

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

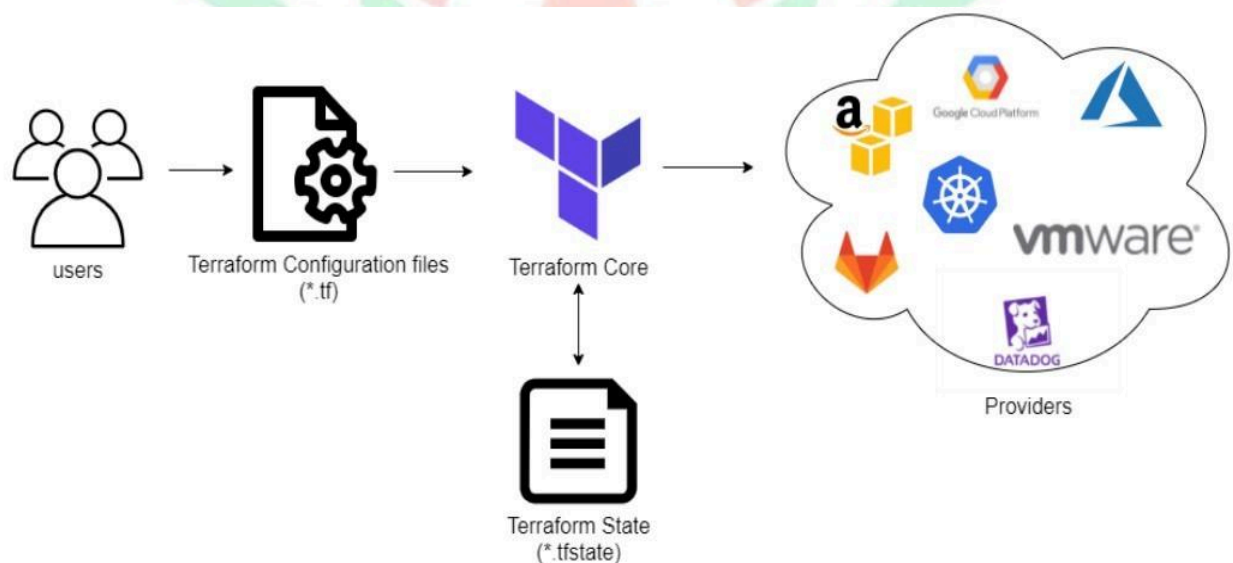
Terraform can apply complex change sets to your infrastructure with minimal human interaction. When you update configuration files, Terraform determines what changed and creates incremental execution plans that respect dependencies.

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 example 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.

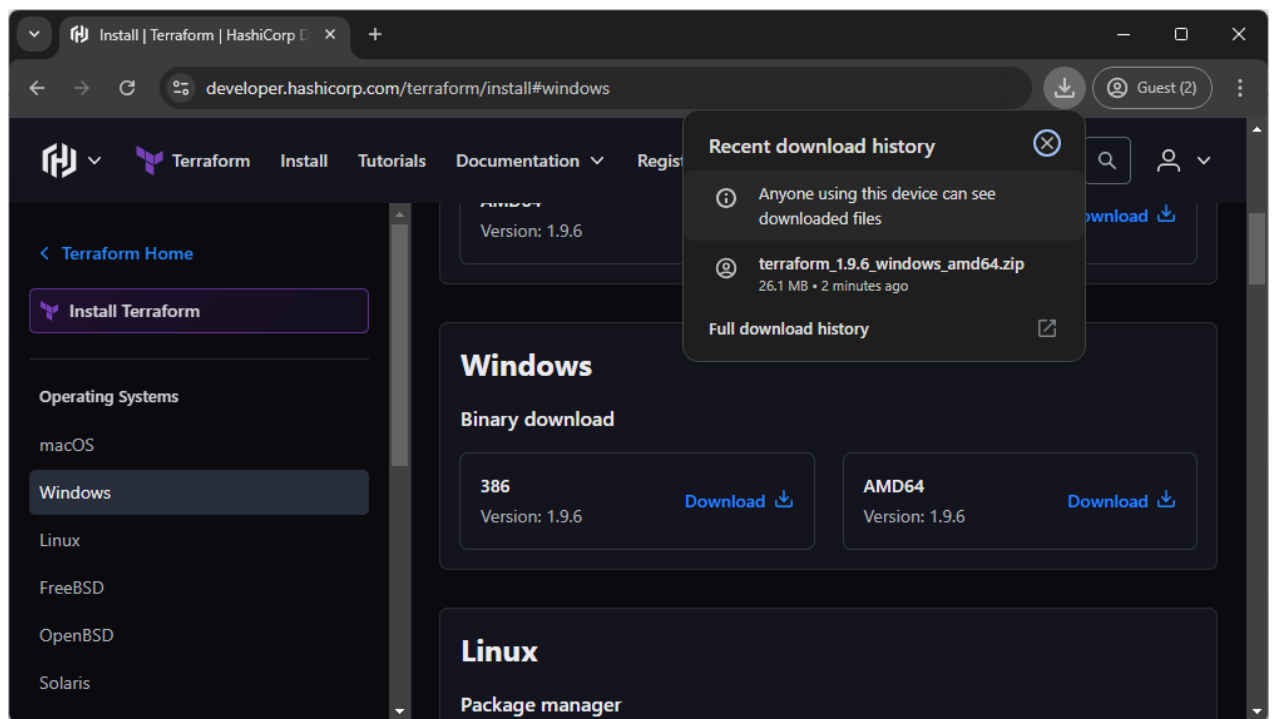
Terraform Core Concepts:

Below are the core concepts/terminologies used in Terraform:

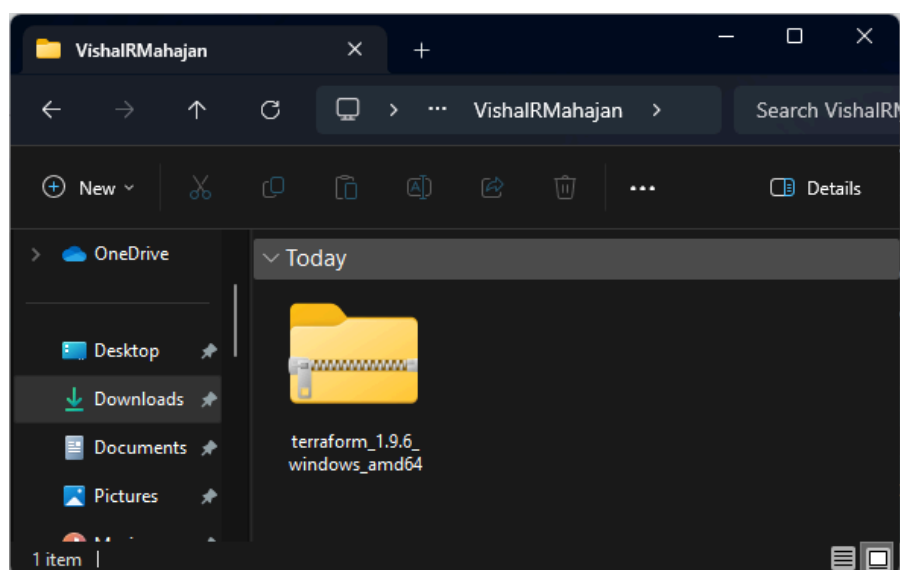
- **Variables:** Also used as input-variables, it is a key-value pair used by Terraform modules to allow customization.
- **Provider:** It is a plugin to interact with APIs of service and access its related resources.
- **Module:** It is a folder with Terraform templates where all the configurations 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.

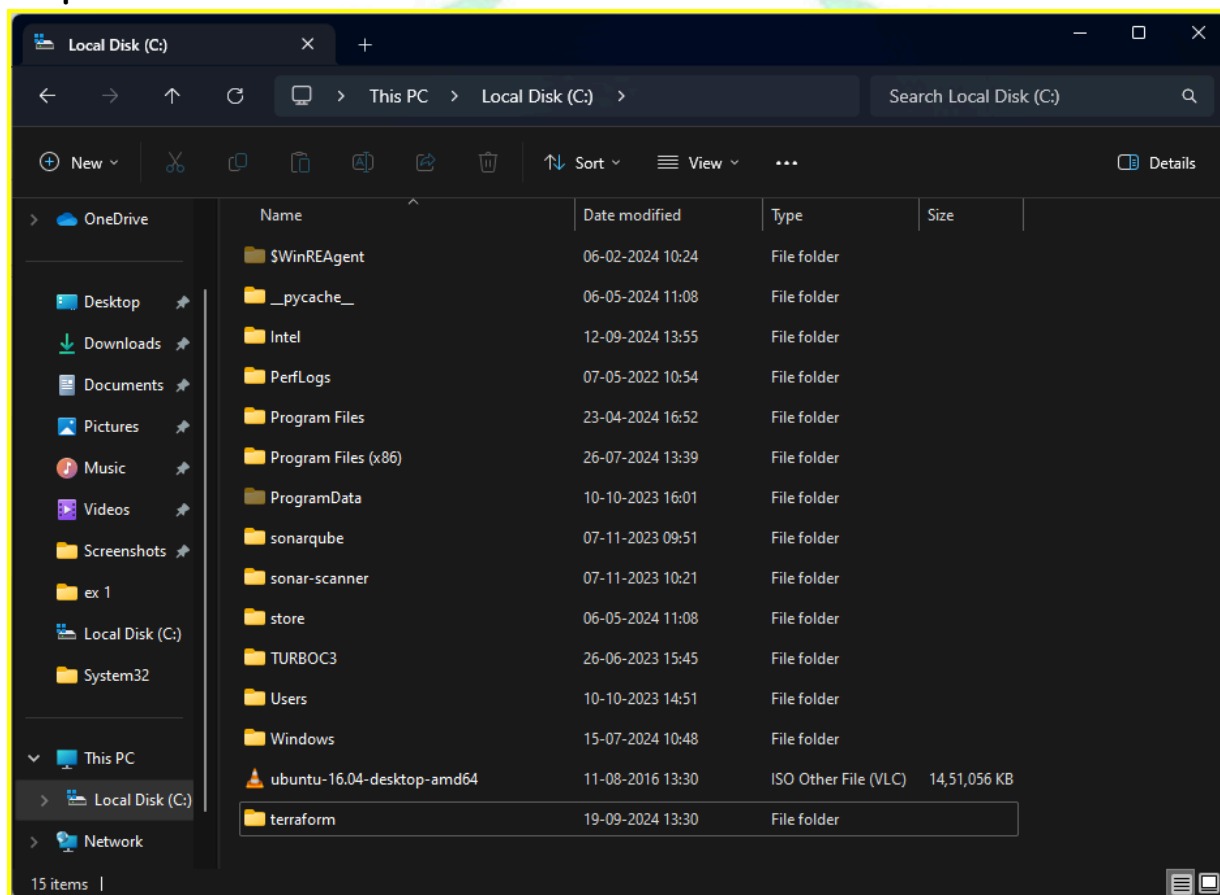
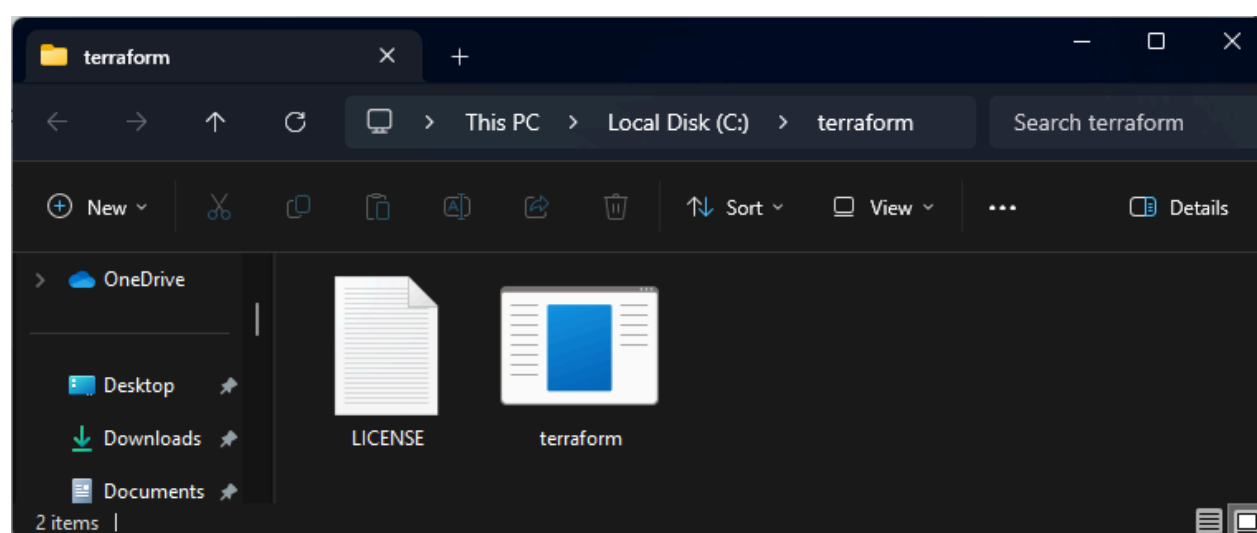
7. Laboratory Exercise

Step 1 : Download appropriate terraform package(.zip) from [terraform.io/downloads.html](https://developer.hashicorp.com/terraform/install#windows) for Windows

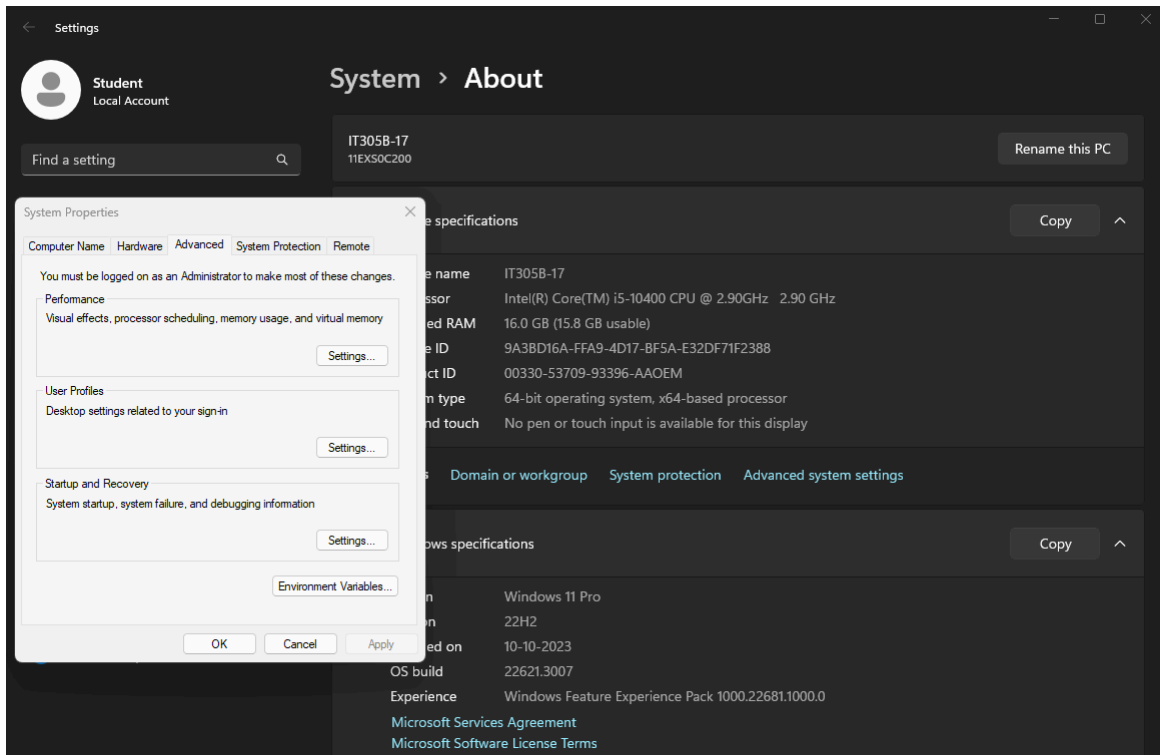


Step 2: Download Terraform for Windows 64-bit / (32-bit).

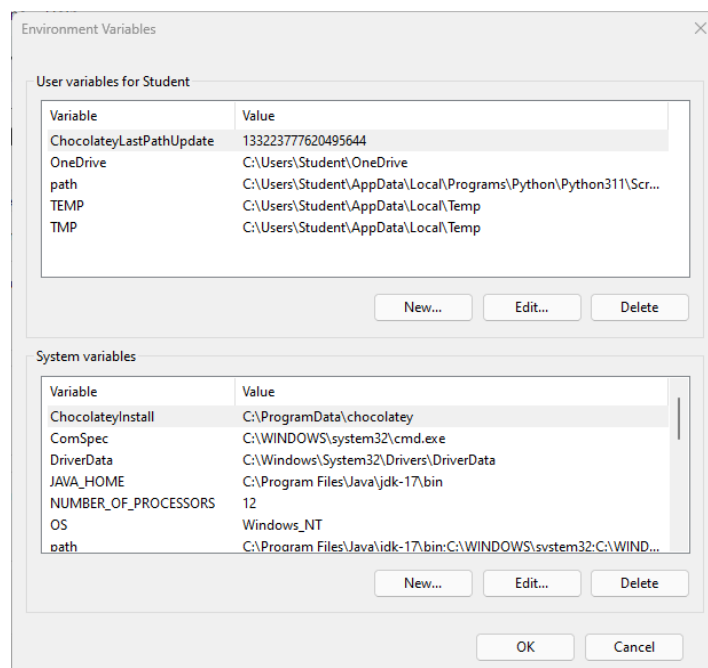


Step 3: Create a folder 'terraform' in drive C .**Step 4 : Extract downloaded zip in to this c:/terraform folder**

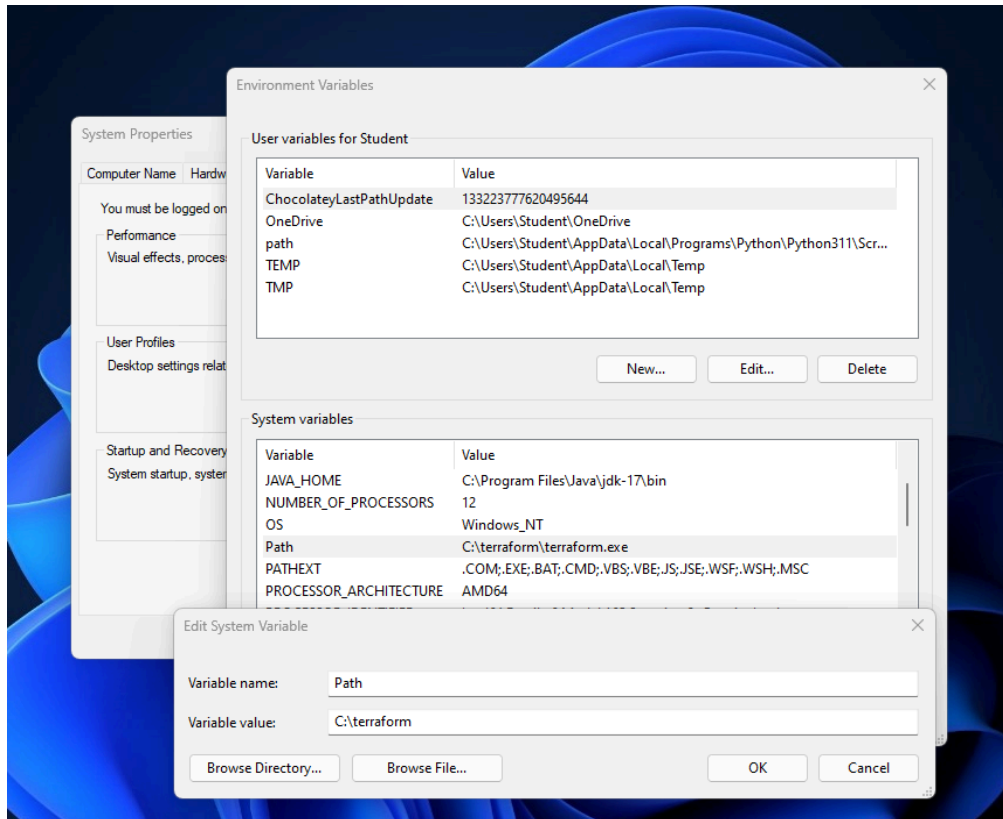
Step 5: Now we need to set a path for terraform. Go to My computer/ This PC, right click, select properties, go to advanced system settings.



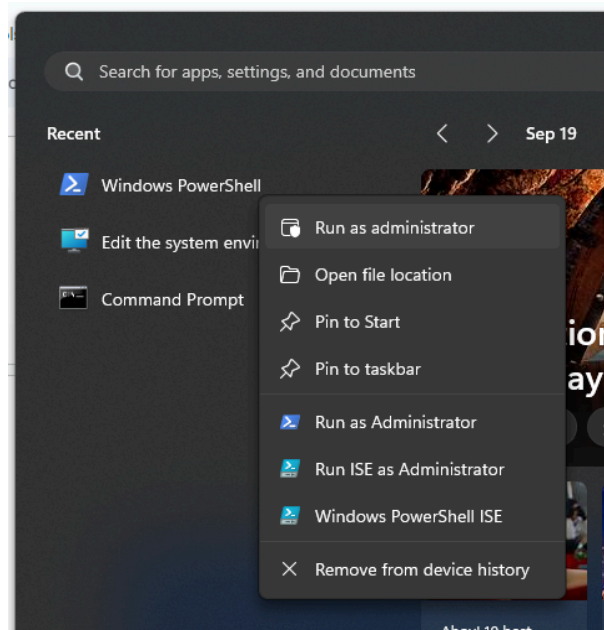
Step 6: click on environment variable



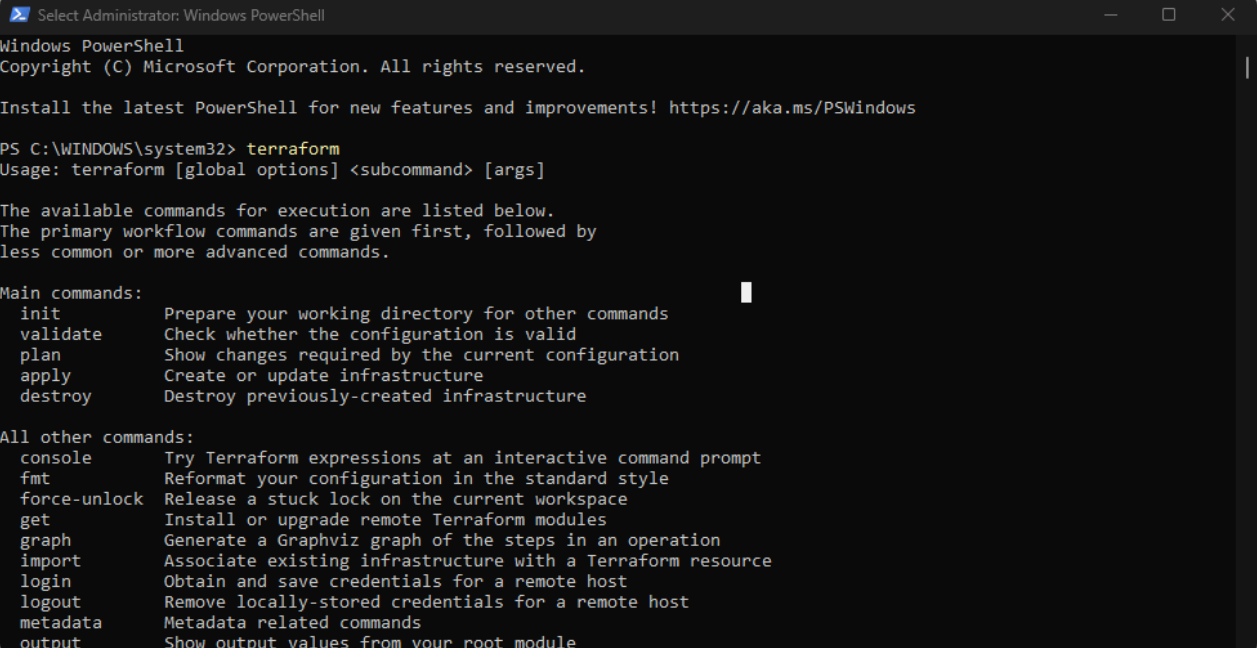
Step 7: Click on New, give variable name = Path, Click on browse directory, select c:/terraform/terra....exe, OK



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



Step 9: Type terraform



```

Select Administrator: Windows PowerShell

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\WINDOWS\system32> terraform
Usage: terraform [global options] <subcommand> [args]

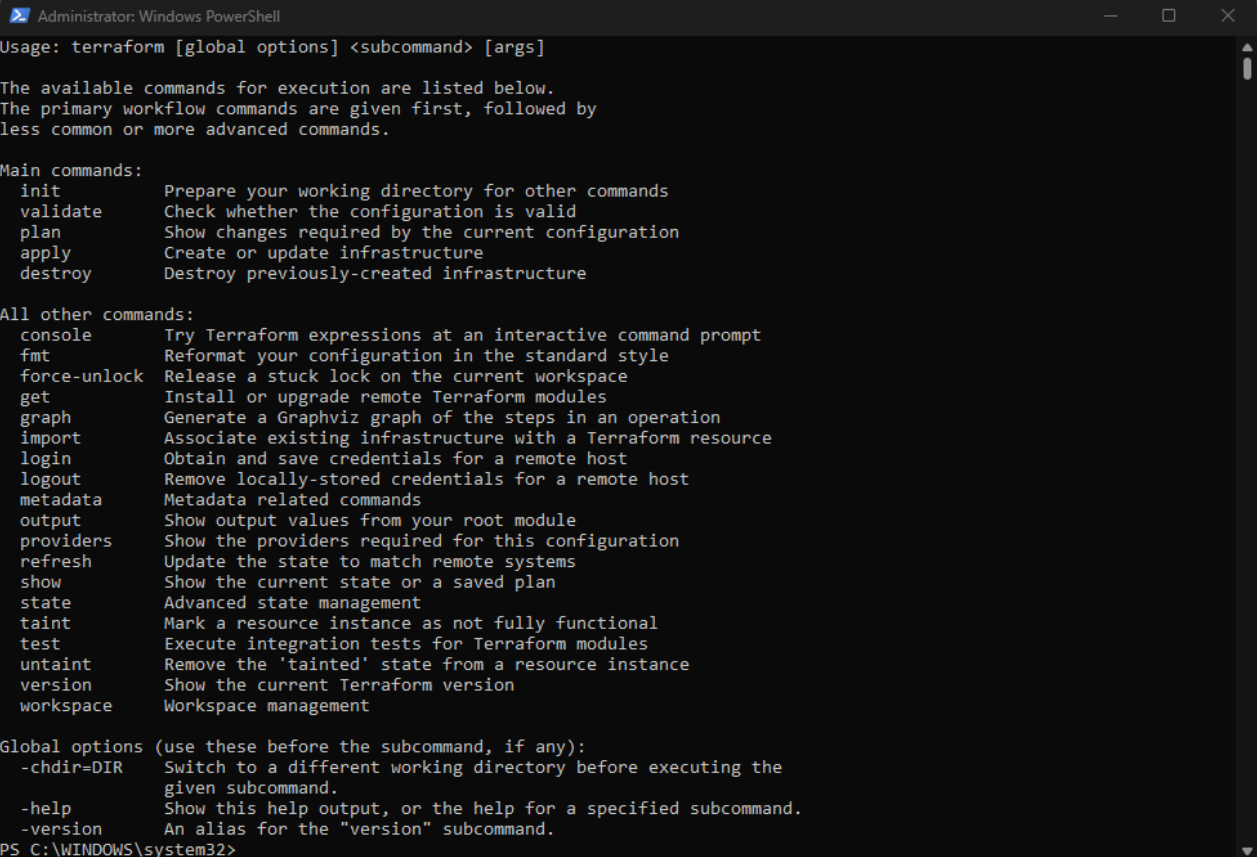
The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

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
  metadata      Metadata related commands
  output        Show output values from your root module

```

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



```

Administrator: Windows PowerShell

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

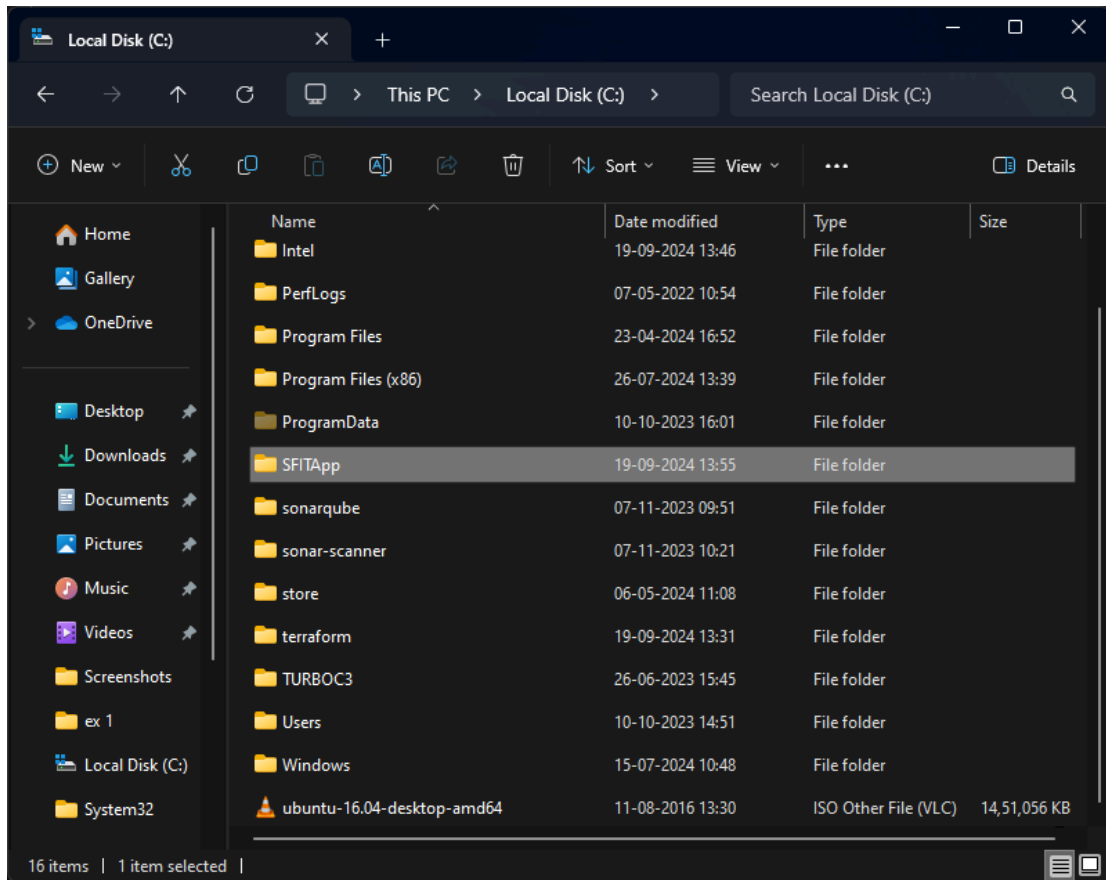
All other commands:
  console       Try Terraform expressions at an interactive command prompt
  fmt           Reformat your configuration in the standard style
  force-unlock  Release a stuck lock on the current workspace
  get           Install or upgrade remote Terraform modules
  graph         Generate 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
  metadata      Metadata related commands
  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          Execute integration tests for Terraform modules
  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):
  -chdir=DIR    Switch to a different working directory before executing the
                given subcommand.
  -help         Show this help output, or the help for a specified subcommand.
  -version      An alias for the "version" subcommand.

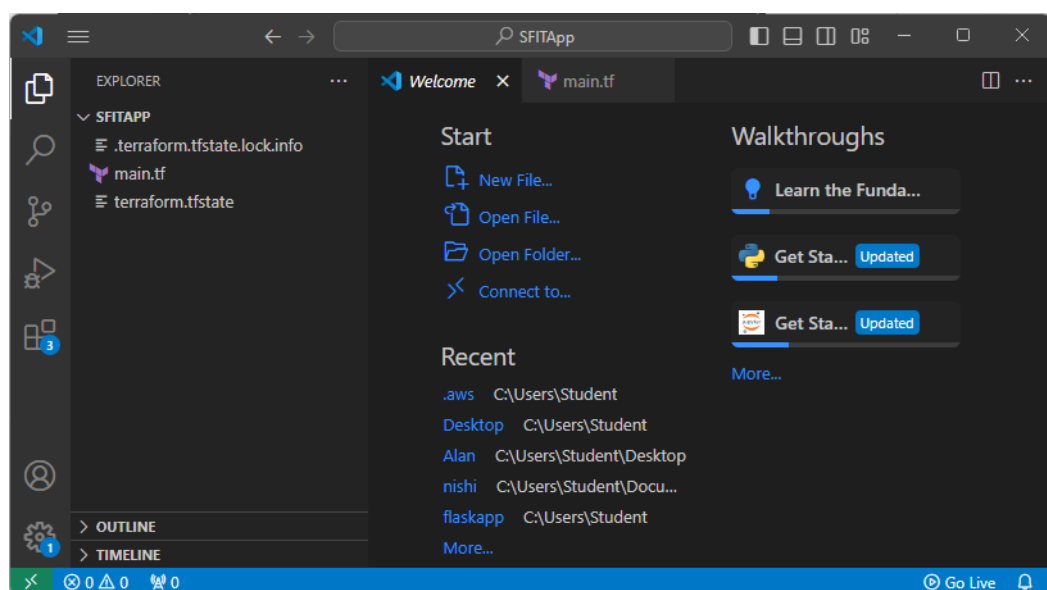
PS C:\WINDOWS\system32>

```

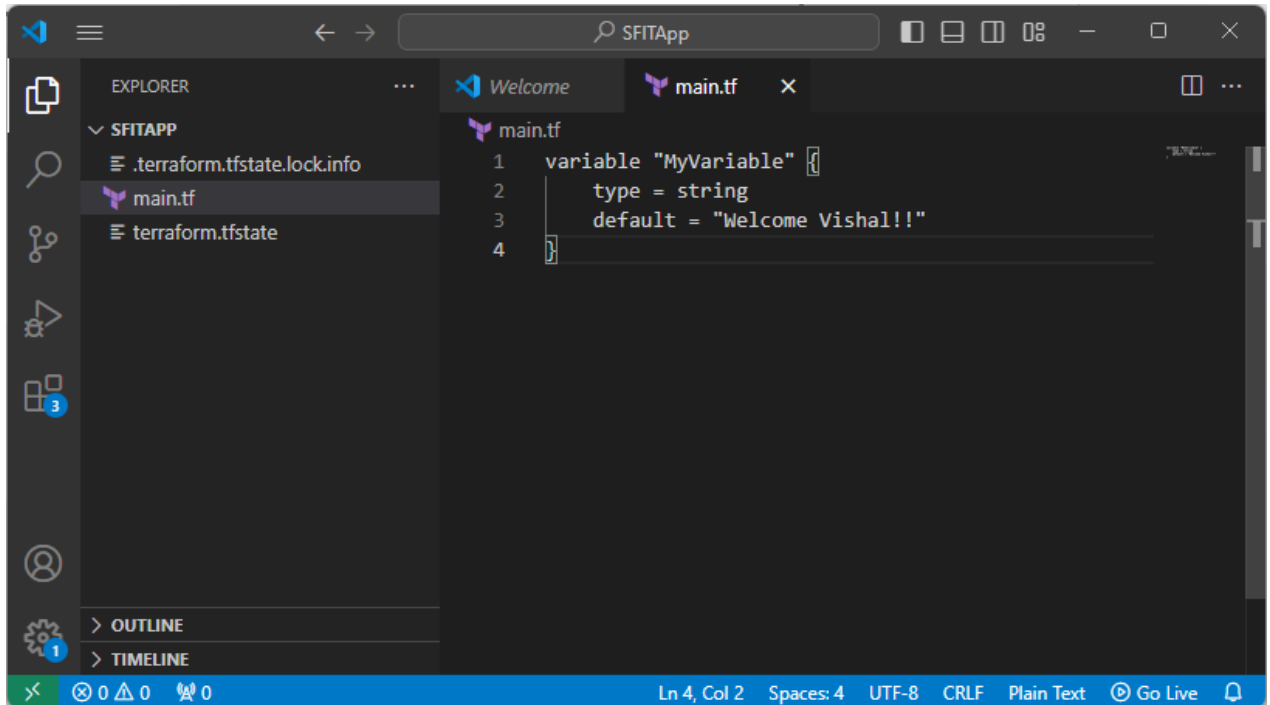

Step 11: Create a folder c:\SFITApp



Step 12 : Open VS Code Editor and Open folder SFITApp

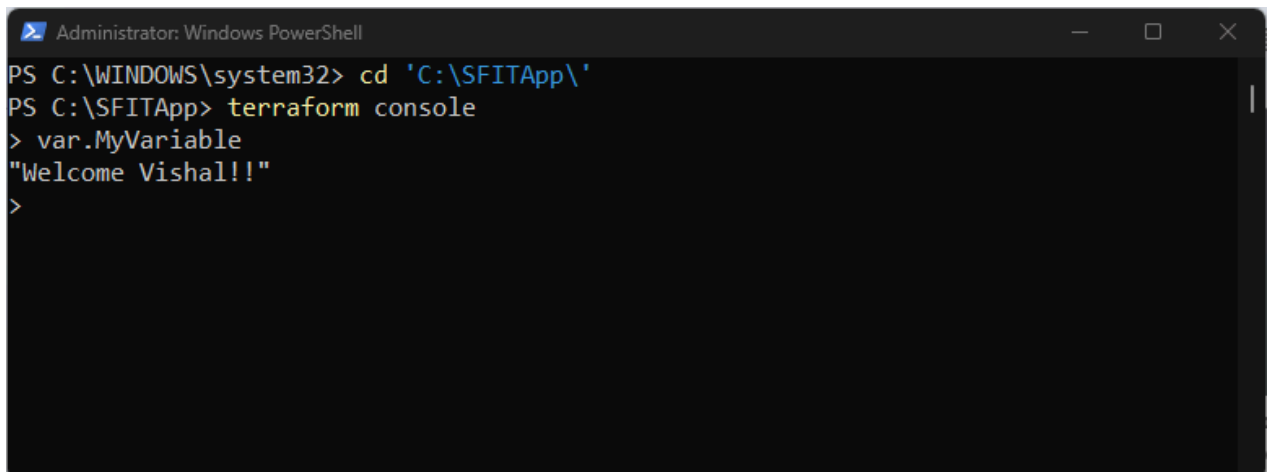


Step 13: write main.tf file with input variables. The input variables, like the one above, use a couple of different types: string, list, map, and Boolean.



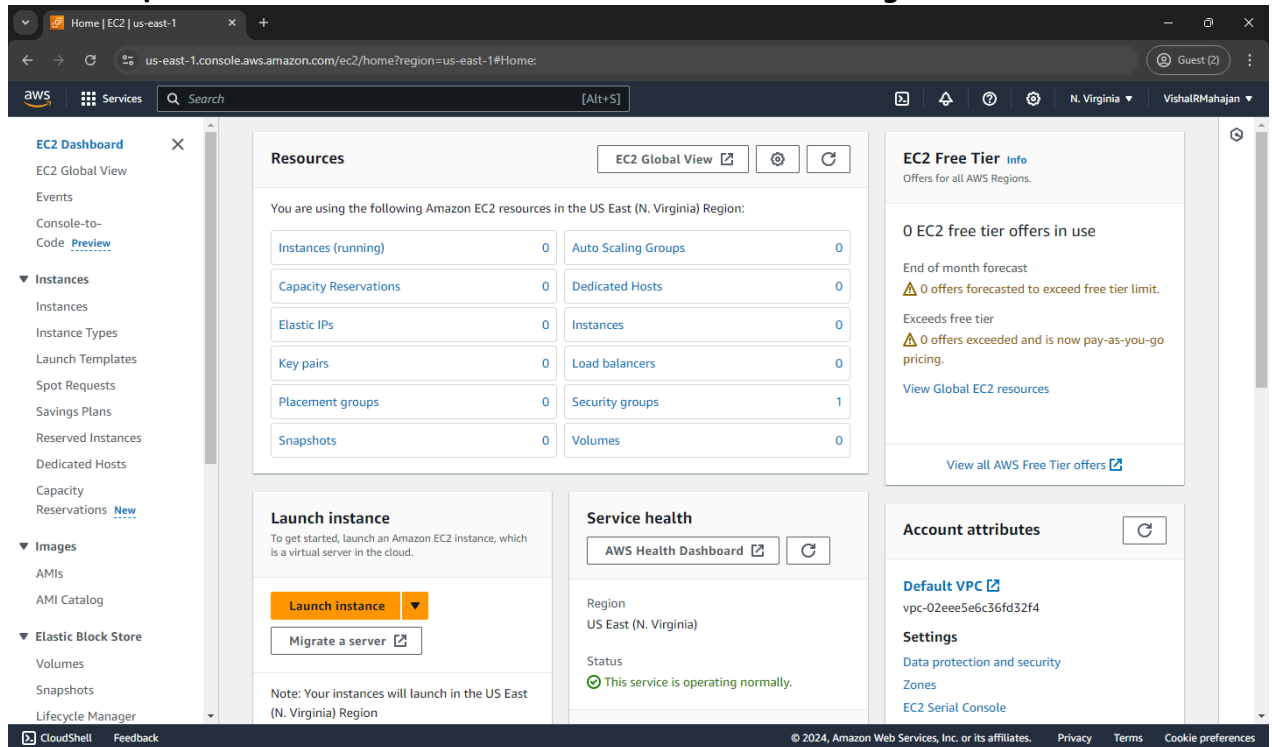
Step 14: Check the output on command Prompt...Go to C:\SFITApp, Type Terraform Console

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

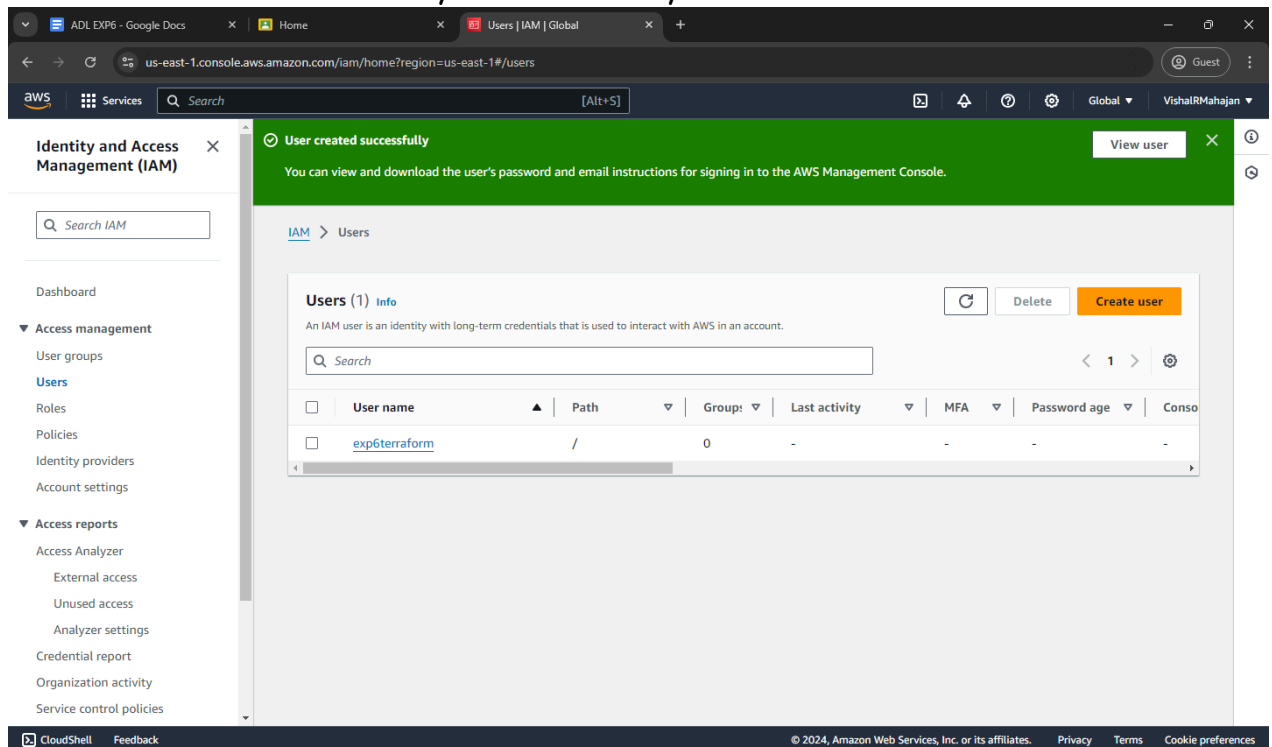


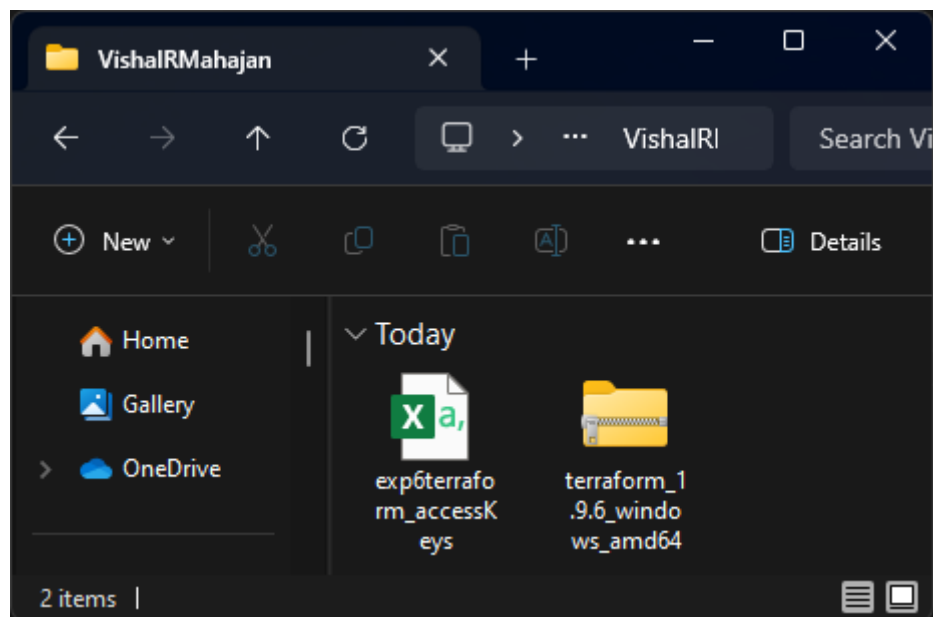
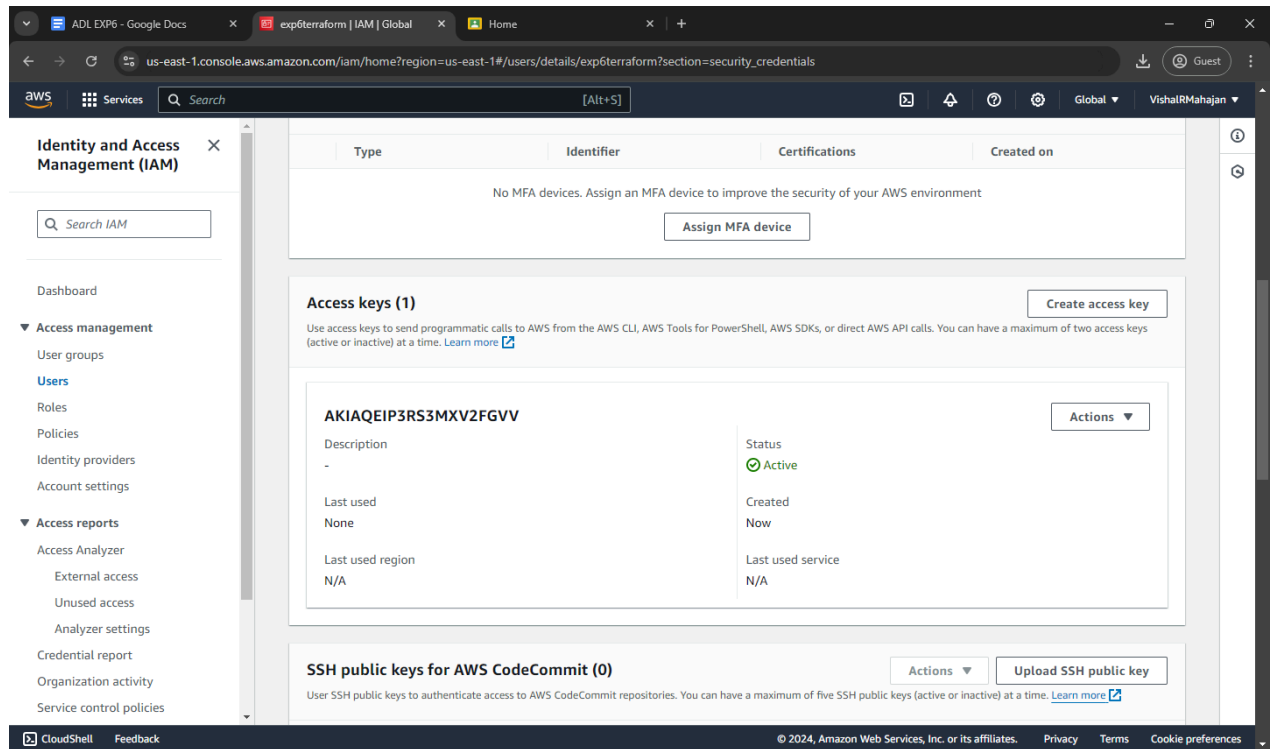
Part B: To build, apply and destroy AWS Resources using Terraform.

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

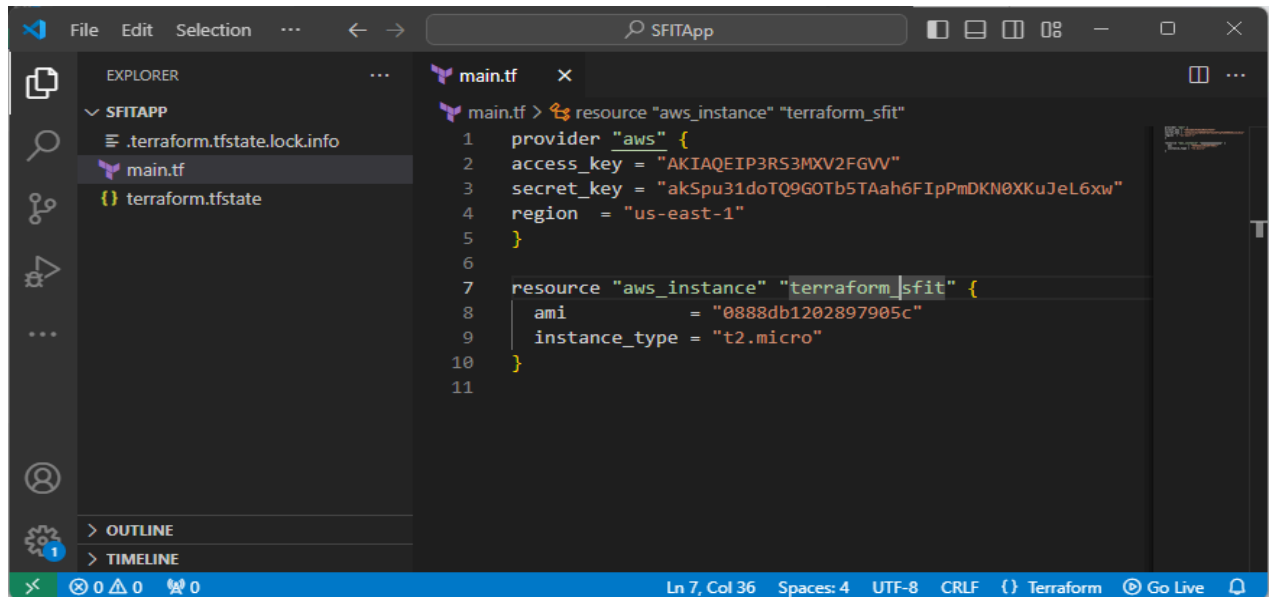


Step 2: Create an IAM user with Programmatic Password, Administrator access and download access key and secret key from download.csv





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

Amazon Machine Image (AMI)

Microsoft Windows Server 2022 Base
ami-0888db1202897905c (64-bit (x86))
Virtualization: hvm ENA enabled: true Root device type: ebs

Free tier eligible

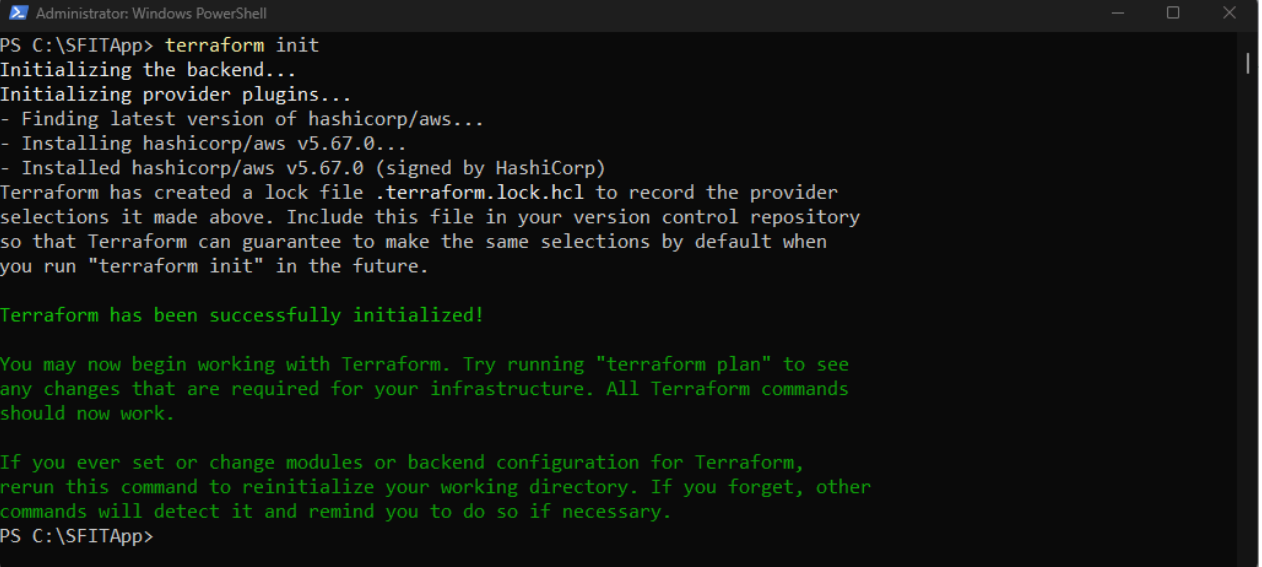
Microsoft Windows Server 2022 Base

Description

Microsoft Windows 2022 Datacenter edition. [English]

Architecture	AMI ID	Username	
64-bit (x86)	ami-0888db1202897905c	root	Verified provider

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



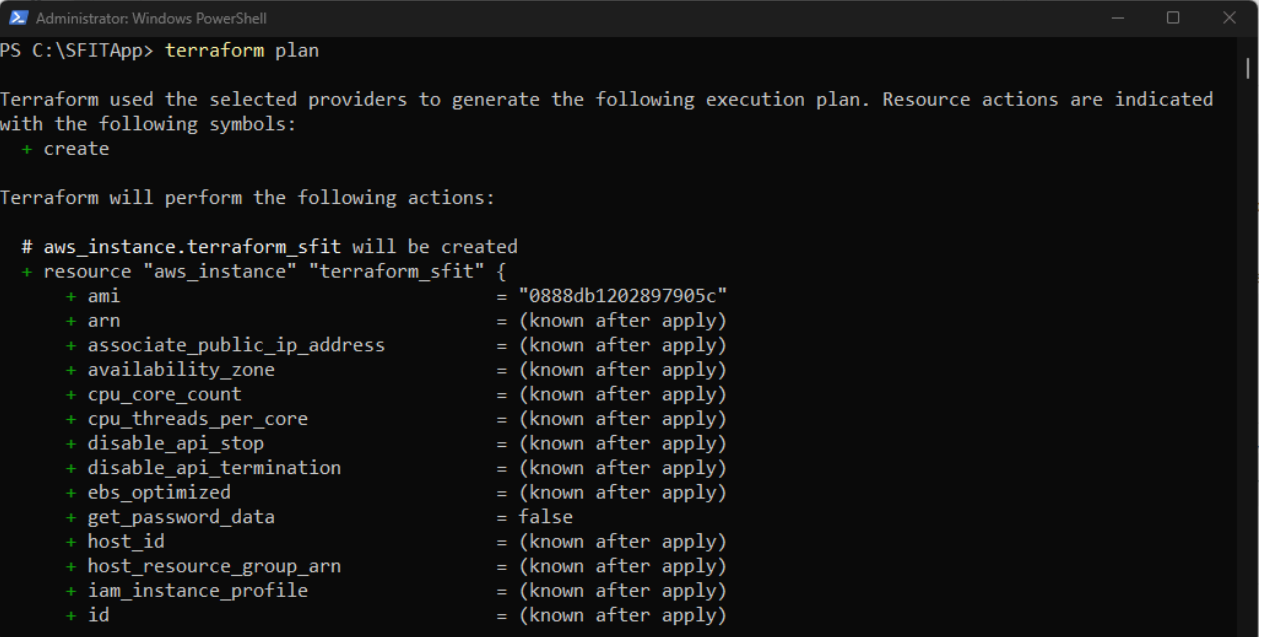
```
Administrator: Windows PowerShell
PS C:\SFITApp> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.67.0...
- Installed hashicorp/aws v5.67.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS C:\SFITApp>
```

Step 5: Run terraform plan



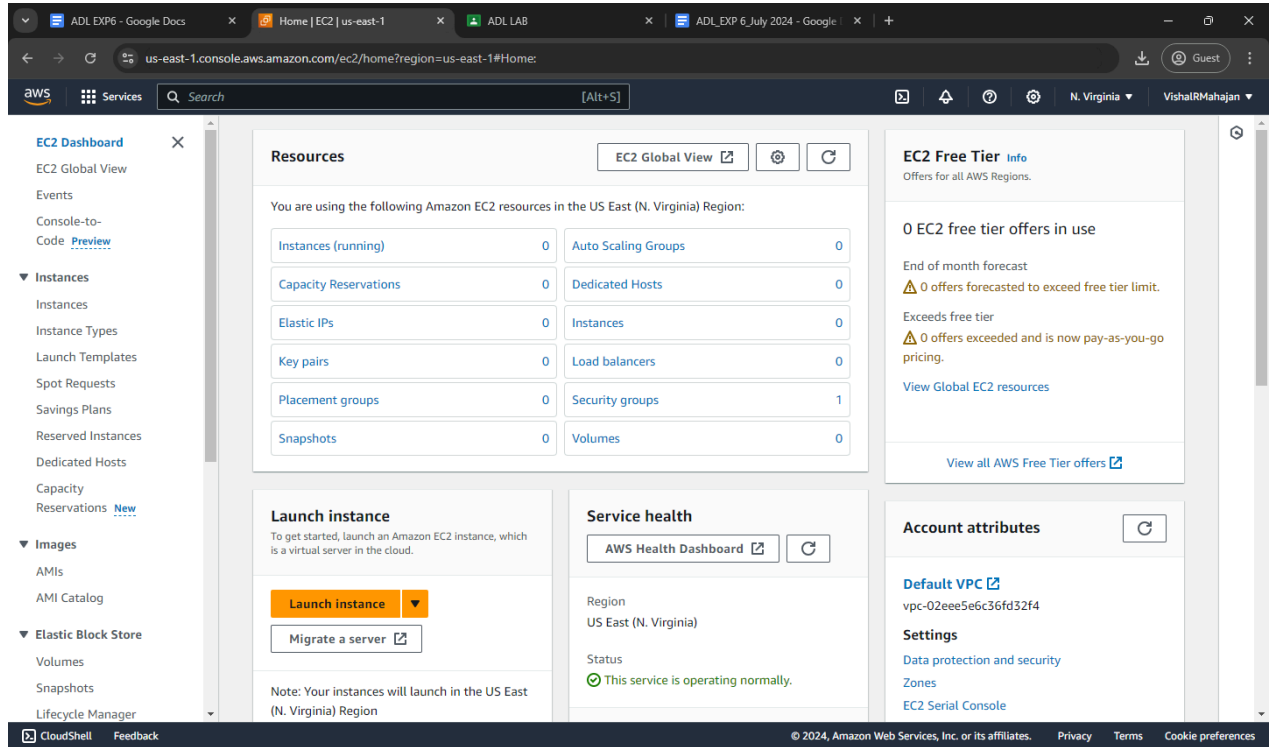
```
Administrator: Windows PowerShell
PS C:\SFITApp> terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated
with the following symbols:
  + create

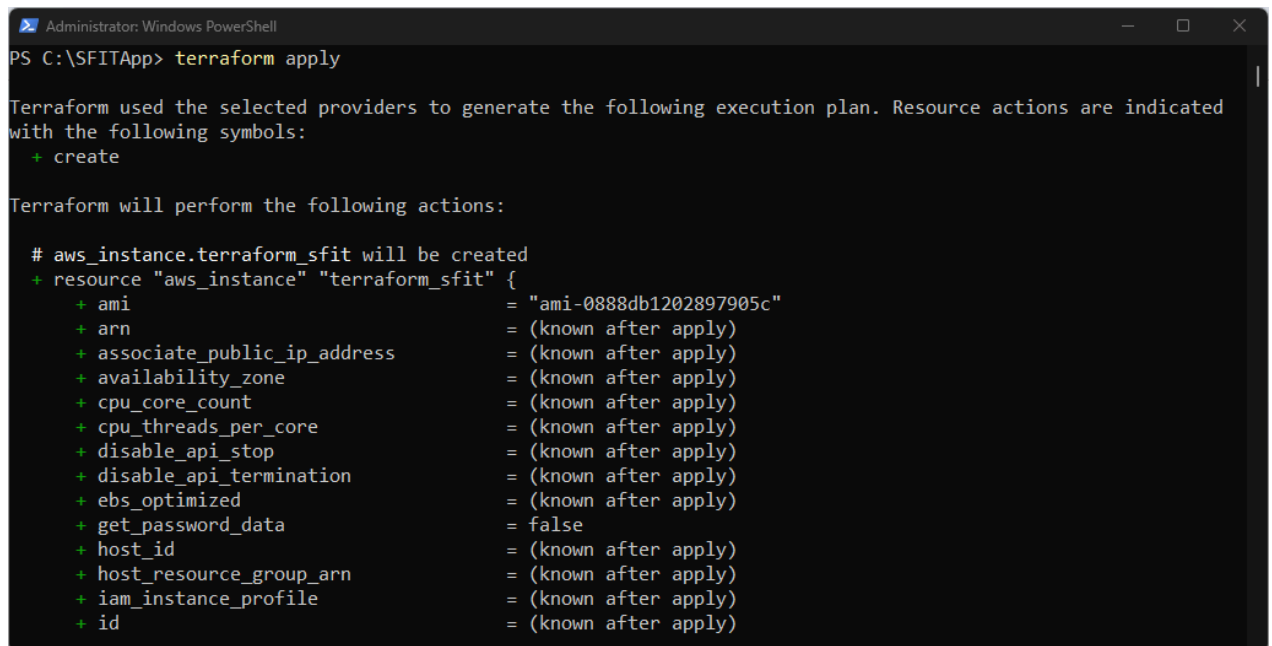
Terraform will perform the following actions:

# aws_instance.terraform_sfit will be created
+ resource "aws_instance" "terraform_sfit" {
  + ami                        = "0888db1202897905c"
  + arn                      = (known after apply)
  + associate_public_ip_address = (known after apply)
  + availability_zone         = (known after apply)
  + cpu_core_count            = (known after apply)
  + cpu_threads_per_core      = (known after apply)
  + disable_api_stop          = (known after apply)
  + disable_api_termination   = (known after apply)
  + ebs_optimized              = (known after apply)
  + get_password_data          = false
  + host_id                   = (known after apply)
  + host_resource_group_arn    = (known after apply)
  + iam_instance_profile       = (known after apply)
  + id                        = (known after apply)
```


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-06d03eb5cad2eab2]

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.

The screenshot shows the AWS Management Console for the US East (N. Virginia) Region. The left sidebar contains navigation links for EC2 Dashboard, EC2 Global View, Events, Console-to-Code, and various EC2 services. The main content area is divided into several sections:

- Resources:** A table showing the following EC2 resources in the US East (N. Virginia) Region:

Resource Type	Count
Instances (running)	1
Capacity Reservations	0
Elastic IPs	0
Key pairs	0
Placement groups	0
Snapshots	0
Auto Scaling Groups	0
Dedicated Hosts	0
Instances	1
Load balancers	0
Security groups	1
Volumes	1
- Launch instance:** A section with a "Launch instance" button and a "Migrate a server" button. It includes a note: "Note: Your instances will launch in the US East (N. Virginia) Region".
- Service health:** A section showing the status of the AWS service. It indicates that the service is operating normally.
- Account attributes:** A section showing account details, including the default VPC (vpc-02eee5e6c36fd32f4) and settings for data protection and security.

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

```

Administrator: Windows PowerShell

- volume_type           = "gp2" -> null
  # (1 unchanged attribute hidden)
}
}

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

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

Enter a value: yes

aws_instance.terraform_sfit: Destroying... [id=i-06d03eb5cad2eab2]
aws_instance.terraform_sfit: Still destroying... [id=i-06d03eb5cad2eab2, 10s elapsed]
aws_instance.terraform_sfit: Still destroying... [id=i-06d03eb5cad2eab2, 20s elapsed]
aws_instance.terraform_sfit: Still destroying... [id=i-06d03eb5cad2eab2, 30s elapsed]
aws_instance.terraform_sfit: Still destroying... [id=i-06d03eb5cad2eab2, 40s elapsed]
aws_instance.terraform_sfit: Still destroying... [id=i-06d03eb5cad2eab2, 50s elapsed]
aws_instance.terraform_sfit: Still destroying... [id=i-06d03eb5cad2eab2, 1m0s elapsed]
aws_instance.terraform_sfit: Destruction complete after 1m3s

Destroy complete! Resources: 1 destroyed.
PS C:\SFITApp>
  
```

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.