**-: IaaC – Terraform (Docker, AWS) :-**

Terraform is an open-source infrastructure as code software tool that provides a consistent CLI workflow to manage hundreds of cloud services. Terraform codifies cloud APIs into declarative configuration files.

**Deliver Infrastructure as Code**

* **Write**

Write infrastructure as code using declarative configuration files. HashiCorp Configuration Language (HCL) allows for concise descriptions of resources using blocks, arguments, and expressions.

* **Plan**

Run terraform plan to check whether the execution plan for a configuration matches your expectations before provisioning or changing infrastructure.

* **Apply**

Apply changes to hundreds of cloud providers with terraform apply to reach the desired state of the configuration.

**Installation of Terraform:** Linux – Ubuntu

* Install HashiCorp's Debian package repository.

sudo apt-get update && sudo apt-get install -y gnupg software-properties-common curl

* Add the HashiCorp GPG Key

curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add –

* Add the official HashiCorp Linux repository.

sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb\_release -cs) main"

* Update to add the repository, and install the Terraform CLI.

sudo apt-get update && sudo apt-get install terraform

**Terraform sections required in configuration file ( main.tf )**

1. Terraform Block
   1. The terraform {} block contains Terraform settings, including the required providers Terraform will use to provision your infrastructure. For each provider, the source attribute defines an optional hostname, a namespace, and the provider type.
2. Providers
   1. The provider block configures the specified provider, in this case aws. A provider is a plugin that Terraform uses to create and manage your resources.
3. Resources
   1. Use resource blocks to define components of your infrastructure. A resource might be a physical or virtual component such as an EC2 instance, or it can be a logical resource such as a Docker application.

**Deploying NGNIX Server using Terraform**

* Ensure Docker engine is installed in your box.
* Create a directory named learn-terraform-docker-container
  + mkdir learn-terraform-docker-container
* Navigate to it
  + cd learn-terraform-docker-container
* Paste the following Terraform configuration into a file and name it main.tf

terraform {

required\_providers {

docker = {

source = "kreuzwerker/docker"

version = "~> 2.13.0"

}

}

}

provider "docker" {}

resource "docker\_image" "nginx" {

name = "nginx:latest"

keep\_locally = false

}

resource "docker\_container" "nginx" {

image = docker\_image.nginx.latest

name = "tutorial"

ports {

internal = 80

external = 8000

}

}

* Initialize the project, which downloads a plugin that allows Terraform to interact with Docker.
  + terraform init
* Provision the NGINX server container with apply. When Terraform asks you to confirm type yes and press ENTER.
  + terraform apply
* Verify the existence of the NGINX container by visiting **localhost:8000** in your web browser or running **docker p**s to see the container.
* To make sure your configuration is syntactically valid and internally consistent by using the terraform validate command
  + terraform validate

Success! The configuration is valid.

* Inspect the current state using terraform show
* To stop the container, run terraform destroy.
  + terraform destroy

**Terraform in AWS: Requirements**

* 1. The Terraform CLI Installed
  2. AWS : AWS CLI Installed, AWS Account with Credentials
* Configure AWS Credentials with : aws configure
* Create a folder – mkdir my\_terraform
* CD to above directory – cd my\_terraform
* Create a terraform file – ***vi main.tf***

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 3.27"

}

}

required\_version = ">= 0.14.9"

}

provider "aws" {

profile = "default"

region = "us-west-2"

}

resource "aws\_instance" "app\_server" {

ami = "ami-830c94e3"

instance\_type = "t2.micro"

tags = {

Name = "VBhaskar\_Training\_Instance"

}

}

1. Format your configuration.
   1. terraform fmt
2. Apply the configuration
   1. terraform apply
3. Inspect the current state
   1. terraform show
4. Destroy terraform resources
   1. terraform plan --destroy
   2. terraform destroy