**\* \* \* \* Terraform \* \* \* \***

**--------------------------------------------------------------------------------------------------------------**

**01-Terraform with AWS-29- JAN-25**

**--------------------------------------------------------------------------------------------------------------**

## Git Hub Repo : <https://github.com/ashokitschool/Terraform_Projects.git> ##

**Que -How to create infrastructure in the cloud ?**

=============================================================

* We can create cloud infrastructure in 2 ways

1) Manually

2) Terraform

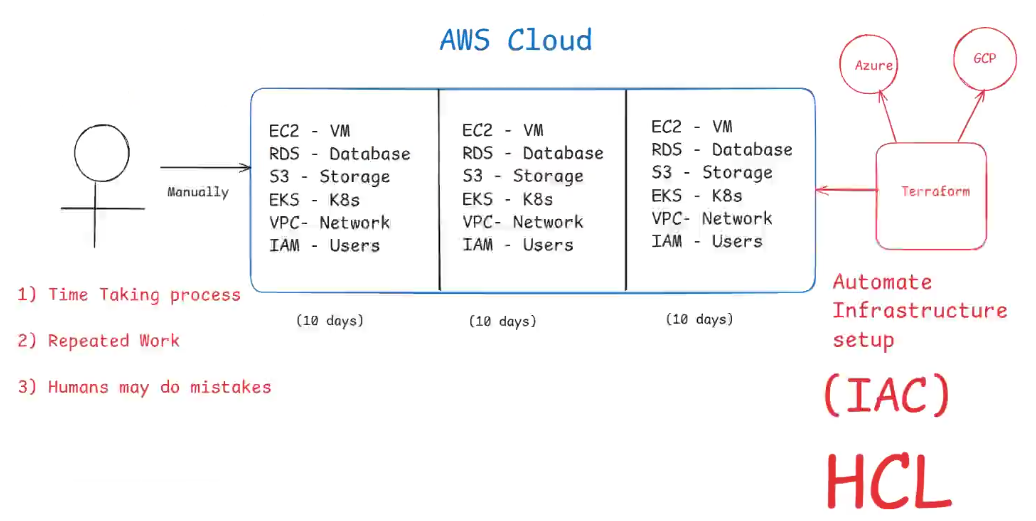
* If we create cloud infrastructure manually then we have below challenges

1) Time taking process

2) Same work again and again

3) Mistakes

* To overcome above problems now companies are using Terraform to software to setup Cloud Infrastructure.

****

**\* Terraform \***

* Developed by Hashicorp
* To create/provision infrastructure in cloud platform
* IAC software (infrastructure as code)
* Supports all most all cloud platforms
* Terraform will use HCL language to provision infrastructure

HCL : Hashicorp configuration language

* We can install terraform in mulitple Operating Systems

Ex: Windows, Linux....

**==============================**

**Terraform Vs Cloud Formation**

**==============================**

\* Cloud Formation is used to create infrastructure only in aws cloud

\* Terraform supports all cloud platforms available in the market.

**====================================**

**Terraform Installation in Windows**

**====================================**

Step-1 : Download terraform for windows & extract zip file

Note: We can see terraform.exe file

Step-2 : Set path for terraform s/w in System environment variables

Step-3 : Verify terraform setup using cmd

$ terraform -v

Step-4 : Download and install VS CODE IDE to write terraform scripts

URL : https://code.visualstudio.com/download

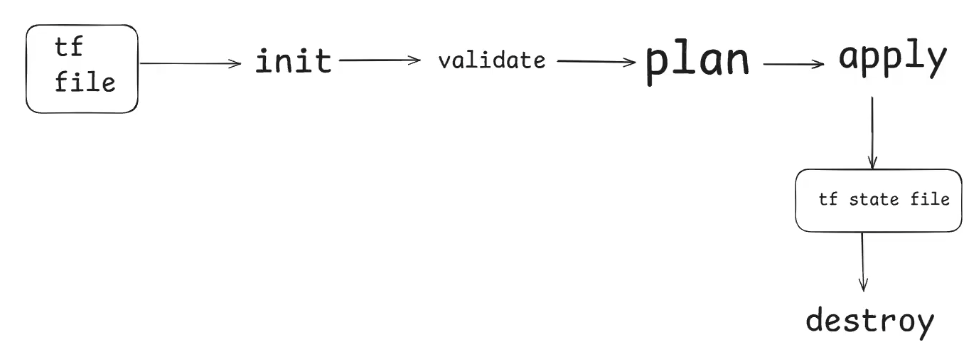
**=========================**

**Terraform Architecture**

**=========================**

\* Terraform will use HCL script to provision infrastructure in cloud platforms.

\* We need to write HCL script and save it in .tf file



\* Below are the terraform commands

**terraform init :** Initialize terraform script (.tf file) \* - **Mandatory**

* Download provider related plugins.

**terraform fmt :** Format terraform script indent spacing (optional)

**terraform validate :** Verify terraform script syntax is valid or not (optional)

**terraform plan :** Create execution plan for terraform script

**terraform apply :** Create actual resources in cloud based on plan \* - **Mandatory**

**Note: tfstate file will be created to track the resources created with our script.**

**terraform destroy :** It is used to delete the resources created with our script.

### Terraform AWS Documentation : https://registry.terraform.io/providers/hashicorp/aws/latest/docs

**==========================================**

**Terraform Script To create EC2 Instance**

**==========================================**

provider "aws" {

region = "ap-south-1"

access\_key = "AKIATCKAMNKD6R2YIMPM"

secret\_key = "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"

}

resource "aws\_instance" "ashokit\_linux\_vm" {

ami = "ami-0e53db6fd757e38c7"

instance\_type = "t2.micro"

key\_name = "awslab"

security\_groups = ["default"]

tags = {

Name = "LinuxVM"

}

}

---------------------------------------

$ terraform init

$ terraform validate

$ terraform fmt

$ terraform plan

$ terraform apply --auto-approve

$ terraform destory --auto-approve

=========================

Variables in Terraform

=========================

\* Variables are used to store data in key-value format

id = 101

name = ashok

\* We can remove hard coded values from resources script using variables

\* Variables we can maintain in seperate .tf file

Ex : input-vars.tf

variable "ami" {

description = "Amazon machine image id"

default = "ami-0e53db6fd757e38c7"

}

variable "instance\_type" {

description = "Represens EC2 instance type"

default = "t2.micro"

}

variable "key\_name" {

description = ""

default = "awslab"

}

\* We can access variables in our resources script like below

resource "aws\_instance" "ashokit\_ec2\_vm" {

ami = "${var.ami}"

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

key\_name = "${var.key\_name}"

security\_groups = ["default"]

tags = {

Name = "AIT-Linux-VM

}

}

=================================

Types of variables in terraform

=================================

1) Input Variables

2) Output Variables

\* Input variables are used to supply values to the terraform script.

Ex : ami, instance\_type, keyname, securitygrp

\* Output variables are used to get the values from terraform script after execution.

Ex-1 : After EC2 VM created, print ec2-vm public ip

Ex-2 : After S3 bucket got created, print bucket info

Ex-3 : After RDS instance got created, print DB endpoint

Ex-4 : After IAM user got created print IAM user info

-------------------provider.tf--------------------

provider "aws" {

region = "ap-south-1"

access\_key = "AKIATCKAMNKD6R2YIMPM"

secret\_key = "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"

}

-------------------input-vars.tf------------------

variable "ami" {

description = "Amazon machine image id"

default = "ami-0e53db6fd757e38c7"

}

variable "instance\_type" {

description = "Represens EC2 instance type"

default = "t2.micro"

}

variable "key\_name" {

description = ""

default = "awslab"

}

--------------------main.tf-------------------

resource "aws\_instance" "ashokit\_ec2\_vm" {

ami = var.ami

instance\_type = var.instance\_type

key\_name = var.key\_name

security\_groups = ["default"]

tags = {

Name = "AIT-Linux-VM"

}

}

---------------output-vars.tf--------------------

output "ec2\_vm\_public\_ip" {

value = aws\_instance.ashokit\_ec2\_vm.public\_ip

}

output "ec2\_private\_ip" {

value = aws\_instance.ashokit\_ec2\_vm.private\_ip

}

output "ec2\_subnet\_id"{

value = aws\_instance.ashokit\_ec2\_vm.subnet\_id

}

output "ec2\_complete\_info"{

value = aws\_instance.ashokit\_ec2\_vm

}

-------------------------------------------------

====================

Creating S3 Bucket

===================

\* S3 is storage service in AWS cloud

\* S3 provides unlimited storage

resource "aws\_s3\_bucket" "mys3b" {

bucket = var.bucket\_name

acl = "private"

versioning {

enabled = true

}

}

=======================================

What is taint and untaint in terraform

=======================================

\* Terraform "taint" is used to replace the resource when we apply the script next time.

\* For example we have created two resources like below

resource "aws\_instance" "vm1"{

// configuration

}

resource "aws\_s3\_bucket" "abt1"{

// configuration

}

\* After sometime we realized that ec2 vm got damaged...

Note : we can taint that ec2 vm using below command to replace when we apply the script next time

$ terraform taint aws\_instance.vm1

$ terraform apply --auto-approve

Note: The alternate for "taint" is "replace"

$ terraform apply -replace=aws\_instance.vm1 --auto-approve

============

Assignment

===========

1) Create Custom VPC

2) Create Ec2 Instance using Custom VPC

===================

Terraform Modules

===================

\* A Terraform module is a set of terraform configuration files available in a single directory.

\* One module can contain one or more .tf files

01-Project

- provider.tf

- main.tf

- input-vars.tf

- output-vars.tf

\* One module can have any no.of child modules in terraform

irctc-app

- provider.tf

- main.tf

- outputs.tf

- ec2

- main.tf

- inputs.tf

- outputs.tf

- s3

- main.tf

- inputs.tf

- outputs.tf

- rds

- main.tf

- inputs.tf

- outputs.tf

Note : Using terraform modules we can achieve re-usability

Note: We will run terraform commands from root module and root module will invoke child modules for execution.

======================================

Terraform project setup with Modules

======================================

### Step-1 : Create Project directory

Ex: 05-TF-Modules

### Step-2 : Create "modules" directory inside project directory

Ex: 05-TF-Modules

- modules

### Step-3 : Create "ec2" & "s3" directories inside "modules" directory

Ex: 05-TF-Modules

- modules

- ec2

- s3

### Step-4 : Create terraform scripts inside "ec2" directory

inputs.tf

main.tf

outputs.tf

### Step-4 : Create terraform scripts inside "S3" directory

inputs.tf

main.tf

outputs.tf

### Step-6 : create "provider.tf" file in root module

### Step-7 : create "main.tf" file in root module and invoke child modules from root module.

module "my\_ec2"{

source = "./modules/ec2"

}

module "my\_s3" {

source = "./modules/s3"

}

### Step-8: Create "ouputs.tf" in project root module and access child modules related outputs.

output "ec2\_public\_ip" {

value = module.my\_ec2.a\_public\_ip

}

output "ec2\_private\_ip" {

value = module.my\_ec2.b\_private\_ip

}

output "s3\_bucket" {

value = module.my\_s3.c\_s3\_info

}

================================

Environments of the project

===============================

\* Env means the platform that is required to run our application

Ex: Servers, Database, Storage, Network....

\* One project contains multiple envs

Ex: DEV, SIT or QA, UAT, PILOT, PROD or LIVE

Dev Env : Developers will use it for code integration testing

SIT / QA Env : Testers will use it for System Integration Testing

UAT Env: Client will use it for Acceptance testing.

Pilot Env : Pre-Prod testing and Performance testing.

Prod Env : Live Environment.

Note: In real-time from environment to environment infrastructure resources configuration might be different

DEV Env : t2.medium

PROD Env : t2.xlarge

\* In order to achieve this requirement we will maintain environment specific input variable file like below

inputs-dev.tf \* Input variables file for DEV env

inputs-sit.tf \* input variables file for SIT env

inputs-uat.tf \* input variables file for UAT env

inputs-pilot.tf \* input variables file for PILOT env

inputs-prod.tf \* input variables file for PROD env

\* When we are executing terraform apply command we can pass inputs variable file like below.

# create infrastructure for DEV Env

$ terraform apply --var-file=inputs-dev.tf

# create infrastructure for PROD Env

$ terraform apply --var-file=inputs-prod.tf

Note: With this approach we can achieve loosely coupling and we can achieve script re-usability.

=========================

Workspace in terraform

=========================

\* To manage infrastructure for multiple environments we will use Terraform workspace concept.

\* When we use workspace, it will maintain seperate state file for every environment/workspace.

Note: We can execute same script for multiple environments.

$ terraform workspace show

$ terraform workspace new <workspace-name>

$ terraform workspace list

$ terraform workspace select <workspace-name>

==================================

Working with terraform workspaces

==================================

Step-1: Create Terraform Project

Step-2: Create "provider.tf" file and configure provider details

Step-3: Create input variables files based on environments and configure variable values.

Ex :

dev.tfvars

sit.tfvars

prod.tfvars

Step-4 : Create main resources script file

Step-6 : Create outputs variable file

Step-7 : Execute "terraform" init command

Step-7 : Create Workspaces

$ terraform workspace new dev

$ terraform workspace new qa

Step-8 : Select workspace

$ terraform workspace select dev

Step-9 : Run script and check state files

$ terraform apply --var-file=dev.tfvars

Note: When we use workspaces concept, it will maintain seperate state file for every environment.

Step-10 : switch to sit workspace and run the script

$ terraform workspace select sit

$ terraform apply --var-file=sit.tfvars

=========================

What is Terraform Vault

=========================

\* Provided by Hashicorp org.

\* It is used to manage secrets such as passwords, tokens, any sensitive information.

Ex-1: Create RDS instance using terraform

provider "aws" {

region = "ap-south-1"

}

resource "aws\_db\_instance" "my\_rds" {

identifier = "my-rds-instance"

allocated\_storage = 20

storage\_type = "gp2"

engine = "mysql"

engine\_version = "8.0"

instance\_class = "db.t3.micro"

username = "admin"

password = "admin@123"

publicly\_accessible = false

skip\_final\_snapshot = true

tags = {

Name = "MyRDSInstance"

}

}

Note: If we terraform script like above anyone can read our DB username and password which is not recommended.

\* To hide this sensitive data from terraform scripts we will use Terraform Vault server. Sensitive data we will store in terraform vault server.

\* Our terraform script file will get sensitive data from vault server like below

resource "aws\_db\_instance" "my\_rds" {

identifier = "my-rds-instance"

allocated\_storage = 20

storage\_type = "gp2"

engine = "mysql"

engine\_version = "8.0"

instance\_class = "db.t3.micro"

username = "vaultserver[db.uname]"

password = "vaultserver[db.pwd]"

publicly\_accessible = false

skip\_final\_snapshot = true

tags = {

Name = "MyRDSInstance"

}

}

===================

Vault Server Setup

===================

Documentation : https://developer.hashicorp.com/vault/tutorials/getting-started/getting-started-install

===================

Vault Server Setup

===================

Documentation : https://developer.hashicorp.com/vault/tutorials/getting-started/getting-started-install

@@ Step-1 :: Create EC2 Instance (Ubuntu AMI) and connect with that

@@ Step-2 :: Install Vault on the EC2 instance

# Install gpg

sudo apt update && sudo apt install gpg

# Download the signing key to a new keyring

wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

# Verify the key's fingerprint

gpg --no-default-keyring --keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg --fingerprint

# add Hashicorp repo to pkg manager

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

# update packages

sudo apt update

# install vault

sudo apt install vault

@@ Step-3 :: Start the vault server (It runs on port number 8200)

$ vault server -dev -dev-listen-address="0.0.0.0:8200"

@@ Step-4 :: Access Vault server UI dashboard (enable 8200 port in inbound rules)

@@ Step-5 : Enable Secret Engine (KV) & create secret and store the data

@@ Step-6 : Connect with Vault server from different ssh terminal

@@ Step-7 : Export Vault addr & Enable approle

$ export VAULT\_ADDR='http://0.0.0.0:8200'

$ vault auth enable approle

@@ Step-8 : Create Policy

---------------------------------------------

vault policy write terraform - <<EOF

path "\*" {

capabilities = ["list", "read"]

}

path "secrets/data/\*" {

capabilities = ["create", "read", "update", "delete", "list"]

}

path "kv/data/\*" {

capabilities = ["create", "read", "update", "delete", "list"]

}

path "secret/data/\*" {

capabilities = ["create", "read", "update", "delete", "list"]

}

path "auth/token/create" {

capabilities = ["create", "read", "update", "list"]

}

EOF

------------------------------------------

@@ Step-9 : Create Role

-----------------------------------------

vault write auth/approle/role/terraform \

secret\_id\_ttl=10m \

token\_num\_uses=10 \

token\_ttl=20m \

token\_max\_ttl=30m \

secret\_id\_num\_uses=40 \

token\_policies=terraform

----------------------------------------

@@ Step-10 : Generate Role\_ID and Secret\_ID and copy them

$ vault read auth/approle/role/terraform/role-id

$ vault write -f auth/approle/role/terraform/secret-id

@@ Step-11 : Write terraform script and read data from terraform vault server

------------------------------------------------------

provider "aws" {

region = "ap-south-1"

access\_key = ""

secret\_key = ""

}

provider "vault" {

address = "http://public-ip:8200"

skip\_child\_token = true

auth\_login {

path = "auth/approle/login"

parameters = {

role\_id = "<>"

secret\_id = "<>"

}

}

}

data "vault\_kv\_secret\_v2" "example" {

mount = "kv"

name = "test-secret"

}

resource "aws\_instance" "ashokit\_vm" {

ami = "ami-08718895af4dfa033"

instance\_type = "t2.micro"

tags = {

Secret = data.vault\_kv\_secret\_v2.example.data["tagname"]

}

}

--------------------------------------------------------------------------------------

Assignment : Create AWS RDS instance by reading db\_username and db\_pwd from Vault Server.

--------------------------------------------------------------------------------------

===========================

1) Infrastructure as code (IAC)

2) Terraform Introduction

3) Terraform Setup (windows)

4) Terraform Architecture

5) Terraform Scripts using HCL

6) Variables (input & output)

7) Resource Taint & Untaint

8) Terraform Modules

9) Project Environments

10) Terraform Workspaces

11) Terraform lock file & state file