Siddhant Sathe **D15A/51**

Aim :To understand terraform lifecycle, core concepts/terminologies and install it on a Linux Machine and Windows.

Theory :

Terraform is an open-source Infrastructure as Code (IaC) tool developed by HashiCorp. It allows users to define and provision infrastructure using a high-level configuration language known as HashiCorp Configuration Language (HCL) or JSON. Terraform supports a wide range of cloud providers, such as AWS, Azure, Google Cloud, and
on-premises solutions, enabling users to manage infrastructure across multiple environments consistently.

Core Concepts and Terminologies

1.Providers:
Providers are plugins that allow Terraform to interact with various APIs of cloud providers, SaaS providers, and other services. Each provider requires configuration and manages resources for that specific service.

2.Resources:
Resources are the most fundamental elements in Terraform. They represent components of your infrastructure, such as virtual machines, databases, networks, and more.

3.Modules:
Modules are containers for multiple resources that are used together. A module can call other modules, creating a hierarchical structure. This makes it easier to organize and reuse code.

4.State:
Terraform maintains a state file that keeps track of the infrastructure managed by Terraform. The state file is crucial as it provides a mapping between the real-world resources and the configuration defined in Terraform.

5.Variables:
Variables in Terraform are used to make configurations dynamic and reusable. They can be defined in the configuration files and assigned values at runtime.

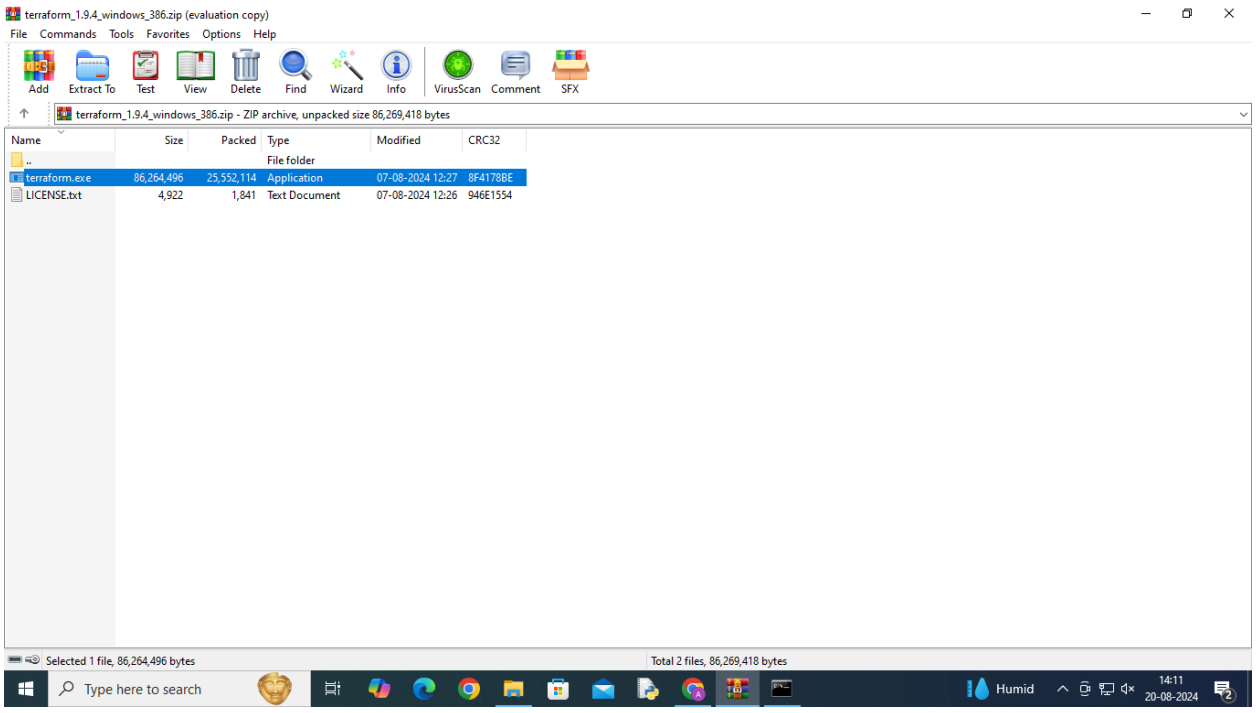
6.Outputs:
Outputs are used to extract information from the Terraform-managed infrastructure and display it after the execution of a Terraform plan or apply.

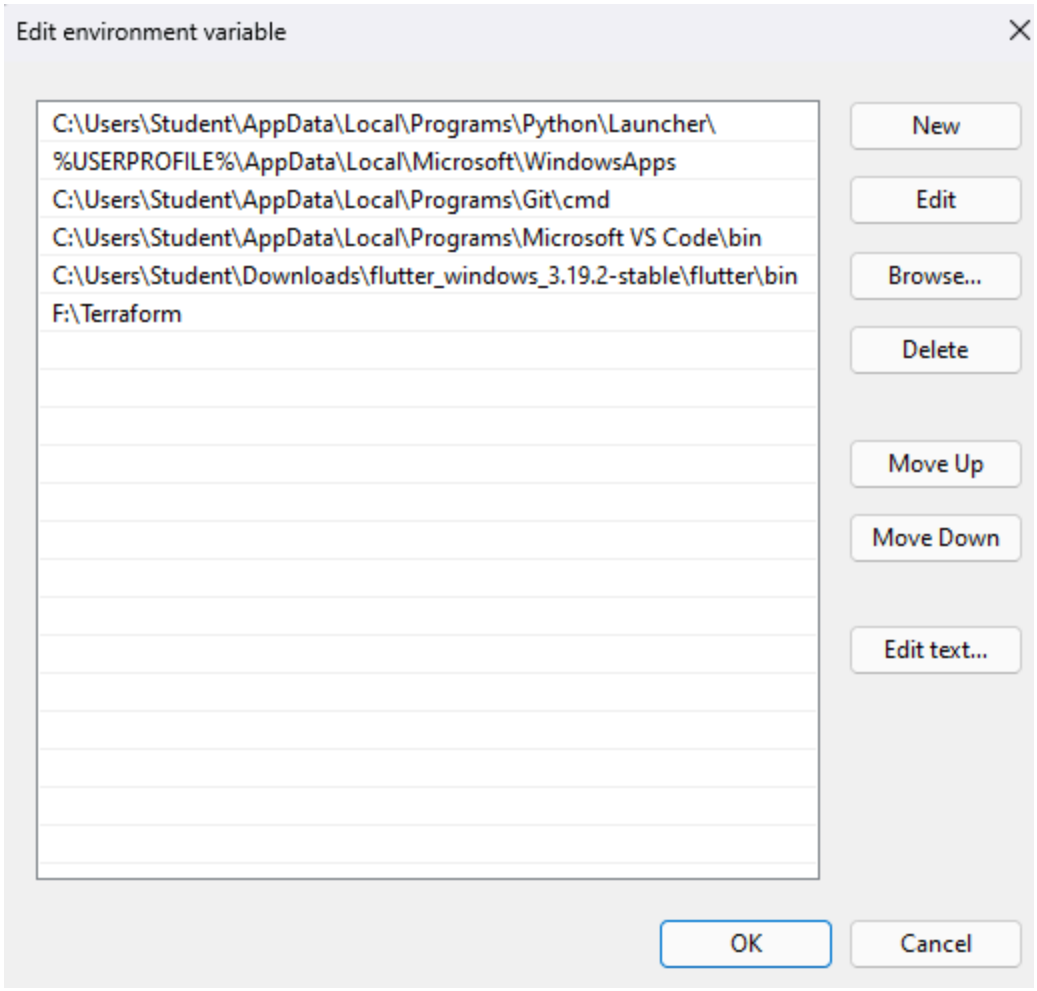
Terraform Lifecycle

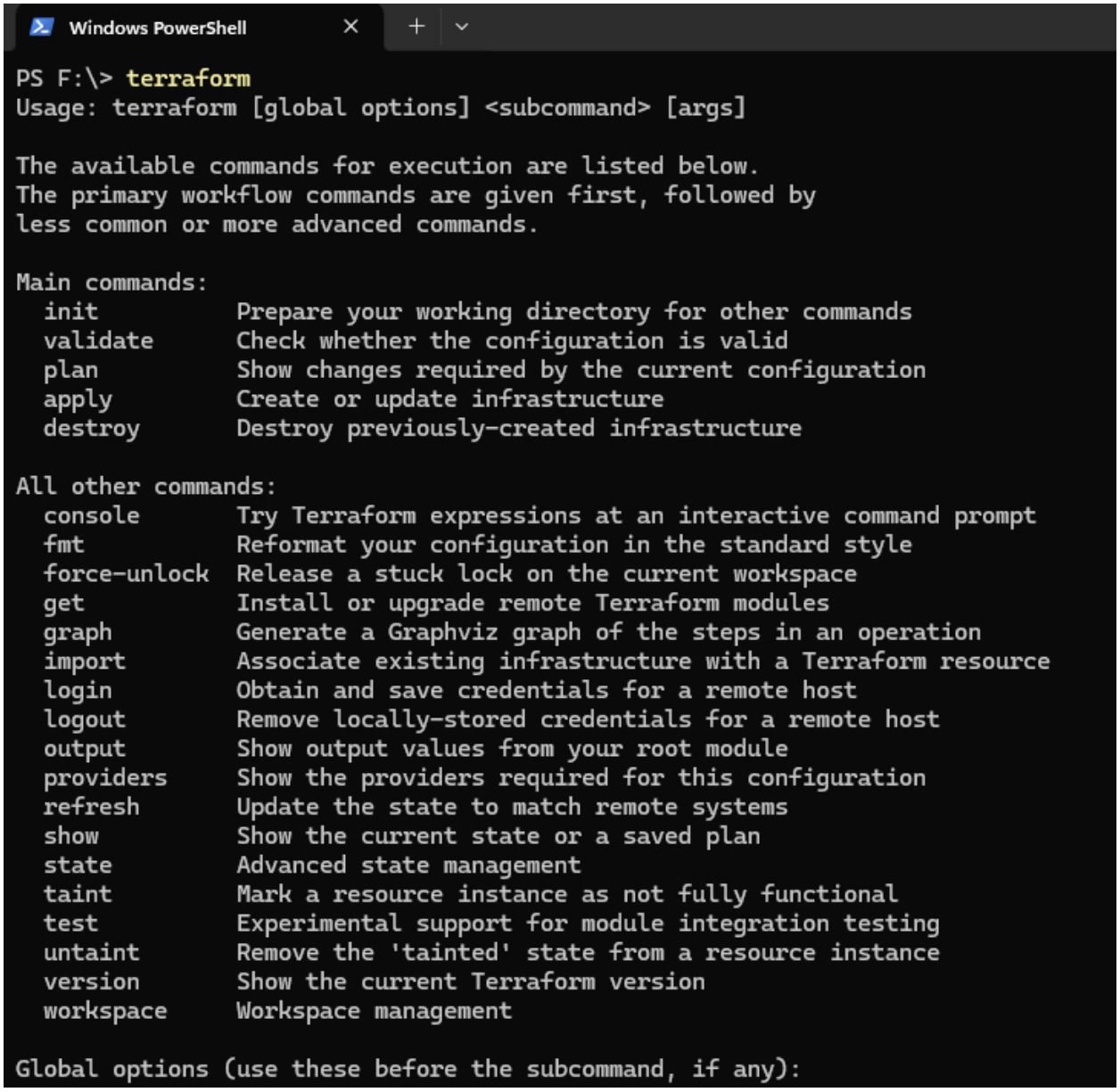
1.Write:
Write the configuration file (typically with .tf extension) using HCL to describe the desired infrastructure.

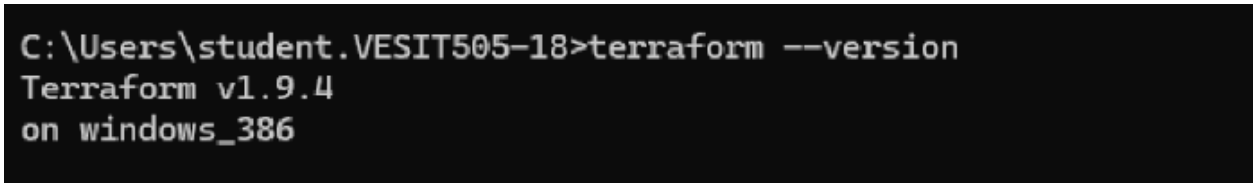
2.Initialize (terraform init):
Initialize the working directory containing the configuration files. This command downloads the necessary provider plugins and sets up the environment.

3.Plan (terraform plan):
Terraform creates an execution plan based on the configuration files. It compares the current state with the desired state and shows the changes that will be made.
4.Apply (terraform apply):
Apply the changes required to reach the desired state of the configuration. Terraform will prompt for confirmation before making any changes.
5.Destroy (terraform destroy):
Destroy the infrastructure managed by Terraform. This command is used to remove all resources defined in the configuration files.




Edit environment variable
C:\Users\Student\AppData\Local\Programs\Python\Launcher\
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\Student\AppData\Local\Programs\Git\cmd
C:\Users\Student\AppData\Local\Programs\Microsoft VS Code\bin
C:\Users\Student\Downloads\flutter_windows_3.19.2-stable\flutter\bin
F:\Terraform
New
Edit
Browse...
Delete
Move Up
Move Down
Edit text...
OK
Cancel


Windows PowerShell
PS F:\> terraform
Usage: terraform [global options] <subcommand> [args]
The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.
Main commands:
init Prepare your working directory for other commands
validate Check whether the configuration is valid
plan Show changes required by the current configuration
apply Create or update infrastructure
destroy Destroy previously-created infrastructure
All other commands:
console Try Terraform expressions at an interactive command prompt
fmt Reformat your configuration in the standard style
force-unlock Release a stuck lock on the current workspace
get Install or upgrade remote Terraform modules
graph Generate a Graphviz graph of the steps in an operation
import Associate existing infrastructure with a Terraform resource
login Obtain and save credentials for a remote host
logout Remove locally-stored credentials for a remote host
output Show output values from your root module
providers Show the providers required for this configuration
refresh Update the state to match remote systems
show Show the current state or a saved plan
state Advanced state management
taint Mark a resource instance as not fully functional
test Experimental support for module integration testing
untaint Remove the 'tainted' state from a resource instance
version Show the current Terraform version
workspace Workspace management
Global options (use these before the subcommand, if any):


C:\Users\student.VESIT505-18>terraform --version
Terraform v1.9.4
on windows_386