St. Francis Institute of Technology, Mumbai-400 103

Department of Information Technology

A.Y. 2024-2025

Class: TE-ITA/B, Semester: V

Experiment – 6: To understand Terraform lifecycle, basic concepts / terminologies, and install it on a Windows/Linux machine and build, apply, and destroy AWS using Terraform.

Subject: **Advanced DevOps Lab**

1. **Aim:** To understand Terraform lifecycle, basic concepts/terminologies and install it on Windows /Linux machine and thereafter to build, apply and destroy AWS using Terraform.
2. **Objectives:** After study of this experiment, the student will be able to
   - Understand basic Terraform concepts
   - Perform installation of Terraform.
   - Write terraform scripts
   - Understand basic Terraform commands and concept of creating instance on EC2 using terraform.
3. **Lab objective mapped :** ITL504.3: To be familiarized with infrastructure as code for provisioning, compliance, and management of any cloud infrastructure and d service.
4. **Prerequisite:** Fundamentals of cloud computing and AWS account
5. **Requirements:** PC and Internet
6. **Pre-Experiment Exercise:**

Brief Theory:

Terraform

Terraform is an infrastructure as code (IaC) tool that allows you to build, change, and version infrastructure safely and efficiently. This includes low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc. Terraform can manage both existing service providers and custom in-house solutions.

Key Features

Infrastructure as Code:

You describe your infrastructure using Terraform's high-level configuration language in human- readable, declarative configuration files. This allows you to create a blueprint that you can version, share, and reuse.

Resource Graph

Terraform builds a resource graph and creates or modifies non-dependent resources in parallel. This allows Terraform to build resources as efficiently as possible and gives you greater insight into your infrastructure.

Change Automation

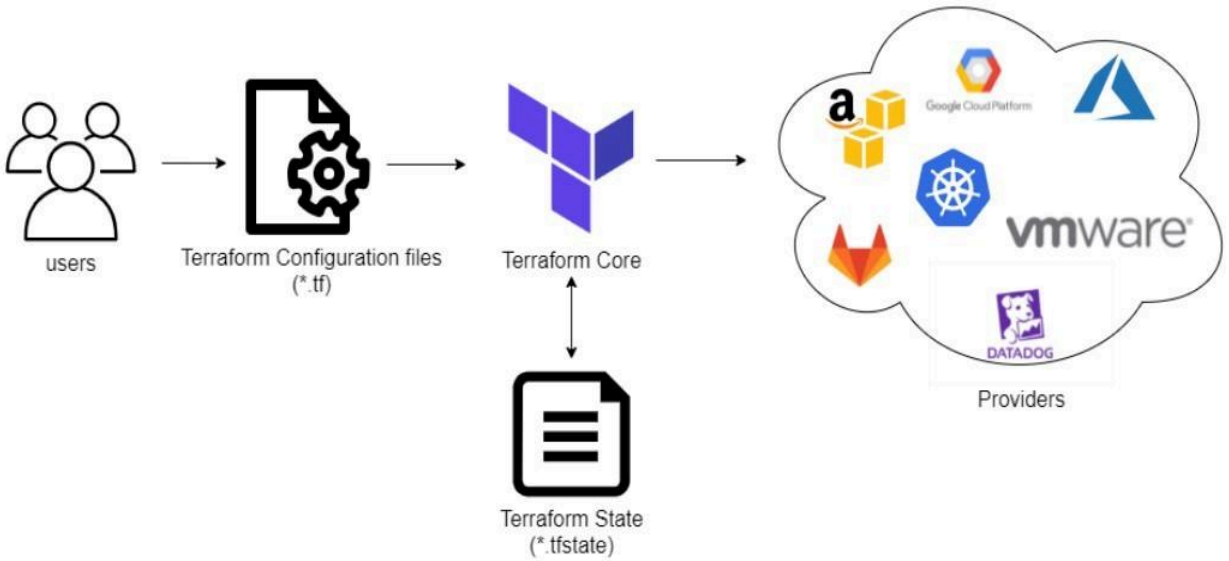
Terraform can apply complex change sets to your infrastructure with minimal human interaction. When you update configuration files, Terraform determines what changed and creates incremental execution plans that respect dependencies.

Terraform Life Cycle:

Terraform actually works, there's sort of two major components:

one is the **terraform core:** it takes the terraform configuration which is being provided by the user and then takes the terraform state which is managed by terraform itself. As such, this gets fed into the core that is responsible for figuring out what is that graph of our different resources for exemple how these different pieces relate to each other or what needs to be created/updated/destroyed, it does all the essential lifecycle management.

On the backside, terraform supports many different **providers**, such as: cloud providers (AWS,GCP,AZURE) and they also could be on-premise infrastructure (VMware, OpenStack.) But this support is not restricted or limited only to Infrastructure As A Service , terraform can also manage higher level like Platform As A Service(Kubernetes, Lambdas..)or even Software As A Service (DataDog, GitHub..)



All of these are important pieces of the infrastructure, they are all part of the logical end-to-end delivery. Terraform has over a hundred providers for different technologies, and each provider gives terraform users access to their resources. It also gives you the ability to create infrastructure at different levels.

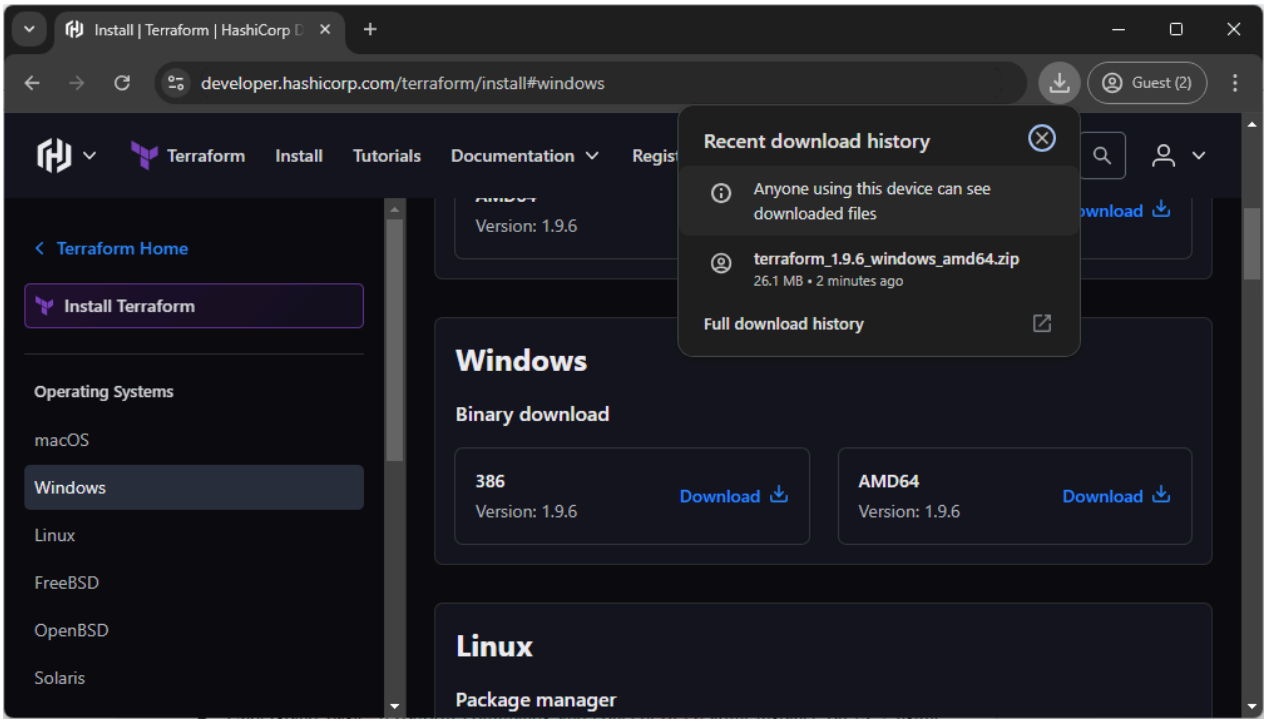
Trraform Core Concepts:

Below are the core concepts/terminologies used in Terraform:

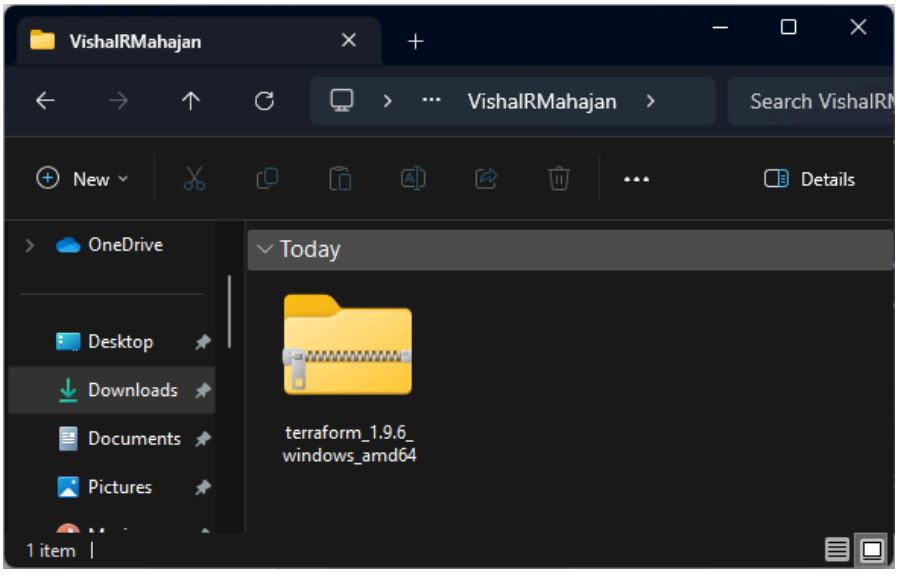
- **Variables:** Also used as input-variables, it is a key-value pair used by Terraform modules to allow customization.
- **Provider:** It is a plugin to interact with APIs of service and access its related resources.
- **Module:** It is a folder with Terraform templates where all the configurations are defined
- **State:** It consists of cached information about the infrastructure managed by Terraform and its related configurations.
- **Resources:** It refers to a block of one or more infrastructure objects (compute instances, virtual networks, etc.), which are used in configuring and managing the infrastructure.
- **Data Source:** It is implemented by providers to return information on external objects to terraform.
- **Output Values:** These are return values of a terraform module that can be used by other configurations.
- **Plan:** It is one of the stages where it determines what needs to be created, updated, or destroyed to move from the real/current state of the infrastructure to the desired state.
- **Apply:** It is one of the stages where it applies the changes in the real/current state of the infrastructure in order to move to the desired state.

7. Laboratory Exercise

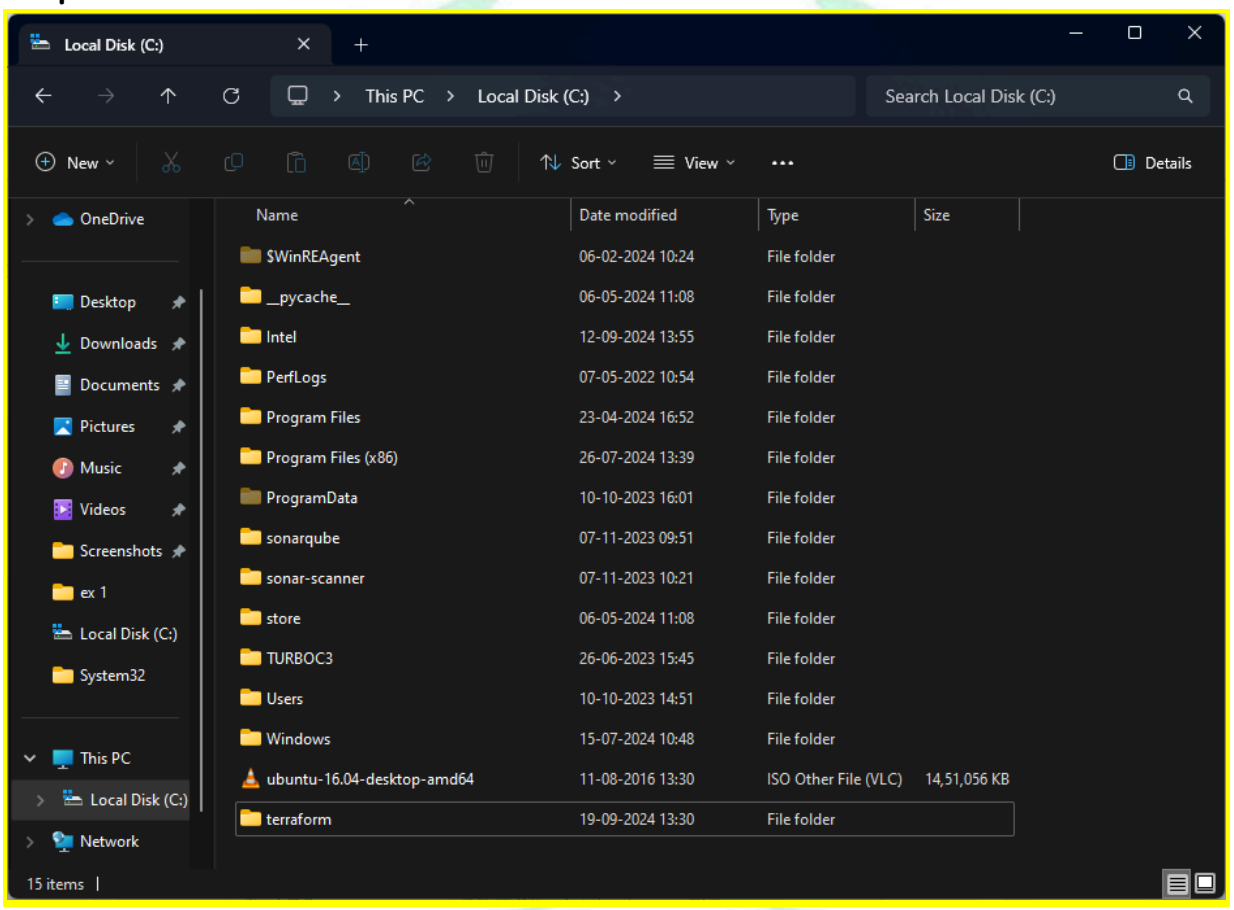
Step 1 : Download appropriate terraform package(.zip) from terraform.io/downloads.html for Windows



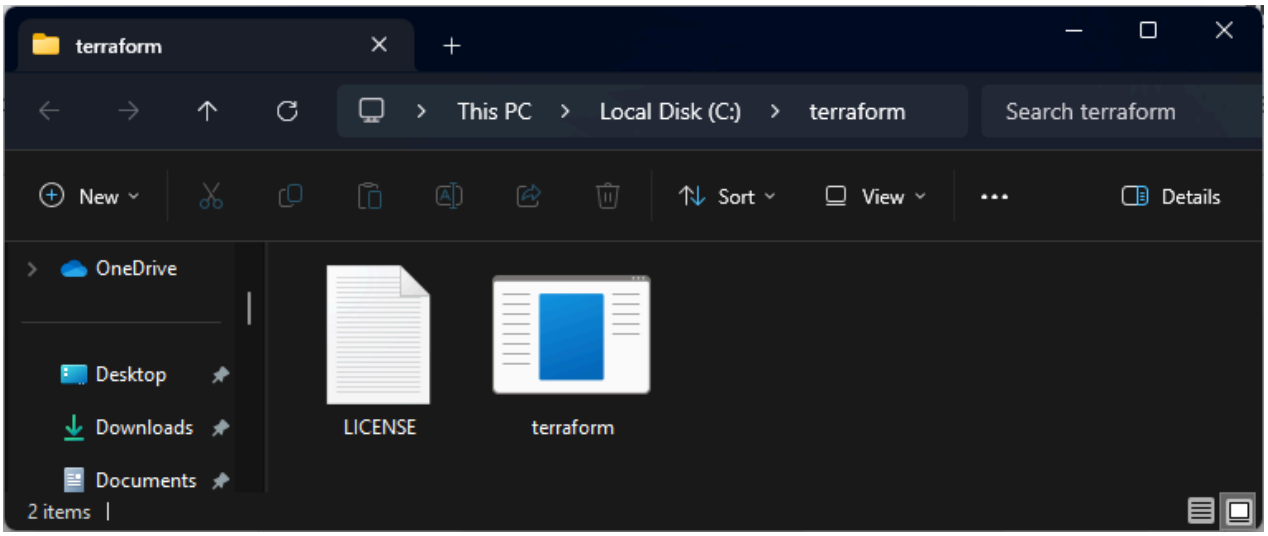
Step 2: Download Terraform for Windows 64-bit / (32-bit).



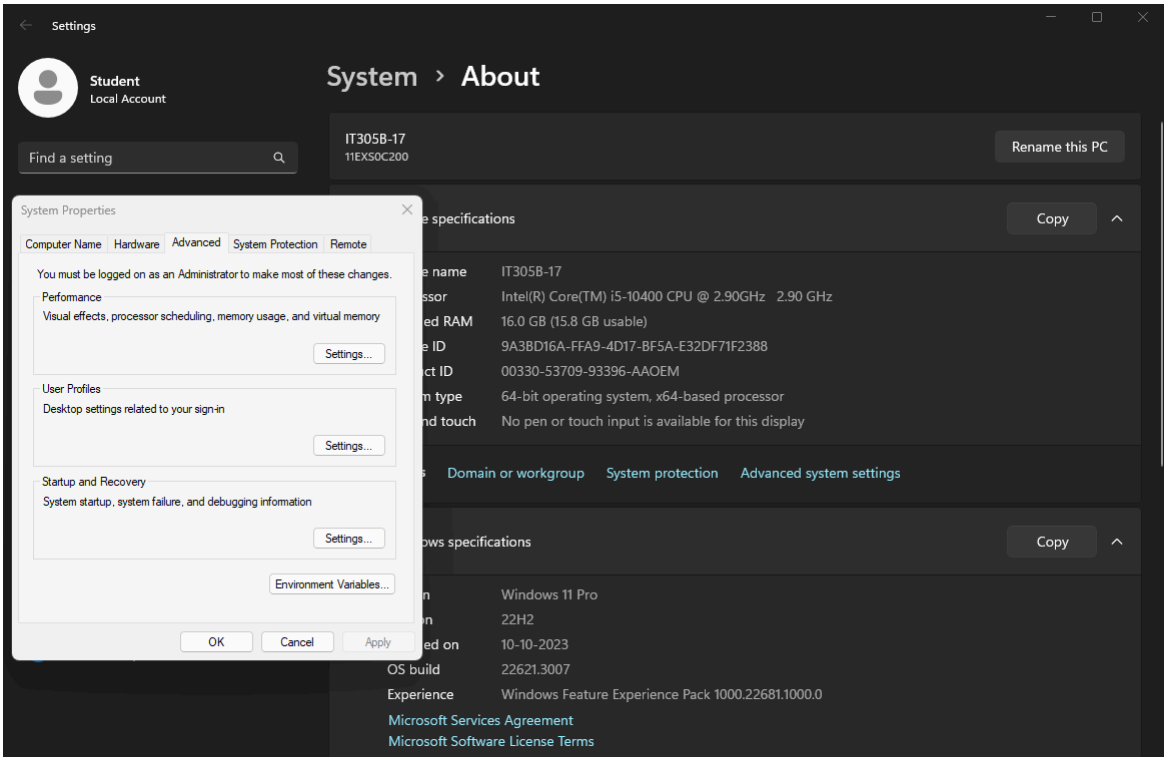
Step 3: Create a folder 'terraform' in drive C .



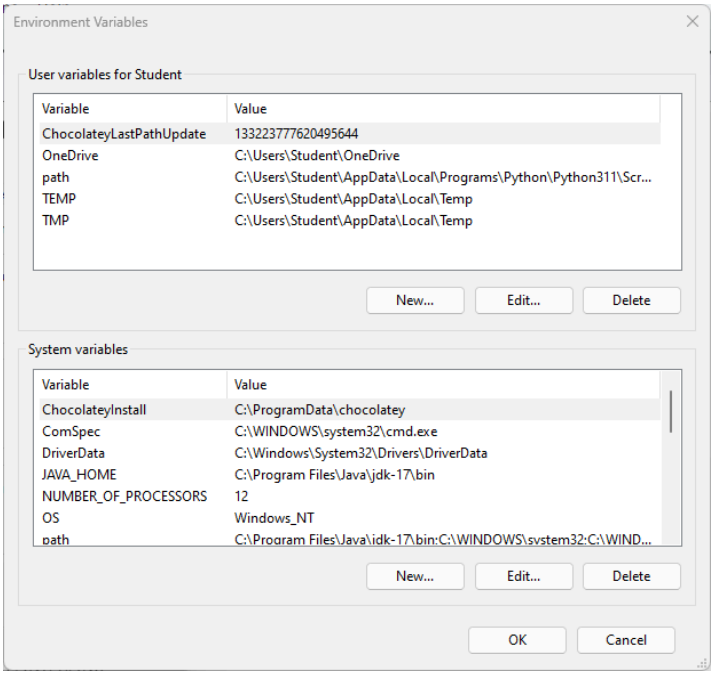
Step 4 : Extract downloaded zip in to this c:/terraform folder



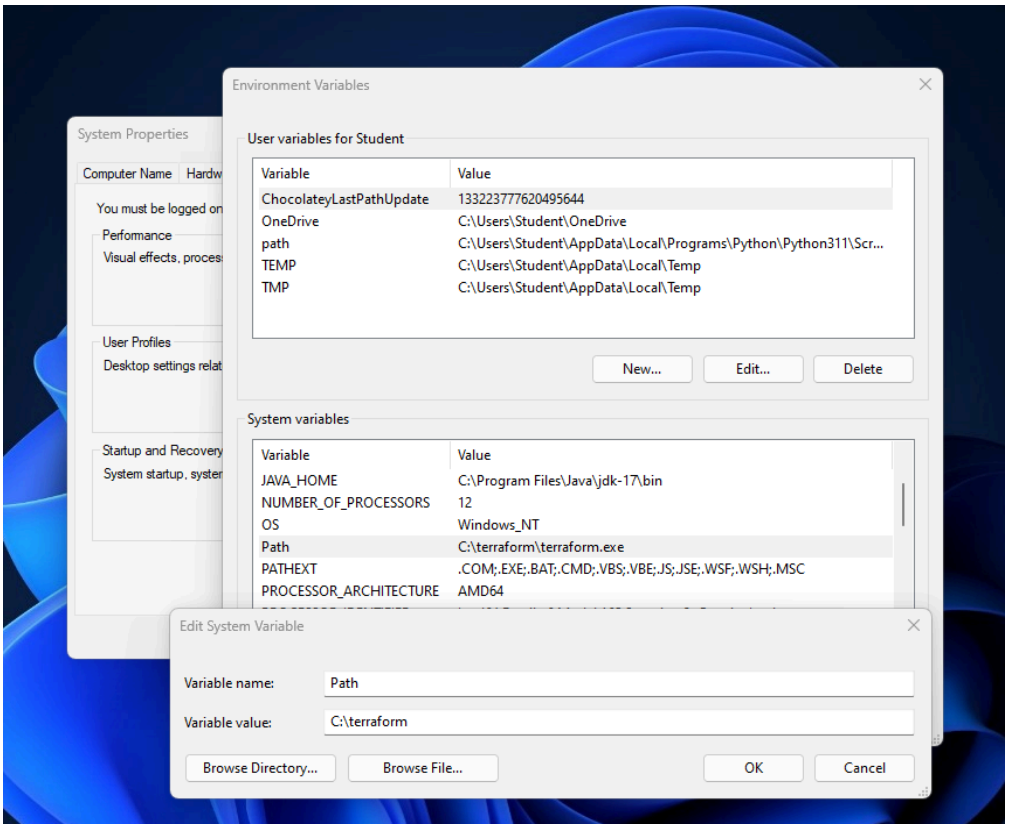
Step 5: Now we need to set a path for terraform. Go to My computer/ This PC, right click, select properties, go to advanced system settings.



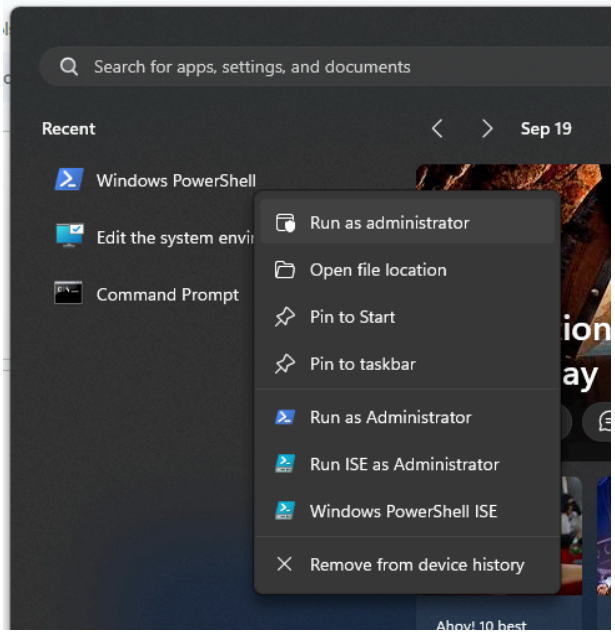
Step 6: click on environment variable



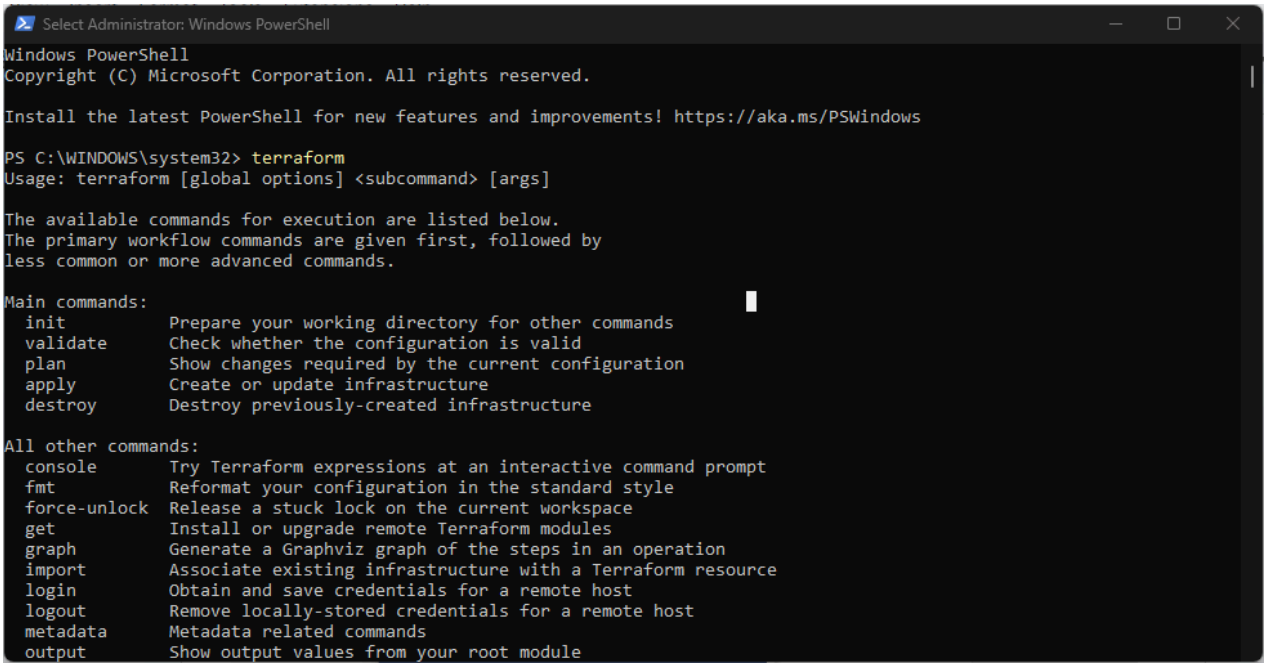
Step 7: Click on New, give variable name = Path, Click on browse directory, select c:/terraforms/terra….exe, OK



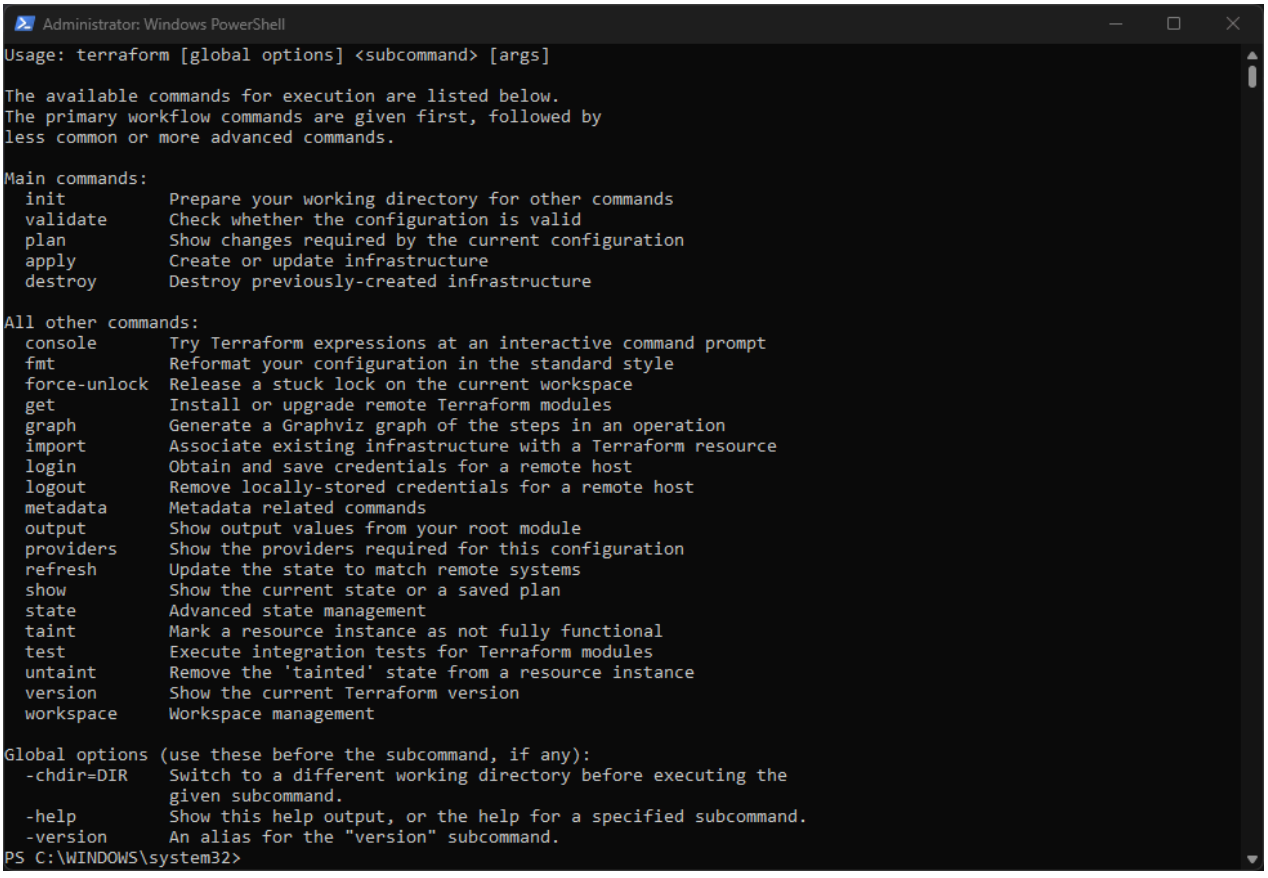
Step 8: Cross verify terraform installed properly or not. go to MS Powershell, run as a administrator



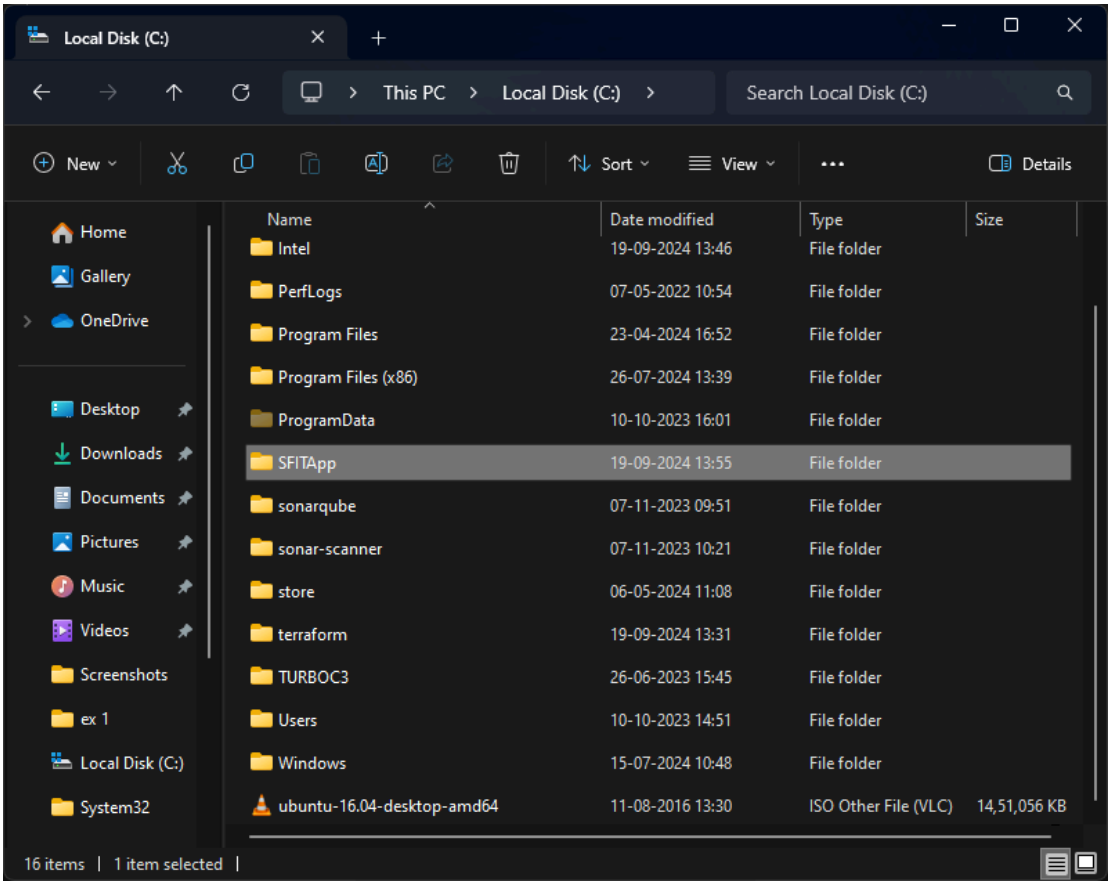
Step 9: Type terraform



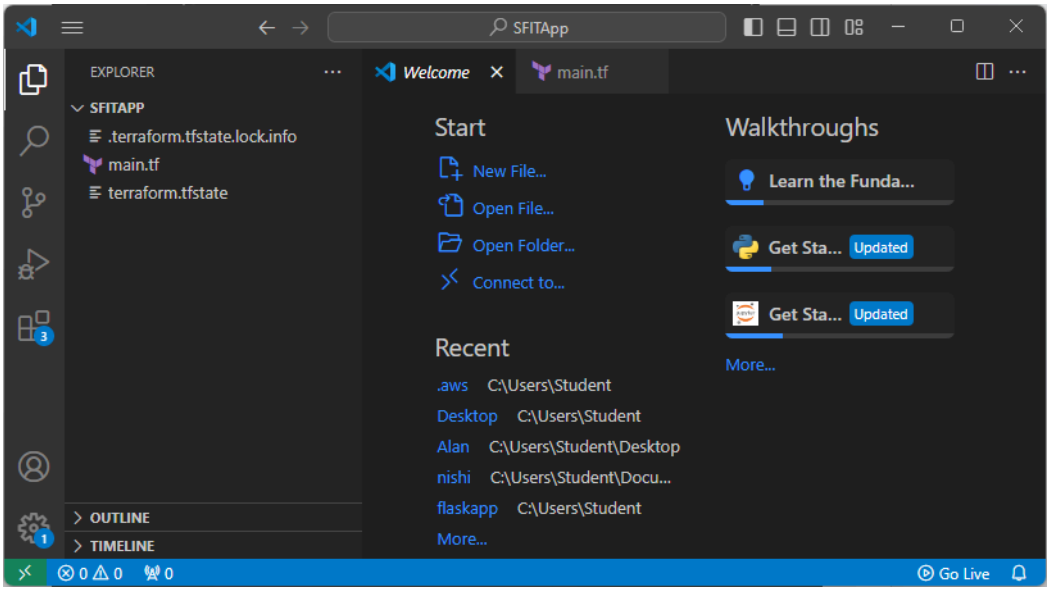
Step 10: You will find init, validate, plan, apply and destroy options means you have installed terraform successfully.



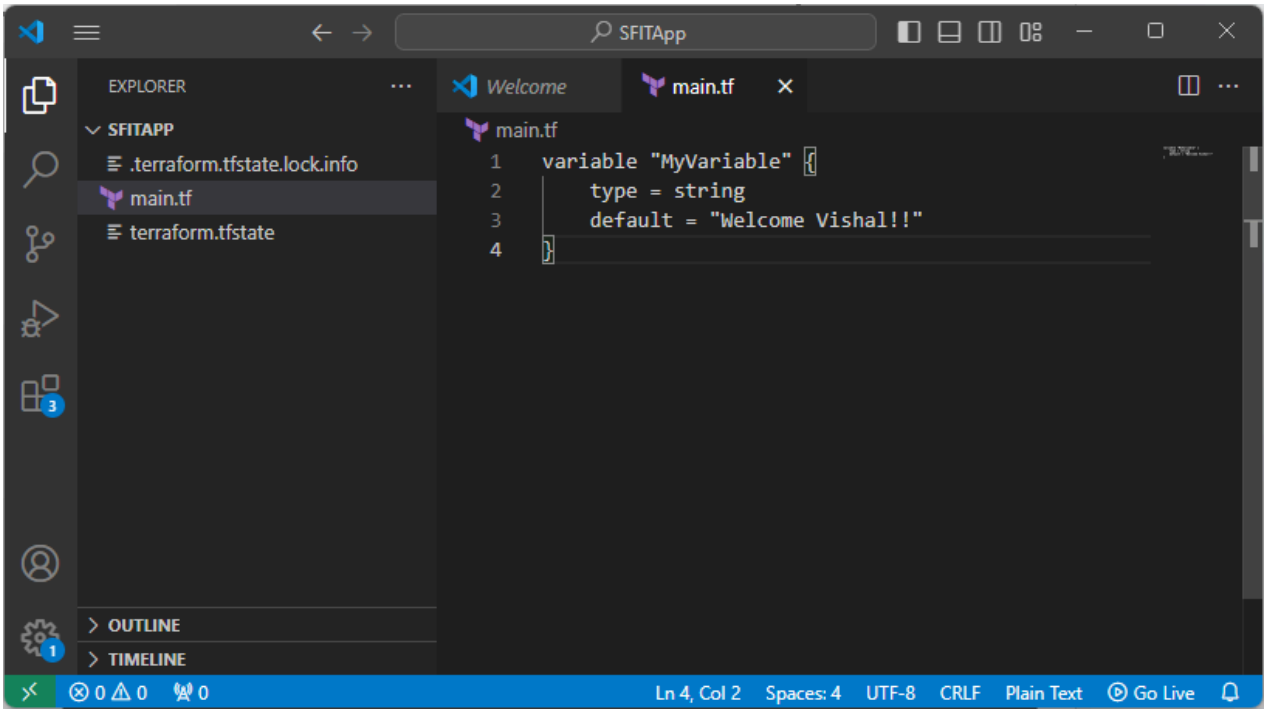
Step 11: Create a folder c:\SFITApp



Step 12 : Open VS Code Editor and Open folder SFITApp

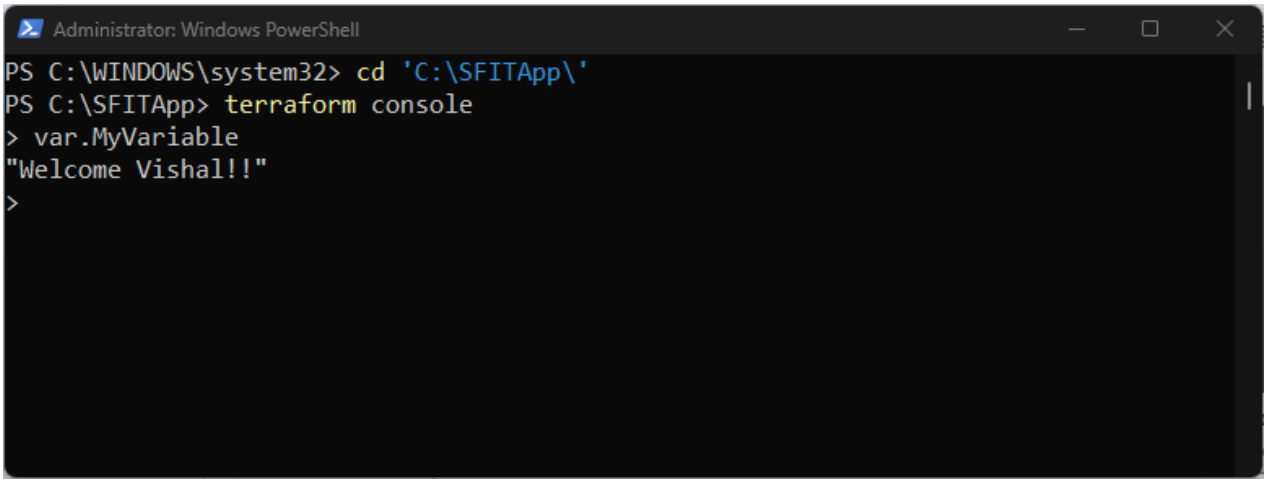


Step 13: write main.tf file with input variables. The input variables, like the one above, use a couple of different types: string, list, map, and Boolean.



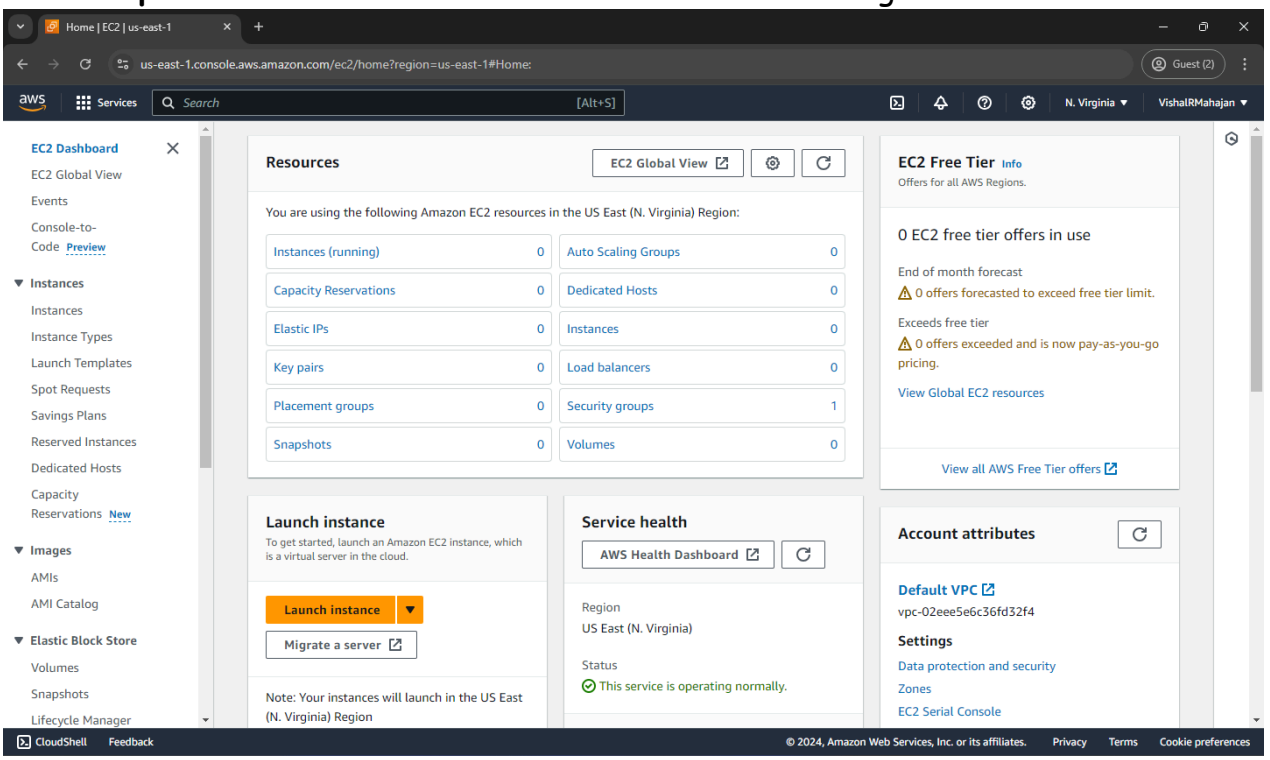
Step 14: Check the output on command Prompt…Go to C:\SFITApp, Type Terraform Console

You will get a terraform prompt, run the .tf with var.MyVariable, You will get Welcome Vishal !! message

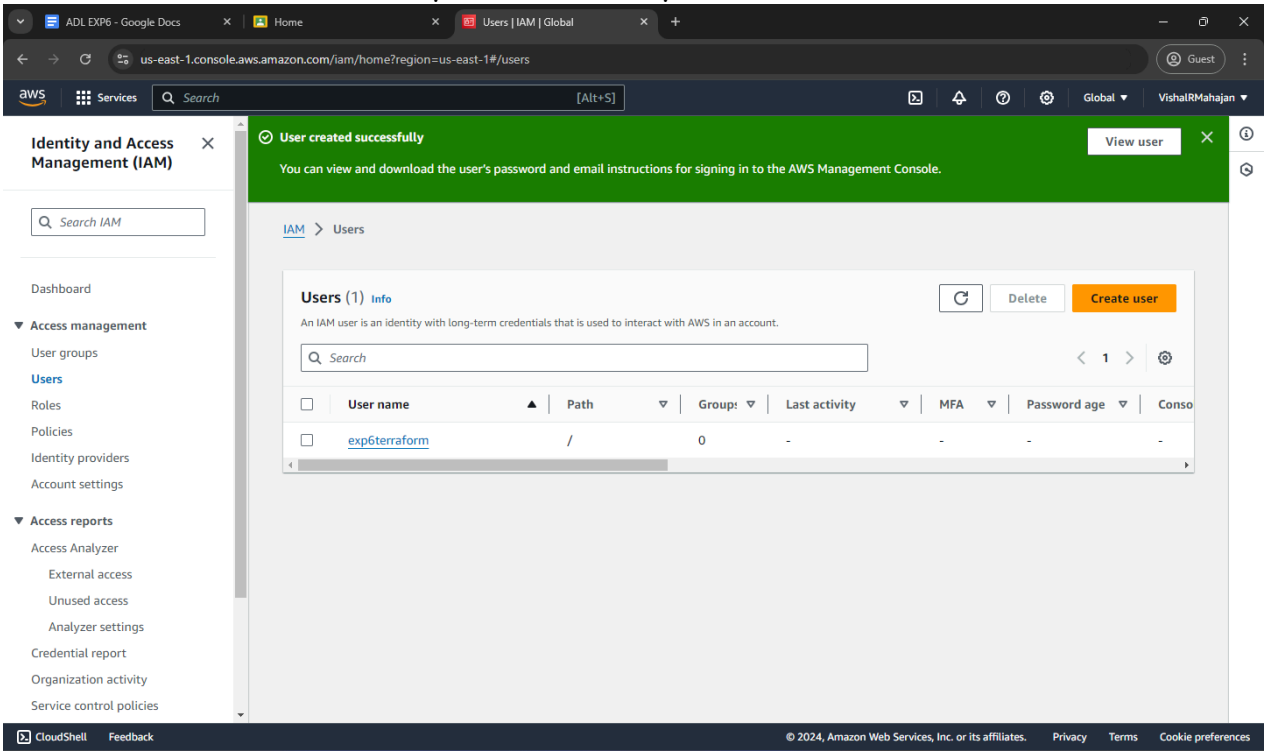


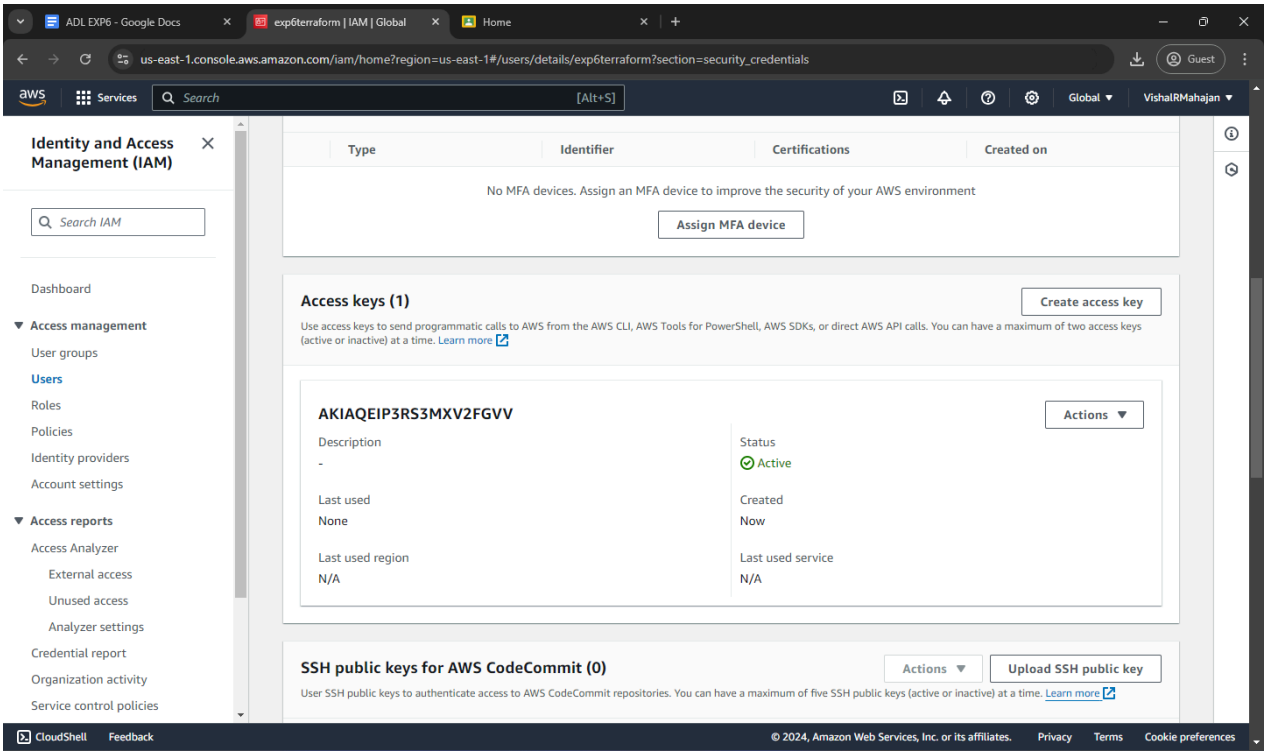
Part B: To build, apply and destroy AWS Resources using Terraform.

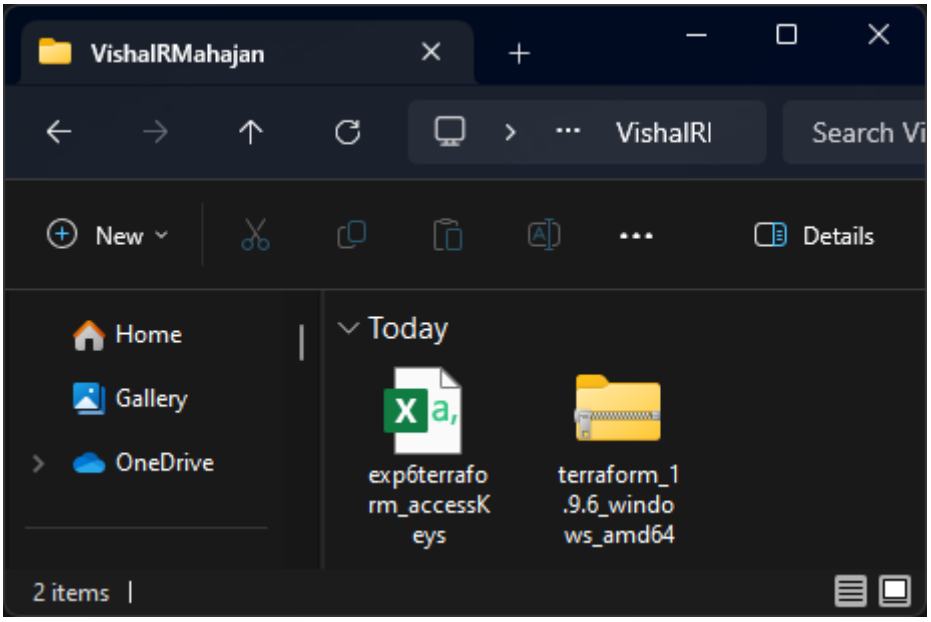
Step 1: First we will check that no instance is running on EC2.



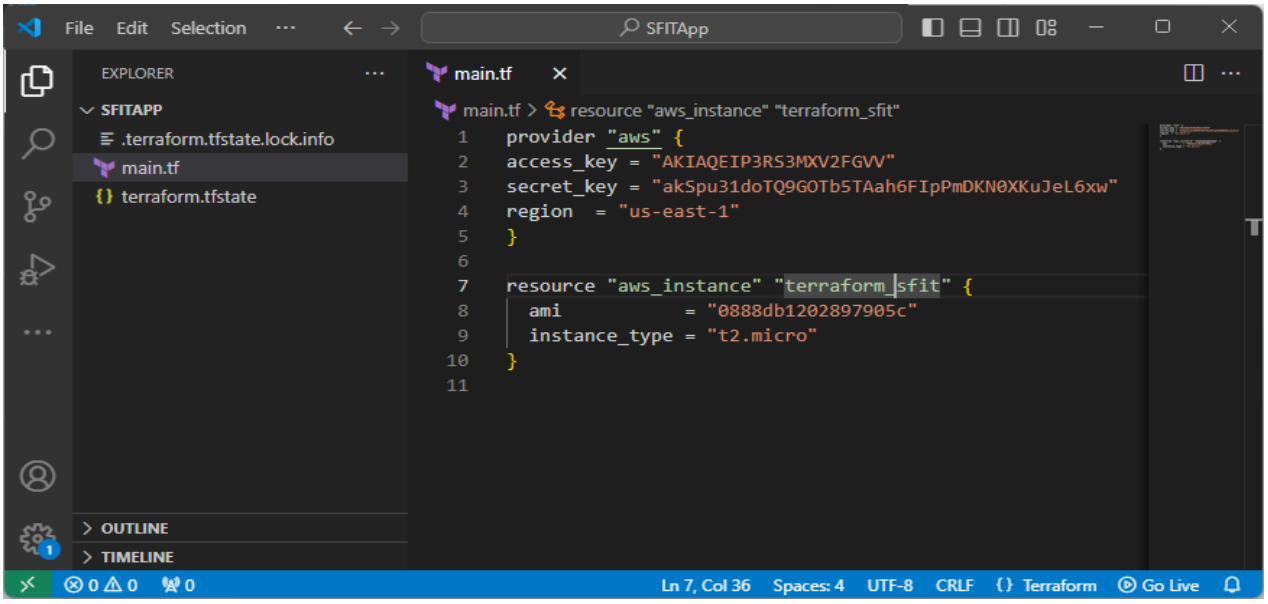
Step 2: Create an IAM user with Programmatic Password, Administrator access and download access key and secret key from download.csv




ADL EXP6 - Google Docs
exp6terraform | IAM | Global
Home
us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/users/details/exp6terraform?section=security_credentials
Guest
aws
Services
Search
[Alt+S]
Global
VishalRMahajan
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Access reports
Access Analyzer
External access
Unused access
Analyzer settings
Credential report
Organization activity
Service control policies
Type
Identifier
Certifications
Created on
No MFA devices. Assign an MFA device to improve the security of your AWS environment
Assign MFA device
Access keys (1)
Create access key
Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. Learn more
AKIAQEIP3RS3MXV2FGVV
Actions
Description
-
Status
Active
Last used
None
Created
Now
Last used region
N/A
Last used service
N/A
SSH public keys for AWS CodeCommit (0)
Actions
Upload SSH public key
User SSH public keys to authenticate access to AWS CodeCommit repositories. You can have a maximum of five SSH public keys (active or inactive) at a time. Learn more
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


VishalRMahajan
VishalRI
Search Vi
New
Details
Home
Gallery
OneDrive
Today
exp6terrafo rm_accessK eys
terraform_1 .9.6_windo ws_amd64
2 items

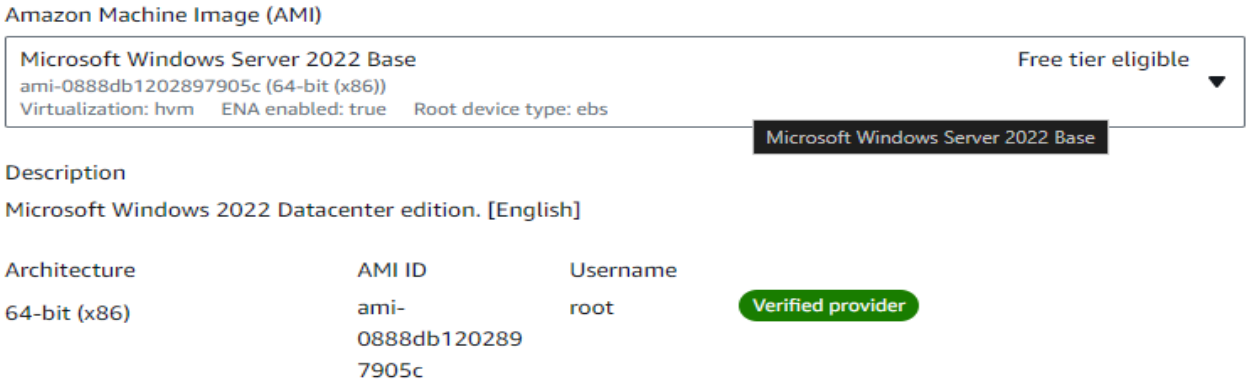
Step 3: Now write a Terraform program in vs code, create new file with .tf extension



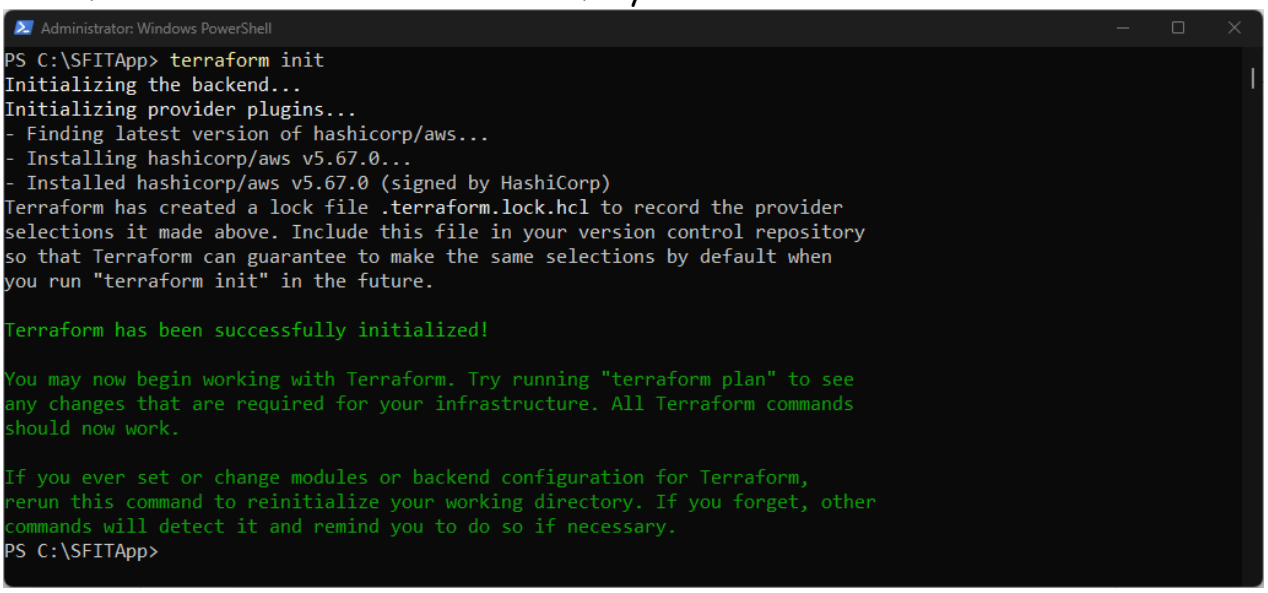
SAMPLE CODE :

```
provider "aws" {
access_key = ""
secret_key = ""
region  = "us-east-1"
}

resource "aws_instance" "terraforma-sfit" {
  ami           = "ami-{code}"
  instance_type = "t2.micro"
}
```

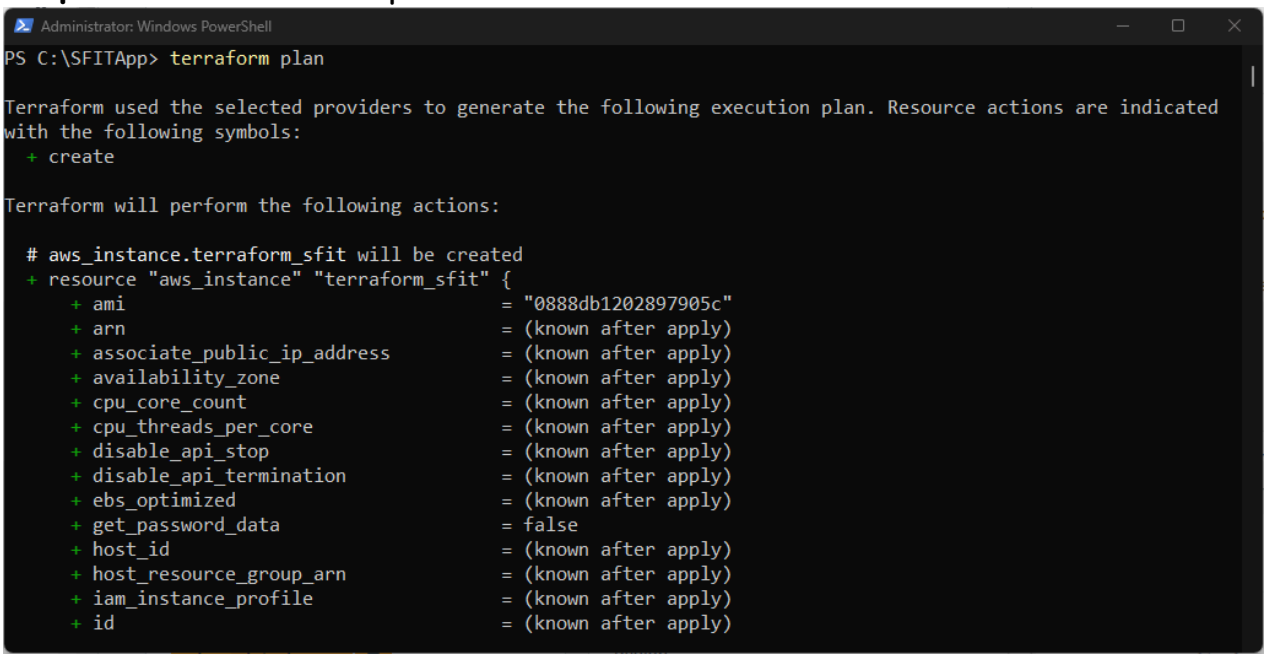
In the EC2 Launch instance, you will get ami : amazon machine image . For Instance type use t2.micro, as it is freely available



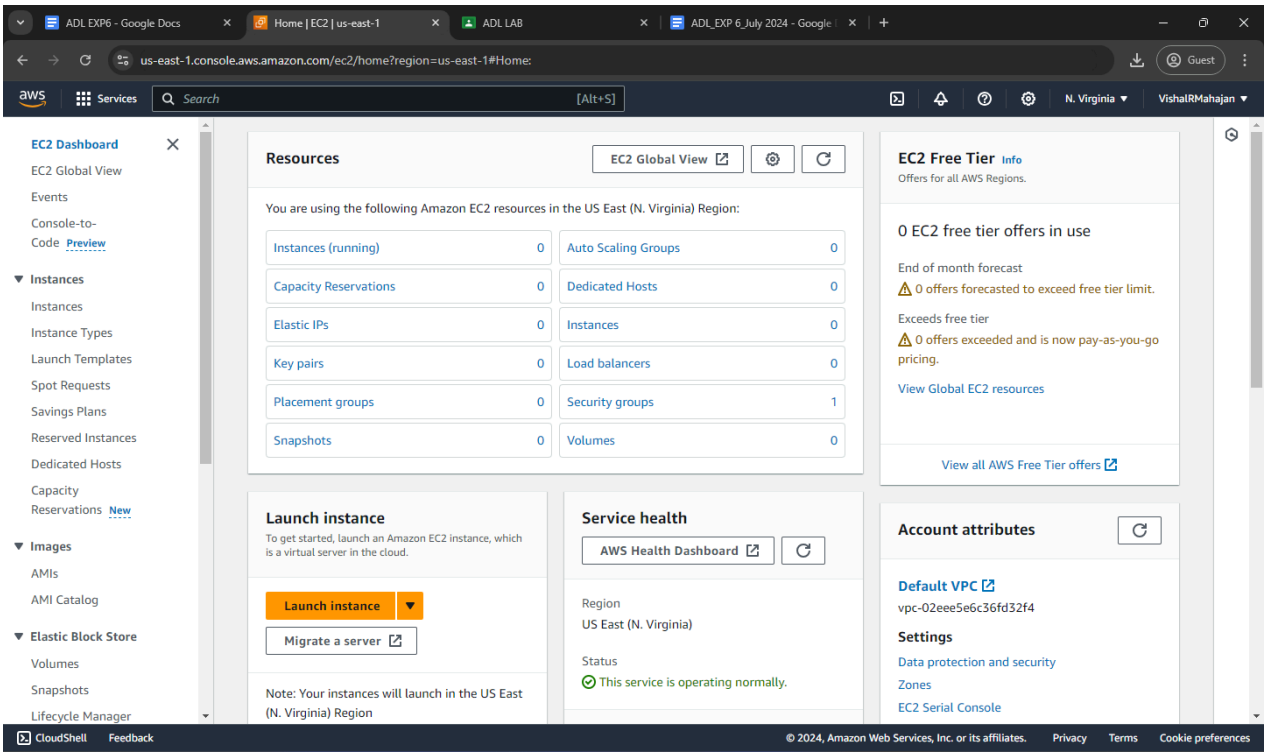
Step 4: Now initialize the terraform …type cd c:\SFITApp then terraform init Terraform has been initialized successfully.



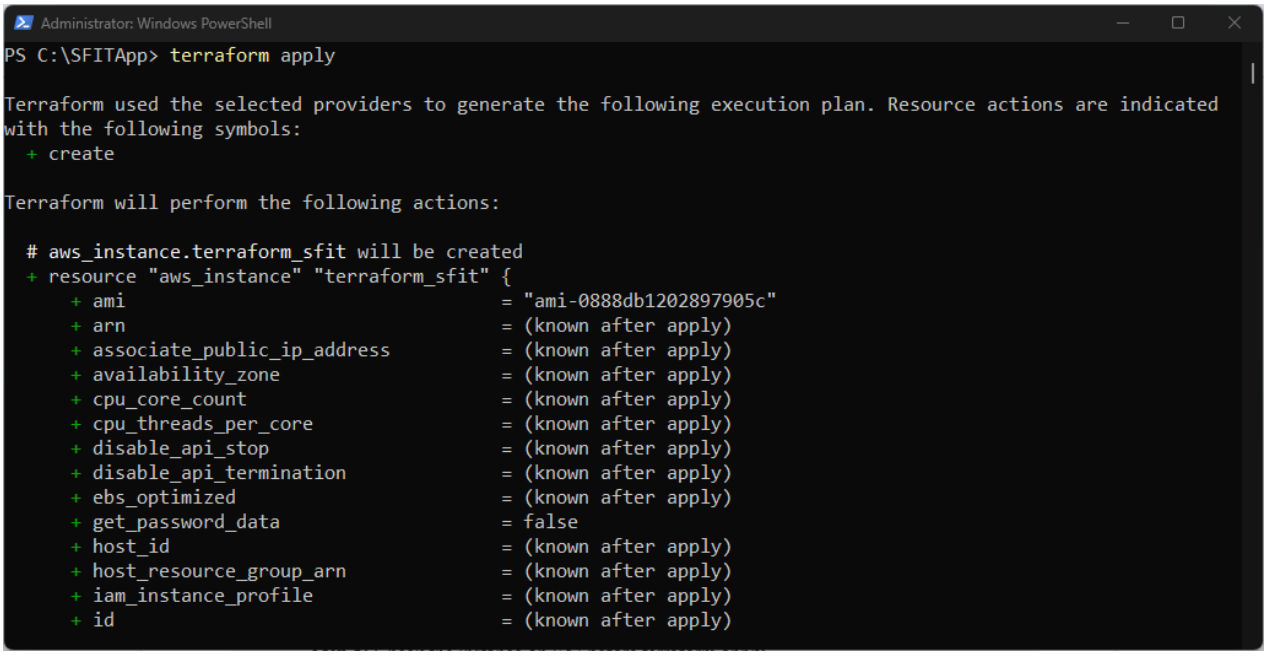
Step 5: Run terraform plan



Step 6: Check the instance on Ec2 before terraform apply Instance is not yet created.

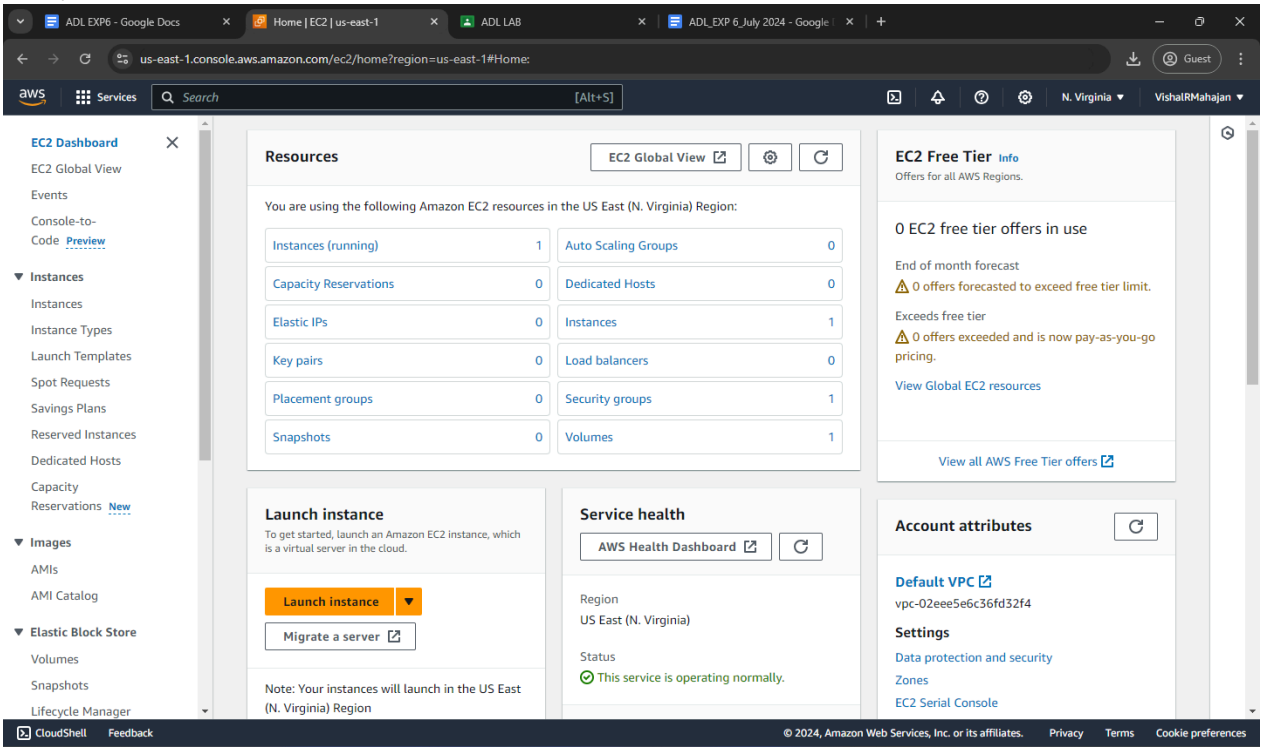


Step 7: Run Terraform apply

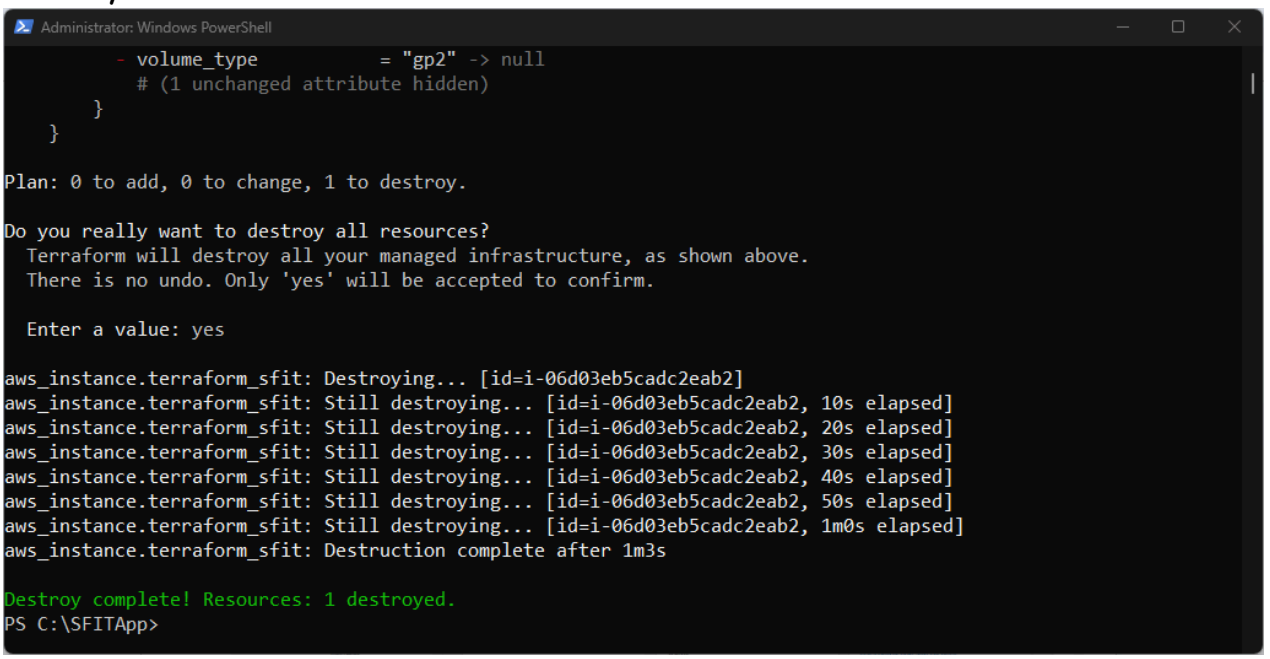


```
Administrator: Windows PowerShell

      + metadata_options (known after apply)

      + network_interface (known after apply)

      + private_dns_name_options (known after apply)

      + root_block_device (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.terraform_sfit: Creating...
aws_instance.terraform_sfit: Still creating... [10s elapsed]
aws_instance.terraform_sfit: Creation complete after 16s [id=i-06d03eb5cadc2eab2]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
PS C:\SFITApp>
```

Step 8: Check terraform created instance on EC2. We have created 1 instance.



Step 9: Now destroy the instance from the command prompt. Run terraform destroy



8. **Post-Experiments Exercise**
   - **A. Extended Theory:**
     - Terraform Vs. Kubernetes (Soft copy)
     - Terraform Vs. Ansible (Soft copy)
     - How to create AWS S3 Bucket using Terraform? (Write only Terraform Code in hand)
   - **B. Questions:**(Soft copy)
     1. Name all version controls supported by Terraform.
     2. Name some major competitors of Terraform.
     3. Why is Terraform preferred as one of the DevOps tools?
   - **C. Conclusion:**
     - A. Write what was performed in the experiment
     - B. Mention a few applications of what was studied.
     - C. Write the significance of the studied topic
   - **D. References:**
     - A. https://www.ibm.com/cloud/learn/terraform#toc-terraform--OoC-5III
     - B. https://www.simplilearn.com/terraform-interview-questions-and-answers-article
     - C. https://aws.amazon.com/microservices/
     - D. https://www.monkeyvault.net/docker-vs-virtualization/
     - E. https://cloudacademy.com/blog/docker-vs-virtualization/
     - F. https://www.terraform.io/docs/language/values/variables.html

1. Terraform VS Kubernetes

Terraform and **Kubernetes** serve different purposes in the DevOps ecosystem, although they complement each other.

- **Terraform** is an Infrastructure as Code (IaC) tool. It allows you to define and provision your infrastructure using a declarative configuration language. Terraform is cloud-agnostic, meaning you can manage resources across multiple cloud platforms like AWS, GCP, and Azure.
- **Kubernetes** is a container orchestration platform. It is designed to automate the deployment, scaling, and management of containerized applications. Kubernetes manages workloads that are already running in a predefined infrastructure.

Key Differences:

1. **Purpose:**
    - **Terraform**: Primarily used for managing infrastructure (compute, storage, network).
    - **Kubernetes**: Manages containerized applications and ensures high availability through automated scheduling and scaling.
2. **Configuration Approach**:
    - **Terraform** uses a declarative model to define the desired state of infrastructure.
    - **Kubernetes** also uses a declarative model, but it focuses on containerized workloads rather than infrastructure.
3. **Integration**:
    - **Terraform** can deploy Kubernetes clusters by provisioning the necessary infrastructure first, and then Kubernetes can take over to manage the applications within those clusters.
4. **State Management**:
    - **Terraform** manages the state of the infrastructure using a state file, which keeps track of the resources.
    - **Kubernetes** maintains the state of applications using an etcd key-value store and its internal API.

2. Terraform VS Ansible

Both **Terraform** and **Ansible** are powerful automation tools in the DevOps world, but their focus and functionality are different.

- **Terraform** is designed for provisioning and managing infrastructure as code. It allows you to define infrastructure resources and ensure that the infrastructure matches the defined state.
- **Ansible** is a configuration management tool that automates tasks such as software installation, configuration, and management of servers.

Key Differences:

1. **State Management**:
    - **Terraform** keeps track of the desired state and current state through a state file. It ensures that any changes are applied to match the desired state.
    - **Ansible** does not maintain a state file; it executes tasks based on playbooks, but it doesn't ensure the infrastructure is in a specific state unless those tasks are re-run.
2. **Idempotency**:
    - Both **Terraform** and **Ansible** strive for idempotency, meaning they ensure that running their commands multiple times will yield the same result.
    - Terraform's state management makes it more reliable for infrastructure provisioning, while Ansible is more commonly used for tasks like server configuration.
3. **Cloud-Agnostic**:
    - Both are cloud-agnostic, but **Terraform** excels at managing cloud infrastructure across multiple platforms, while **Ansible** focuses more on task automation, including software configuration and application deployment.
4. **Use Cases**:
    - **Terraform**: Provisioning infrastructure resources such as VMs, storage, and networking.
    - **Ansible**: Configuring software, deploying applications, and managing services on top of already provisioned infrastructure.

8B) Questions:

1. Name all version controls supported by Terraform:

Terraform supports various version control systems (VCS) through Terraform Cloud or Terraform Enterprise. These include:

- GitHub
- GitLab
- Bitbucket
- Azure DevOps
- AWS CodeCommit

2. Name some major competitors of Terraform:

Some major competitors of Terraform are:

- Pulumi
- AWS CloudFormation
- Ansible
- Chef
- SaltStack

3. Why is Terraform preferred as one of the DevOps tools?:

- Terraform is preferred due to its **cloud-agnostic nature**, allowing users to manage resources across multiple cloud providers.
- Its **declarative configuration language (HCL)** simplifies infrastructure management.
- **State management** ensures that infrastructure remains consistent over time.
- It promotes **reusability** with modules and allows for **versioning** of infrastructure.
- Terraform integrates well with other DevOps tools and platforms, making it a core part of infrastructure provisioning in many CI/CD pipelines.