**Terraform**

We can normally create the resources using the services of the AWS through User interface in AWS console.

Suppose there is a request for creation of more resources like 50 EC2 instances, S3 buckets, etc which will take more time to create. For this AWS came with a solution where we can use the AWS CFT to create the resources using a template written in JSON or YAML.

We can create the entire infrastructure using the CFT in AWS due to which it is also called as Infrastructure as Code (IAC).

Here to communicate with different cloud platforms we can use the API’s of the cloud platforms.

Its ok we can create in AWS but if your organization is using more than one cloud platform then it will be difficult as each cloud provider has different templates.

Terraform came up with a universal approach and in terraform we have to define the cloud provider with which the terraform will communicate with that particular cloud provider.

Terraform is written in HashiCorp Configuration Language(HCL). The terraform converts the HCLs into API’s of the cloud providers.

Through CLI if you want to create resources in AWS we need to configure using the Access key and Secret Access Key.

This allows the terraform to communicate with AWS account with API calls.

**Commands:**

terraform init – to identify the cloud provider and authenticate it using the main.tf file

terraform plan – shows what will be created after the resource is created

terraform apply – creates the resource

terraform destroy – will destroy the resources created by the terraform

**Creating EC2 instance in AWS with terraform:**

Create a file with name **main.tf**

provider "aws" {

region = "us-east-1" # Choose your preferred AWS region

}

resource "aws\_instance" "example" {

ami = "ami-0c614dee691cbbf37"

instance\_type = "t2.micro"

subnet\_id = “ ”

key\_name = “ ” }

After writing the code in file save it.

Use command terraform init >> terraform plan >> terraform apply.

**Terraform state file (terraform.tfstate):**

This will record all the details of resources that are created by terraform in the state file.

**Providers:**

The cloud providers name is mentioned at the beginning of the code. Based on mentioned name the terraform will convert the code into API calls of that particular provider.

We have providers like AWS, Azure, GCP, Kubernetes, etc.

**Multi-region:**

We have to create resources in different regions.

# Configure AWS provider for us-east-1

provider "aws" {

alias = "us\_east\_1"

region = "us-east-1"

}

# Configure AWS provider for us-west-2

provider "aws" {

alias = "us\_west\_2"

region = "us-west-2"

}

# Create an EC2 instance in the us-east-1 region

resource "aws\_instance" "us\_east\_instance" {

provider = aws.us\_east\_1

ami = "ami-0c55b159cbfafe1f0" # Example Amazon Linux 2 AMI (change based on your region)

instance\_type = "t2.micro" }

}

# Create an EC2 instance in the us-west-2 region

resource "aws\_instance" "us\_west\_instance" {

provider = aws.us\_west\_2

ami = "ami-0c55b159cbfafe1f0" # Example Amazon Linux 2 AMI (change based on your region)

instance\_type = "t2.micro" } }

**Multi- cloud:**

We can create resources in multiple cloud platform in a single code.

# Configure AWS provider

provider "aws" {

alias = "aws"

region = "us-east-1" # AWS region

}

# Configure Azure provider

provider "azurerm" {

alias = "azure"

features {}

}

# Create an EC2 instance in AWS

resource "aws\_instance" "example" {

provider = aws

ami = "ami-0c55b159cbfafe1f0" # Amazon Linux 2 AMI

instance\_type = "t2.micro"

}

}

resource "azurerm\_resource\_group" "example" {

provider = azurerm

name = "example-resources"

location = "East US"

}

resource "azurerm\_virtual\_network" "example" {

provider = azurerm name = "example-network"

address\_space = ["10.0.0.0/16"]

location = azurerm\_resource\_group.example.location resource\_

group\_name = azurerm\_resource\_group.example.name

}

**Variables:**

Hardcoding the code is not a good practice as we have to use different codes for different requirements. For this we have used variables so that the user can change the AMI image type, name of instance, etc. Using a variable will give us an advantage so that we can pass the parameters.

We have two types of variables

**Output variable:**

Gives the output after the resource is created like IP address, instance ID.

**Input Variable:**

Parameterizing the variables.

**Code:**

# Variables Demo

# Define an input variable for the EC2 instance type

variable "instance\_type" {

description = "EC2 instance type"

type = string

default = "t2.micro"

}

# Define an input variable for the EC2 instance AMI ID

variable "ami\_id" {

description = "EC2 AMI ID"

type = string

}

# Configure the AWS provider using the input variables

provider "aws" {

region = "us-east-1"

}

# Create an EC2 instance using the input variables

resource "aws\_instance" "example\_instance" {

ami = var.ami\_id

instance\_type = var.instance\_type

}

# Define an output variable to expose the public IP address of the EC2 instance

output "public\_ip" {

description = "Public IP address of the EC2 instance"

value = aws\_instance.example\_instance.public\_ip

}

Here we have Input, Resource and Output section.

We have declared two variables for input and the inputs are taken during the terraform apply.

The output variable will give the required details from the instance creation.

Writing the code in a single file will make clumsy and not easy for understanding.

As a solution for this we can split the code into multiple files as follows below

provider.tf - we mention the cloud provider

input.tf - we use the variables

resource.tf - write the required resource

output.tf - outputs we needed from the created resource.

terraform.tfvars - here we will give the inputs to variables.

**Conditions:**

In programming we have IF to work with conditions. Similarly, we have conditional operators.

condition ? true\_val : false\_val

**Code:**

variable "environment" {

description = "Environment type"

type = string

default = "development"

}

variable "production\_subnet\_cidr" {

description = "CIDR block for production subnet"

type = string

default = "10.0.1.0/24"

}

variable "development\_subnet\_cidr" {

description = "CIDR block for development subnet"

type = string

default = "10.0.2.0/24"

}

resource "aws\_security\_group" "example" {

name = "example-sg"

description = "Example security group"

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = var.environment == "production" ? [var.production\_subnet\_cidr] : [var.development\_subnet\_cidr]

}}

Here the var.environment will be given manually and if the environment is production then the production related subnet will be assigned to the resource if not then development subnet will be assigned.