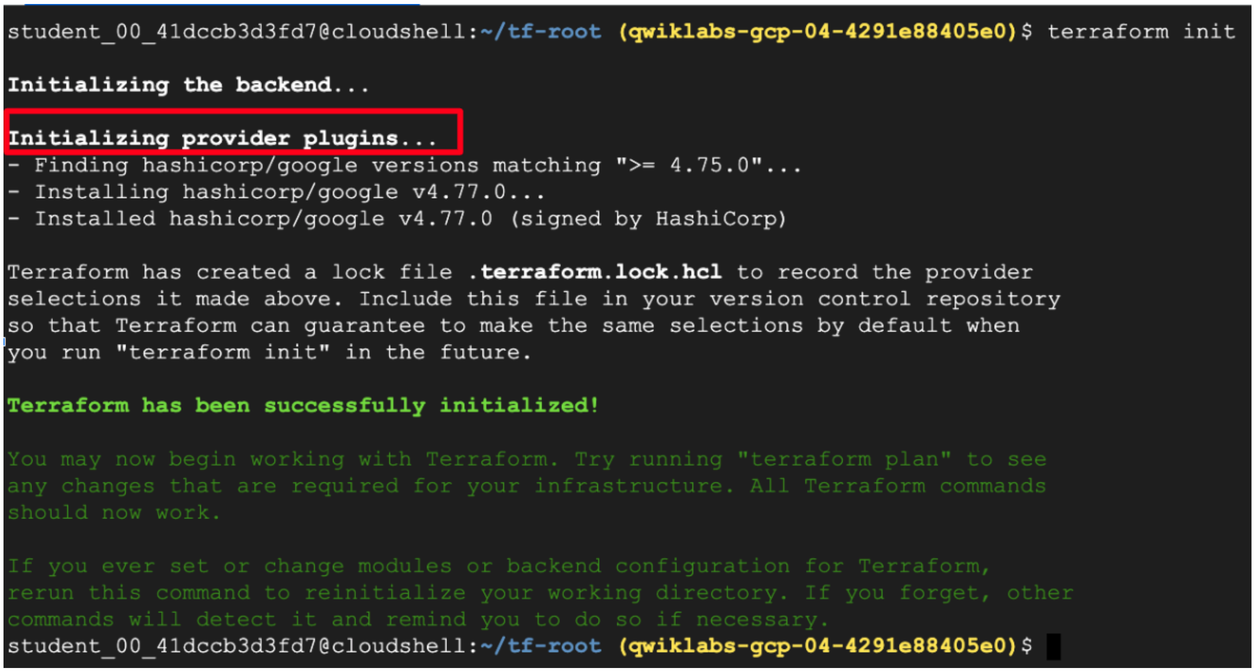
In the other tab, switch to cloud Terminal.

Execute the following command to initialize Terraform and install a defined google provider.

Command:

```
terraform init
```

Example Output:



```
student_00_41dccb3d3fd7@cloudshell:~/tf-root (qwiklabs-gcp-04-4291e88405e0)$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/google versions matching ">= 4.75.0"...
- Installing hashicorp/google v4.77.0...
- Installed hashicorp/google v4.77.0 (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_41dccb3d3fd7@cloudshell:~/tf-root (qwiklabs-gcp-04-4291e88405e0)$
```

Task 1.2 : Create VPC, subnet, GCE instance & Firewall rules

Update Main.tf file with relevant TF code to create resources

Step 1.2.1 - Create VPC

Do not auto-create subnetworks
Provider Reference

File: Main.tf

```
## Creates "my-vpc" in the specified project
resource "google_compute_network" "my_vpc" {
  project                 = "<Your-project-id>"
  name                    = "my-vpc"
  auto_create_subnetworks = false
}
```

Step 1.2.2 : Create a Subnet

Give it a CIDR of 192.168.0.0/24
User region as “us-central1”

Provider Reference

File: Main.tf

```
## Creates "my-subnet" in "my-vpc" VPC
resource "google_compute_subnetwork" "my_subnet" {
  name          = "my-subnet"
  project       = google_compute_network.my_vpc.project
  ip_cidr_range = "192.168.0.0/24"
  region        = "us-central1"
  network       = google_compute_network.my_vpc.id
}
```

Step 1.2.3 : Create a GCE instance (VM)

Configure the VM with following settings

- Machine type is e2-micro
- Lives in zone “us-central-b”
- Use image debian-cloud/debian-11
- Assign a network tag “www”
- Run “apt update && apt install -y nginx” at startup

Provider Reference

File: Main.tf

```
resource "google_compute_instance" "my_www_vm_1" {
  project      = google_compute_network.my_vpc.project
  name         = "my-www-vm-1"
  machine_type = "e2-micro"
  zone         = "us-central1-b"

  tags = ["www"]

  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-11"
    }
  }
  network_interface {
    network    = google_compute_network.my_vpc.id
    subnetwork = google_compute_subnetwork.my_subnet.id
  }
   allow_stopping_for_update = true
  metadata_startup_script = "apt update && apt install -y
nginx"
}
```

Step 1.2.4 : Create a firewall rule to allow ingress traffic on port TCP 80

- It should match source 192.168.0.0/24
- It should match destination instances with tag "www”

Provider Reference

File: Main.tf

```
  resource "google_compute_firewall" "allow_www" {
    name          = "allow-www"
    project       = google_compute_network.my_vpc.project
    network       = google_compute_network.my_vpc.id
    source_ranges = ["192.168.0.0/24"]
    target_tags   = ["www"]
    allow {
      protocol = "tcp"
      ports    = ["80"]
    }
  }
```

Step 1.2.5 : Create a firewall rule to allow ingress traffic for IAP

- It should match source 192.168.0.0/24
- Matches destination protocol tcp, port 22
- It should be applied to all instances

Provider Reference

File: Main.tf

```
  resource "google_compute_firewall" "allow_iap" {
    name          = "allow-iap"
    project       = google_compute_network.my_vpc.project
    network       = google_compute_network.my_vpc.id
    source_ranges = ["35.235.240.0/20"]
    allow {
      protocol = "tcp"
      ports    = ["22"]
    }
    }
```

Step 1.2.6 : Deploy the resources

Execute **Terraform plan**, review the resources that will be created.

Command:

```
terraform plan
```

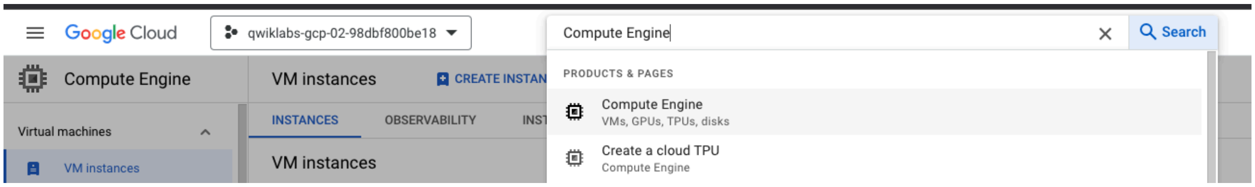
Execute **Terraform apply** to actually create the resources.

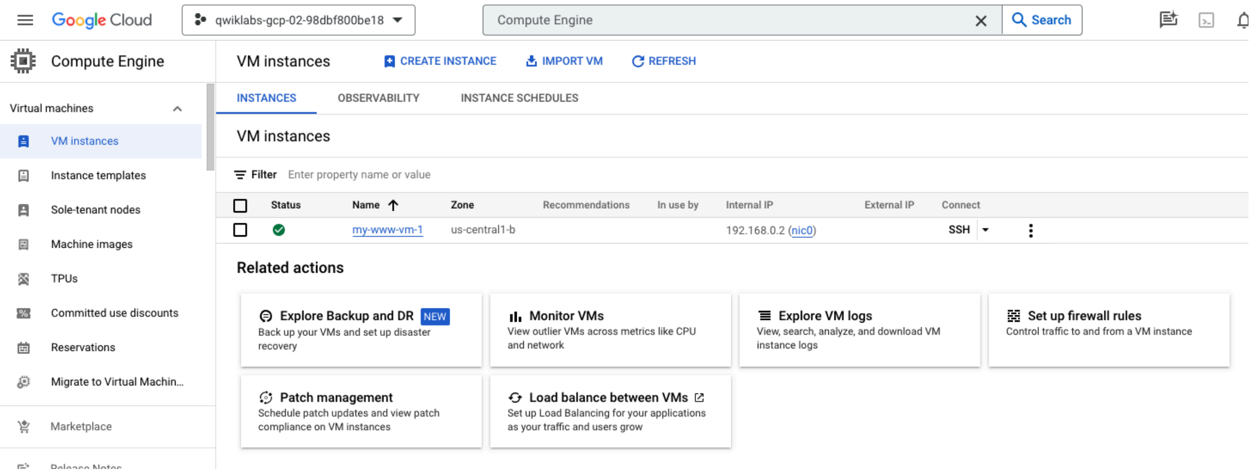
Command:

```
terraform apply
```

Verification

1. Verify a Compute instance with name “my-www-vm-1” is created from the console.





☐	Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP	Connect
☐	✓	my-www-vm-1	us-central1-b			192.168.0.2 (nic0)		SSH ▾ ⋮

Related actions

Explore Backup and DR NEW
Back up your VMs and set up disaster recovery

Monitor VMs
View outlier VMs across metrics like CPU and network

Explore VM logs
View, search, analyze, and download VM instance logs

Set up firewall rules
Control traffic to and from a VM instance

Patch management
Schedule patch updates and view patch compliance on VM instances

Load balance between VMs
Set up Load Balancing for your applications as your traffic and users grow

2. Verify 2 firewall rules “allow-www” & “allow-iap” are created

Lab 2

Task 2.1 : Modify GCE instance to use variables & output relevant values.

Step 2.1.1 : Define variable to configure VM name prefix & another variable for the region

- ○ "**vm_prefix**" of type "String" & default value of "vm"
- ○ "**region**" of type "String" & default value of "us-central1"

File: Variables.tf

```
variable "vm_prefix" {
  type        = string
  description = "the common VM names prefix"
  default     = "vm"
}

variable "region" {
  type        = string
  description = "the region where the resources live"
  default     = "us-central1"
}
```

Step 2.1.2 : Modify compute resource to infer name & region from defined variables

File: Main.tf - modify the value of name attribute to use variable "vm_prefix"

```
resource "google_compute_instance" "my_www_vm_1" {
  project      = google_compute_network.my_vpc.project
  name         = "${var.vm_prefix}-1"
  machine_type = "e2-micro"
  zone         = "us-central1-b"
}
```

Task 2.2 : Add VM name as Output

- Define vm_1_name output variable in outputs.tf file

File: Outputs.tf

```
output "vm_1_name" {
  description = "The www VM 1 name."
  value       = google_compute_instance.my_www_vm_1.name
}
```

Task 2.3 : Define local value representing GCP zone & modify the GCE instance to use the local

- Define locals in main.tf & Modify compute_instance resource code to use the defined local

File: Main.tf - Define “**gce_zone**” as Local variable and modify value for “zone” attribute to use the local

```
locals {
  gce_zone = "${var.region}-b"
}

resource "google_compute_instance" "my_www_vm_1" {
  project      = google_compute_network.my_vpc.project
  name         = "${var.vm_prefix}-1"
  machine_type = "e2-micro"
  zone         = local.gce_zone
}
```

Task 2.4 : Count - Use Variable to decide whether the VM should be created or not

- Add a new variable “create_www_instance” variables of type “bool” to variables.tf

File: Variables.tf

```
variable "create_www_instance" {
  type        = bool
  description = "Whether the www VM should be created."
}
```

- Modify compute_instance resource to include a condition using count and bool value

File: Main.tf - Add **count** and condition using defined bool variable “**create_www_instance**”
New - **Output.tf - Note at this step Comment Output.tf**

```
resource "google_compute_instance" "my_www_vm_1" {
  count        = var.create_www_instance ? 1 : 0
  name         = "${var.vm_prefix}-1"
  project      = google_compute_network.my_vpc.project
  machine_type = "e2-micro"
  zone         = local.gce_zone

  tags = ["www"]
}
```

- Execute Terraform plan & apply to deploy resources

- Execute "**Terraform plan"** to confirm the changes and "**Terraform apply"** to deploy the changes.

- Terraform plan will prompt for value of variable "create_www_instance"
- Answer false and notice no changes will be shown and true where new instance changes are shown

- To avoid this prompt and provide a value to the "create_www_instace" variable, open terraform.tfvars and assign value to the variable

File: terraform.tfvars

```
create_www_instance = true
```

- Test again by executing **Terraform plan** & **Terraform apply** and the instance should be created now

Verify
Validate VM Name & zone details are interpreted correctly as expected Validate "Terraform apply" returns output vm_1_name with the instance name

Task 2.5 : Create 3 GCE instances using a loop

Step 2.5.1 - Create a list variable

- Modify **variables.tf** to define a new variable "my_vms" of type "list(string)"

File: Variables.tf

```
variable "my_vms" {
  type        = list(string)
  description = "My VMs."
}
```

Step 2.5.2 - Assign value to the list variable created above

- Assign value to “my_vms” variable

File: terraform.tfvars

```
my_vms = ["1", "2", "3"]
```

Step 2.5.3 : Modify compute resource to index through list variable

- Modify compute instance resource group to index through my_vms list and create 3 instances

File: Main.tf - Add **for_each** to parse thru values of the list “**my_vms**” and modify **“name”** attribute value to use the passed value through the list variable.

```
resource "google_compute_instance" "my_www_vms" {
  for_each     = toset(var.my_vms)
  project      = google_compute_network.my_vpc.project
  name         = "vm-${index(var.my_vms, each.value)}"
  machine_type = "e2-micro"
}
```

Note: alternate way to parse through the list us using “each” for e.g. in the above code name can be defined as

```
name  = "${var.vm-prefix}-${each.value}"
```

Step 2.5.4 : Change Code to express both VM name suffix and machine type through code using **map** construct

- Define new variable “www-instances” of type “map(any)”

File: Variables.tf

```
variable "www_instances" {
  type        = map(any)
  description = "The www VMs to create"
  default     = {}
}
```

- Assign map of values of :www-instances” variable

File: [terraform.tfvars]()

```
www_instances = {
  machine_1 = {
    name_suffix  = "1"
    machine_type = "n2-standard-2"
  },
  machine_2 = {
    name_suffix  = "2"
    machine_type = "n2-standard-2"
  },
 machine_3 = {
    name_suffix  = "3"
    machine_type = "n2-standard-2"
  },
}
```

- Modify google_compute_instance to use the above variable and define the VM Name and machine type

File: [Main.tf]() **-** Modify google_compute_instance resource name “**my_www_vms**” and **for_each** list to point to “**www_instances**” variable. Also, modify **name** attribute value and **machine_type** attribute value to use the values from “**www_instances**” variable

```
resource "google_compute_instance" "my_www_vms" {
  for_each      = var.www_instances
  project       = google_compute_network.my_vpc.project
  name          = "${var.vm_prefix}-${each.value.name_suffix}"
  machine_type  = "${each.value.machine_type}"
}
```

Command:

```
terraform plan
```

- Execute **Terraform apply** to actually create the resources.

Command:

```
terraform apply
```

Task 2.6 : Modify code to output all the 3 VM Names and their ip addresses using **for** expression

File: Outputs.tf

```
output "vms" {
  description = "The www VM names."
  value       = ({
    for _,vm in google_compute_instance.my_www_vms
    : vm.name => vm.network_interface[0].network_ip
  })
}
```

Task 2.7 : Create an Unmanaged Instance group and add 3 VM instances.

- Also define an “http” port that points to port 80
  Provider Reference

File: Main.tf

```
resource "google_compute_instance_group" "my_www_vms" {
  name        = "www-servers"
  description = "The instance group to the www servers"
  project     = google_compute_network.my_vpc.project
  instances   = [for _,vm in google_compute_instance.my_www_vms
: vm.self_link]

  named_port {
    name = "http"
    port = "80"
  }

  zone = local.gce_zone
}
```

- Execute **Terraform plan**, review the resources that will be created.

Command:

```
terraform plan
```

- Execute **Terraform apply** to actually create the resources.

Command:

```
terraform apply
```

Lab 3

Task 3.1 : Add an L4 Internal load balancer

- Add an L4 Internal Load Balancer (ILB), using the corresponding Fabric module.
- It should point to the UIG just created earlier.

File: Main.tf

```
module "my_ilb" {
  source        =
"github.com/GoogleCloudPlatform/cloud-foundation-fabric//module
s/net-lb-int"
  project_id    = google_compute_network.my_vpc.project
  region        = "us-central1"
  name          = "lb-test"
  ports         = [80]

  vpc_config    = {
    network       = google_compute_network.my_vpc.self_link
    subnetwork    =
google_compute_subnetwork.my_subnet.self_link
  }

  backends = [{
    group          =
google_compute_instance_group.my_www_vms.self_link
    }
  ]
}
```

Task 3.2 : Activate Cloud NAT

- Allow Internet connection to VMs by configuring Cloud NAT using the corresponding Fabric Module.

- Activate Cloud NAT in us-central1 region

File: Main.tf

```
module "my_nat" {
  source         =
"github.com/GoogleCloudPlatform/cloud-foundation-fabric//modules/net-cloudnat"
  name           = "my-nat"
  project_id     = google_compute_network.my_vpc.project
  region         = "us-central1"
  router_network = google_compute_network.my_vpc.self_link
}
```

Task 3.3 : Allow health checks to contact VM by configuring firewall rule

- It should match source 35.191.0.0/16 and 130.211.0.0/22
- Matches destination protocol tcp, port 80
- It should be applied to the instances tagged www
  Provider Reference

File: Main.tf

```
  resource "google_compute_firewall" "allow_hc" {
    name          = "allow-hc"
    project       = google_compute_network.my_vpc.project
    network       = google_compute_network.my_vpc.id
    source_ranges = ["35.191.0.0/16", "130.211.0.0/22"]
    target_tags   = ["www"]

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

Task 3.4 : Create a client VM

Step 3.4.1 : Create client VM using gcloud command

- Manually create a client VM in GCP using gcloud, in order to test your connectivity.

- Name it my-client, resides in us-central1, runs debian-cloud/debian-11 and machine type is set to e2-micro.
  gcloud reference

Command: replace project_id, vpc_id and subnet_id with appropriate values

```
gcloud compute instances create my-client \
   --project="<project_id>" \
   --image-project=debian-cloud \
   --image-family=debian-11 \
   --machine-type=e2-micro \
   --zone=us-central1-b \
   --network="<vpc_id>" \
   --subnet="<subnet_id>" \
   --no-address \
   --no-service-account \
   --no-scopes
```

Step 3.4.2 : Test connectivity from client to www backend

Command: replace <my_project_id> with you project id

```
gcloud compute forwarding-rules list \
  --project="my_project_id" \
  | grep IP_ADDRESS

# Connect to your client VM
gcloud compute ssh my-client \
  --zone=us-central1-b \
  --project=my_project_id

$ curl x.x.x.x
```

Task 3.5 : Import client VM created manually to Terraform State file

Step 3.5.1 : Write terraform first, then import the state

Reference

File: **Main.tf**

```
resource "google_compute_instance" "my_client" {
  name         = "my-client"
  project      = google_compute_network.my_vpc.project
  machine_type = "e2-micro"
  zone         = "us-central1-b"

  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-11"
    }
  }
  network_interface {
    network    = google_compute_network.my_vpc.id
    subnetwork = google_compute_subnetwork.my_subnet.id
  }
}
```

Step 3.5.2 : Import the GCE instance

Terraform import reference

Command:

```
terraform import google_compute_instance.my_client \
projects/<YOUR_PROJECT_ID>/zones/us-central1-b/instances/my-client
```

Task 3.6 : Transfer the Terraform state from local to remote

Step 3.6.1 : Create GCS bucket

- Give it a unique name.
- Configure uniform bucket access level.
- Make it europe-based.
- Enable versioning.
- Set force destroy to true.

File: Main.tf

```
resource "google_storage_bucket" "default" {
  name                          = "${google_compute_network.my_vpc.project}-bucket-tfstate"
  project                       = google_compute_network.my_vpc.project
  location                    = "US"
  uniform_bucket_level_access = true
  force_destroy               = true

  versioning {
    enabled = true
  }
}
```

Step 3.6.2 : Setup your shared backend configuration

File: backend.tf - replace <YOUR_BUCKET_NAME>

```
terraform {
  backend "gcs" {
    bucket = "YOUR_BUCKET_NAME"
    prefix = "terraform/state"
  }
}
```

Step 3.6.3 : Transfer local state to shared GCS bucket

- Re-initialize Terraform to migrate the state

Command:

```
terraform init [-migrate-state]
```

Step 3.6.4 : Verify the new state

Command:

```
cat .terraform/terraform.tfstate
```

Example Output:

Verify for your cloud storage bucket name in the output

```
…
"backend": {
  "type": "gcs",
  "config": { ...
    "bucket": "YOUR_BUCKET_NAME",
```

Lab 4 (Optional)

Clean up all your resources
Task 4.1 : Comment / remove remote state config in backend.tf
File: backend.tf

```
terraform {
  ...
  /*
  backend "gcs" {
    bucket = "YOUR_BUCKET_NAME"
    prefix = "terraform/state"
  }
  */
}
```

Task 4.2 : Migrate back to local state
Command:

```
terraform init [-migrate-state]
```

Task 4.3 : Destroy everything with Terraform
Command:

```
terraform destroy
```

Summary

After completing this lab you should have a firm grasp of the following concepts:

- Basic Terraform constructs like providers, resources etc..
- Define variables & outputs
- Usage of Count, for_each, for loops etc..

- Terraform state management
- Terraform advanced commands like state, import, replace etc…
- Write Terraform code consuming external modules
- Terraform root module structure and best practices