```
==========
Terraform
==========

=> Developed by Hashicorp company

=> To create/provision infrastructure in cloud platform

=> IAC software (infrastructure as code)

=> Terraform will use HCL language to provision infrastructure

                HCL : Hashicorp configuration language

=> Supports almost all cloud platforms (Ex: AWS, Azure, GCP)

=> 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

        ## URL to download : https://developer.hashicorp.com/terraform/install

        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 Installation in Amazon linux
=======================================

sudo yum install -y yum-utils shadow-utils
sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
sudo yum -y install terraform
terraform -v

=======================================
Terraform Installation in Ubuntu
=======================================

wget -O - https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o
```

```
/usr/share/keyrings/hashicorp-archive-keyring.gpg
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
sudo apt update && sudo apt install terraform
terraform -v

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


terraform init : Initialize terraform script (.tf file) and download provider plugins

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

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

Step-1 : Create IAM user with Adminstrator permission

Step-2 : Login as IAM user and generae access keys

Step-3 : Create below script file in VS Code IDE


provider "aws" {
    region = "ap-south-1"
    access_key = ""
    secret_key = ""
}

resource "aws_instance" "ashokitvm" {
  ami           = "ami-05c179eced2eb9b5b"
  instance_type = "t2.micro"
  key_name = "ashokit"
  security_groups = ["default"]

  tags = {
    Name = "HelloWorld"
  }
}

Step-4 : Execute terraform commands

$ terraform init
$ terraform validate
$ terraform fmt
$ terraform plan
$ terraform apply
```

```
$ terraform destroy

==========================
What is Lock File
==========================

=> The lock file is used to lock the provider versions in your Terraform configuration.

=> It records the exact versions of the providers that Terraform is using for the current
configuration.

=> This helps to avoid issues with provider version mismatches across different runs or different
team members.

==========================
What is State File
==========================

=> The state file contains the current state of the infrastructure that Terraform manages. It keeps
track of resources, their attributes, and their dependencies.

=> It is a JSON file that stores all the information Terraform needs to know about the infrastructure
it has created or modified. This includes resource IDs, configurations, and relationships between
resources.


========================
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-05c179eced2eb9b5b"
}

variable "instance_type" {
    description = "Represens EC2 instance type"
    default = "t2.micro"
}

variable "key_name" {
    description = ""
    default = "ashokit"
}

=> We can access variables in our resources script like below

resource "aws_instance" "abc" {
  ami             = var.ami
  instance_type   = var.instance_type
  key_name        = var.key_name
  security_groups = ["default"]
```

```
  tags = {
    Name = "VM-1"
  }
}

================================
Types of variables in terraform
================================

=> We have 2 type of variables in terraform

    1) Input Variables

    2) Output Variables

=> Input variables are used to supply input values to the terraform script.

      Ex : ami, instance_type, keyname, securitygrp

=> Output variables are used to get the values from terraform after script 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 = ""
  secret_key = ""
}
----------------- input-vars.tf----------------------
variable "ami" {
  description = "Amazon machine image id"
  default = "ami-05c179eced2eb9b5b"
}

variable "instance_type" {
    description = "Represens EC2 instance type"
    default = "t2.micro"
}

variable "key_name" {
    description = ""
    default = "ashokit"
}

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

resource "aws_instance" "abc" {
  ami             = var.ami
  instance_type   = var.instance_type
  key_name        = var.key_name
  security_groups = ["default"]

  tags = {
    Name = "VM-1"
  }
}
```

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

output "ec2_vm_public_ip"{
    value = aws_instance.abc.public_ip
}

output "ec2_vm_private_ip"{
    value = aws_instance.abc.private_ip
}

output "ec2_vm_info"{
    value = aws_instance.abc
}

-----------------------------------------------------------
Assignment-1 : Create 3 EC2 Instances with Diff Tag names

Assignment-2 : Create S3 bucket using Terraform

Assignment-3 : Create RDS instance using Terraform
-----------------------------------------------------------


=======================================
What is taint and untaint in terraform
=======================================

=> For example we have created two resources like below using terraform


resource "aws_instance" "vm1"{
    // configuration
}

resource "aws_s3_bucket" "bkt1"{
    // configuration
}

=> After sometime we realized that ec2 vm got damaged. we want to replace existing ec2 vm with new
ec2 vm (we don't want make any changes to s3 bucket).

Note: In this scenario we can use taint concept.

=> Terraform "taint" is used to replace the resource 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"


# delete particular resource
terraform destroy -target=aws_instance.vm1




==================
Terraform Modules
==================
```

```
=> Any directory/folder which contains set of terraform configuration files is called as one module.

=> One module contains one or more .tf files like below

        01-EC2
          - provider.tf
          - inputs.tf
          - outputs.tf
          - main.tf


=> One module can have any no.of child modules in terraform

        sbi-infra-app

            - ec2
                - inputs.tf
                - main.tf
                - outputs.tf

            - rds
                - inputs.tf
                - outputs.tf
                - main.tf

            - s3
                - inputs.tf
                - ouputs.tf
                - main.tf

Note : Using terraform modules we can achieve re-usability


=====================================
Terraform project setup with Modules
=====================================

### Step-1 : Create Project directory (root module)

    Ex : SBI-Infra-App

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

    Ex: SBI-Infra-App
            - modules

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

    Ex : SBI-Infra-App

            - modules

                - ec2

                - s3

### Step-4 : Create terraform scripts inside "ec2" directory to create ec2-instance.

            inputs.tf
            main.tf
            outputs.tf

### Step-5 : Create terraform scripts inside "s3" directory to create s3 bucket
```

```
            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 "ait_vm_public_ip"{
    value = module.my_ec2.a1
}

output "ait_vm_private_ip" {
    value = module.my_ec2.a2
}

Note: here a1 and a2 are ouput variables declared in ec2 module ouputs.tf file

=============================================================================================
Assignment : Create custom VPC and create EC2 VM in that custom VPC using Terraform Script
=============================================================================================

================================
Environments of the project
================================

=> Env means the platform that is required to run our application

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

=> One Project contains multiple environments

        1) DEV Env
        2) SIT Env
        3) UAT Env
        4) Pilot Env
        5) Prod Env


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.micro
```

```
        SIT Env ==> t2.medium

        UAT Env ==> t2.large

        PROD Env ==> t2.xlarge
```

Note: To create EC2 instance for multiple environments then we have to change value of variable which is not recommended option.

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

inputs-dev.tfvars : input variables for dev env

inputs-sit.tfvars : input variables for SIT env

inputs-uat.tfvars : input variables for UAT env

inputs-pilot.tfvars : input variables for PILOT env

inputs-prod.tfvars : input variables 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.tfvars

# create infrastructure for PROD env
$ terraform apply --var-file=inputs-prod.tfvars
```

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

```
==================================
Working with terraform workspaces
==================================
```

=> To manage infrastructure for multiple environments we will use Terraform workspace concept.

=> When we use workspaces, it will maintain seperate state file for every environment/workspace.

Note: We can execute same script for multiple environments without effecting other env infrastructure resources.

```
# display current workspace name
$ terraform workspace show

# create new workspace 'dev'
$ terraform workspace new dev

# create new workspace 'sit'
$ terraform workspace new sit

# display workspaces available
$ terraform workspace list

# switch to particualar workspace
$ terraform workspace select dev
```

```
==================================
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
        qa.tfvars
        uat.tfvars
        prod.tfvars

Step-4 : Create main resources script file

Step-6 : Create outputs variable file

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-9 : switch to qa workspace and run the script

        $ terraform workspace select qa

        $ terraform plan --var-file=qa.tfvars

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


=========================
What is Terraform Vault
=========================

@@ Terraform Vault setup : https://youtu.be/OOBbBRdQt2I

=> Provided by Hashicorp org.

=> It is used to manage secrets such as passwords, tokens, any sensitive information

Ex:
----

1) While creating RDS instance we need to specify username and pwd

2) While creating iAM user we need to specify username and pwd

Note: IT is not recommended to configure those credentials in terraform script directley.

=> Vault allows you to store and manage secrets securely, reducing the risk of exposing sensitive
data in your Terraform configurations.
```

```
==================
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 a code (IAAC)

2) Terraform Introduction

3) Terraform Setup (Windows)

4) Terraform Architecture

5) Terraform Commands

6) Terraform Scripts

7) Variables (input & output)

8) Terraform Modules

9) Project Environments & Env specific inputs

10) Terraform Workspaces

11) Resource Tainting or Replace

12) Lock File Vs State File