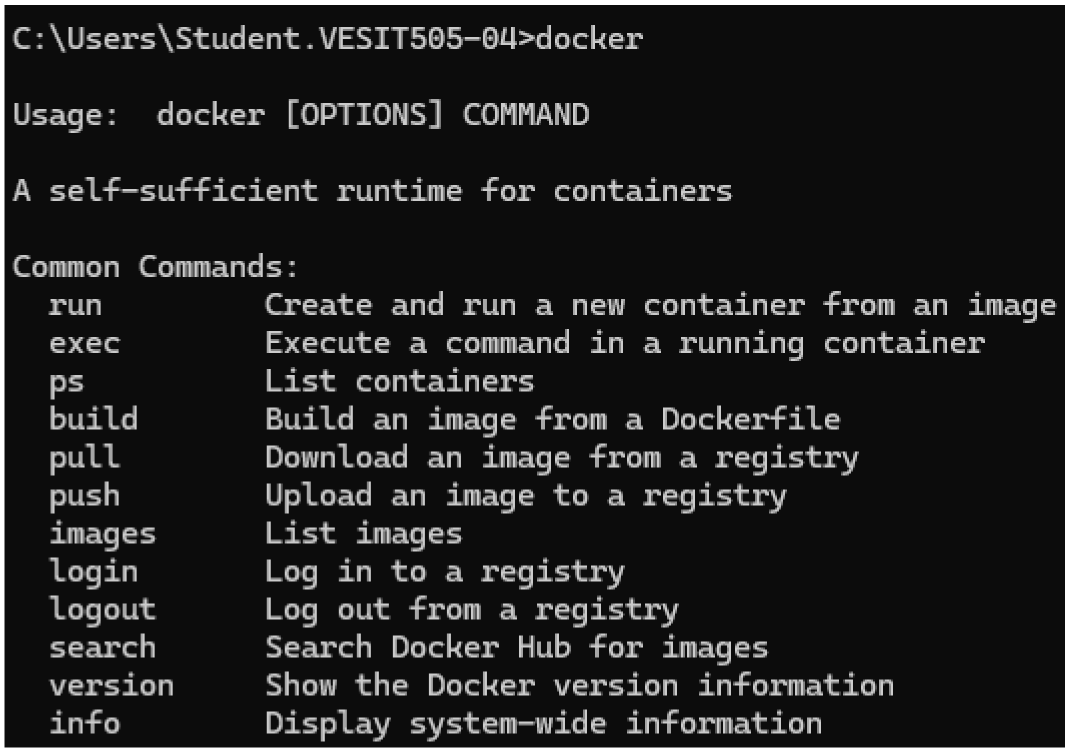
Experiment No.: 6

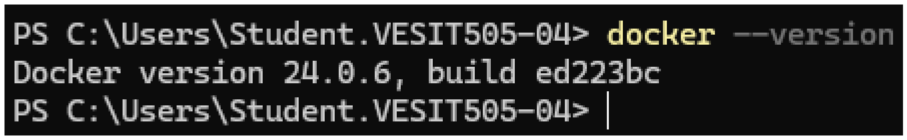
A. Creating docker image using terraform

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

Step 1: Check the docker functionality



```
C:\Users\Student.VESIT505-04>docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:
  run         Create and run a new container from an image
  exec        Execute a command in a running container
  ps          List containers
  build       Build an image from a Dockerfile
  pull        Download an image from a registry
  push        Upload an image to a registry
  images      List images
  login       Log in to a registry
  logout      Log out from a registry
  search      Search Docker Hub for images
  version     Show the Docker version information
  info        Display system-wide information
```



```
PS C:\Users\Student.VESIT505-04> docker --version
Docker version 24.0.6, build ed223bc
PS C:\Users\Student.VESIT505-04> |
```

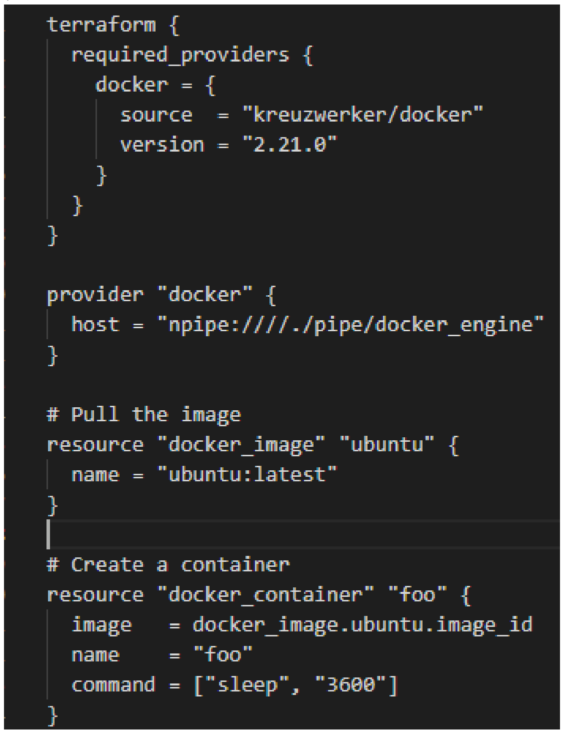
Now, create a folder named 'Terraform Scripts' in which we save our different types of scripts which will be further used in this experiment.

Step 2: Firstly create a new folder named 'Docker' in the 'TerraformScripts' folder. Then create a new docker.tf file using Atom editor and write the followingcontents into it to create a Ubuntu Linux container.

Script:

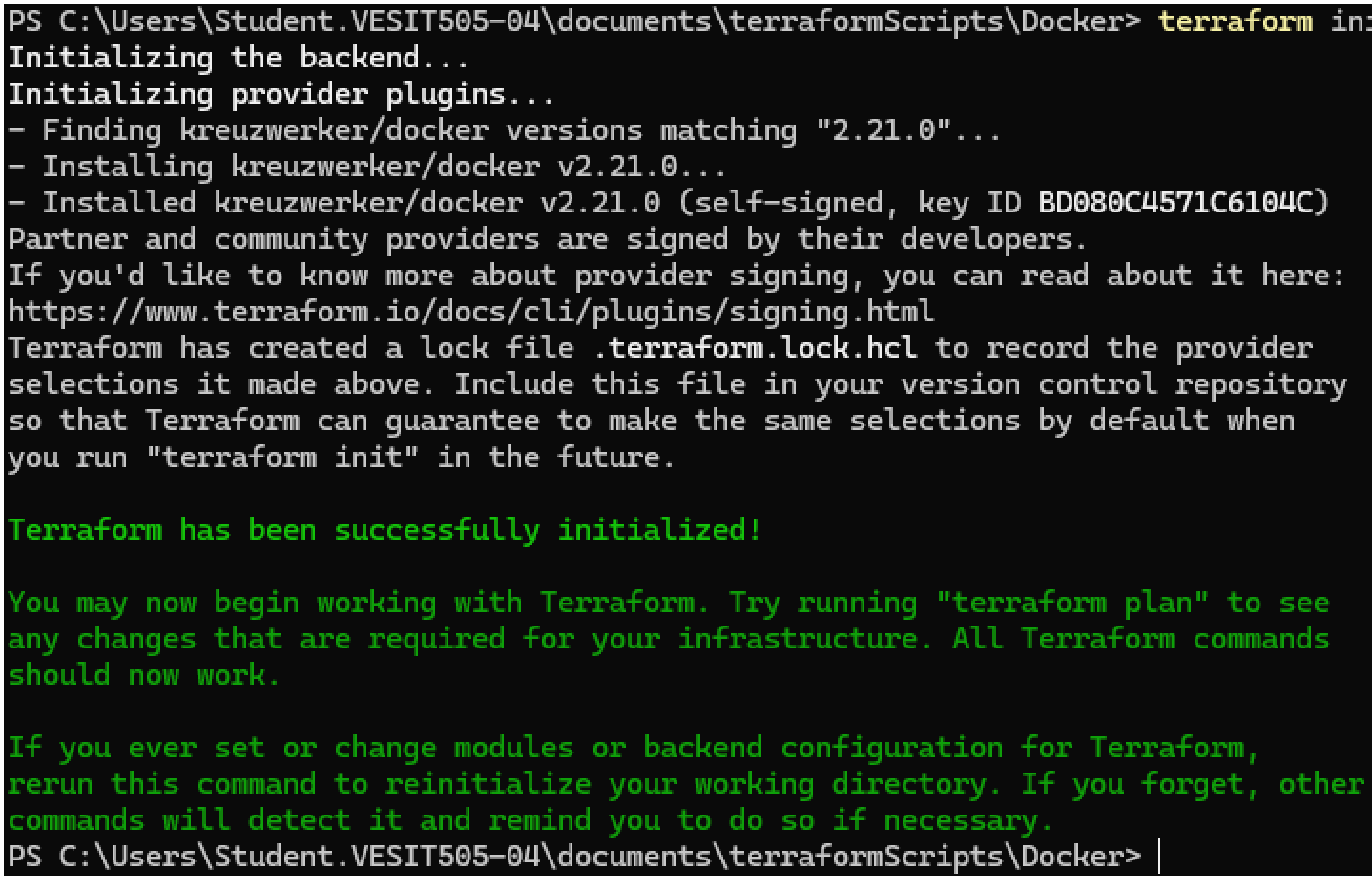
```
terraform
  { required_providers
  {docker = {
    source = "kreuzwerker/docker"
```

```
      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_idname =
  "foo"
}
```



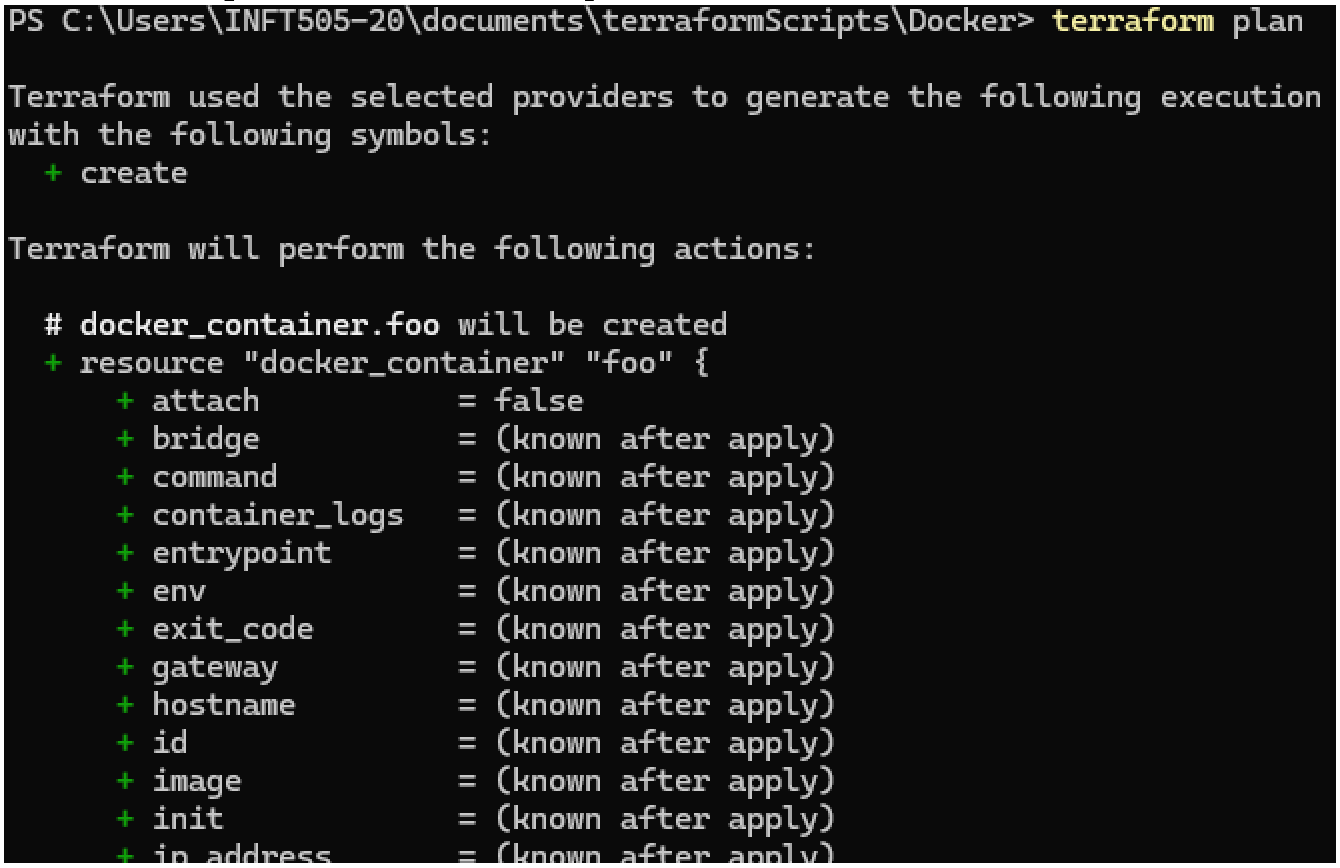
```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "2.21.0"
    }
  }
}

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

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

# Create a container
resource "docker_container" "foo" {
  image   = docker_image.ubuntu.image_id
  name    = "foo"
  command = ["sleep", "3600"]
}
```

Step 3: Execute Terraform Init command to initialize the resources

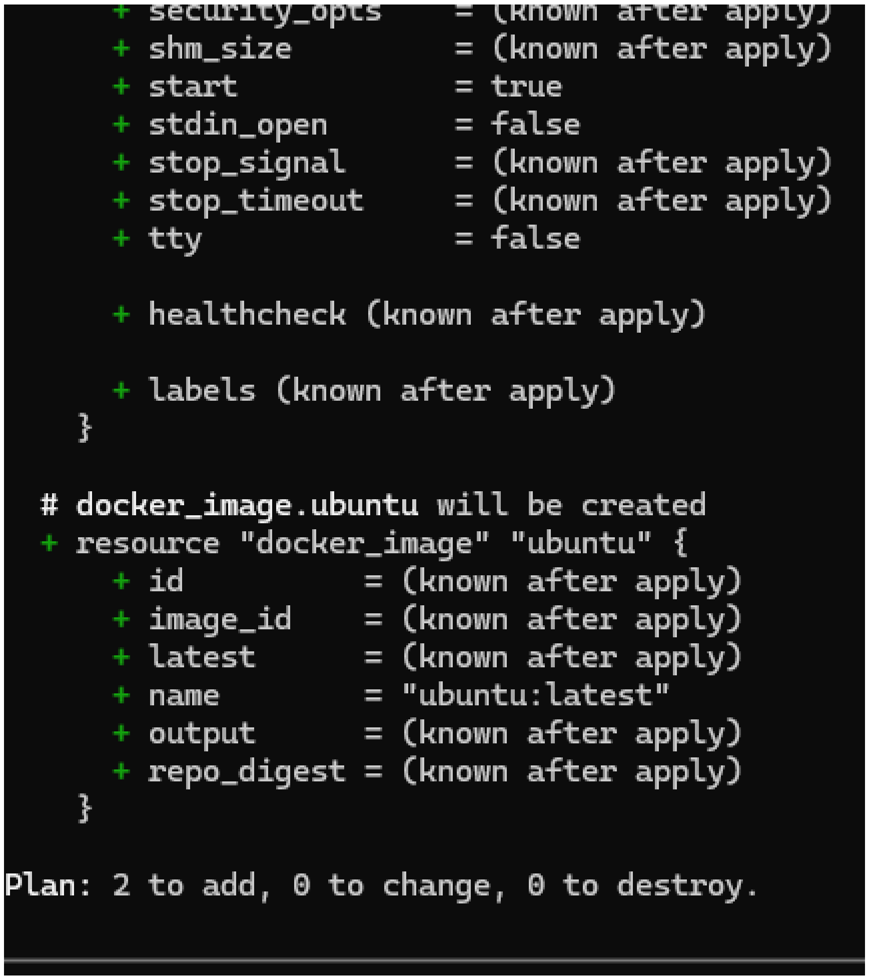


```
PS C:\Users\Student.VESIT505-04\documents\terraformScripts\Docker> terraform in
Initializing the backend...
Initializing provider plugins...
- Finding kreuzwerker/docker versions matching "2.21.0"...
- Installing kreuzwerker/docker v2.21.0...
- Installed kreuzwerker/docker v2.21.0 (self-signed, key ID BD080C4571C6104C)
Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html
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.
PS C:\Users\Student.VESIT505-04\documents\terraformScripts\Docker>
```

Step 4: Execute Terraform plan to see the available resources

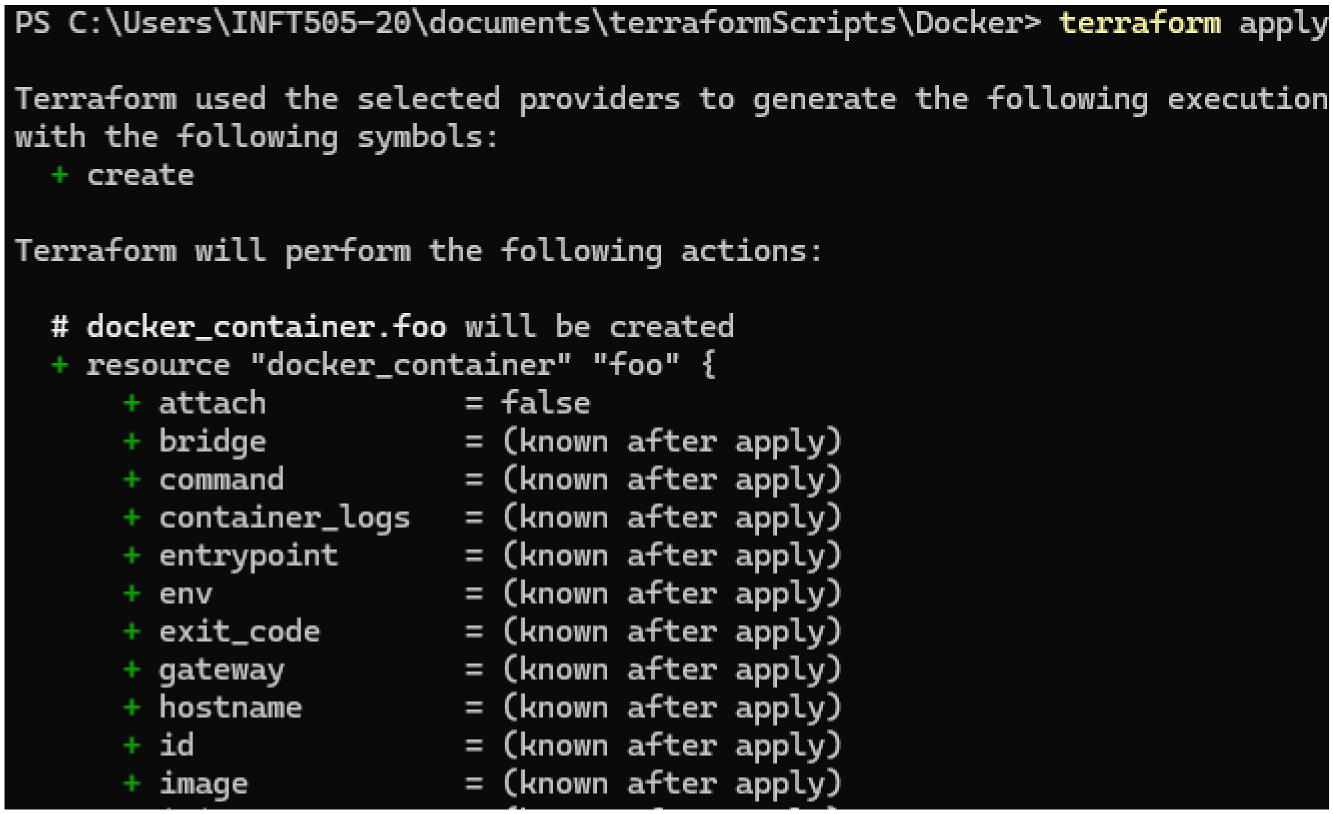


```
PS C:\Users\INFT505-20\documents\terraformScripts\Docker> terraform plan

Terraform used the selected providers to generate the following execution
with the following symbols:
  + create

Terraform will perform the following actions:

  # docker_container.foo will be created
  + resource "docker_container" "foo" {
      + attach               = false
      + bridge               = (known after apply)
      + command              = (known after apply)
      + container_logs       = (known after apply)
      + entrypoint           = (known after apply)
      + env                  = (known after apply)
      + exit_code            = (known after apply)
      + gateway              = (known after apply)
      + hostname             = (known after apply)
      + id                   = (known after apply)
      + image                = (known after apply)
      + init                 = (known after apply)
      + ip_address           = (known after apply)
```



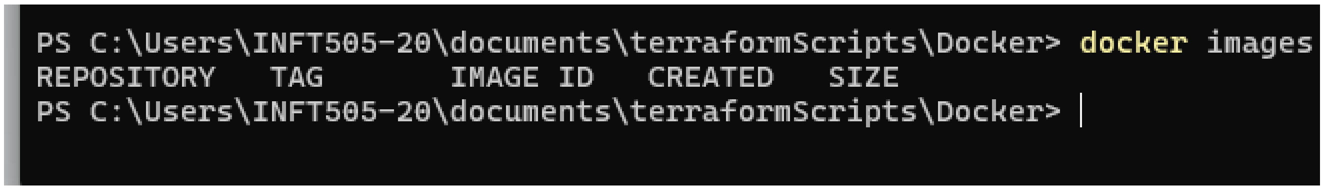
```
      + security_opts     = (known after apply)
      + shm_size          = (known after apply)
      + start             = true
      + stdin_open        = false
      + stop_signal       = (known after apply)
      + stop_timeout      = (known after apply)
      + tty               = false

      + healthcheck (known after apply)

      + labels (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.
```

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”**



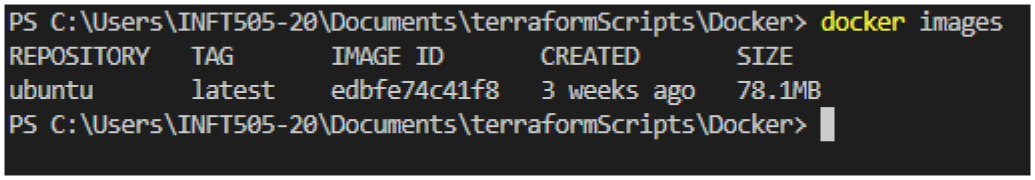
```
PS C:\Users\INFT505-20\documents\terraformScripts\Docker> terraform apply

Terraform used the selected providers to generate the following execution
with the following symbols:
  + create

Terraform will perform the following actions:

  # docker_container.foo will be created
  + resource "docker_container" "foo" {
      + attach              = false
      + bridge              = (known after apply)
      + command             = (known after apply)
      + container_logs      = (known after apply)
      + entrypoint          = (known after apply)
      + env                 = (known after apply)
      + exit_code           = (known after apply)
      + gateway             = (known after apply)
      + hostname            = (known after apply)
      + id                  = (known after apply)
      + image               = (known after apply)
```

Run Docker images, Before Executing Apply step:



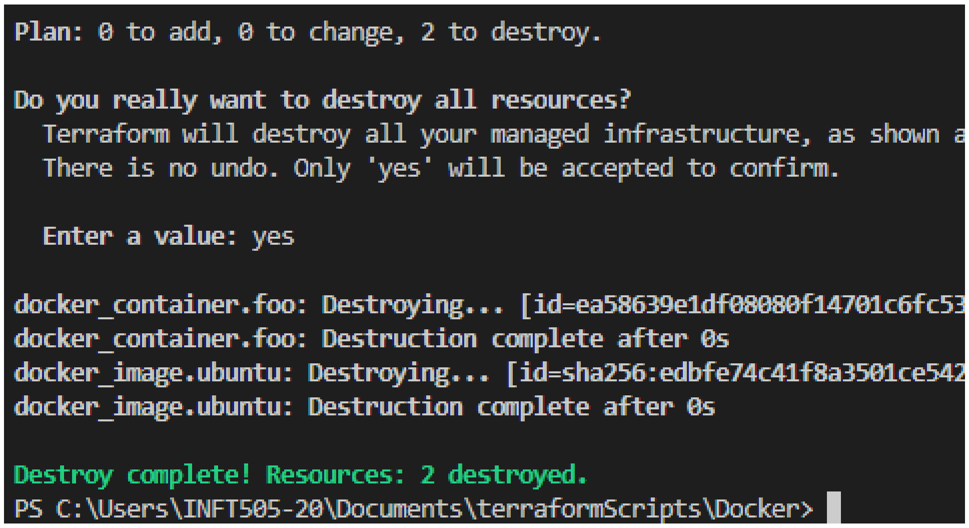
```
PS C:\Users\INFT505-20\documents\terraformScripts\Docker> docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
PS C:\Users\INFT505-20\documents\terraformScripts\Docker>
```

Docker images, After Executing Apply step:



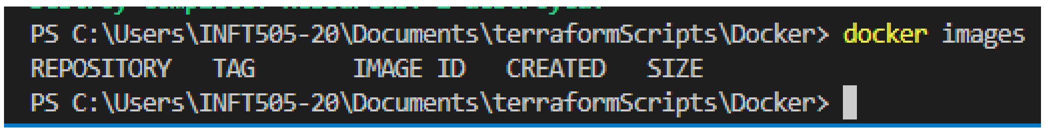
```
PS C:\Users\INFT505-20\Documents\terraformScripts\Docker> docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
ubuntu       latest    edbfe74c41f8   3 weeks ago   78.1MB
PS C:\Users\INFT505-20\Documents\terraformScripts\Docker>
```

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



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

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown a
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

docker_container.foo: Destroying... [id=ea58639e1df08080f14701c6fc53
docker_container.foo: Destruction complete after 0s
docker_image.ubuntu: Destroying... [id=sha256:edbfe74c41f8a3501ce542
docker_image.ubuntu: Destruction complete after 0s

Destroy complete! Resources: 2 destroyed.
PS C:\Users\INFT505-20\Documents\terraformScripts\Docker>
```

Docker images After Executing Destroy step



```
PS C:\Users\INFT505-20\Documents\terraformScripts\Docker> docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
PS C:\Users\INFT505-20\Documents\terraformScripts\Docker>
```