Name: Manav Jawrani

Roll No.: 19

Subject: Advanced DevOps

Experiment No.: 6

Experiment 6.

Aim: To Build, Change and destroy AWS / GCP / Microsoft Azure / Digital ocean infrastructure using Terraform.

Theory:

Terraform is an open-source "Infrastructure as Code" tool created by Hashi Corp.

- What is Infrastructure as Code?
  Infrastructure as Code (IaC) is a widespread terminology among DevOps professionals and a key DevOps practice in the industry. It is the process of managing and provisioning the complete infrastructure comprising of both physical and virtual machines using machine readable definition files.

- Terraform providers
  A provider is responsible for understanding API interactions and exposing resources. It is an executable plug-in that contains code necessary to interact with API of the service. Terraform configurations must declare which providers they require. So that Terraform can install and use them.

Terraform Configuration Files

Configuration files are a set of files used to describe infrastructure in Terraform and have the file extensions .tf and .tf.json. Terraform uses a declarative model for defining infrastructure. Configuration files let you write a configuration that declares your desired state. Configuration files are made up of resources with settings and values representing the desired state of your infrastructure.

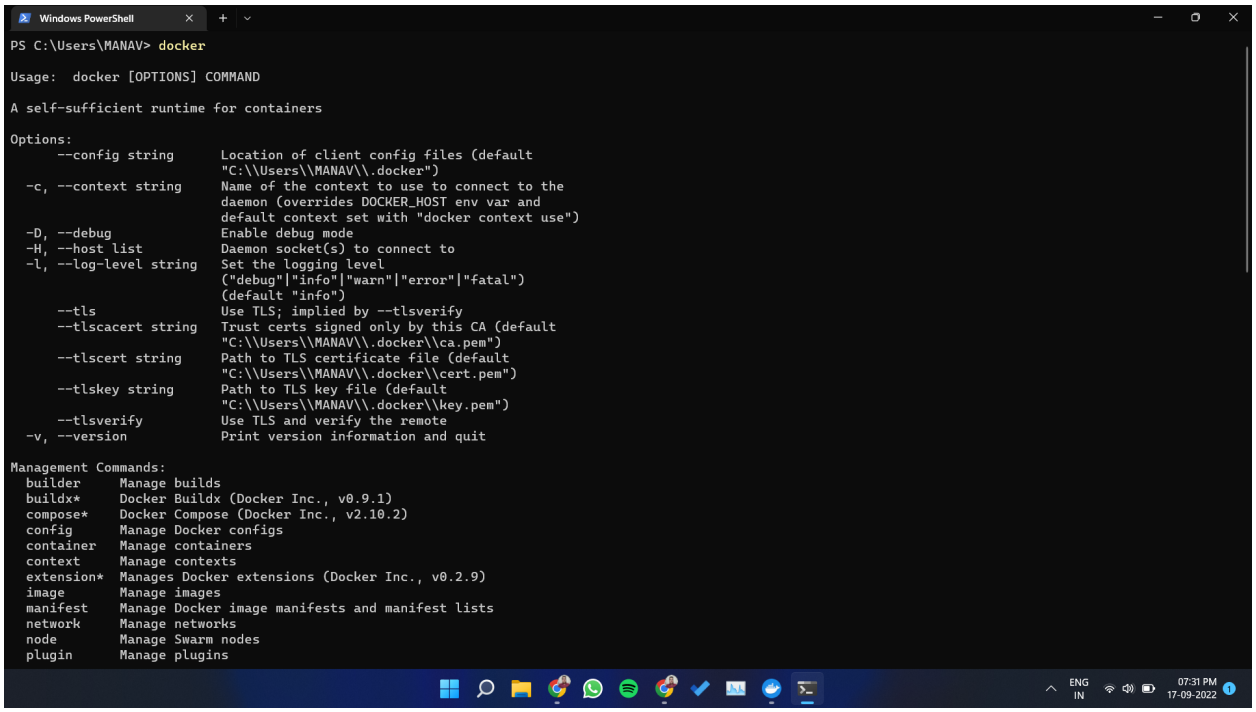
Implementation:

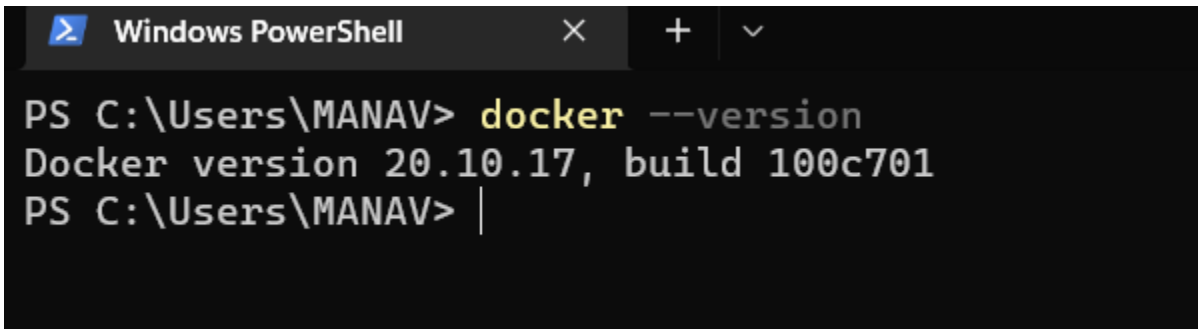
A. Creating docker image using terraform

Prerequisite:

1) Download and Install Docker Desktop from https://www.docker.com/

Step 1: Check the docker functionality





Now, create a folder named 'Terraform Scripts' in which we save our different types of scripts which will be further used in this experiment.

Step 2: Firstly create a new folder named ‘Docker’ in the ‘Terraform Scripts’ folder. Then create a new docker.tf file using Atom editor and write the following contents into it to create a Ubuntu Linux container.

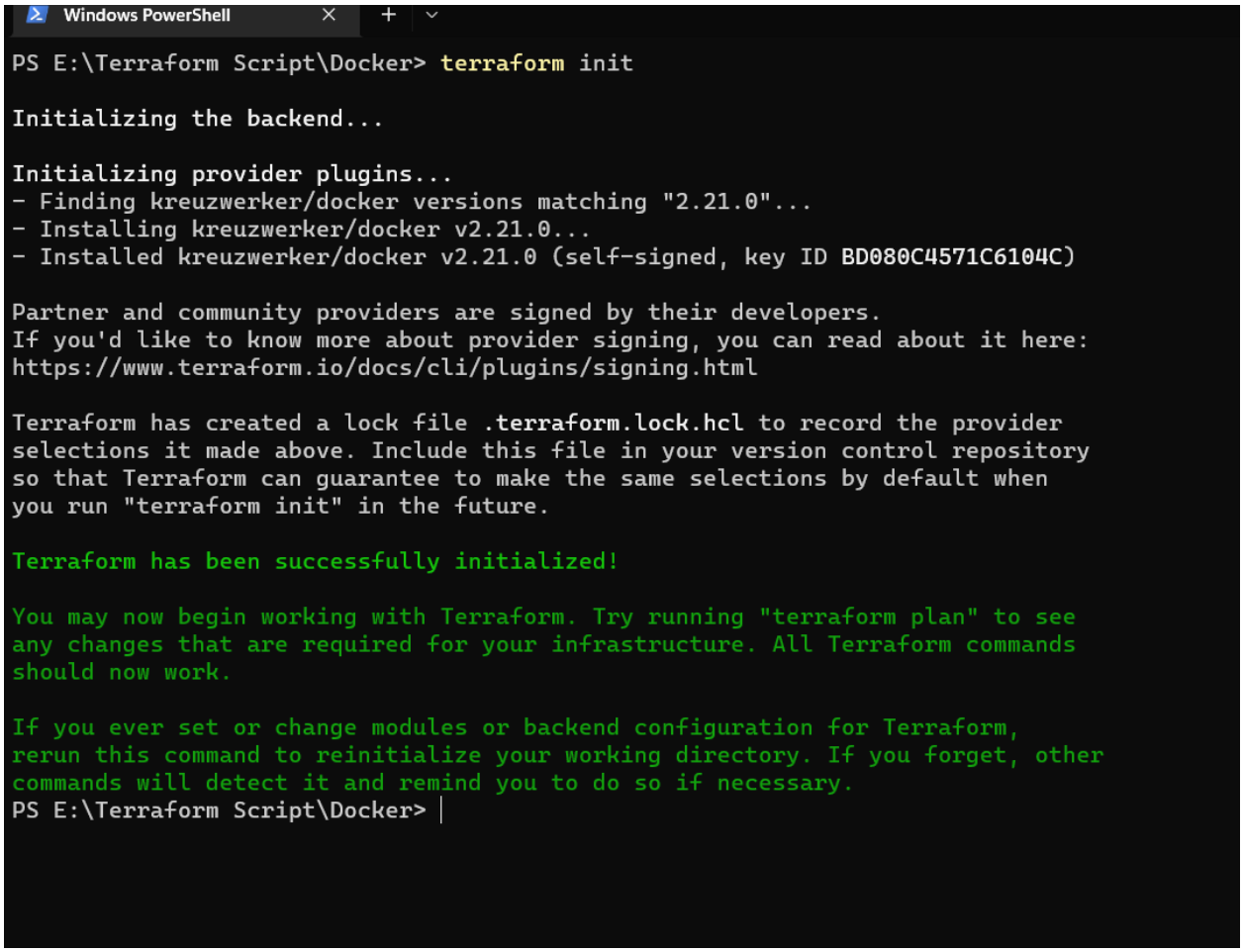
Script:

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "2.21.0"
    }
  }
}

provider "docker" {
  host = "npipe:////.//pipe//docker_engine"
}

# Pulls the image
resource "docker_image" "ubuntu" {
  name = "ubuntu:latest"
}

# Create a container
resource "docker_container" "foo" {
  image = docker_image.ubuntu.image_id
  name  = "foo"
}
```

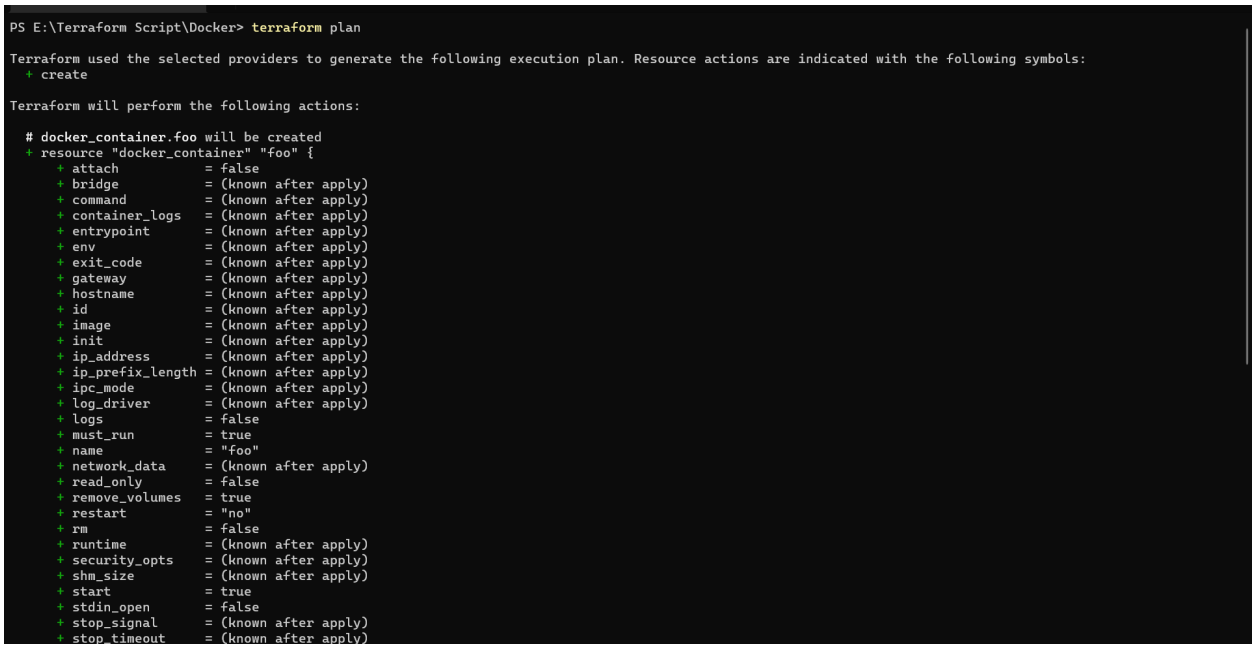
docker.tf

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "2.21.0"
    }
  }
}

provider "docker" {
  host = "npipe:////.//pipe//docker_engine"
}

# Pulls the image
resource "docker_image" "ubuntu" {
  name = "ubuntu:latest"
}

# Create a container
resource "docker_container" "foo" {
  image = docker_image.ubuntu.image_id
  name  = "foo"
}

```

Step 3: Execute Terraform Init command to initialize the resources



Step 4: Execute Terraform plan to see the available resources



```
      + rm               = false
      + runtime          = (known after apply)
      + security_opts    = (known after apply)
      + shm_size         = (known after apply)
      + start            = true
      + stdin_open       = false
      + stop_signal      = (known after apply)
      + stop_timeout     = (known after apply)
      + tty              = false

      + healthcheck {
          + interval     = (known after apply)
          + retries      = (known after apply)
          + start_period = (known after apply)
          + test         = (known after apply)
          + timeout      = (known after apply)
        }

      + labels {
          + label = (known after apply)
          + value = (known after apply)
        }
    }

  # docker_image.ubuntu will be created
  + resource "docker_image" "ubuntu" {
      + id          = (known after apply)
      + image_id    = (known after apply)
      + latest      = (known after apply)
      + name        = "ubuntu:latest"
      + output      = (known after apply)
      + repo_digest = (known after apply)
    }

Plan: 2 to add, 0 to change, 0 to destroy.

─────────────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
PS E:\Terraform Script\Docker>
```

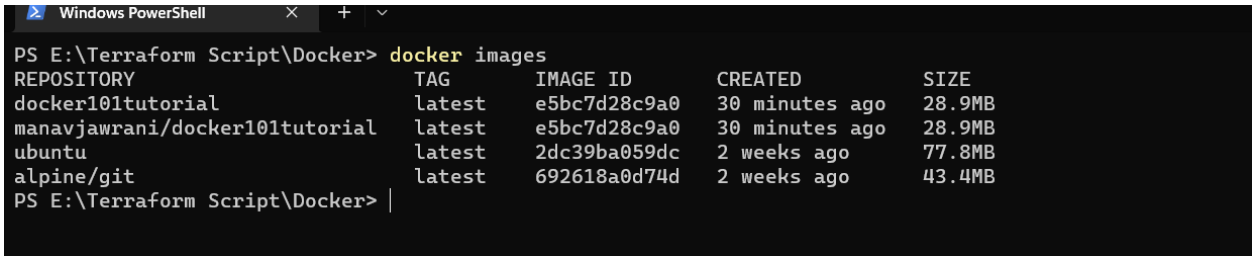
Step 5: Execute Terraform apply to apply the configuration, which will automatically create and run the Ubuntu Linux container based on our configuration. Using command : "**terraform apply"**

```
  # docker_image.ubuntu will be created
  + resource "docker_image" "ubuntu" {
      + id          = (known after apply)
      + image_id    = (known after apply)
      + latest      = (known after apply)
      + name        = "ubuntu:latest"
      + output      = (known after apply)
      + repo_digest = (known after apply)
    }

Plan: 2 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

docker_image.ubuntu: Creating...
docker_image.ubuntu: Still creating... [10s elapsed]
docker_image.ubuntu: Still creating... [20s elapsed]
docker_image.ubuntu: Still creating... [30s elapsed]
docker_image.ubuntu: Creation complete after 30s [id=sha256:2dc39ba059dcd42ade30aae30147b5692777ba9ff0779a62ad93a74de02e3e1fubuntu:latest]
docker_container.foo: Creating...
```

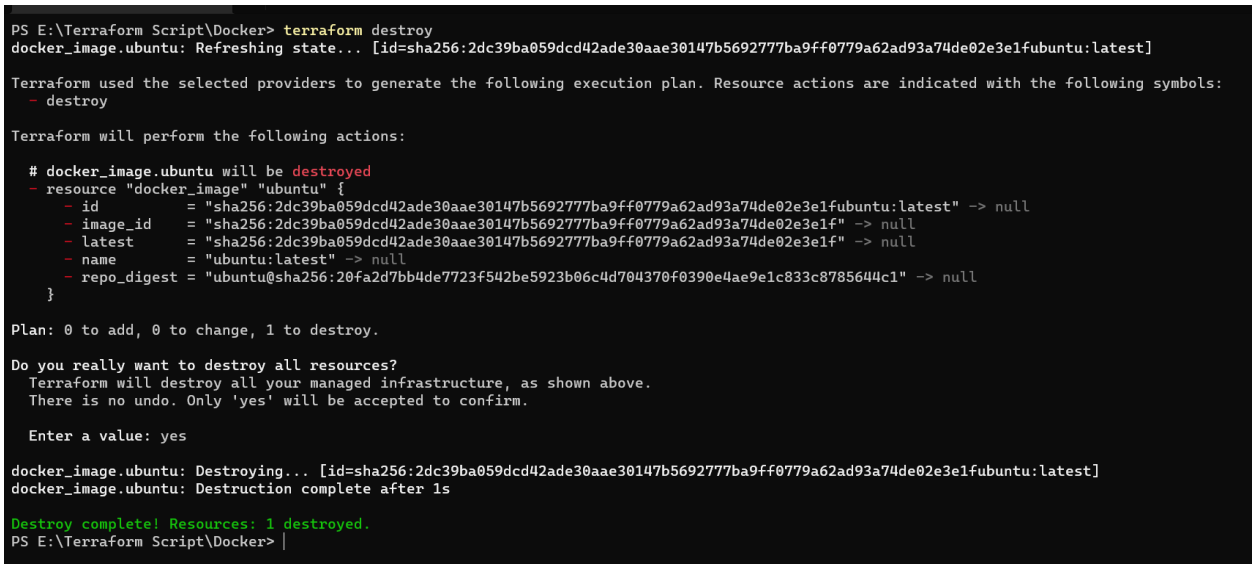
Docker images, Before Executing Apply step:

```
PS E:\Terraform Script\Docker> docker images
REPOSITORY                          TAG       IMAGE ID       CREATED          SIZE
docker101tutorial                   latest    e5bc7d28c9a0   33 minutes ago   28.9MB
manavjawrani/docker101tutorial      latest    e5bc7d28c9a0   33 minutes ago   28.9MB
alpine/git                          latest    692618a0d74d   2 weeks ago      43.4MB
PS E:\Terraform Script\Docker>
```

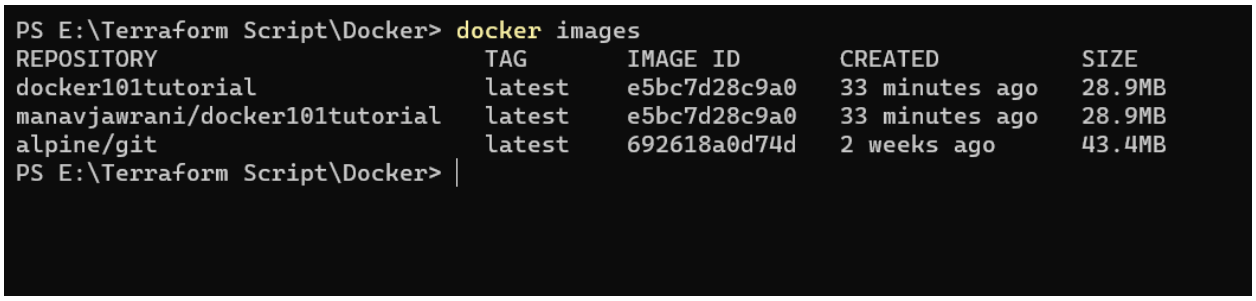
Docker images, After Executing Apply step:



Step 6: Execute Terraform destroy to delete the configuration, which will automatically delete the Ubuntu Container.



Docker images After Executing Destroy step:



Conclusion :

In this experiment, we learned how to create, update and delete Dockes images using Terraform and its commands. When preparing the environment for the implementation, we ran into a few problems with path and configuration but we were able to fix them and the implementation for creating images was successful.