**TERRAFORM**

* **Terraform is an open source s/w created by HashiCorp and written in Go programming language**
* Terraform is an **infrastructure as code** (IaaC) software tool,
* Infrastructure as code is the process of managing infrastructure in a file or files rather than manually configuring resources in a user interface (UI)
* In Terraform resources are nothing but Virtual machines, Elastic IP, Security Groups, Network interfaces
* Terraform Code is written in the **HashiCorp Configuration language (HCL)** in files with the extension **.tf**
* Terraform allows users to use HashiCorp Configuration Language (HCL) to create the files containing definitions of the their desired resources
* Terraform supports all most all cloud providers (AWS, AZURE, GCP, Openstack etc.). To automate **infrastructure creation we will use Terraform**.

**Terraform v/s Cloud Formation**

|  |  |
| --- | --- |
| **Terraform** | **Cloud Formation** |
| Terraform developed by HashiCorp | CloudFormation developed by AWS |
| Terraform supports many cloud providers | Cloud Formation will support only in AWS |
| Terraform uses HashiCorp configuration language (HCL) which built by HashiCorp. It is fully compatible with JSON. | AWS Cloud Formation utilizes either JSON or YAML. Cloud formation has a limit of 51,000 bytes for the template body itself. |

**Terraform V/s Ansible**

|  |  |
| --- | --- |
| **Terraform** | **Ansible** |
| Terraform developed by HashiCorp | Ansible is also an open source software |
| Terraform is an infrastructure as a Code, which means they are designed to provision the servers themselves. | Ansible is a configuration management tool. This means ansible designed to install and manage software on existing servers. |
| Terraform is ideal for creating, managing and improving infrastructure. | Ansible is ideal for software provisioning, application deployment and configuration management. |

**Pre-Requisites**

1) Any Cloud Provider (AWS, Azure, GCP, Openstack etc.)

2) IAM User account (Secret Key and Access Key)

3) IAM User should have AmazonEc2FullAccess

1) Create EC2 instance (RED HAT Linux)

2) Connect to EC2 VM using Mobaxterm

3) Swith to root user

**$ sudo su -**

4) Install unzip software

**$ yum install wget unzip vim -y**

5) Download Terraform Software (https://www.terraform.io/downloads)

**$ sudo yum install -y yum-utils**

**$ sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/RHEL/hashicorp.repo**

**$ sudo yum -y install terraform**

6) Check Terraform Version

**$ terraform -v**

7) Create IAM User with Programmatic Access and Provide 'AmazonEc2FullAccess'

8) Download Secret Key and Access Key

9) Write Terraform Script

**$ mkdir terraformscript**

**$ cd terraformscripts**

**$ vi FirstTFScript.tf**

provider "aws" {

region = "ap-south-1"

access\_key = "your-access-key"

secret\_key = "your-secret-key"

}

resource "aws\_instance" "AWSServer" {

ami = "ami-05c8ca4485f8b138a"

instance\_type = "t2.micro"

key\_name = "linux"

security\_groups = ["launch-wizard-1"]

tags = {

Name = "Terrafrom Server"

}

}

10) Initialize Terraform using init command

**$ terraform init**

11) Format your script (indent spaces)

**$ terraform fmt**

12) Validate Your Script

**$ terraform validate**

13) Create Execution Plan For Your Script

**$ terraform plan**

14) Create Infrastructure

**$ terraform apply**

**$ terraform -auto-aprove**

**Note:** When the script got executed it will store that state in a file. If we execute script again it will not create. If you delete that state file and execute script again then it will create it.

15) Destory Infrastructure

**$ terraform destroy -auto-aprove**

**In first script we kept provider and resources info in single script file. We can keep provider and resources information in separate files**

Ex: proder.tf & main.tf

**Script to create multiple Ec2 instances**

provider "aws" {

region = "ap-south-1"

access\_key = "your-access-key"

secret\_key = "your-secret-key"

}

resource "aws\_instance" "AWSVM\_Server" {

count = "3"

ami = "ami-05c8ca4485f8b138a"

instance\_type = "t2.micro"

key\_name = "linux"

security\_groups = ["ashokit\_security\_group"]

tags = {

Name = "REDHAT-EC2-VM1"

}

}

**Note: Once it is created, then destory infrastructure using below command**

**$ terraform destroy -auto-approve**

Variables in TypeScript

We can maintain variables in seperate file

**$ vi vars.tf**

variable "ami"{

description="Amazon Machine Image value"

default = "ami-05c8ca4485f8b138a"

}

variable "instance\_type"{

description="Amazon Instance Type"

default = "t2.micro"

}

variable "instances\_count"{

description="Total No.of Instances"

default = "2"

}

Create main tf file using variables

**$ vi main.tf**

provider "aws" {

region = "ap-south-1"

access\_key = "AKIA4GJG"

secret\_key = "d04co9pmnE5xNj8XiubpUPhI"

}

resource "aws\_instance" "AWSServer" {

count="${var.instances\_count}"

ami = "${var.ami}"

instance\_type = "${var.instance\_type}"

key\_name = "linux"

security\_groups = ["launch-wizard-1"]

tags = {

Name = "EC2 VM - ${count.index}"

}

}

**Note: We can supply variables in runtime also**

Remove instances\_count variable from var.tf file and pass like below

**$ terraform apply -var instances\_count="2" -auto-aprove**

Comments in Terraform Script

# - single line comment

// - single line comment (java style)

/\* and \*/ - Multi line comments

**Dealing with Secret Key and Access Key**

We have configure secret\_key and access\_key in terraform script file. Instead of that we can configure them as environment variables.

**$ export AWS\_ACCESS\_KEY\_ID="AKIA4MGQ5UW7B"**

**$ export AWS\_SECRET\_ACCESS\_KEY="kzpje2T8S1lHsywFLTOI89XU7sRy8"**

To verify environment variables we can use echo command

**$ echo $AWS\_ACCESS\_KEY**

**$ echo $AWS\_SECRET\_ACCESS\_KEY**

Now remove credentials from terraform script and execute it.

**Note: We are setting provider credentials in terminal so these variables will be available for current session. If we want to set permanently add them in .bashrc file**

**Working with User Data**

It is used to execute script when instance launched for first time.

Create Userdata in one file

**$ vi installHttpd.sh**

#!/bin/bash

sudo su

yum install httpd -y

cd /var/www/html

echo "<html><h1>Welcome to Ashok IT...!!</h1></html>" > index.html

service httpd start

$ chmod u+x installHttpd.sh

create main scrit in main.tf file

vi main.tf

provider "aws" {

region = "ap-south-1"

access\_key = "your-access-key"

secret\_key = "your-secret-key"

}

resource "aws\_instance" "AWSServer" {

ami = "ami-05c8ca4485f8b138a"

instance\_type = "t2.micro"

key\_name = "linux"

security\_groups = ["ashokit\_security\_group"]

user\_data = "${file("installHttpd.sh")}"

tags = {

Name = "Terrafrom Server"

}

}

**Creating S3 bucket using Terraform script**

Add S3 policy for IAM user

Execute below terraform script to create s3 bucket in AWS

provider "aws"{

region = "ap-south-1"

access\_key = "your-access-key"

secret\_key = "your-secret-key"

}

resource "aws\_s3\_bucket" "s3bucketashokit"{

bucket = "s3bucketashokit"

acl="private"

versionining{

enabled = true

}

tags = {

Name = "S3 Bucket By Ashok"

}

}

**Create MySQL DB in AWS using Terraform**

Provider RDS access for IAM user

Execute below script to create MySQL DB in AWS cloud

provider "aws"{

region = "ap-south-1"

access\_key = "your-access-key"

secret\_key = "your-secret-key"

}

resource "aws\_db\_instance" "default" {

allocated\_storage = 100

engine = "mysql"

engine\_version = "5.7"

instance\_class = "db.t3.micro"

name = "mydb"

username = "foo"

password = "foobarbaz"

parameter\_group\_name = "default.mysql5.7"

skip\_final\_snapshot = true

}