Project 3 – Organizing and Reusing Configuration with Terraform Modules (Kavitha B)

Objective:

- Creating a Terraform configuration with a module to automate the deployment of Google Cloud infrastructure.
- As the configuration changes, Terraform can create incremental execution plans, which allows to build the overall configuration step by step.
- The instance module allows to re-use the same resource configuration for multiple resources while providing properties as input variables. You can leverage the configuration and module that are created as a starting point for future deployments.

Step 1: Setup Terraform and Cloud Shell

Terraform –version
Create a directory-tfinfra

Provider.tf

```
provider "google" {}
```

Terraform uses a plugin-based architecture to support the numerous infrastructure and service providers available. Each "provider" is its own encapsulated binary distributed separately from Terraform itself. Initialize Terraform by setting Google as the provider.
To initialize terraform:
In tfinfra folder,
Command: Terraform init

Step 2: Create mynetwork and its resources

Tfinfra/mynetwork.tf

```
# Create the mynetwork network
resource "google_compute_network" "mynetwork" {
name = "mynetwork"
# RESOURCE properties go here
auto_create_subnetworks = "true"
}
# Add a firewall rule to allow HTTP, SSH, RDP and ICMP traffic on mynetwork
resource "google_compute_firewall" "mynetwork-allow-http-ssh-rdp-icmp" {
name = "mynetwork-allow-http-ssh-rdp-icmp"
# RESOURCE properties go here
network = google_compute_network.mynetwork.self_link
allow {
   protocol = "tcp"
   ports    = ["22", "80", "3389"]
   }
allow {
   protocol = "icmp"
   }
```

Project 3 – Organizing and Reusing Configuration with Terraform Modules (Kavitha B)

```
 source_ranges = ["0.0.0.0/0"]
 }
 # Create the mynet-us-vm instance
 module "mynet-us-vm" {
  source          = "./instance"
  instance_name    = "mynet-us-vm"
  instance_zone    = "Zone"
  instance_network = google_compute_network.mynetwork.self_link
 }
 # Create the mynet-eu-vm" instance
 module "mynet-eu-vm" {
  source          = "./instance"
  instance_name    = "mynet-eu-vm"
  instance_zone    = "Zone"
  instance_network = google_compute_network.mynetwork.self_link
 }
```

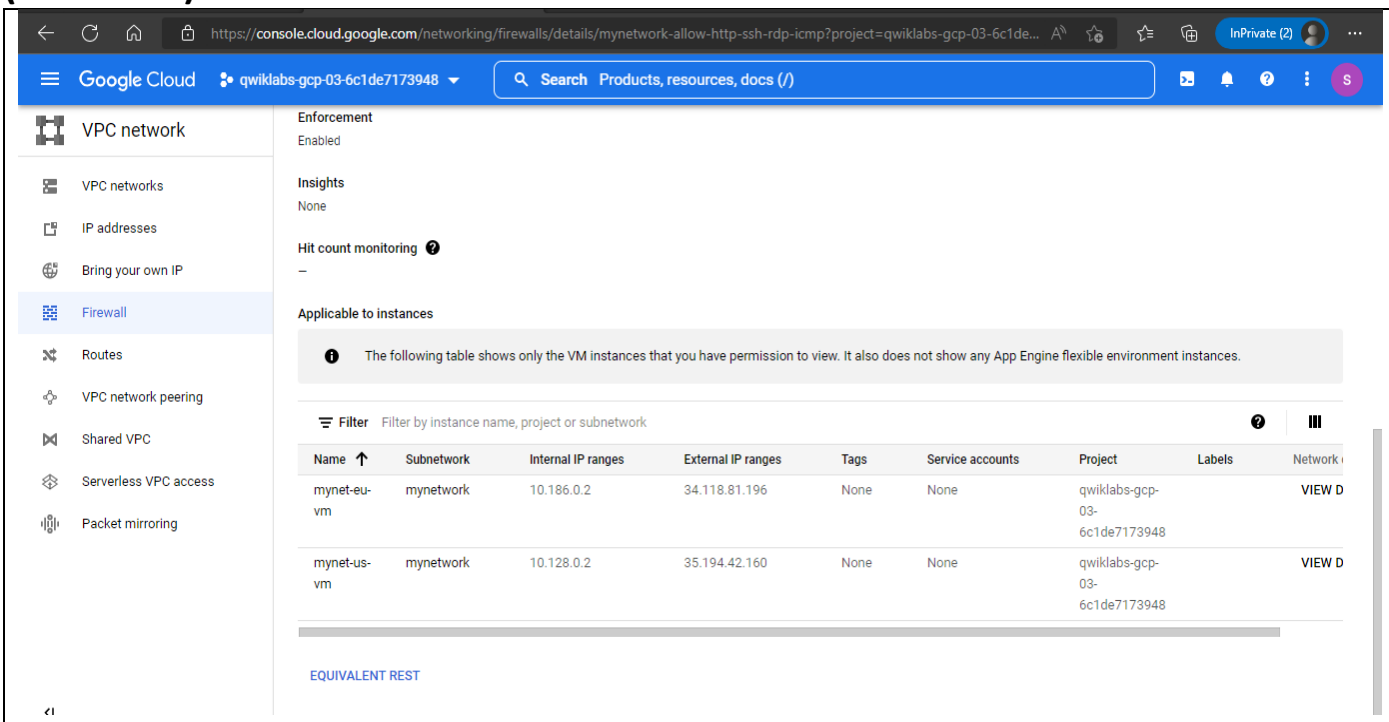
- **Configure mynetwork**

google_compute_network resource is a VPC network
auto_create_subnetworks to **true** :
Auto mode network automatically creates a subnetwork in each region.

- **Firewall rule configuration**

This firewall rule depends on its network, using **google_compute_network.mynetwork.self_link** reference to instruct Terraform to resolve these resources in a dependent order.
In this case, the network is created before the firewall rule.
Allow protocol-tcp(ports:22,80,3389)
Allow protocol-icmp
Source Ranges: [“0.0.0.0/0”]

Project 3 – Organizing and Reusing Configuration with Terraform Modules (Kavitha B)

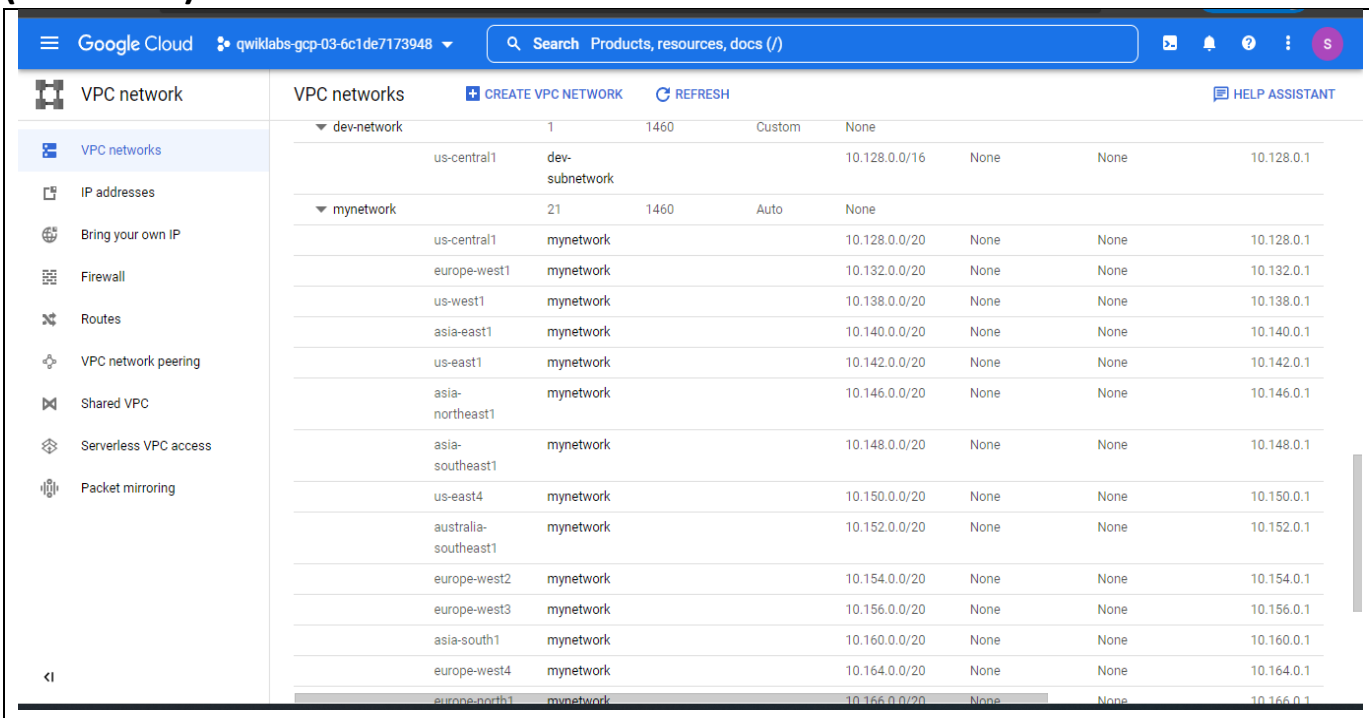


- **Adding the VM instances**

These resources are leveraging the module in the **instance** folder and provide the name, zone, and network as inputs. Because these instances depend on a VPC network, you are using the **google_compute_network.mynetwork.self_link** reference to instruct Terraform to resolve these resources in a dependent order. In this case, the network is created before the instance.

The benefit of writing a Terraform module is that it can be reused across many configurations. Instead of writing your own module, you can also leverage existing modules from the Terraform Module registry.

Project 3 – Organizing and Reusing Configuration with Terraform Modules (Kavitha B)



Step 3: Configure the VM instance/s

To define the VM instances create a VM instance module. A module is a reusable configuration inside a folder. In this lab, using this module for both VM instances.

Create a new folder instance inside the tfinfra folder.
/tfinfra/instance/main.tf

```
resource "google_compute_instance" "vm_instance" {
  name         = "${var.instance_name}"
  zone         = "${var.instance_zone}"
  machine_type = "${var.instance_type}"
  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-11"
      }
  }
  network_interface {
    network = "${var.instance_network}"
    access_config {
      # Allocate a one-to-one NAT IP to the instance
    }
  }
}
```

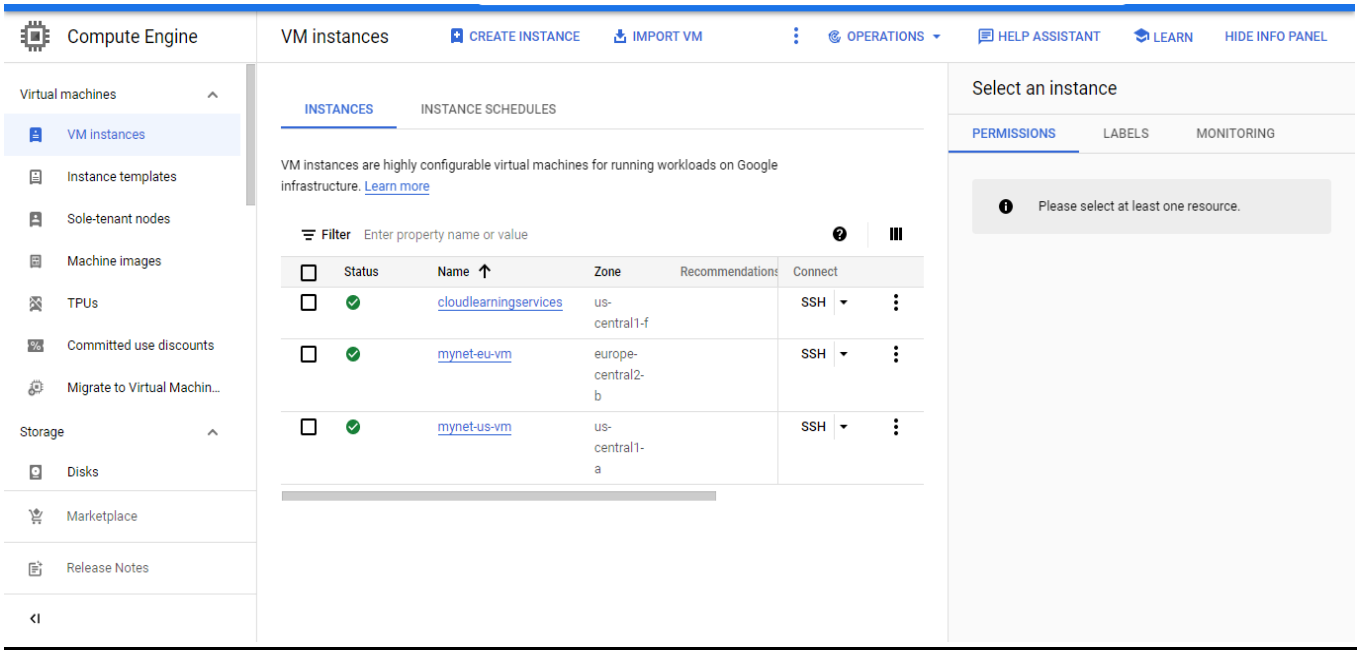
google_compute_instance - resource is a Compute Engine instance

Project 3 – Organizing and Reusing Configuration with Terraform Modules (Kavitha B)

zone and machine_type - properties define the zone and machine type of the instance as input variables.

Boot_disk - This property defines the boot disk to use the Debian 11 OS image. Because both VM instances will use the same image, this property is hard-coded in the module.

Network_interface - This property defines the network interface by providing the network name as an input variable and the access configuration. Leaving the access configuration empty results in an ephemeral external IP address (required in this project). To create instances with only an internal IP address, remove the access_config section.



Step 4: Configure variables

/tfinfra/instance/variables.tf

```
variable "instance_name" {}
variable "instance_zone" {}
variable "instance_type" {
  default = "n1-standard-1"
  }
variable "instance_network" {}
```

By giving **instance_type** a default value, you make the variable optional.

Step 5: Create mynetwork and its resources

This step is to apply the mynetwork configuration.

To rewrite the Terraform configuration files to a canonical format and style, run the following command:

Command : terraform fmt

Project 3 – Organizing and Reusing Configuration with Terraform Modules (Kavitha B)

Next,

Command : Terraform init

Command : Terraform plan

Terraform determined that the following 4 resources need to be added:

Name	Description
mynetwork	VPC network
mynetwork-allow-http-ssh-rdp-icmp	Firewall rule to allow HTTP, SSH, RDP and ICMP
mynet-us-vm	VM instance in `Zone`
mynet-eu-vm	VM instance in `Zone`

To effect the changes below command is run-

Command: terraform apply

Step 6:Verify the deployment

Verify the network in the Cloud Console

1. In the Cloud Console, on the **Navigation menu** (☰), click **VPC network** > **VPC networks**.
2. View the **mynetwork** VPC network with a subnetwork in every region.
3. On the **Navigation menu**, click **VPC network** > **Firewall**.
4. Sort the firewall rules by **Network**.

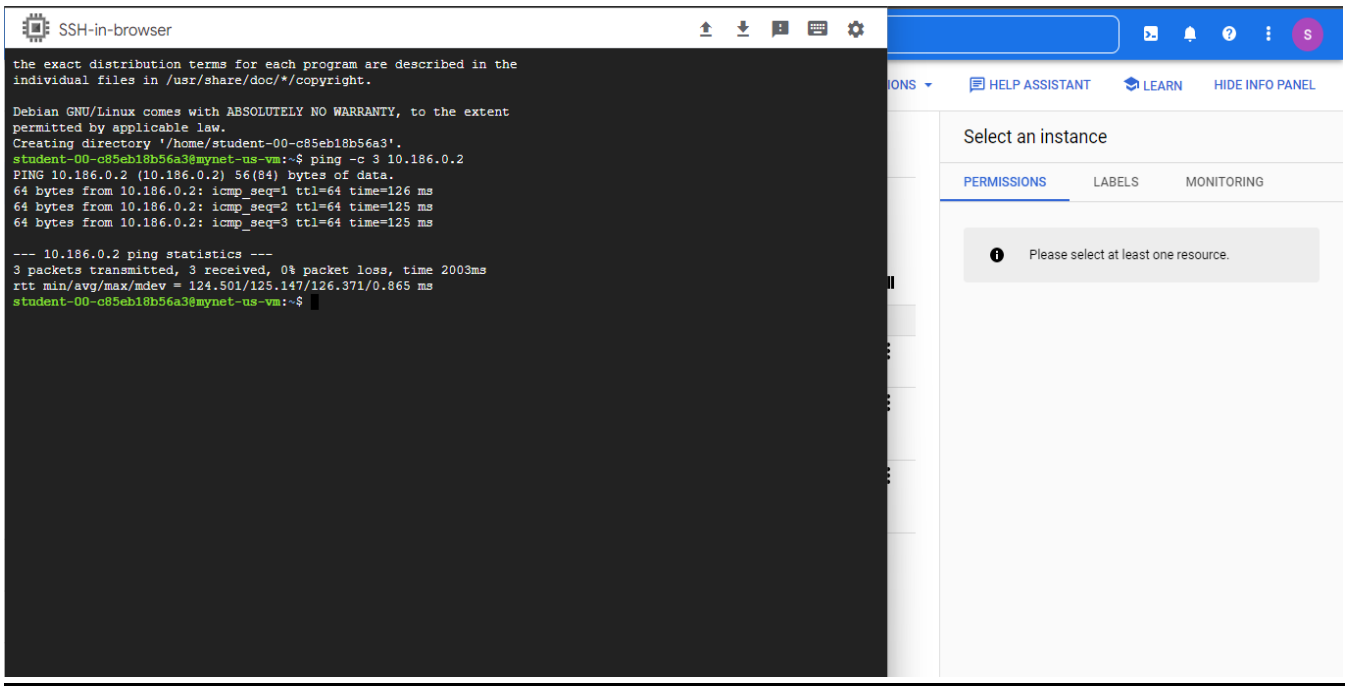
5. View the **mynetwork-allow-http-ssh-rdp-icmp** firewall rule for **mynetwork**.

Verify the VM instances in the Cloud Console

1. On the **Navigation menu** (☰), click **Compute Engine** > **VM instances**.
2. View the **mynet-us-vm** and **mynet-eu-vm** instances.
3. Note the internal IP address for **mynet-eu-vm**.
4. For **mynet-us-vm**, click **SSH** to launch a terminal and connect.
5. To test connectivity to **mynet-eu-vm**'s internal IP address, run the following command in the SSH terminal (replacing mynet-eu-vm's internal IP address with the value noted earlier):

   Command: ping -c 3 < mynet-eu-vm's internal IP >

(This should work because both VM instances are on the same network, and the firewall rule allows ICMP traffic)



Step 7: Destroy created resources to avoid charges

Project 3 – Organizing and Reusing Configuration with Terraform Modules (Kavitha B)

Accomplished:

- Deploy one auto mode network with a firewall rule and two VM instances.
- Create a configuration for an auto mode network
- Create a configuration for a firewall rule
- Create a module for VM instances
- Create and deploy a configuration
- Verify the deployment of a configuration