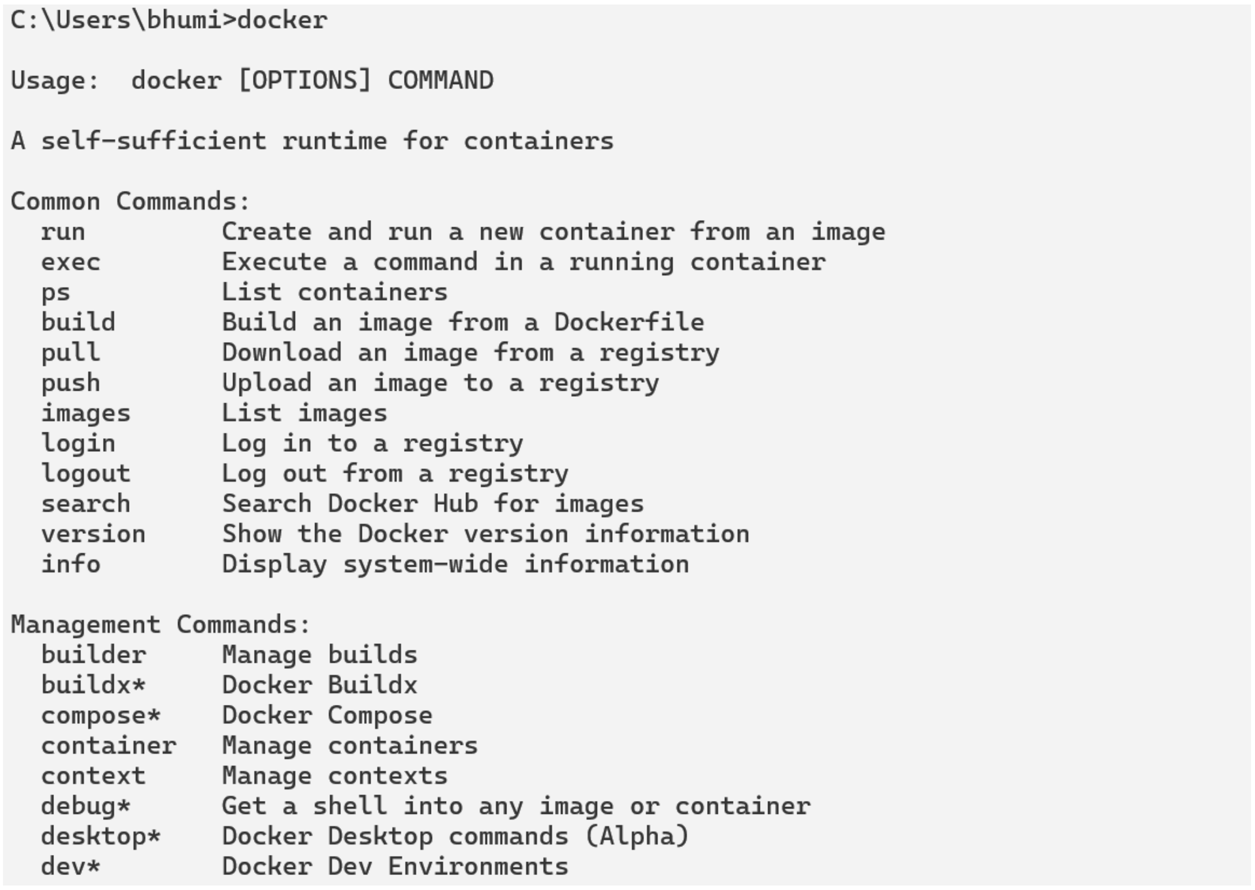
Aim: To Build, change, and destroy AWS / GCP /Microsoft Azure/ DigitalOcean infrastructure Using Terraform. (S3 bucket or Docker) fdp.

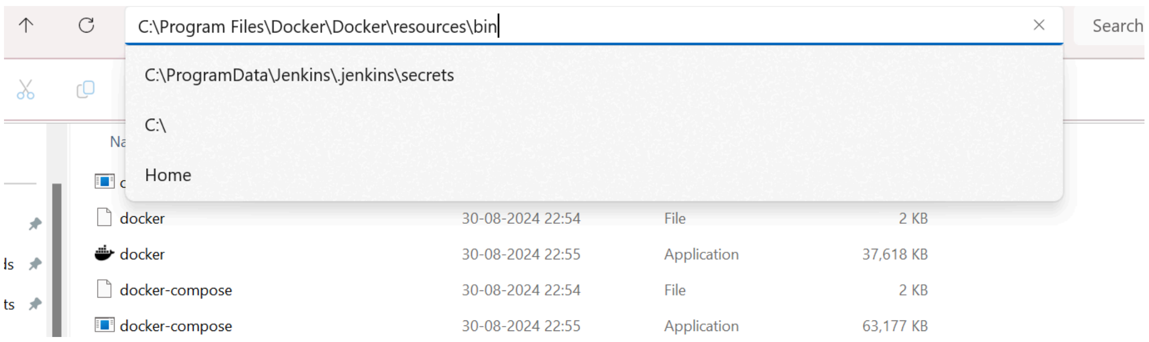
Step 1: For this experiment, you need to install docker on your computer. Go to https://www.docker.com/ and download the file according to the OS you have. Open the file and start the installation. Once installed, open your terminal and run ‘docker’ command. If this is your output, then docker is installed successfully.



```
C:\Users\bhumi>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

Management Commands:
  builder     Manage builds
  buildx*     Docker Buildx
  compose*    Docker Compose
  container   Manage containers
  context     Manage contexts
  debug*      Get a shell into any image or container
  desktop*    Docker Desktop commands (Alpha)
  dev*        Docker Dev Environments
```

If you det an error like ‘docker is not an internal or external command’, you need to add the bin path of docker to your environment variables.

Go to File Explorer, and follow this path: C drive → Program Files → Docker → Docker → Resources → bin. Copy this path by clicking on the bar having the path and using shortcut CTRL + C



C:\Program Files\Docker\Docker\resources\bin

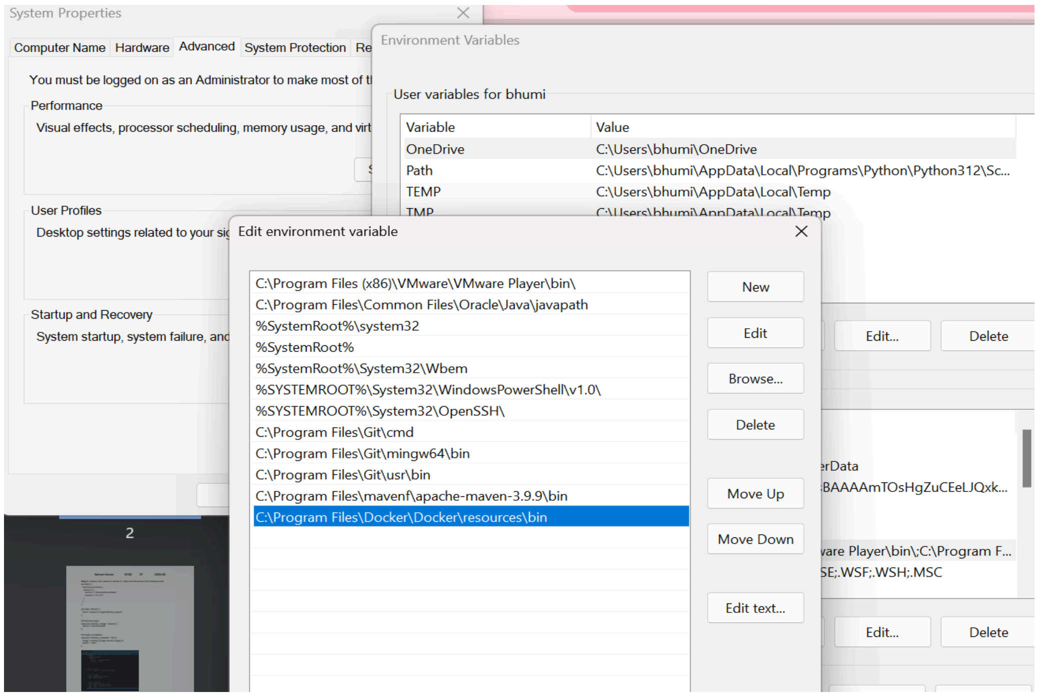
C:\ProgramData\Jenkins\.jenkins\secrets

C:\

Home

docker	30-08-2024 22:54	File	2 KB
docker	30-08-2024 22:55	Application	37,618 KB
docker-compose	30-08-2024 22:54	File	2 KB
docker-compose	30-08-2024 22:55	Application	63,177 KB

Open ‘Edit the System Environment Variables’ on your system. Click on Environment Variables. Now, check for a ‘Path’ variable under System variables, if it exists, click on it, then click on edit. Else, click on New and add the variable ‘Path’. If the variable existed, click on Edit, then on New. This will give you a text box. Paste the path you copied here and click on ok until you close all the tabs.



User variables for bhumi

Variable	Value
OneDrive	C:\Users\bhumi\OneDrive
Path	C:\Users\bhumi\AppData\Local\Programs\Python\Python312\Sc...
TEMP	C:\Users\bhumi\AppData\Local\Temp
TMP	C:\Users\bhumi\AppData\Local\Temp

Edit environment variable

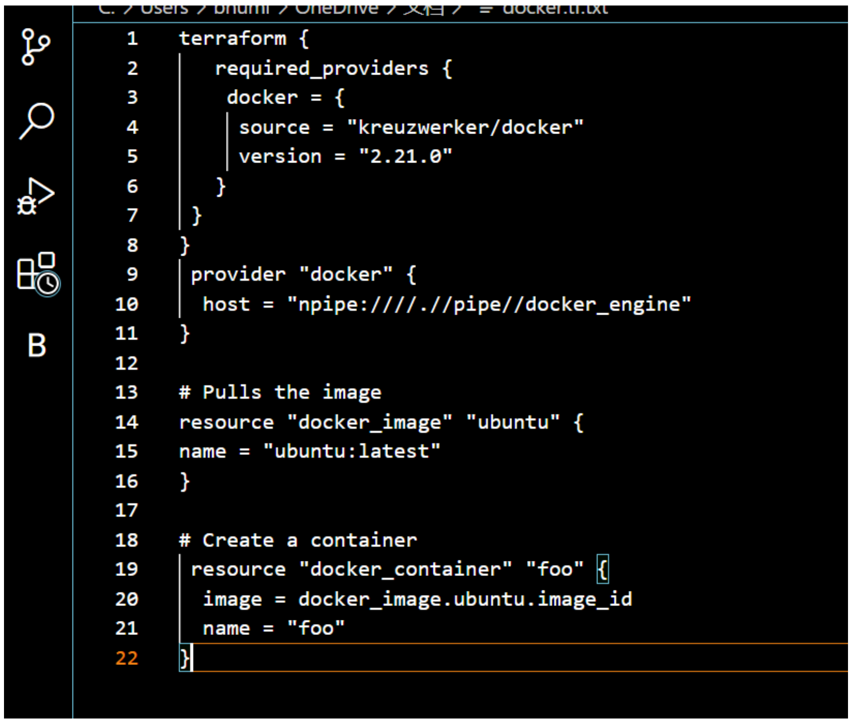
```
C:\Program Files (x86)\VMware\VMware Player\bin\
C:\Program Files\Common Files\Oracle\Java\javapath
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\Git\cmd
C:\Program Files\Git\mingw64\bin
C:\Program Files\Git\usr\bin
C:\Program Files\maven\apache-maven-3.9.9\bin
C:\Program Files\Docker\Docker\resources\bin
```

Now run the docker command again and the output would appear.
Alternatively, you could also run ‘docker –version’ to check whether docker is started on the terminal.

```
C:\Users\bhumi>docker --version
Docker version 27.1.1, build 6312585
```

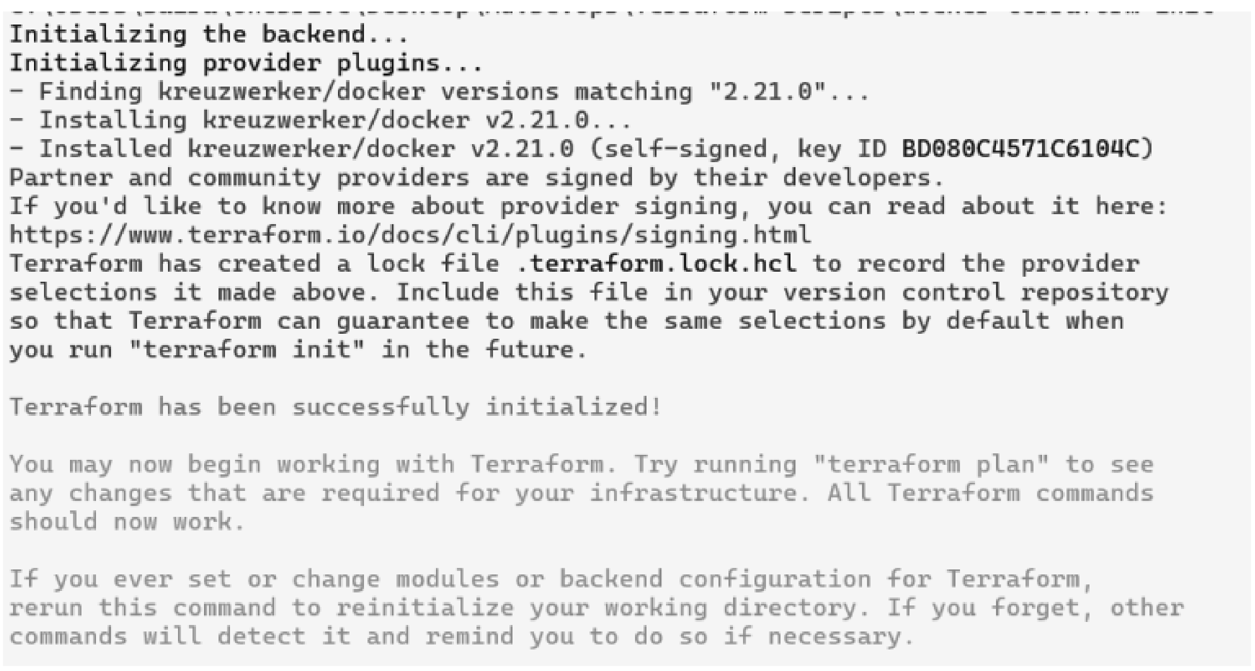
Step 2: Create a file called 'docker.tf'. Open the file and put the following code.

```
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"
}
```



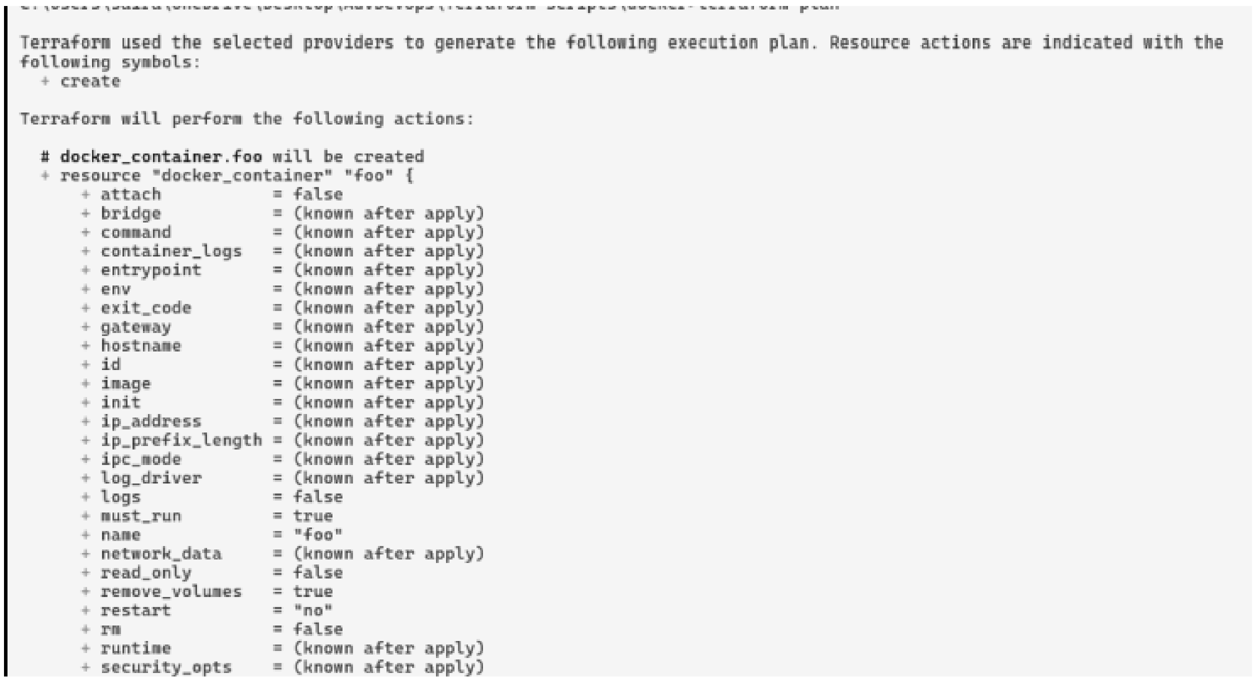
```
C: > Users > bhumi > OneDrive > 文档 > ≡ docker.tf.txt
terraform {
    required_providers {
     docker = {
      source = "kreuzwerker/docker"
      version = "2.21.0"
     }
   }
 }
 provider "docker" {
  host = "npipe:////.//pipe//docker_engine"
 }

# 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: Open the folder where the docker.tf is present on your terminal. Execute the command 'terrafom init'. This will initialize terraform in the directory.



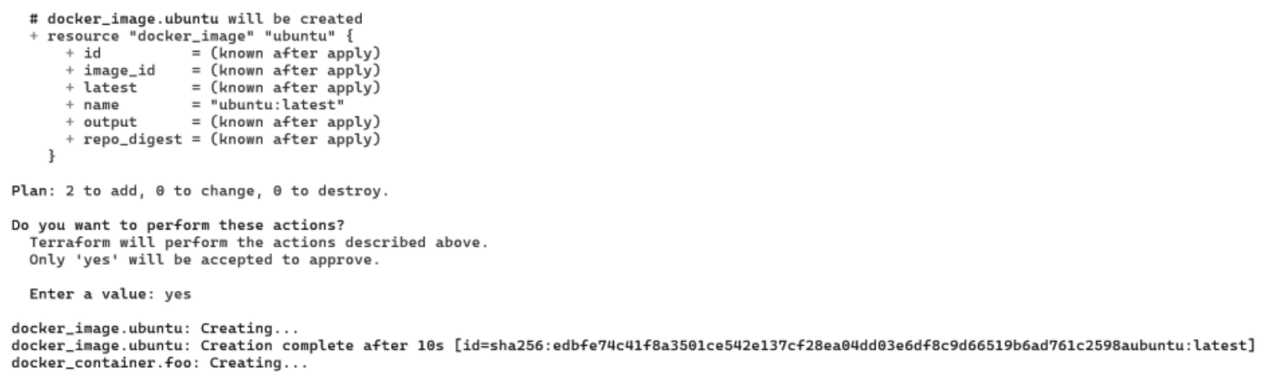
```
Initializing the backend...
Initializing provider plugins...
- Finding kreuzwerker/docker versions matching "2.21.0"...
- Installing kreuzwerker/docker v2.21.0...
- Installed kreuzwerker/docker v2.21.0 (self-signed, key ID BD080C4571C6104C)
Partner and community providers are signed by their developers.
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.
```

Step 4: Run the command ‘terraform plan’. This creates an execution plan and lets you overview changes that are going to happen in your infrastructure.



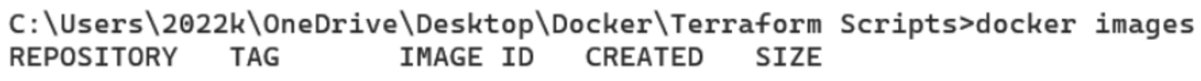
```
Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
  + create

Terraform will perform the following actions:

  # 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)
      + ip_prefix_length = (known after apply)
      + ipc_mode         = (known after apply)
      + log_driver       = (known after apply)
      + logs             = false
      + must_run         = true
      + name             = "foo"
      + network_data     = (known after apply)
      + read_only        = false
      + remove_volumes   = true
      + restart          = "no"
      + rm               = false
      + runtime          = (known after apply)
      + security_opts    = (known after apply)
```

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



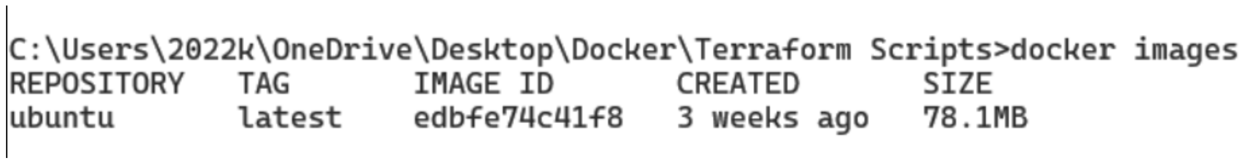
```
  # docker_image.ubuntu will be created
  + resource "docker_image" "ubuntu" {
      + id          = (known after apply)
      + image_id    = (known after apply)
      + latest      = (known after apply)
      + name        = "ubuntu:latest"
      + output      = (known after apply)
      + repo_digest = (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: Creation complete after 10s [id=sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598aubuntu:latest]
docker_container.foo: Creating...
```

Docker images, Before Executing Apply step:



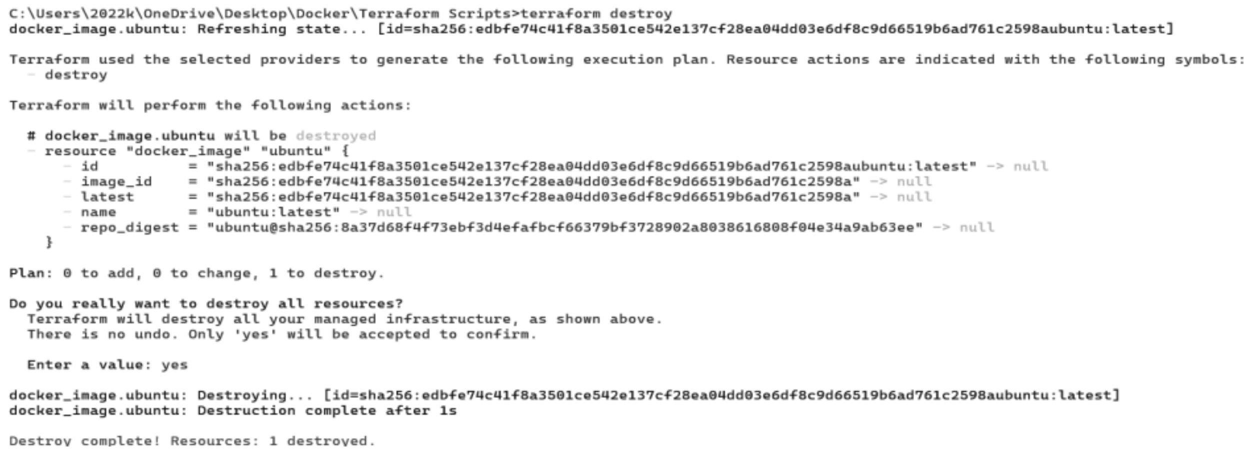
```
C:\Users\2022k\OneDrive\Desktop\Docker\Terraform Scripts>docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
```

Docker images, After Executing Apply step:



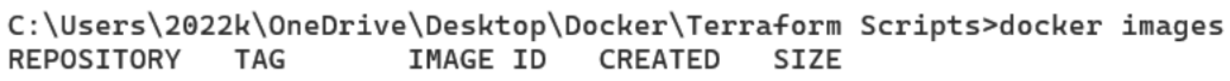
```
C:\Users\2022k\OneDrive\Desktop\Docker\Terraform Scripts>docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
ubuntu       latest    edbfe74c41f8   3 weeks ago   78.1MB
```

Step 6: Execute Terraform destroy to delete the configuration, which will automatically Delete the ubuntu container.



```
C:\Users\2022k\OneDrive\Desktop\Docker\Terraform Scripts>terraform destroy
docker_image.ubuntu: Refreshing state... [id=sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598aubuntu:latest]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  # docker_image.ubuntu will be destroyed
  - resource "docker_image" "ubuntu" {
      - id          = "sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598aubuntu:latest" -> null
      - image_id    = "sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598a" -> null
      - latest      = "sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598a" -> null
      - name        = "ubuntu:latest" -> null
      - repo_digest = "ubuntu@sha256:8a37d68f4f73ebf3d4efafbcf66379bf3728902a8038616808f04e34a9ab63ee" -> null
    }

Plan: 0 to add, 0 to change, 1 to destroy.

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

  Enter a value: yes

docker_image.ubuntu: Destroying... [id=sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598aubuntu:latest]
docker_image.ubuntu: Destruction complete after 1s

Destroy complete! Resources: 1 destroyed.
```

Docker images After Executing Destroy step.



```
C:\Users\2022k\OneDrive\Desktop\Docker\Terraform Scripts>docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
```