**Challenge #1**

**A 3-tier environment is a common setup. Use a tool of your choosing/familiarity create these**

**resources. Please remember we will not be judged on the outcome but more focusing on the**

**approach, style and reproducibility**.

Answer:

For the above task we are going to create an 3tier environment by using Google Cloud Platform(GCP). So inorder to do this we are creating the Custom VPC , Instance Group and Load Balancer.

creating a 3 Tier environment . For this I am creating a VPC Network and Two Subnets. After that i am creating an instance template which I am going to use in the instance group while creating. And after that I am going to create an HTTP Load Balancer using all of the aboveResources. For solving the above task ,I am using a IAC tool Terraform. I have wriiten a terraform code for creating the desired environment.

After writing the “main.tf” file ,which container details of the provider(GCP) and resources.

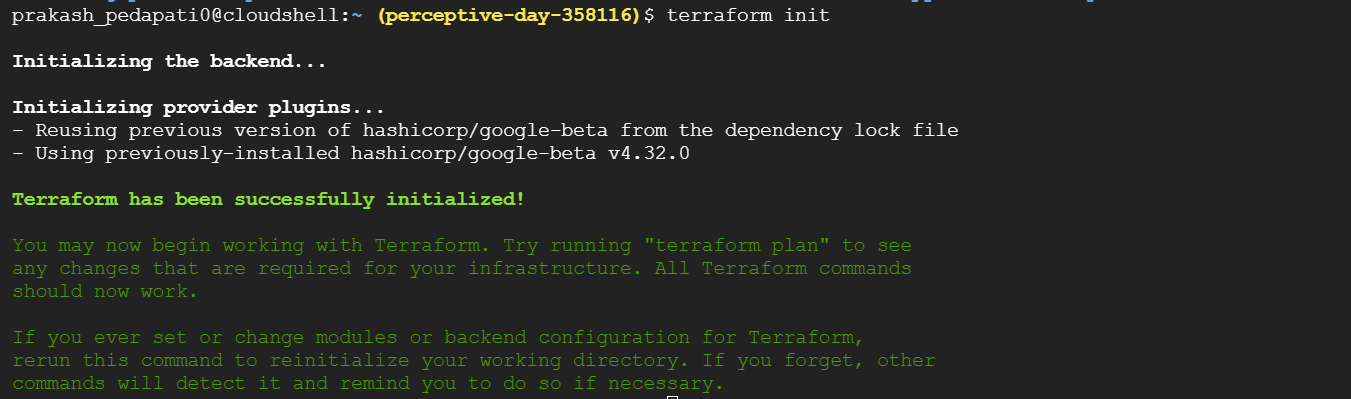
First we need to run “terraform init” So it downloads all the dependencies it need . All the dependencies it downloaded are stored in a file “.terraform” , which was automatically created when the “terraform init” command is Run. below is an screenshot of the output after using terraform init.

Terraform code:

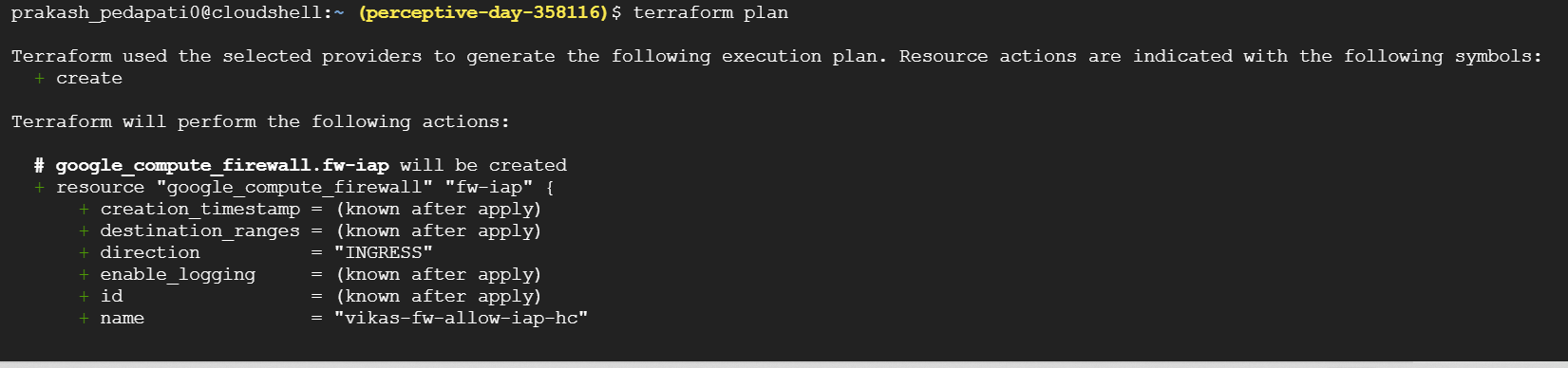
--------------------------------------------------------------------------------------------------------------------------------------

#main.tf

|  |
| --- |
| # VPC network |
|  | resource "google\_compute\_network" "vpc\_network" { |
|  | name = "vpc-network" |
|  | provider = google-beta |
|  | auto\_create\_subnetworks = false |
|  | } |
|  |  |
|  | # proxy-only subnet |
|  | resource "google\_compute\_subnetwork" "proxy\_subnet" { |
|  | name = "vikas-subnet1" |
|  | provider = google-beta |
|  | ip\_cidr\_range = "10.0.1.0/24" |
|  | region = "asia-east1" |
|  | purpose = "INTERNAL\_HTTPS\_LOAD\_BALANCER" |
|  | role = "ACTIVE" |
|  | network = google\_compute\_network.vpc\_network.id |
|  | } |
|  |  |
|  | # backend subnet |
|  | resource "google\_compute\_subnetwork" "vpc\_subnet" { |
|  | name = "vikas-subnet2" |
|  | provider = google-beta |
|  | ip\_cidr\_range = "10.0.2.0/24" |
|  | region = "asia-east1" |
|  | network = google\_compute\_network.vpc\_network.id |
|  | } |
|  |  |
|  | # forwarding rule |
|  | resource "google\_compute\_forwarding\_rule" "google\_compute\_forwarding\_rule" { |
|  | name = "vikas-forwarding-rule" |
|  | provider = google-beta |
|  | region = "asia-east1" |
|  | depends\_on = [google\_compute\_subnetwork.proxy\_subnet] |
|  | ip\_protocol = "TCP" |
|  | load\_balancing\_scheme = "INTERNAL\_MANAGED" |
|  | port\_range = "80" |
|  | target = google\_compute\_region\_target\_http\_proxy.default.id |
|  | network = google\_compute\_network.vpc\_network.id |
|  | subnetwork = google\_compute\_subnetwork.vpc\_subnet.id |
|  | network\_tier = "PREMIUM" |
|  | } |
|  |  |
|  | # HTTP target proxy |
|  | resource "google\_compute\_region\_target\_http\_proxy" "default" { |
|  | name = "vikas-target-http-proxy" |
|  | provider = google-beta |
|  | region = "asia-east1" |
|  | url\_map = google\_compute\_region\_url\_map.default.id |
|  | } |
|  |  |
|  | # URL map |
|  | resource "google\_compute\_region\_url\_map" "default" { |
|  | name = "vikas-regional-url-map" |
|  | provider = google-beta |
|  | region = "asia-east1" |
|  | default\_service = google\_compute\_region\_backend\_service.default.id |
|  | } |
|  |  |
|  | # backend service |
|  | resource "google\_compute\_region\_backend\_service" "default" { |
|  | name = "vikas-backend-subnet" |
|  | provider = google-beta |
|  | region = "asia-east1" |
|  | protocol = "HTTP" |
|  | load\_balancing\_scheme = "INTERNAL\_MANAGED" |
|  | timeout\_sec = 10 |
|  | health\_checks = [google\_compute\_region\_health\_check.default.id] |
|  | backend { |
|  | group = google\_compute\_region\_instance\_group\_manager.mig.instance\_group |
|  | balancing\_mode = "UTILIZATION" |
|  | capacity\_scaler = 1.0 |
|  | } |
|  | } |
|  |  |
|  | # instance template |
|  | resource "google\_compute\_instance\_template" "instance\_template" { |
|  | name = "vikas-template" |
|  | provider = google-beta |
|  | machine\_type = "e2-small" |
|  | tags = ["http-server"] |
|  |  |
|  | network\_interface { |
|  | network = google\_compute\_network.vpc\_network.id |
|  | subnetwork = google\_compute\_subnetwork.vpc\_subnet.id |
|  | access\_config { |
|  | # add external ip to fetch packages |
|  | } |
|  | } |
|  | disk { |
|  | source\_image = "debian-cloud/debian-10" |
|  | auto\_delete = true |
|  | boot = true |
|  | } |
|  |  |
|  | # install nginx and serve a simple web page |
|  | metadata = { |
|  | startup-script = <<-EOF2 |
|  | #! /bin/bash |
|  | set -euo pipefail |
|  | export DEBIAN\_FRONTEND=noninteractive |
|  | apt-get update |
|  | apt-get install -y nginx-light jq |
|  | NAME=$(curl -H "Metadata-Flavor: Google" "http://metadata.google.internal/computeMetadata/v2/instance/hostname") |
|  | IP=$(curl -H "Metadata-Flavor: Google" "http://metadata.google.internal/computeMetadata/v2/instance/network-interfaces/0/ip") |
|  | METADATA=$(curl -f -H "Metadata-Flavor: Google" "http://metadata.google.internal/computeMetadata/v2/instance/attributes/?recursive=True" | jq 'del(.["startup-script"])') |
|  | cat <<EOF > /var/www/html/index.html |
|  | <pre> |
|  | Name: $NAME |
|  | IP: $IP |
|  | Metadata: $METADATA |
|  | </pre> |
|  | EOF |
|  | EOF2 |
|  | } |
|  | lifecycle { |
|  | create\_before\_destroy = true |
|  | } |
|  | } |
|  |  |
|  | # health check |
|  | resource "google\_compute\_region\_health\_check" "default" { |
|  | name = "vikas-hc" |
|  | provider = google-beta |
|  | region = "asia-east1" |
|  | http\_health\_check { |
|  | port\_specification = "USE\_SERVING\_PORT" |
|  | } |
|  | } |
|  |  |
|  | # MIG |
|  | resource "google\_compute\_region\_instance\_group\_manager" "mig" { |
|  | name = "vikas-mig1" |
|  | provider = google-beta |
|  | region = "asia-east1" |
|  | version { |
|  | instance\_template = google\_compute\_instance\_template.instance\_template.id |
|  | name = "primary" |
|  | } |
|  | base\_instance\_name = "vm" |
|  | target\_size = 2 |
|  | } |
|  |  |
|  | # allow all access from IAP and health check ranges |
|  | resource "google\_compute\_firewall" "fw-iap" { |
|  | name = "vikas-fw-allow-iap-hc" |
|  | provider = google-beta |
|  | direction = "INGRESS" |
|  | network = google\_compute\_network.vpc\_network.id |
|  | source\_ranges = ["130.211.0.0/22", "35.191.0.0/16", "35.235.240.0/20"] |
|  | allow { |
|  | protocol = "tcp" |
|  | } |
|  | } |
|  |  |
|  | # allow http from proxy subnet to backends |
|  | resource "google\_compute\_firewall" "fw-ilb-to-backends" { |
|  | name = "vikas-fw-allow-ilb-to-backends" |
|  | provider = google-beta |
|  | direction = "INGRESS" |
|  | network = google\_compute\_network.vpc\_network.id |
|  | source\_ranges = ["10.0.1.0/24"] |
|  | target\_tags = ["http-server"] |
|  | allow { |
|  | protocol = "tcp" |
|  | ports = ["80", "443", "8080"] |
|  | } |
|  | } |
|  |  |
|  | # test instance |
|  | resource "google\_compute\_instance" "vm-test" { |
|  | name = "vikas-test-vm" |
|  | provider = google-beta |
|  | zone = "asia-east1-b" |
|  | machine\_type = "e2-small" |
|  | network\_interface { |
|  | network = google\_compute\_network.vpc\_network.id |
|  | subnetwork = google\_compute\_subnetwork.vpc\_subnet.id |
|  | } |
|  | boot\_disk { |
|  | initialize\_params { |
|  | image = "debian-cloud/debian-10" |
|  | } |
|  | } |
|  | }  ------------------------------------------------------------------------------ |



After using terraform init , then we need to run the command “terraform plan” . terraform plan let us know what the terraform is going to execute or going to do. Below is a screenshot of the output after using the 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\_firewall.fw-iap will be created

+ resource "google\_compute\_firewall" "fw-iap" {

+ creation\_timestamp = (known after apply)

+ destination\_ranges = (known after apply)

+ direction = "INGRESS"

+ enable\_logging = (known after apply)

+ id = (known after apply)

+ name = "vikas-fw-allow-iap-hc"

+ network = (known after apply)

+ priority = 1000

+ project = (known after apply)

+ self\_link = (known after apply)

+ source\_ranges = [

+ "130.211.0.0/22",

+ "35.191.0.0/16",

+ "35.235.240.0/20",

]

+ allow {

+ ports = []

+ protocol = "tcp"

}

}

# google\_compute\_firewall.fw-ilb-to-backends will be created

+ resource "google\_compute\_firewall" "fw-ilb-to-backends" {

+ creation\_timestamp = (known after apply)

+ destination\_ranges = (known after apply)

+ direction = "INGRESS"

+ enable\_logging = (known after apply)

+ id = (known after apply)

+ name = "vikas-fw-allow-ilb-to-backends"

+ network = (known after apply)

+ priority = 1000

+ project = (known after apply)

+ self\_link = (known after apply)

+ source\_ranges = [

+ "10.0.1.0/24",

]

+ target\_tags = [

+ "http-server",

]

+ allow {

+ ports = [

+ "80",

+ "443",

+ "8080",

]

+ protocol = "tcp"

}

}

# google\_compute\_forwarding\_rule.google\_compute\_forwarding\_rule will be created

+ resource "google\_compute\_forwarding\_rule" "google\_compute\_forwarding\_rule" {

+ creation\_timestamp = (known after apply)

+ id = (known after apply)

+ ip\_address = (known after apply)

+ ip\_protocol = "TCP"

+ label\_fingerprint = (known after apply)

+ load\_balancing\_scheme = "INTERNAL\_MANAGED"

+ name = "vikas-forwarding-rule"

+ network = (known after apply)

+ network\_tier = "PREMIUM"

+ port\_range = "80"

+ project = (known after apply)

+ psc\_connection\_id = (known after apply)

+ psc\_connection\_status = (known after apply)

+ region = "asia-east1"

+ self\_link = (known after apply)

+ service\_name = (known after apply)

+ subnetwork = (known after apply)

+ target = (known after apply)

+ service\_directory\_registrations {

+ namespace = (known after apply)

+ service = (known after apply)

}

}

# google\_compute\_instance.vm-test will be created

+ resource "google\_compute\_instance" "vm-test" {

+ 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 = "e2-small"

+ metadata\_fingerprint = (known after apply)

+ min\_cpu\_platform = (known after apply)

+ name = "vikas-test-vm"

+ project = (known after apply)

+ self\_link = (known after apply)

+ tags\_fingerprint = (known after apply)

+ zone = "asia-east1-b"

+ 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"

+ source = (known after apply)

+ initialize\_params {

+ image = "debian-cloud/debian-10"

+ labels = (known after apply)

+ size = (known after apply)

+ type = (known after apply)

}

}

+ confidential\_instance\_config {

+ enable\_confidential\_compute = (known after apply)

}

+ network\_interface {

+ ipv6\_access\_type = (known after apply)

+ name = (known after apply)

+ network = (known after apply)

+ network\_ip = (known after apply)

+ stack\_type = (known after apply)

+ subnetwork = (known after apply)

+ subnetwork\_project = (known after apply)

}

+ reservation\_affinity {

+ type = (known after 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)

}

}

}

# google\_compute\_instance\_template.instance\_template will be created

+ resource "google\_compute\_instance\_template" "instance\_template" {

+ can\_ip\_forward = false

+ id = (known after apply)

+ machine\_type = "e2-small"

+ metadata = {

+ "startup-script" = <<-EOT

#! /bin/bash

set -euo pipefail

export DEBIAN\_FRONTEND=noninteractive

apt-get update

apt-get install -y nginx-light jq

NAME=$(curl -H "Metadata-Flavor: Google" "http://metadata.google.internal/computeMetadata/v1/instance/hostname")

IP=$(curl -H "Metadata-Flavor: Google" "http://metadata.google.internal/computeMetadata/v1/instance/network-interfaces/0/ip")

METADATA=$(curl -f -H "Metadata-Flavor: Google" "http://metadata.google.internal/computeMetadata/v1/instance/attributes/?recursive=True" | jq 'del(.["startup-script"])')

cat <<EOF > /var/www/html/index.html

<pre>

Name: $NAME

IP: $IP

Metadata: $METADATA

</pre>

EOF

EOT

}

+ metadata\_fingerprint = (known after apply)

+ name = "vikas-template"

+ name\_prefix = (known after apply)

+ project = (known after apply)

+ region = (known after apply)

+ self\_link = (known after apply)

+ tags = [

+ "http-server",

]

+ tags\_fingerprint = (known after apply)

+ confidential\_instance\_config {

+ enable\_confidential\_compute = (known after apply)

}

+ disk {

+ auto\_delete = true

+ boot = true

+ device\_name = (known after apply)

+ disk\_size\_gb = (known after apply)

+ disk\_type = (known after apply)

+ interface = (known after apply)

+ mode = (known after apply)

+ source\_image = "debian-cloud/debian-10"

+ type = (known after apply)

}

+ network\_interface {

+ ipv6\_access\_type = (known after apply)

+ name = (known after apply)

+ network = (known after apply)

+ stack\_type = (known after apply)

+ subnetwork = (known after apply)

+ subnetwork\_project = (known after apply)

+ access\_config {

+ nat\_ip = (known after apply)

+ network\_tier = (known after apply)

+ public\_ptr\_domain\_name = (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)

}

}

}

# google\_compute\_network.vpc\_network will be created

+ resource "google\_compute\_network" "vpc\_network" {

+ auto\_create\_subnetworks = false

+ delete\_default\_routes\_on\_create = false

+ gateway\_ipv4 = (known after apply)

+ id = (known after apply)

+ internal\_ipv6\_range = (known after apply)

+ mtu = (known after apply)

+ name = "vpc-network"

+ project = (known after apply)

+ routing\_mode = (known after apply)

+ self\_link = (known after apply)

}

# google\_compute\_region\_backend\_service.default will be created

+ resource "google\_compute\_region\_backend\_service" "default" {

+ connection\_draining\_timeout\_sec = 0

+ creation\_timestamp = (known after apply)

+ fingerprint = (known after apply)

+ health\_checks = (known after apply)

+ id = (known after apply)

+ load\_balancing\_scheme = "INTERNAL\_MANAGED"

+ name = "vikas-backend-subnet"

+ port\_name = (known after apply)

+ project = (known after apply)

+ protocol = "HTTP"

+ region = "asia-east1"

+ self\_link = (known after apply)

+ session\_affinity = (known after apply)

+ timeout\_sec = 10

+ backend {

+ balancing\_mode = "UTILIZATION"

+ capacity\_scaler = 1

+ failover = (known after apply)

+ group = (known after apply)

}

+ cdn\_policy {

+ cache\_mode = (known after apply)

+ client\_ttl = (known after apply)

+ default\_ttl = (known after apply)

+ max\_ttl = (known after apply)

+ negative\_caching = (known after apply)

+ serve\_while\_stale = (known after apply)

+ signed\_url\_cache\_max\_age\_sec = (known after apply)

+ cache\_key\_policy {

+ include\_host = (known after apply)

+ include\_named\_cookies = (known after apply)

+ include\_protocol = (known after apply)

+ include\_query\_string = (known after apply)

+ query\_string\_blacklist = (known after apply)

+ query\_string\_whitelist = (known after apply)

}

+ negative\_caching\_policy {

+ code = (known after apply)

+ ttl = (known after apply)

}

}

+ log\_config {

+ enable = (known after apply)

+ sample\_rate = (known after apply)

}

}

# google\_compute\_region\_health\_check.default will be created

+ resource "google\_compute\_region\_health\_check" "default" {

+ check\_interval\_sec = 5

+ creation\_timestamp = (known after apply)

+ healthy\_threshold = 2

+ id = (known after apply)

+ name = "vikas-hc"

+ project = (known after apply)

+ region = "asia-east1"

+ self\_link = (known after apply)

+ timeout\_sec = 5

+ type = (known after apply)

+ unhealthy\_threshold = 2

+ http\_health\_check {

+ port\_specification = "USE\_SERVING\_PORT"

+ proxy\_header = "NONE"

+ request\_path = "/"

}

+ log\_config {

+ enable = (known after apply)

}

}

# google\_compute\_region\_instance\_group\_manager.mig will be created

+ resource "google\_compute\_region\_instance\_group\_manager" "mig" {

+ base\_instance\_name = "vm"

+ distribution\_policy\_target\_shape = (known after apply)

+ distribution\_policy\_zones = (known after apply)

+ fingerprint = (known after apply)

+ id = (known after apply)

+ instance\_group = (known after apply)

+ name = "vikas-mig1"

+ project = (known after apply)

+ region = "asia-east1"

+ self\_link = (known after apply)

+ status = (known after apply)

+ target\_size = 2

+ wait\_for\_instances = false

+ wait\_for\_instances\_status = "STABLE"

+ update\_policy {

+ instance\_redistribution\_type = (known after apply)

+ max\_surge\_fixed = (known after apply)

+ max\_surge\_percent = (known after apply)

+ max\_unavailable\_fixed = (known after apply)

+ max\_unavailable\_percent = (known after apply)

+ min\_ready\_sec = (known after apply)

+ minimal\_action = (known after apply)

+ most\_disruptive\_allowed\_action = (known after apply)

+ replacement\_method = (known after apply)

+ type = (known after apply)

}

+ version {

+ instance\_template = (known after apply)

+ name = "primary"

}

}

# google\_compute\_region\_target\_http\_proxy.default will be created

+ resource "google\_compute\_region\_target\_http\_proxy" "default" {

+ creation\_timestamp = (known after apply)

+ id = (known after apply)

+ name = "vikas-target-http-proxy"

+ project = (known after apply)

+ proxy\_id = (known after apply)

+ region = "asia-east1"

+ self\_link = (known after apply)

+ url\_map = (known after apply)

}

# google\_compute\_region\_url\_map.default will be created

+ resource "google\_compute\_region\_url\_map" "default" {

+ creation\_timestamp = (known after apply)

+ default\_service = (known after apply)

+ fingerprint = (known after apply)

+ id = (known after apply)

+ map\_id = (known after apply)

+ name = "vikas-regional-url-map"

+ project = (known after apply)

+ region = "asia-east1"

+ self\_link = (known after apply)

}

# google\_compute\_subnetwork.proxy\_subnet will be created

+ resource "google\_compute\_subnetwork" "proxy\_subnet" {

+ creation\_timestamp = (known after apply)

+ external\_ipv6\_prefix = (known after apply)

+ fingerprint = (known after apply)

+ gateway\_address = (known after apply)

+ id = (known after apply)

+ ip\_cidr\_range = "10.0.1.0/24"

+ ipv6\_cidr\_range = (known after apply)

+ name = "vikas-subnet1"

+ network = (known after apply)

+ private\_ipv6\_google\_access = (known after apply)

+ project = (known after apply)

+ purpose = "INTERNAL\_HTTPS\_LOAD\_BALANCER"

+ region = "asia-east1"

+ role = "ACTIVE"

+ secondary\_ip\_range = (known after apply)

+ self\_link = (known after apply)

+ stack\_type = (known after apply)

}

# google\_compute\_subnetwork.vpc\_subnet will be created

+ resource "google\_compute\_subnetwork" "vpc\_subnet" {

+ creation\_timestamp = (known after apply)

+ external\_ipv6\_prefix = (known after apply)

+ fingerprint = (known after apply)

+ gateway\_address = (known after apply)

+ id = (known after apply)

+ ip\_cidr\_range = "10.0.2.0/24"

+ ipv6\_cidr\_range = (known after apply)

+ name = "vikas-subnet2"

+ network = (known after apply)

+ private\_ipv6\_google\_access = (known after apply)

+ project = (known after apply)

+ purpose = (known after apply)

+ region = "asia-east1"

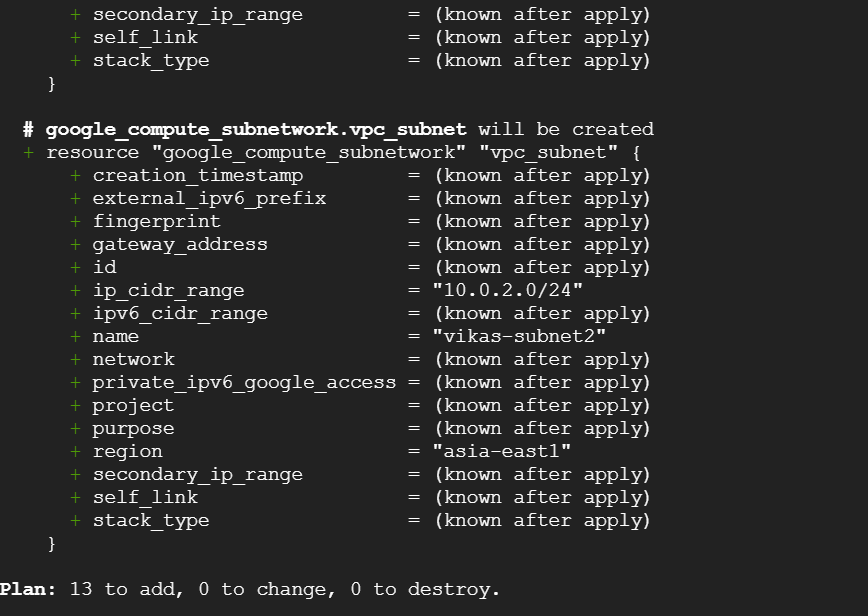
+ secondary\_ip\_range = (known after apply)

+ self\_link = (known after apply)

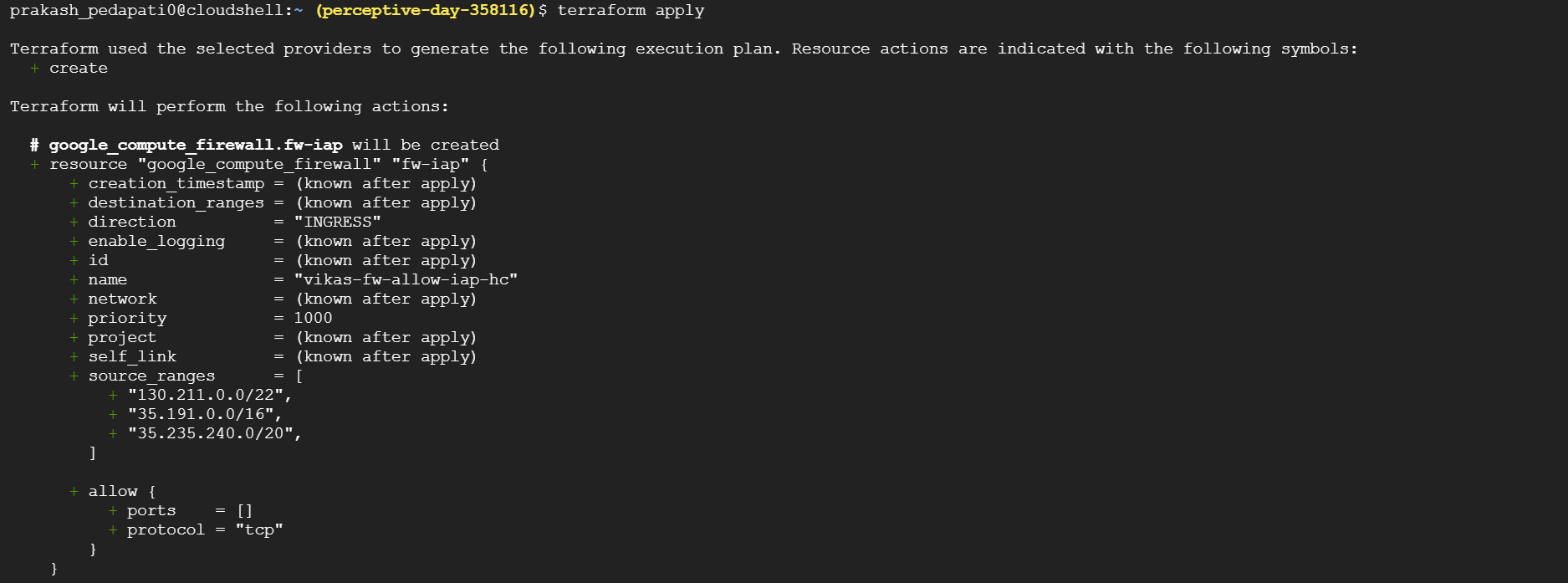
+ stack\_type = (known after apply)

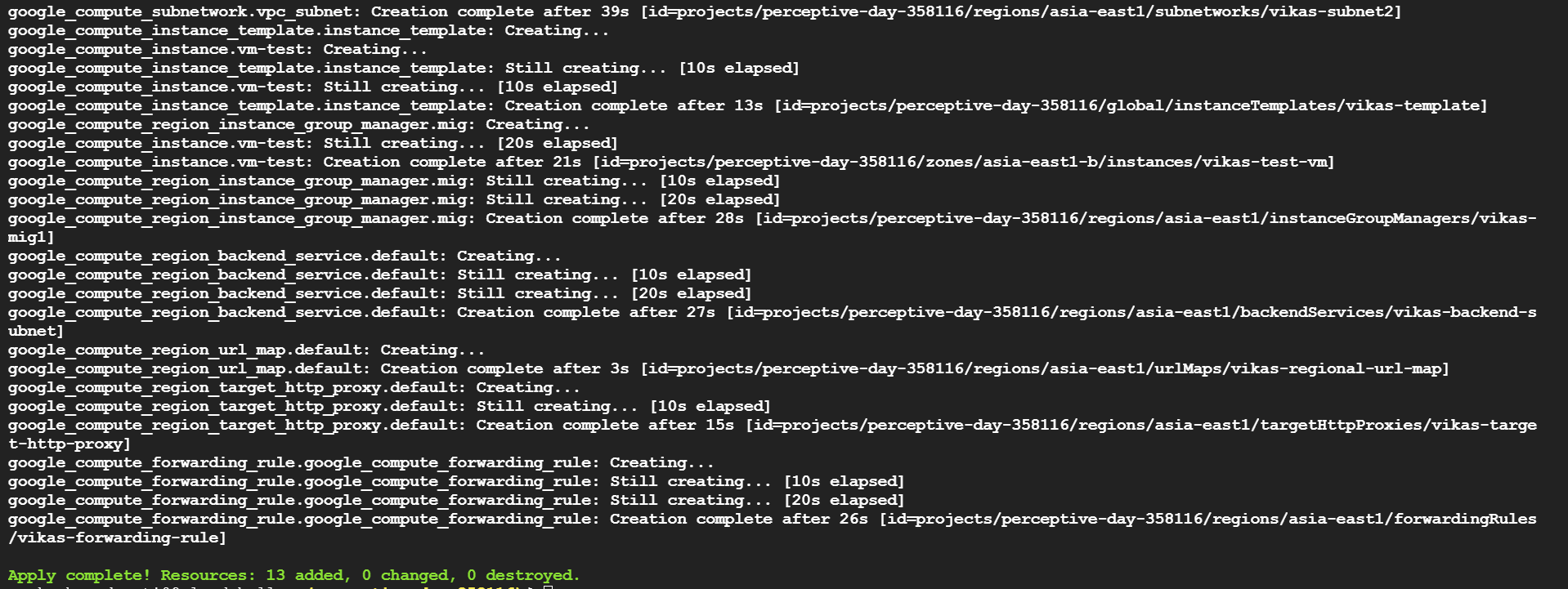
}

Plan: 13 to add, 0 to change, 0 to destroy.



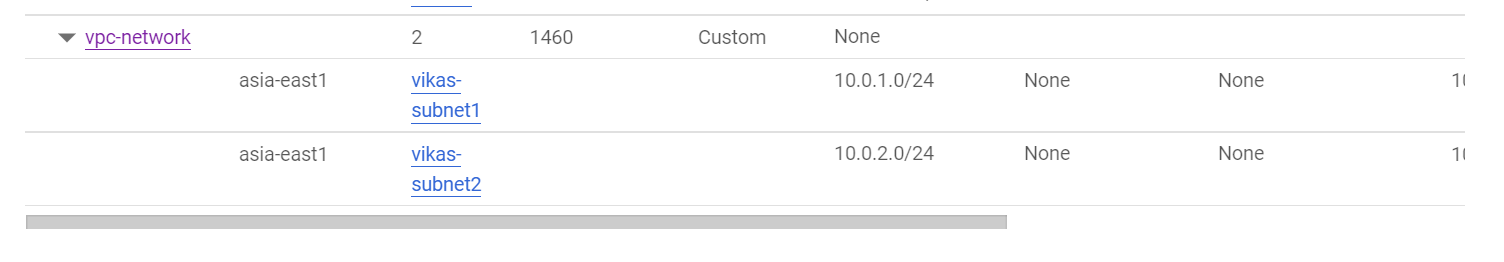
After using the terraform plan, we need to run the command “terraform apply”.this terraform apply command will do the following changes below as the terraform plan showed.below is a screenshot of the output of terraform apply.



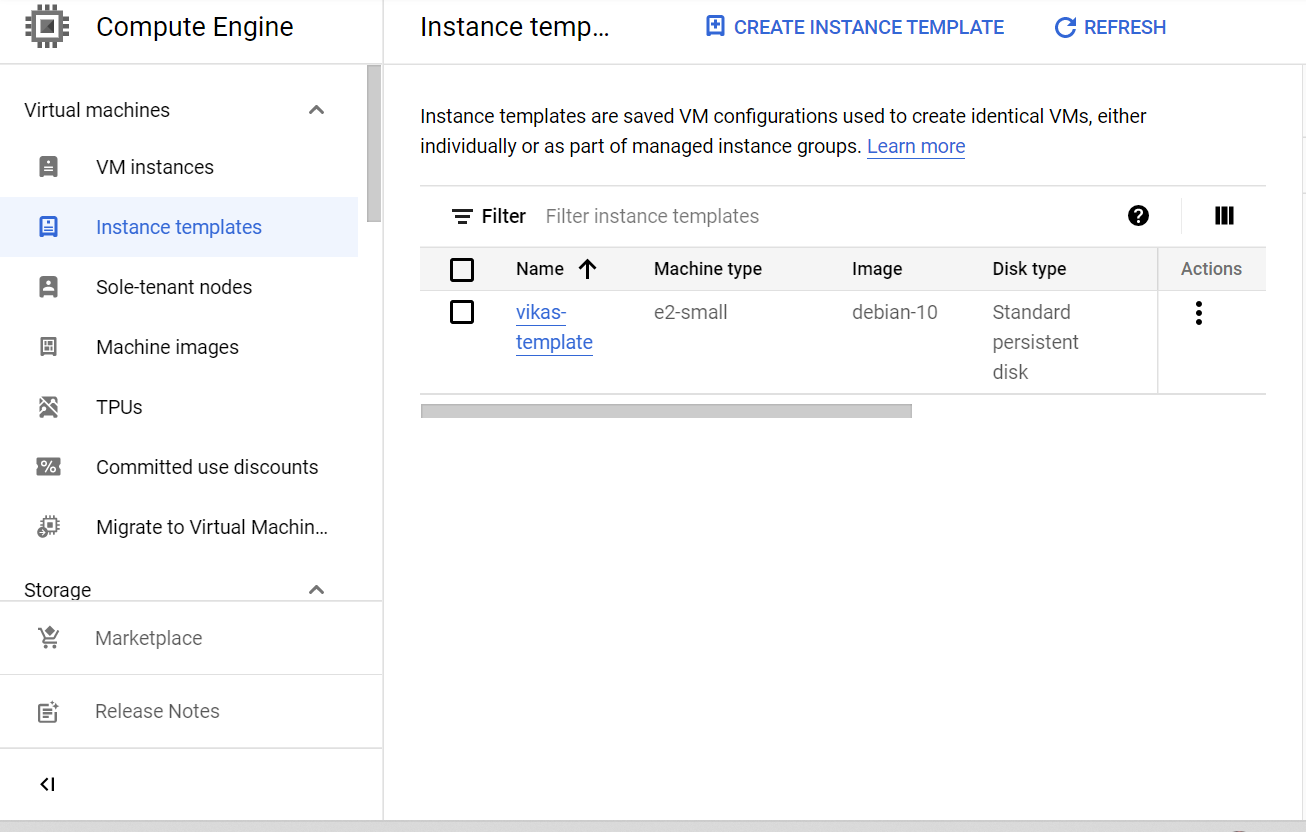


Now we can go and check wheather the Resources are created or not .

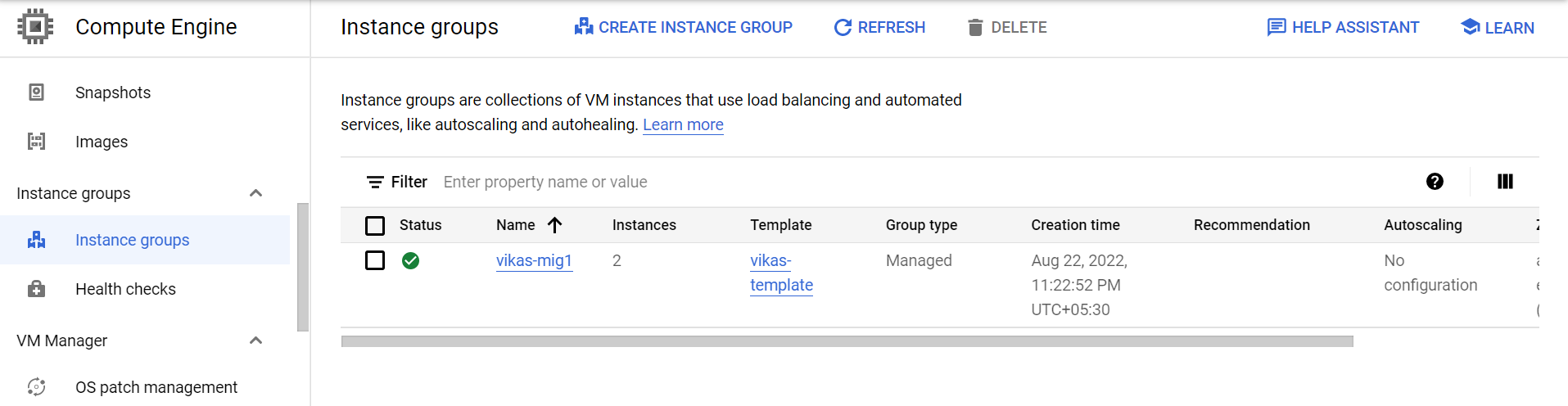
Below is an VPC Network “vpc-network” created by terraform



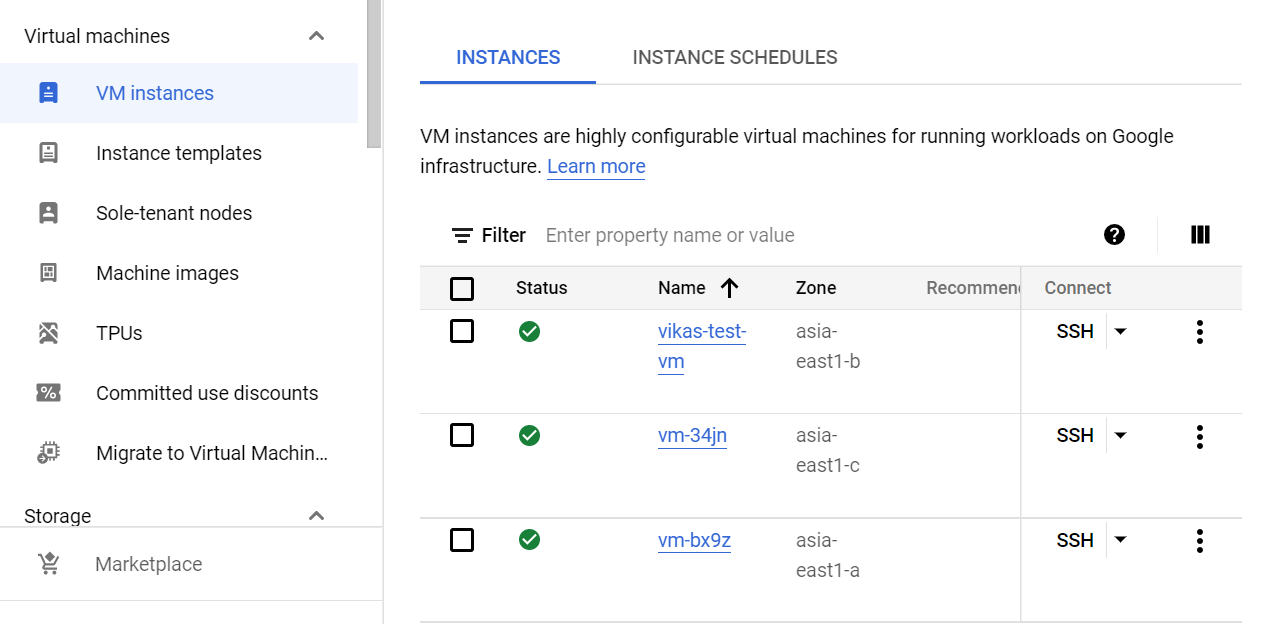
Below is the Instance template with the name “vikas-template” created by the terraform



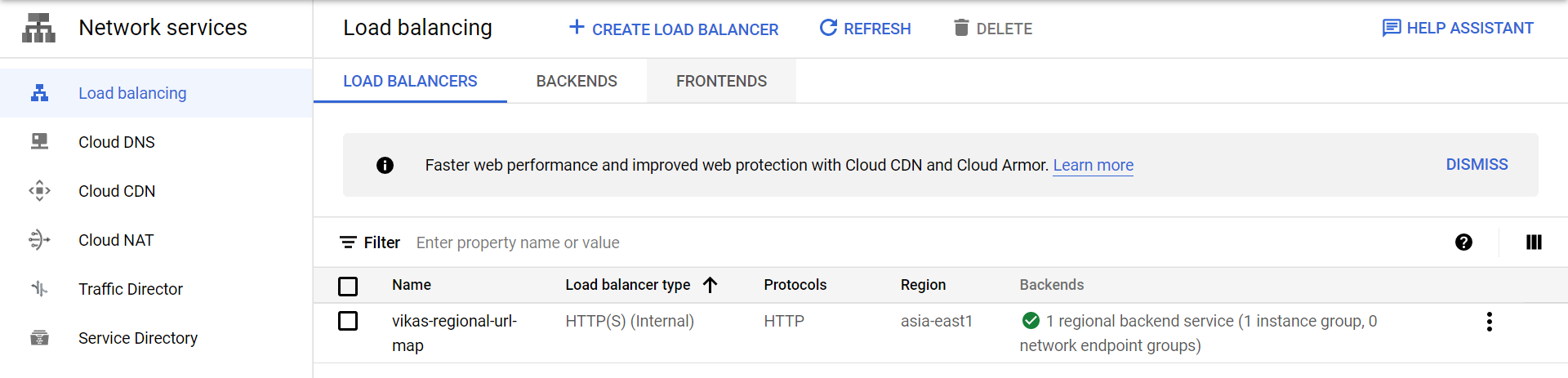
Below is the Instance Group Created by the terraform .



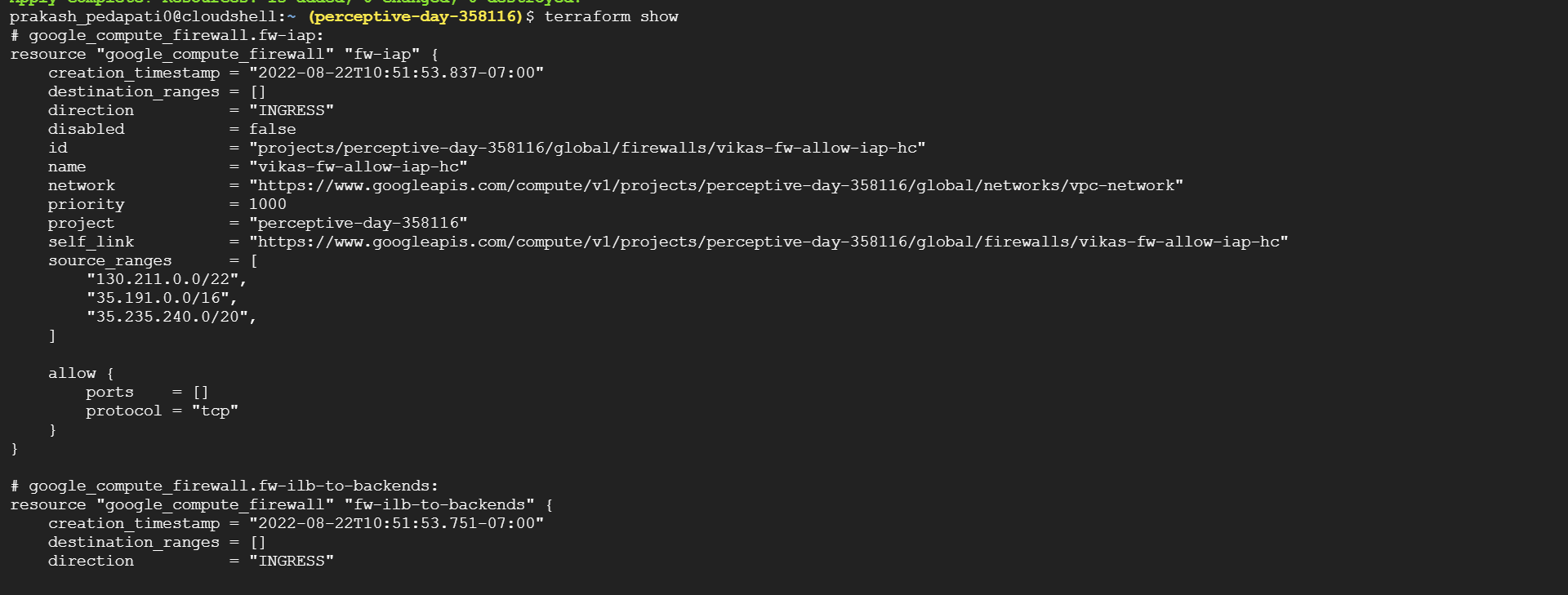
Below are the Instance created as part of the 3 tier environment.

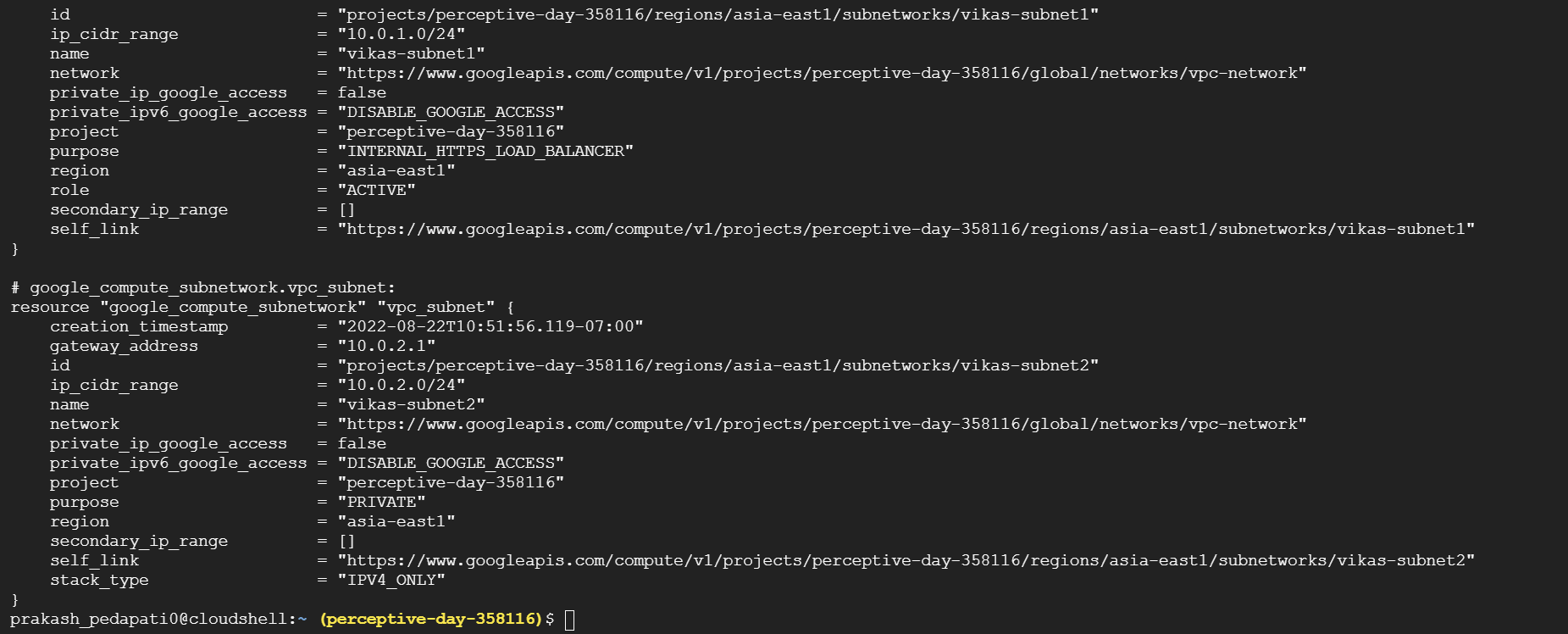


Below is the HTTP Load Balancer created by the terraform .

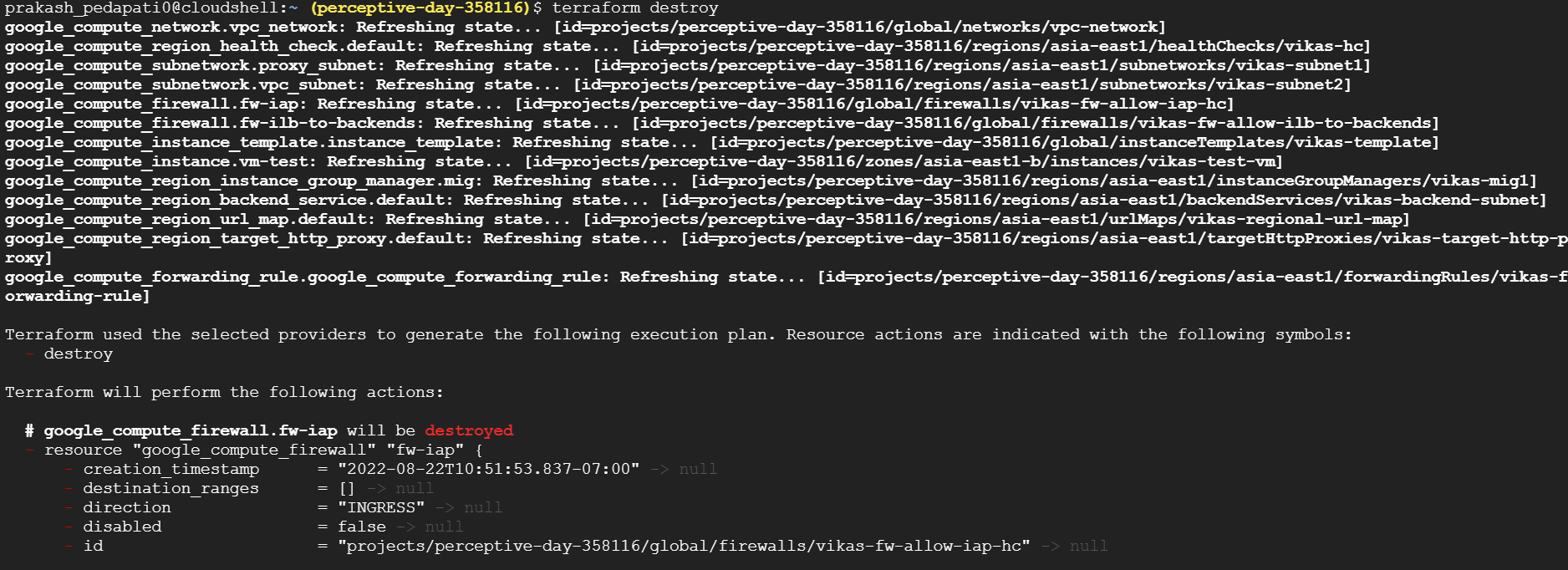


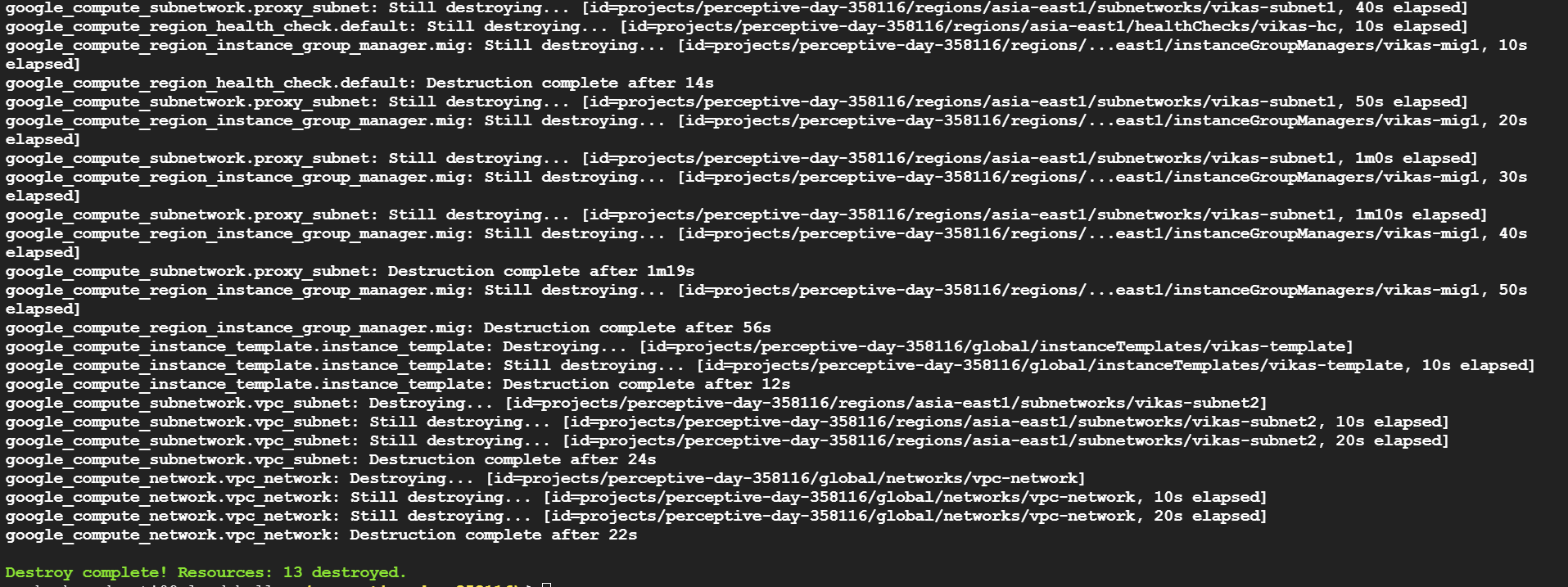
Terraform mainly groups the resources together and All the resources created by the terraform are in the terraforms control. The file “terraform.tf “ holds the information about the resources terraform created and if the terraform.tf file is deleted information in the terraform.tf file is lost . If we want to see the resources which are in the control of terraform,we can use the command “terraform show” .Below is the output of the “terraform show “ command.





Now for destroying the resources we can use the command “terraform destroy”. Below the output of the command “terraform destroy”.





CONCLUSION:

Using terraform we can create resource and also can group the resources together. By using the commands “terraform init “ terraform plan”, “terraform apply”, terraform destroy”, “terraform show” we can manage the resources with help of terraform code .

Make note that “.terraform” folder contains dependencies the terraform downloaded once we have used the “terraform init” and “terraform.tf” file contains information about the resources terraform is going to manage.