

Name: Chinmay Chaudhari
Div: D15C
Roll NO:06

Experiment No.: 6

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

```
PS C:\Users\91773> 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
```

```
PS C:\Users\91773> docker --version
Docker version 27.0.3, build 7d4bcd8
PS C:\Users\91773> |
```

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 following contents into it to create a Ubuntu Linux container.

Script:

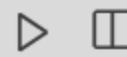
```
terraform {
  required_providers {
    docker = {
      source = "kreuzwerker/docker"
      version = "2.21.0"
    }
  }
}

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

# 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"]
}
```

Y docker.tf X



Y docker.tf

```
1 terraform {
2   required_providers {
3     docker = {
4       source = "kreuzwerker/docker"
5       version = "2.21.0"
6     }
7   }
8 }
9
10 provider "docker" {
11   host = "npipe://///pipe/docker_engine"
12 }
13
14 # Pull the image
15 resource "docker_image" "ubuntu" {
16   name = "ubuntu:latest"
17 }
18
19 # Create a container
20 resource "docker_container" "foo" {
21   image = docker_image.ubuntu.image_id
22   name   = "foo"
23   command = ["sleep", "3600"]
24 }
25
```

Step 3: Execute Terraform Init command to initialize the resources

```
PS C:\Users\91773\Desktop\College Resources\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 plan to see the available resources

```
PS C:\Users\91773\Desktop\College Resources\TerraformScripts\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)
```

```

+ 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\91773\Desktop\College Resources\TerraformScripts\Docker> terraform apply
docker_image.ubuntu: Refreshing state... [id=sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad763
tu:latest]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated
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",
+   ]
}

```

Docker images, Before Executing Apply step:

```

PS C:\Users\91773\Desktop\College Resources\TerraformScripts\Docker> docker images
REPOSITORY      TAG          IMAGE ID      CREATED      SIZE

```

Docker images, After Executing Apply step:

```
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
PS C:\Users\91773\Desktop\College Resources\TerraformScripts\Docker> docker images
REPOSITORY    TAG          IMAGE ID      CREATED      SIZE
ubuntu        latest       edbfe74c41f8  2 weeks ago  78.1MB
PS C:\Users\91773\Desktop\College Resources\TerraformScripts\Docker> |
```

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

```
PS C:\Users\91773\Desktop\College Resources\TerraformScripts\Docker> terraform destroy
docker_image.ubuntu: Refreshing state... [id=sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598a]
docker_container.foo: Refreshing state... [id=f03a28e4658896c23c9992f7a98eb1011befc7d014e997ea9fc6372da70b7903]

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

# docker_image.ubuntu will be destroyed
- resource "docker_image" "ubuntu" {
  - id              = "sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598a" -> 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=f03a28e4658896c23c9992f7a98eb1011befc7d014e997ea9fc6372da70b7903]
docker_container.foo: Destruction complete after 0s
docker_image.ubuntu: Destroying... [id=sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598a]
docker_image.ubuntu: Destruction complete after 1s

Destroy complete! Resources: 2 destroyed.
```

Docker images After Executing Destroy step

```
PS C:\Users\91773\Desktop\College Resources\TerraformScripts\Docker> docker images
REPOSITORY    TAG          IMAGE ID      CREATED      SIZE
```