Name: Shejal Tiwari Div/Roll no: D15C/57

AdvanceDevops Experiment: 6

AIM: To Build, change, and destroy AWS /GCP/ Microsoft Azure/ Digital Ocean infrastructure using Terraform. (S3 bucket or Docker)

To Create docker image using terraform

We need Download and Install Docker Desktop from https://www.docker.com/

Step 1: Check the docker functionality

```
PS C:\Users\INFT505-16> 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\INFT505-16> docker --version
Docker version 24.0.6, build ed223bc
PS C:\Users\INFT505-16>
```

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. Copy the Script into it. **Script:**

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

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

# Pulls the Ubuntu 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

```
C:\Users\INFT505-16>cd desktop\TerraformScripts\Docker

C:\Users\INFT505-16\Desktop\TerraformScripts\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.
```

Step 4: 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”

```
C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>terraform apply

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",
          + "3600",
        ]
      + 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)
```

```
      + 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: Still creating... [40s elapsed]
docker_image.ubuntu: Still creating... [50s elapsed]
docker_image.ubuntu: Creation complete after 53s [id=sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598aubuntu:latest]
docker_container.foo: Creating...
docker_container.foo: Creation complete after 2s [id=353bd0cae537e335931797ed86d2b603c682520b71c2af7d5b72f3c09eed2b11]
```

Docker images, After Executing Apply step:

```
C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
ubuntu       latest    edbfe74c41f8   3 weeks ago      78.1MB
sonarqube    latest    3183d6818c6e   10 months ago    716MB
```

Step 5: Execute Terraform destroy to delete the configuration, which will automatically delete the Ubuntu Container.

```
C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>terraform destroy
docker_image.ubuntu: Refreshing state... [id=sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598aubuntu:latest]
docker_container.foo: Refreshing state... [id=353bd0cae537e335931797ed86d2b603c682520b71c2af7d5b72f3c09eed2b11]

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",
          - "3600",
        ] -> 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
```

```
  # 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=353bd0cae537e335931797ed86d2b603c682520b71c2af7d5b72f3c09eed2b11]
docker_container.foo: Destruction complete after 0s
docker_image.ubuntu: Destroying... [id=sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598aubuntu:latest]
docker_image.ubuntu: Destruction complete after 1s

Destroy complete! Resources: 2 destroyed.
```

Step 6: This command outputs the state or plan in a human-readable format, helping you review the details of your configuration..

```
C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>terraform show
# docker_container.foo:
resource "docker_container" "foo" {
    attach            = false
    bridge            = null
    command           = [
        "sleep",
        "3600",
    ]
    cpu_set           = null
    cpu_shares        = 0
    domainname        = null
    entrypoint        = []
    env               = []
    gateway           = "172.17.0.1"
    hostname          = "353bd0cae537"
    id                = "353bd0cae537e335931797ed86d2b603c682520b71c2af7d5b72f3c09eed2b11"
    image             = "sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598a"
    init              = false
    ip_address        = "172.17.0.2"
    ip_prefix_length  = 16
```

Step 7: This command generates a visual graph of your Terraform resources, which can help you understand the dependencies and relationships between them.

```
C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>terraform graph
digraph G {
  rankdir = "RL";
  node [shape = rect, fontname = "sans-serif"];
  "docker_container.foo" [label="docker_container.foo"];
  "docker_image.ubuntu" [label="docker_image.ubuntu"];
  "docker_container.foo" -> "docker_image.ubuntu";
}
```

Step 8: This command lists all the resources tracked by the Terraform state, allowing you to see which resources have been created and are being managed by Terraform..

```
C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>terraform state list
docker_container.foo
docker_image.ubuntu

C:\Users\INFT505-16\Desktop\TerraformScripts\Docker>
```