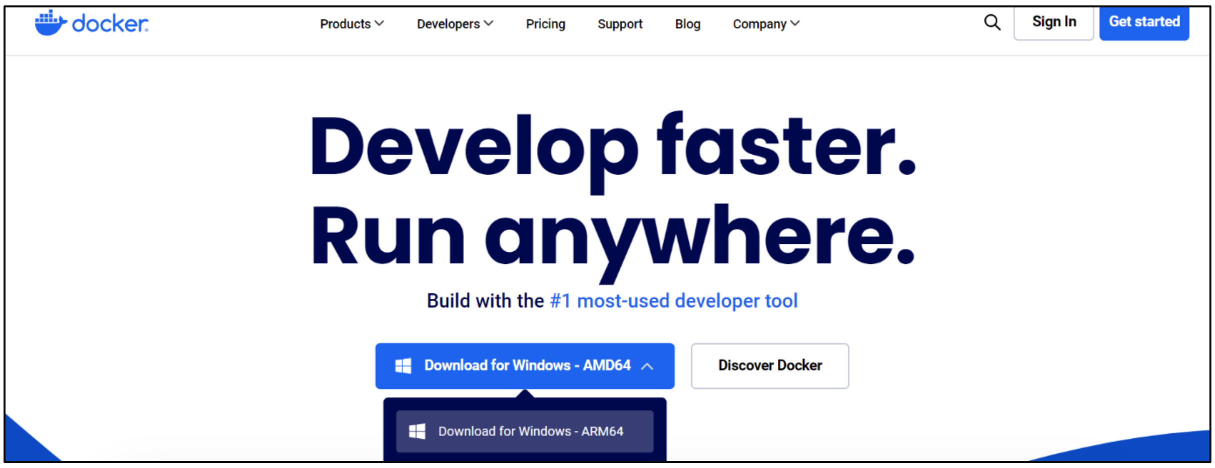
Advance DevOps

Experiment 6

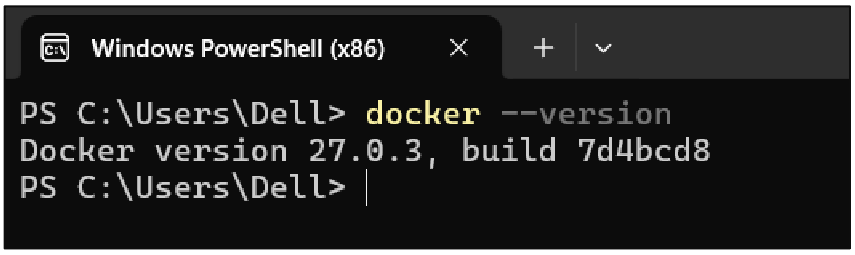
Creating docker image using terraform

Steps:

1. Firstly, we will have to download and install docker from its official website.

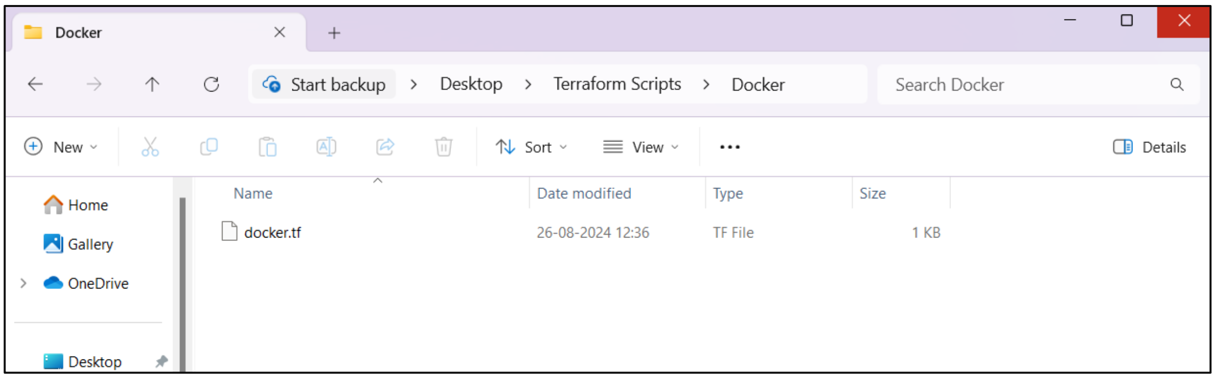

docker
Products
Developers
Pricing
Support
Blog
Company
Sign In
Get started
Develop faster. Run anywhere.
Build with the #1 most-used developer tool
Download for Windows - AMD64
Discover Docker
Download for Windows - ARM64

2. To check if docker has been correctly installed we can use the command "docker –version". If it returns the version then installation has been successful.



```
PS C:\Users\Dell> docker --version
Docker version 27.0.3, build 7d4bcd8
PS C:\Users\Dell>
```

3. Now we will create a folder “Terraform Scripts” and inside this folder we will create another folder “Docker” where our docker script will be saved. Using an editor create a file named docker and save it with .tf extension inside the above-mentioned directory structure.



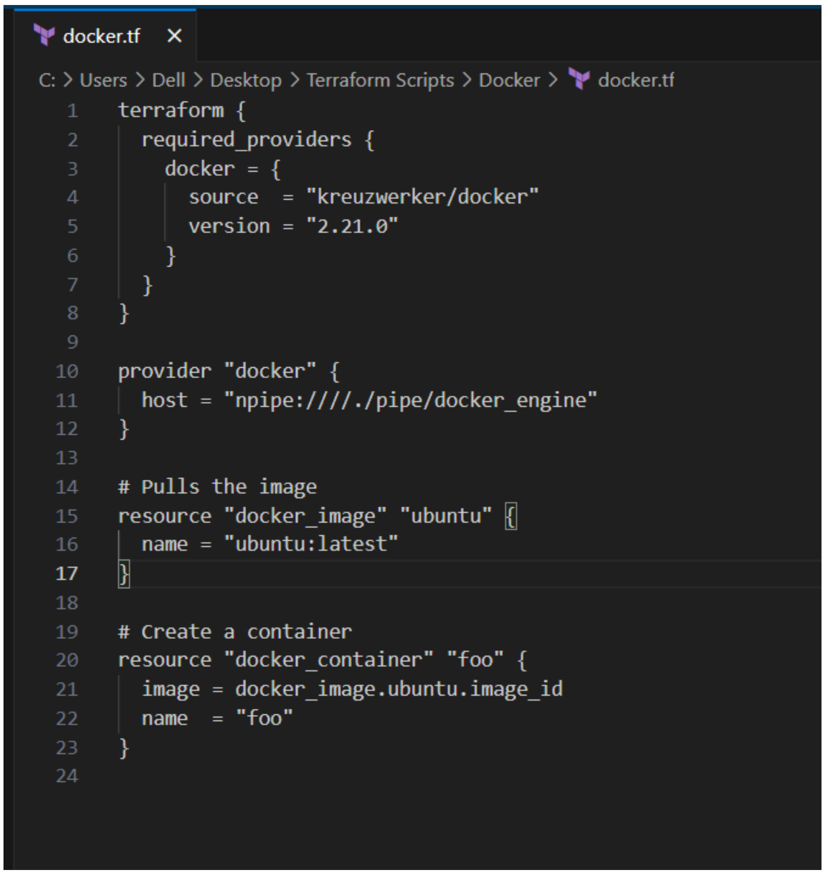
Name	Date modified	Type	Size
docker.tf	26-08-2024 12:36	TF File	1 KB

4. The docker.tf file will contain the following:

```
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"
}
```

Ensure correct block formatting to avoid errors.

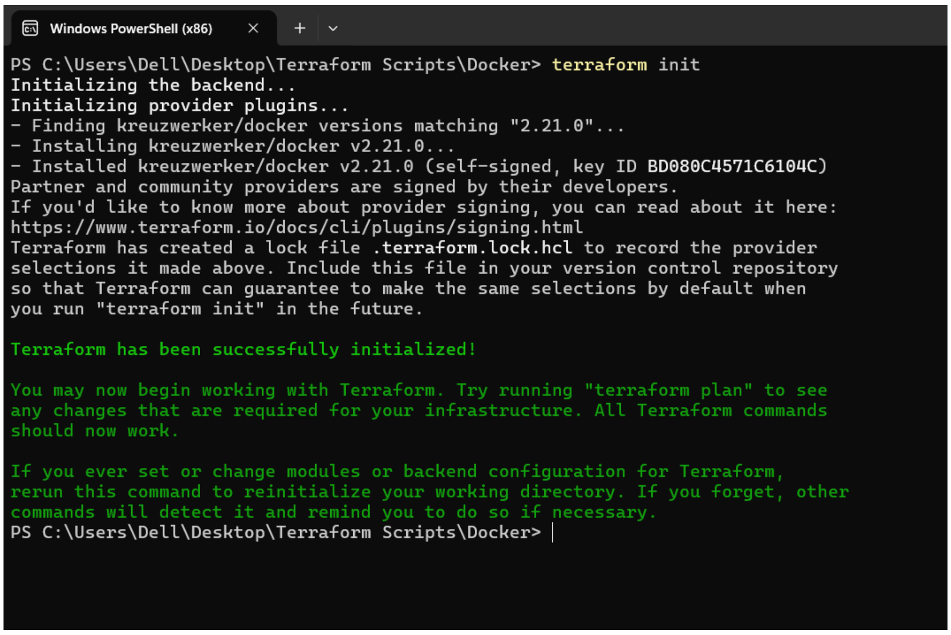


docker.tf ×

C: > Users > Dell > Desktop > Terraform Scripts > Docker > 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"
}
```

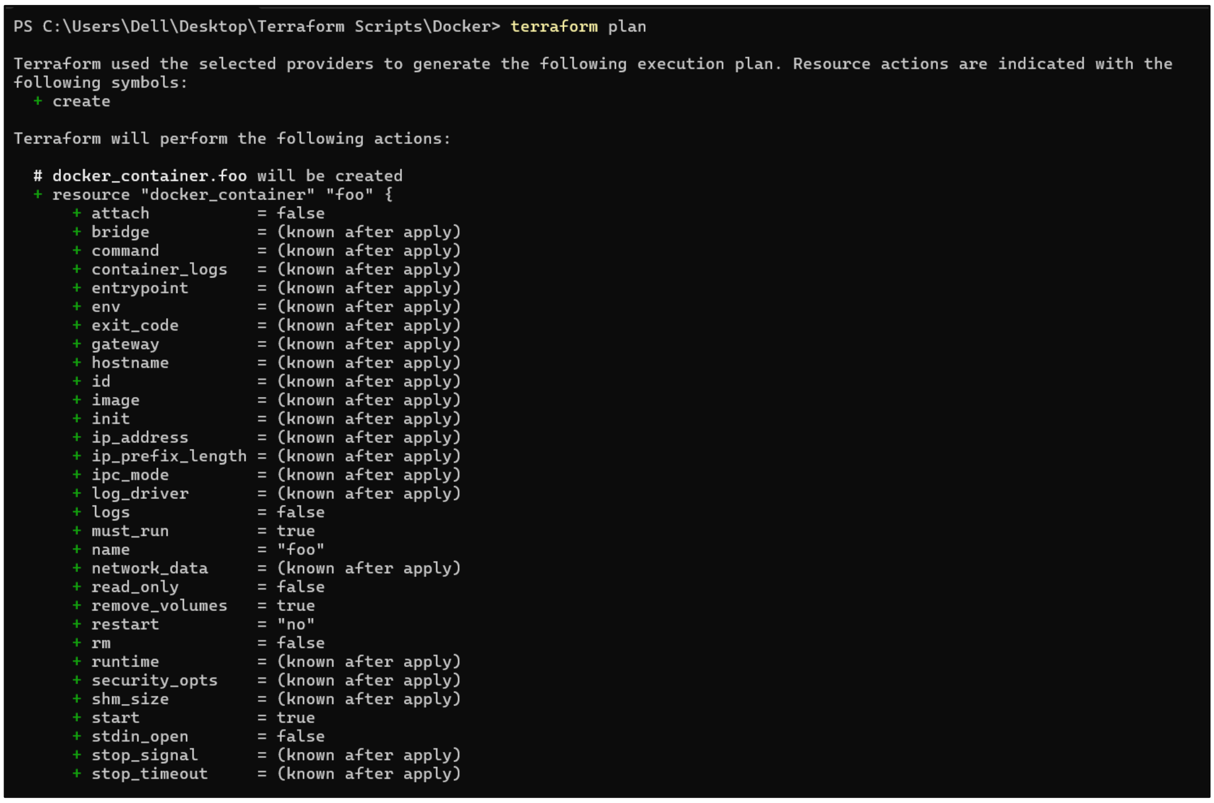
5. Now execute "terraform init" command in power shell. This command will initialize your working directory and download the necessary provider plugins. (Make sure you are in the "Docker" directory)



Windows PowerShell (x86)

```
PS C:\Users\Dell\Desktop\Terraform Scripts\Docker> terraform init
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\Dell\Desktop\Terraform Scripts\Docker> |
```

6. Next execute "terraform plan" command. The terraform plan command is used to create an execution plan for terraform. This plan shows you what changes terraform will make to your infrastructure based on your current configuration files and the state of your existing infrastructure.

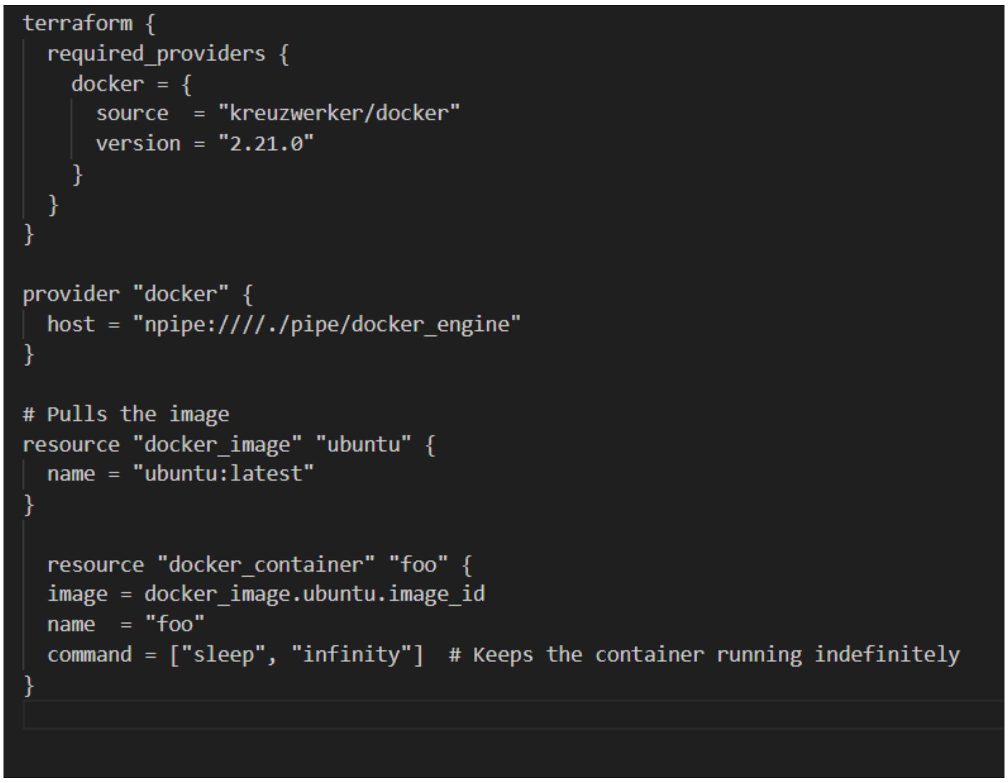


```
PS C:\Users\Dell\Desktop\Terraform Scripts\Docker> terraform plan

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:

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

7. After this execute the "terraform apply" command. The terraform apply command executes the actions proposed in terraform plan. It is used to deploy your infrastructure.

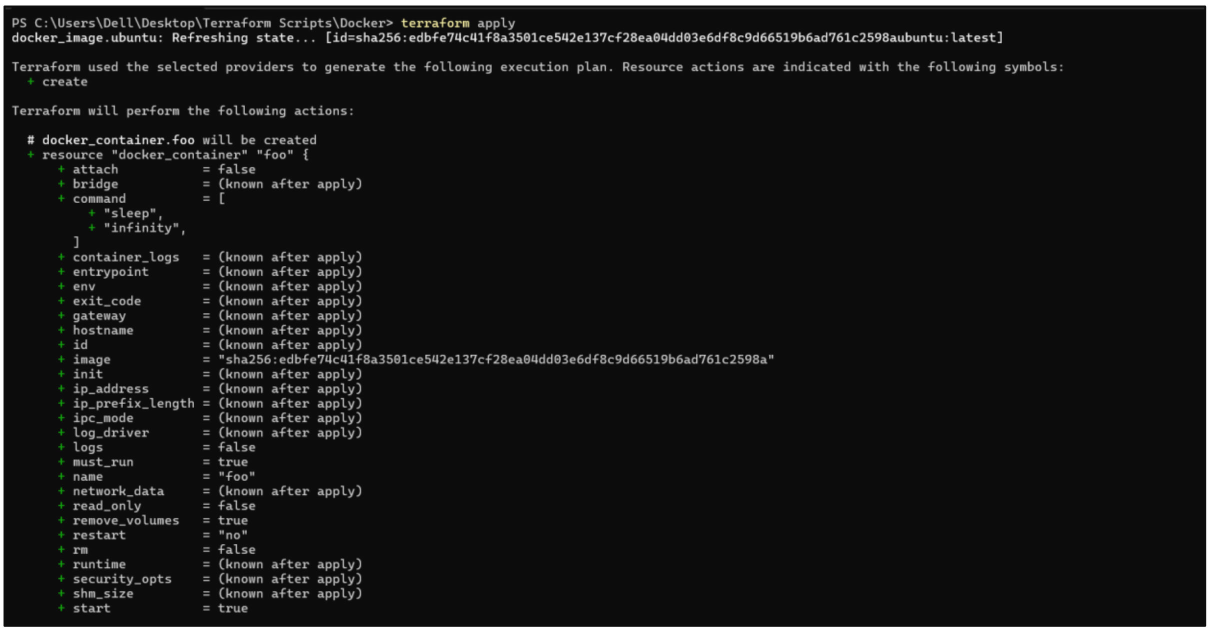
   In this step I got an error saying "container exited immediately" which indicates that the Docker container created by terraform configuration exited immediately after being started. In order to keep the container running I updated the script file with the following code:

   command = ["sleep", "infinity"] # Keeps the container running

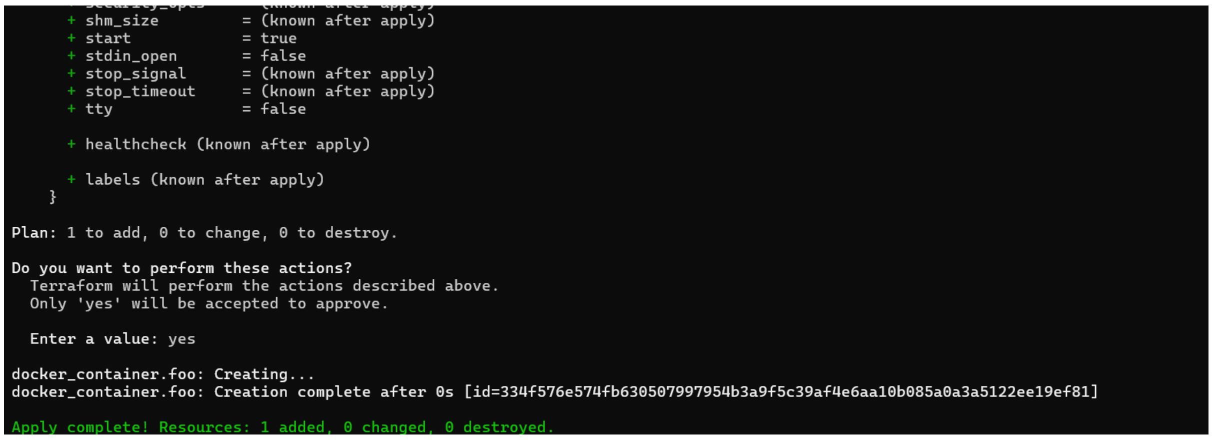


```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "2.21.0"
    }
  }
}

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

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

  resource "docker_container" "foo" {
  image = docker_image.ubuntu.image_id
  name  = "foo"
  command = ["sleep", "infinity"]  # Keeps the container running indefinitely
}
```

8. After updating the script, the error will be solved and we can again run apply command.



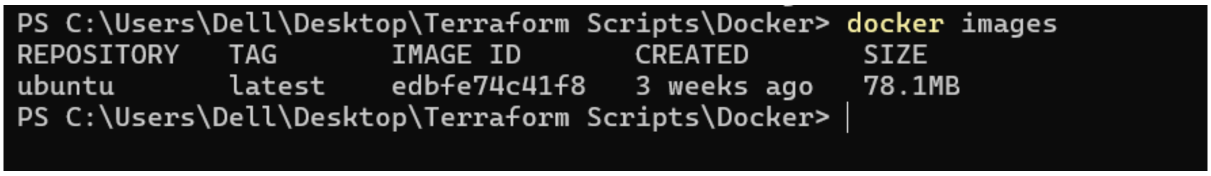
```
PS C:\Users\Dell\Desktop\Terraform Scripts\Docker> terraform apply
docker_image.ubuntu: Refreshing state... [id=sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598aubuntu:latest]

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

Terraform will perform the following actions:

  # docker_container.foo will be created
  + resource "docker_container" "foo" {
      + attach            = false
      + bridge            = (known after apply)
      + command           = [
          + "sleep",
          + "infinity",
        ]
      + 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             = "sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598a"
      + 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)
      + shm_size          = (known after apply)
      + start             = true
```



```
      + 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)
    }

Plan: 1 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_container.foo: Creating...
docker_container.foo: Creation complete after 0s [id=334f576e574fb630507997954b3a9f5c39af4e6aa10b085a0a3a5122ee19ef81]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

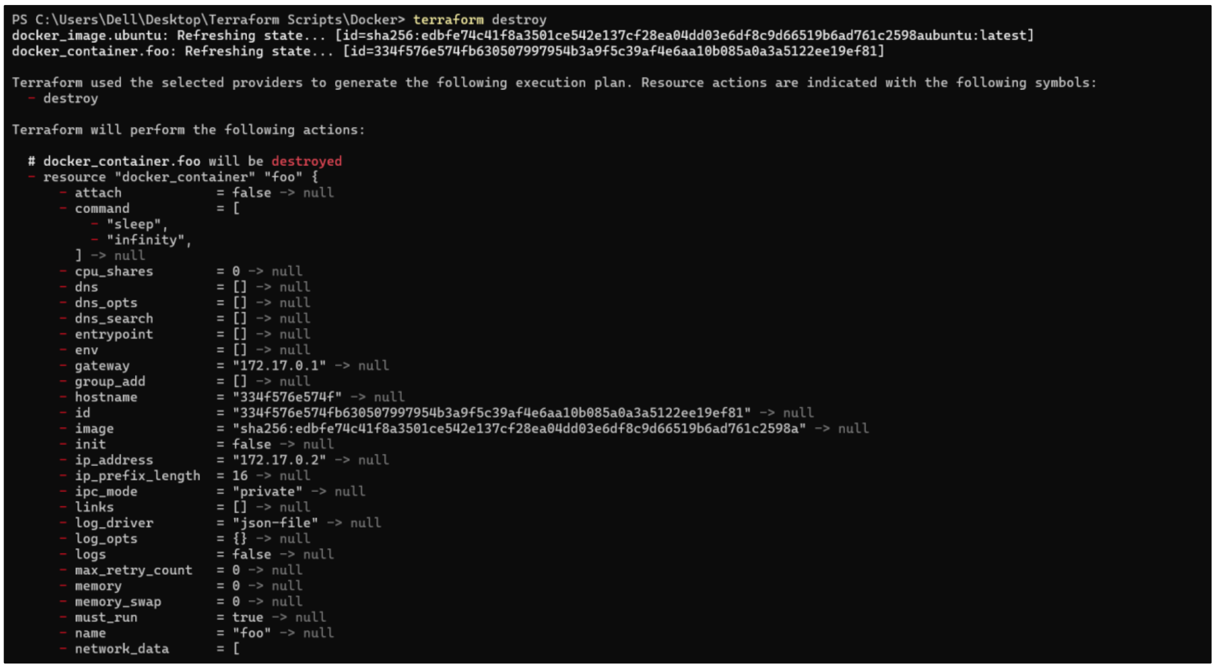
Docker image has been created.

9. The image created can be checked by using docker images command. It will show the repository, tag, image id, time of creation and size as shown below.

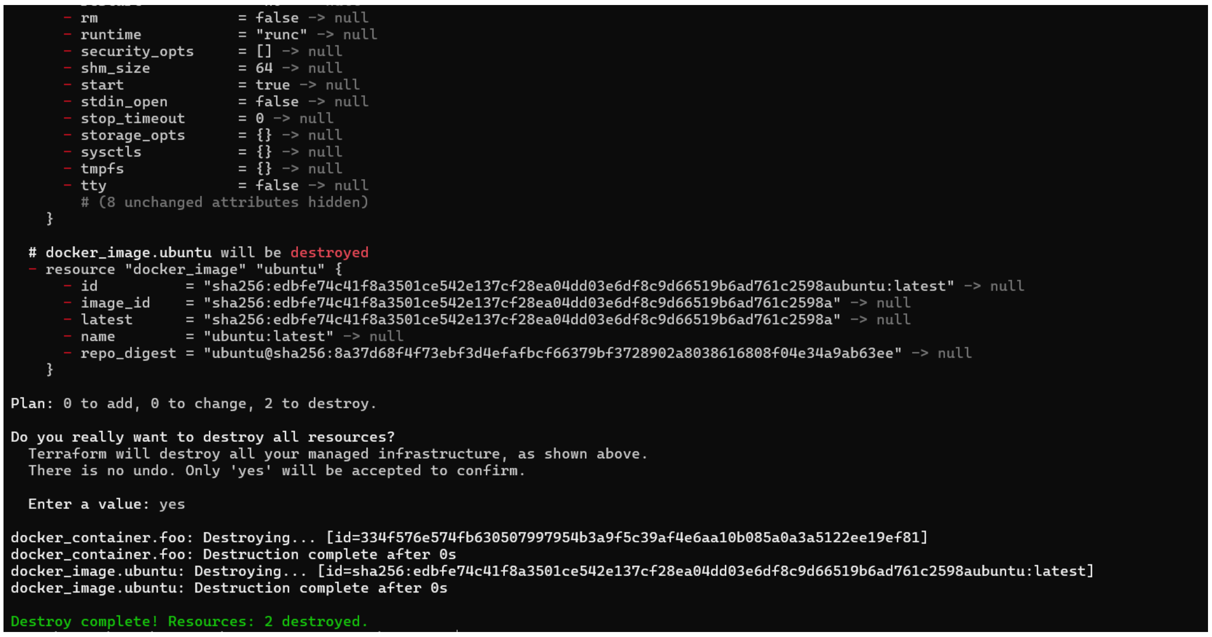


```
PS C:\Users\Dell\Desktop\Terraform Scripts\Docker> docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
ubuntu       latest    edbfe74c41f8   3 weeks ago   78.1MB
PS C:\Users\Dell\Desktop\Terraform Scripts\Docker> |
```

10. Using terraform destroy we can destroy the container.

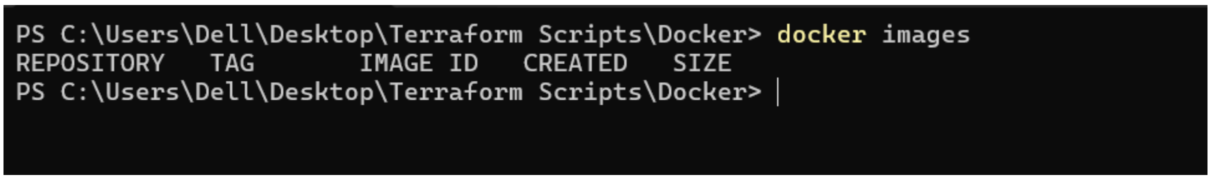


```
PS C:\Users\Dell\Desktop\Terraform Scripts\Docker> terraform destroy
docker_image.ubuntu: Refreshing state... [id=sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598aubuntu:latest]
docker_container.foo: Refreshing state... [id=334f576e574fb630507997954b3a9f5c39af4e6aa10b085a0a3a5122ee19ef81]

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_container.foo will be destroyed
  - resource "docker_container" "foo" {
      - attach            = false -> null
      - command           = [
          - "sleep",
          - "infinity",
        ] -> null
      - cpu_shares        = 0 -> null
      - dns               = [] -> null
      - dns_opts          = [] -> null
      - dns_search        = [] -> null
      - entrypoint        = [] -> null
      - env               = [] -> null
      - gateway           = "172.17.0.1" -> null
      - group_add         = [] -> null
      - hostname          = "334f576e574f" -> null
      - id                = "334f576e574fb630507997954b3a9f5c39af4e6aa10b085a0a3a5122ee19ef81" -> null
      - image             = "sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598a" -> null
      - init              = false -> null
      - ip_address        = "172.17.0.2" -> null
      - ip_prefix_length  = 16 -> null
      - ipc_mode          = "private" -> null
      - links             = [] -> null
      - log_driver        = "json-file" -> null
      - log_opts          = {} -> null
      - logs              = false -> null
      - max_retry_count   = 0 -> null
      - memory            = 0 -> null
      - memory_swap       = 0 -> null
      - must_run          = true -> null
      - name              = "foo" -> null
      - network_data      = [
```



```
      - rm                = false -> null
      - runtime           = "runc" -> null
      - security_opts     = [] -> null
      - shm_size          = 64 -> null
      - start             = true -> null
      - stdin_open        = false -> null
      - stop_timeout      = 0 -> null
      - storage_opts      = {} -> null
      - sysctls           = {} -> null
      - tmpfs             = {} -> null
      - tty               = false -> null
        # (8 unchanged attributes hidden)
    }

  # 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, 2 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_container.foo: Destroying... [id=334f576e574fb630507997954b3a9f5c39af4e6aa10b085a0a3a5122ee19ef81]
docker_container.foo: Destruction complete after 0s
docker_image.ubuntu: Destroying... [id=sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598aubuntu:latest]
docker_image.ubuntu: Destruction complete after 0s

Destroy complete! Resources: 2 destroyed.
```

Now when we run docker images we will see that the container information is not visible that means it has been deleted successfully.



```
PS C:\Users\Dell\Desktop\Terraform Scripts\Docker> docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
PS C:\Users\Dell\Desktop\Terraform Scripts\Docker>
```