Shravani Anil Patil
D15A-38

EXPERIMENT NO. 6

Aim :To Build, change, and destroy AWS infrastructure Using Terraform (S3 bucket or Docker) .

Theory :

Terraform is an open-source tool that enables developers and operations teams to define, provision, and manage cloud infrastructure through code. It uses a declarative language to specify the desired state of infrastructure, which can include servers, storage, networking components, and more. With Terraform, infrastructure changes can be automated, versioned, and tracked efficiently.

Building Infrastructure

When you build infrastructure using Terraform, you define the desired state of your infrastructure in configuration files. For example, you may want to create an S3 bucket or deploy a Docker container on an EC2 instance. Terraform reads these configuration files and, using the specified cloud provider (such as AWS), it provisions the necessary resources to match the desired state.

- **Docker on AWS:** Terraform can deploy Docker containers on AWS infrastructure. This often involves setting up an EC2 instance and configuring it to run Docker containers, which encapsulate applications and their dependencies.

Changing Infrastructure

As your needs evolve, you may need to modify the existing infrastructure. Terraform makes it easy to implement changes by updating the configuration files to reflect the new desired state. For instance, you might want to change the storage settings of an S3 bucket, add new security policies, or modify the Docker container's configuration.

Terraform's "plan" command helps you preview the changes that will be made to your infrastructure before applying them. This step ensures that you understand the impact of your changes and can avoid unintended consequences.

Destroying Infrastructure

When certain resources are no longer needed, Terraform allows you to destroy them in a controlled manner. This might involve deleting an S3 bucket or terminating an EC2 instance running Docker containers. By running the "destroy" command, Terraform ensures that all associated resources are properly de-provisioned and removed.

Destroying infrastructure with Terraform is beneficial because it helps avoid unnecessary costs associated with unused resources and ensures that the environment remains clean and free of clutter.

Benefits of Using Terraform for AWS Infrastructure

1. **Consistency:** Terraform ensures that infrastructure is consistent across environments by applying the same configuration files.

2. **Automation:** Manual processes are reduced, and infrastructure is provisioned, updated, and destroyed automatically based on code.
3. **Version Control:** Infrastructure configurations can be stored in version control systems (like Git), allowing teams to track changes, collaborate, and roll back if necessary.
4. **Scalability:** Terraform can manage complex infrastructures, scaling them up or down as needed, whether for small projects or large-scale applications.
5. **Modularity:** Terraform configurations can be broken down into reusable modules, making it easier to manage and scale infrastructure.

Implementation :

Terraform and Docker -

Step 1 : check docker installation and version

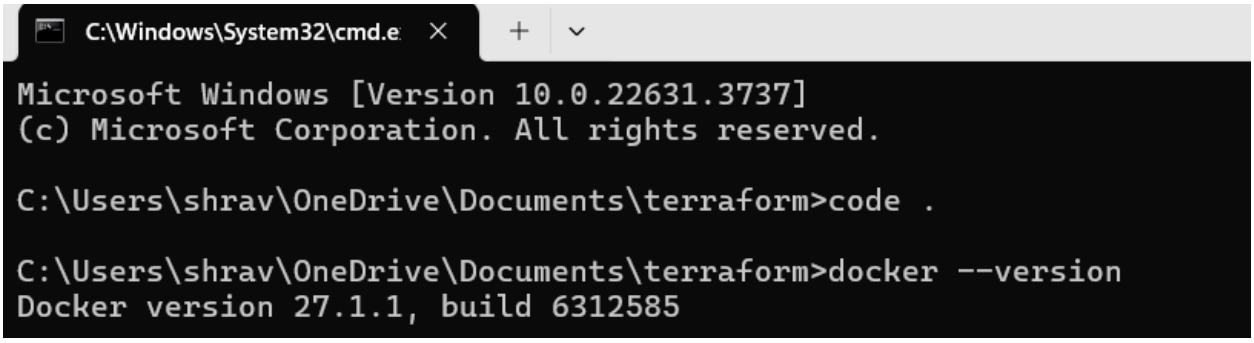
```
C:\Users\shrav\OneDrive\Documents\terraform>docker

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
```



Step 2 : create docker.tf file and write following code for terraform and docker

Code -

```
terraform { required_providers {
    docker = {
      source = "kreuzwerker/docker" version =
      "~> 3.0.1"
   }
  }
}
provider "docker" {
host         = "npipe:////.//pipe//docker_engine"
}
resource "docker_image" "nginx" {
  name = "nginx:latest" keep_locally = false
} resource "docker_container" "nginx" {
  image   =   docker_image.nginx.image_id
  name = "tutorial" ports {
    internal = 80 external
    = 8000 }
}
```

credentials.txt | provider.tf | docker.tf | main.tf

docker > docker.tf > provider "docker"

```
terraform {
    required_providers {
docker = {
source = "kreuzwerker/docker"
version = "~> 3.0.1"
}
}
}
provider "docker" {
host    = "npipe:////.//pipe//docker_engine"
}
 resource "docker_image" "nginx" {
name    = "nginx:latest"
keep_locally = false
}
resource "docker_container" "nginx" {
image = docker_image.nginx.image_id
name = "tutorial"
ports {
    internal = 80
     external = 8000
     }
}

```

Step 3 : Type terraform init command to initialize terraform backend

```
shrav@LAPTOP-0MELEBGI MINGW64 ~/OneDrive/Documents/terraform/docker
$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding kreuzwerker/docker versions matching "~> 3.0.1"...
- Installing kreuzwerker/docker v3.0.2...
- Installed kreuzwerker/docker v3.0.2 (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.
```

Step 4(EXTRA) : type terraform fmt and validate commands .
The two Terraform commands – terraform validate and terraform fmt – are used to maintain a clean, error-free, and well-structured Terraform codebase.

```
shrav@LAPTOP-0MELEBGI MINGW64 ~/OneDrive/Documents/terraform/docker
$ terraform fmt
docker.tf

shrav@LAPTOP-0MELEBGI MINGW64 ~/OneDrive/Documents/terraform/docker
$ terraform validate
Success! The configuration is valid.
```

Step 5 : Type Terraform plan command to create execution plan .

```
shrav@LAPTOP-0MELEBGI MINGW64 ~/OneDrive/Documents/terraform/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.nginx will be created
  + resource "docker_container" "nginx" {
      + attach                                      = false
      + bridge                                      = (known after apply)
      + command                                     = (known after apply)
      + container_logs                              = (known after apply)
      + container_read_refresh_timeout_milliseconds = 15000
      + entrypoint                                  = (known after apply)
      + env                                         = (known after apply)
      + exit_code                                   = (known after apply)
      + hostname                                    = (known after apply)
      + id                                          = (known after apply)
      + image                                       = (known after apply)
      + init                                        = (known after apply)
      + ipc_mode                                    = (known after apply)
      + log_driver                                  = (known after apply)
      + logs                                        = false
      + must_run                                    = true
      + name                                        = "tutorial"
      + network_data                                = (known after apply)
      + read_only                                   = false
      + remove_volumes                              = true
      + restart                                     = "no"
      + rm                                          = false
      + runtime                                     = (known after apply)
      + security_opts                               = (known after apply)
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  PORTS                                  bash -

      + 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
      + wait                                        = false
      + wait_timeout                                = 60

      + healthcheck (known after apply)

      + labels (known after apply)

      + ports {
          + external = 8000
          + internal = 80
          + ip       = "0.0.0.0"
          + protocol = "tcp"
        }
    }

  # docker_image.nginx will be created
  + resource "docker_image" "nginx" {
      + id           = (known after apply)
      + image_id     = (known after apply)
      + keep_locally = false
      + name         = "nginx:latest"
      + 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.
```

Step 6 : Type terraform apply to apply changes .

```
docker_image.nginx: Still creating... [1m40s elapsed]
docker_image.nginx: Still creating... [1m50s elapsed]
docker_image.nginx: Still creating... [2m0s elapsed]
docker_image.nginx: Still creating... [2m10s elapsed]
docker_image.nginx: Still creating... [2m20s elapsed]
docker_image.nginx: Still creating... [2m30s elapsed]
docker_image.nginx: Still creating... [2m40s elapsed]
docker_image.nginx: Still creating... [2m50s elapsed]
docker_image.nginx: Still creating... [3m0s elapsed]
docker_image.nginx: Creation complete after 3m1s [id=sha256:5ef79149e0ec84a7a9f9284c3f91aa3c20608f8391f5445eabe92ef07dbda03cnginx:latest]
docker_container.nginx: Creating...
docker_container.nginx: Creation complete after 8s [id=0b844cbfdc3ee0fca75bbf5a6577f7a7d824bd4867787cd570085b011ecdb82e]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

shrav@LAPTOP-0MELEBGI MINGW64 ~/OneDrive/Documents/terraform/docker
$
```

Step 7 : Docker container after step 6 execution BEFORE -
AFTER -

```
shrav@LAPTOP-0MELEBGI MINGW64 ~/OneDrive/Documents/terraform/docker
$ docker container list
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                   NAMES
0b844cbfdc3e   5ef79149e0ec   "/docker-entrypoint.…"   2 minutes ago   Up 2 minutes   0.0.0.0:8000->80/tcp   tutorial

shrav@LAPTOP-0MELEBGI MINGW64 ~/OneDrive/Documents/terraform/docker
$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
nginx        latest    5ef79149e0ec   2 weeks ago   188MB
```

Step 8 (EXTRA) : Execution of change .

docker.tf

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0.1"
    }
  }
}

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

resource "docker_image" "nginx" {
  name         = "nginx:latest"
  keep_locally = false
}

resource "docker_container" "nginx" {
  image = docker_image.nginx.image_id
  name  = "tutorial"
  ports {
    internal = 80
    external = 8080
  }
}

```

```
PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE   PORTS

          + tmpfs                                    = (known after apply)
          + tty                                      = (known after apply)
          + user                                     = (known after apply)
          + userns_mode                              = (known after apply)
          + wait                                     = (known after apply)
          + wait_timeout                             = (known after apply)
          + working_dir                              = (known after apply)
        } -> (known after apply)

      ~ ports {
          ~ external = 8000 -> 8080 # forces replacement
            # (3 unchanged attributes hidden)
        }
    }

Plan: 1 to add, 0 to change, 1 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.nginx: Destroying... [id=0b844cbfdc3ee0fca75bbf5a6577f7a7d824bd4867787cd570085b011ecdb82e]
docker_container.nginx: Destruction complete after 3s
docker_container.nginx: Creating...
```

Step 9 : terraform destroy to destroy infrastructure.

```
shrav@LAPTOP-0MELEBGI MINGW64 ~/OneDrive/Documents/terraform/docker
$ terraform destroy
docker_image.nginx: Refreshing state... [id=sha256:5ef79149e0ec84a7a9f9284c3f91aa3c20608f8391f5445eabe92ef07dbda03cnginx:latest]
docker_container.nginx: Refreshing state... [id=fd0e4e24940ef385ecf3b680240ce727964686a2af4e4cfea5b7d70b4cd9a32d]

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.nginx will be destroyed
  - resource "docker_image" "nginx" {
      - id           = "sha256:5ef79149e0ec84a7a9f9284c3f91aa3c20608f8391f5445eabe92ef07dbda03cnginx:latest" -> null
      - image_id     = "sha256:5ef79149e0ec84a7a9f9284c3f91aa3c20608f8391f5445eabe92ef07dbda03c" -> null
      - keep_locally = false -> null
      - name         = "nginx:latest" -> null
      - repo_digest  = "nginx@sha256:447a8665cc1dab95b1ca778e162215839ccbb9189104c79d7ec3a81e14577add" -> 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.
```

```
  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.nginx: Destroying... [id=sha256:5ef79149e0ec84a7a9f9284c3f91aa3c20608f8391f5445eabe92ef07dbda03cnginx:latest]
  docker_image.nginx: Destruction complete after 2s

  Destroy complete! Resources: 1 destroyed.

  shrav@LAPTOP-0MELEBGI MINGW64 ~/OneDrive/Documents/terraform/docker
  $
```

Step 10 : Docker after destroy command.