

Name: Manav Jawrani

Roll No.: 19

Subject: Advanced DevOps

Experiment No.: 6

Experiment 6.

Aim: To Build, Change and destroy AWS/GCP/Microsoft Azure /Digital ocean Infrastructure using Terraform.

Theory:

Terraform is an open-source "Infrastructure as Code" tool created by Hashi Corp.

What is Infrastructure as Code?

Infrastructure as Code (IaC) is a widespread terminology among DevOps professionals and a key DevOps practice in the industry. It is the process of managing and provisioning the complete infrastructure comprising of both physical and virtual machines using machine readable definition files.

Terraform providers

A provider is responsible for understanding API interactions and exposing resources. It is an executable plug-in that contains code necessary to interact with API of the service. Terraform Configuration must declare which providers they require. So that Terraform can install and use them.

• Terraform Configuration Files

Configuration files are a set of files used to describe infrastructure in Terraform and have the file extensions `.tf` and `.tf.json`. Terraform uses a declarative model for defining infrastructure. Configuration files let you write a configuration that declares your desired state. Configuration files are made up of resources with settings and values representing the desired state of your infrastructure.

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

```
Windows PowerShell
PS C:\Users\MANAV> docker

Usage: docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
  --config string      Location of client config files (default
                        "C:\\Users\\MANAV\\.docker")
  -c, --context string  Name of the context to use to connect to the
                        daemon (overrides DOCKER_HOST env var and
                        default context set with "docker context use")
  -D, --debug           Enable debug mode
  -H, --host list       Daemon socket(s) to connect to
  -l, --log-level string Set the logging level
                        ("debug"|"info"|"warn"|"error"|"fatal")
                        (default "info")
  --tls                Use TLS; implied by --tlsverify
  --tlscacert string    Trust certs signed only by this CA (default
                        "C:\\Users\\MANAV\\.docker\\ca.pem")
  --tlscert string      Path to TLS certificate file (default
                        "C:\\Users\\MANAV\\.docker\\cert.pem")
  --tlskey string        Path to TLS key file (default
                        "C:\\Users\\MANAV\\.docker\\key.pem")
  --tlsverify           Use TLS and verify the remote
  -v, --version         Print version information and quit

Management Commands:
  builder              Manage builds
  buildx*              Docker Buildx (Docker Inc., v0.9.1)
  compose*             Docker Compose (Docker Inc., v2.10.2)
  config               Manage Docker configs
  container             Manage containers
  context               Manage contexts
  extension*           Manages Docker extensions (Docker Inc., v0.2.9)
  image                Manage images
  manifest              Manage Docker image manifests and manifest lists
  network              Manage networks
  node                 Manage Swarm nodes
  plugin               Manage plugins
```

```
Windows PowerShell
PS C:\Users\MANAV> docker --version
Docker version 20.10.17, build 100c701
PS C:\Users\MANAV> |
```

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 ‘Terraform Scripts’ 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"  
}  
  
# 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"  
}
```

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 # Pulls 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 }
24
```

Step 3: Execute Terraform Init command to initialize the resources

```
Windows PowerShell
PS E:\Terraform Script\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 E:\Terraform Script\Docker> |
```

Step 4: Execute Terraform plan to see the available resources

```
PS E:\Terraform Script\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)
```

```

+ 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)
+ tty               = false

+ healthcheck {
+   interval = (known after apply)
+   retries  = (known after apply)
+   start_period = (known after apply)
+   test     = (known after apply)
+   timeout  = (known after apply)
+ }

+ labels {
+   label = (known after apply)
+   value = (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.

```

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
PS E:\Terraform Script\docker> |

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**”

```

# 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: Still creating... [10s elapsed]
docker_image.ubuntu: Still creating... [20s elapsed]
docker_image.ubuntu: Still creating... [30s elapsed]
docker_image.ubuntu: Creation complete after 30s [id=sha256:2dc39ba059dcd42ade30aae30147b5692777ba9ff0779a62ad93a74de02e3efubuntu:latest]
docker_container.foo: Creating...

```

Docker images, Before Executing Apply step:

```

PS E:\Terraform Script\docker> docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
docker101tutorial   latest      e5bc7d28c9a0  33 minutes ago  28.9MB
manavjawrani/docker101tutorial   latest      e5bc7d28c9a0  33 minutes ago  28.9MB
alpine/git          latest      692618a0d74d  2 weeks ago    43.4MB
PS E:\Terraform Script\docker> |

```


Docker images, After Executing Apply step:

```
Windows PowerShell
PS E:\Terraform Script\Docker> docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
docker101tutorial   latest      e5bc7d28c9a0  30 minutes ago 28.9MB
manavjawrani/docker101tutorial latest      e5bc7d28c9a0  30 minutes ago 28.9MB
ubuntu              latest      2dc39ba059dc  2 weeks ago   77.8MB
alpine/git          latest      692618a0d74d  2 weeks ago   43.4MB
PS E:\Terraform Script\Docker> |
```

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

```
PS E:\Terraform Script\Docker> terraform destroy
docker_image.ubuntu: Refreshing state... [id=sha256:2dc39ba059dcd42ade30aae30147b5692777ba9ff0779a62ad93a74de02e3e1fubuntu: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       = "sha256:2dc39ba059dcd42ade30aae30147b5692777ba9ff0779a62ad93a74de02e3e1fubuntu:latest" -> null
  - image_id = "sha256:2dc39ba059dcd42ade30aae30147b5692777ba9ff0779a62ad93a74de02e3e1f" -> null
  - latest   = "sha256:2dc39ba059dcd42ade30aae30147b5692777ba9ff0779a62ad93a74de02e3e1f" -> null
  - name     = "ubuntu:latest" -> null
  - repo_digest = "ubuntu@sha256:20fa2d7bb4de7723f542be5923b06c4d704370f0390e4ae9e1c833c8785644c1" -> 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=sha256:2dc39ba059dcd42ade30aae30147b5692777ba9ff0779a62ad93a74de02e3e1fubuntu:latest]
docker_image.ubuntu: Destruction complete after 1s

Destroy complete! Resources: 1 destroyed.
PS E:\Terraform Script\Docker> |
```

Docker images After Executing Destroy step:

```
PS E:\Terraform Script\Docker> docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
docker101tutorial   latest      e5bc7d28c9a0  33 minutes ago 28.9MB
manavjawrani/docker101tutorial latest      e5bc7d28c9a0  33 minutes ago 28.9MB
alpine/git          latest      692618a0d74d  2 weeks ago   43.4MB
PS E:\Terraform Script\Docker> |
```

Conclusion :

In this experiment, we learned how to create, update and delete Docker images using Terraform and its commands. When preparing the environment for the implementation, we ran into a few problems with path and configuration but we were able to fix them and the implementation for creating images was successful.