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](https://www.terraform.io/docs/language/index.html) 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

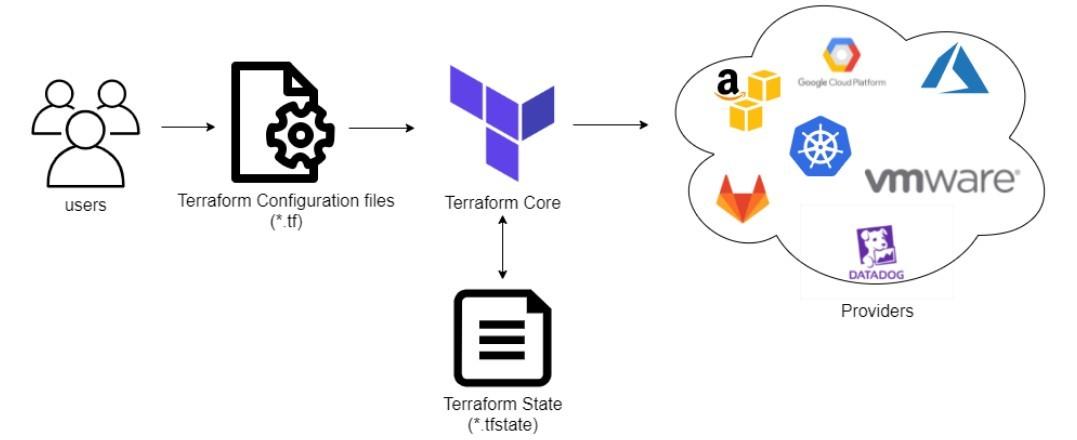
Terraform can apply complex change sets to your infrastructure with minimal human interaction. When you update configuration files, Terraform determines what changed and creates incremental execution plans that respect dependencies.

# 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.

Terraform Core Concepts:

Below are the core concepts/terminologies used in Terraform:

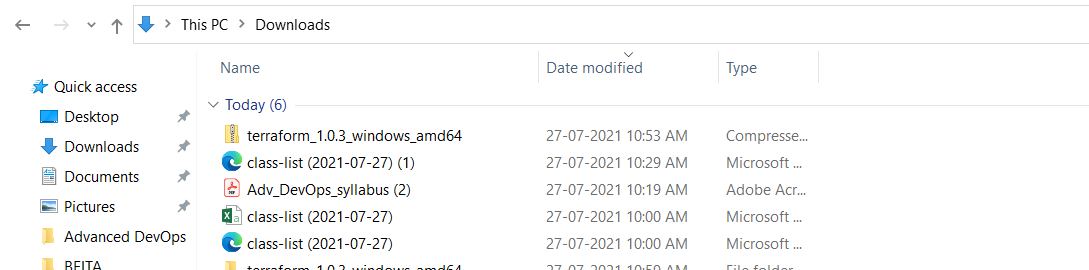
* **Variables:** Also used as input-variables, it is a key-value pair used by Terraform modules to allow customization.
* **Provider:** It is a plugin to interact with APIs of service and access its related resources.
* **Module:** It is a folder with Terraform templates where all the configurations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

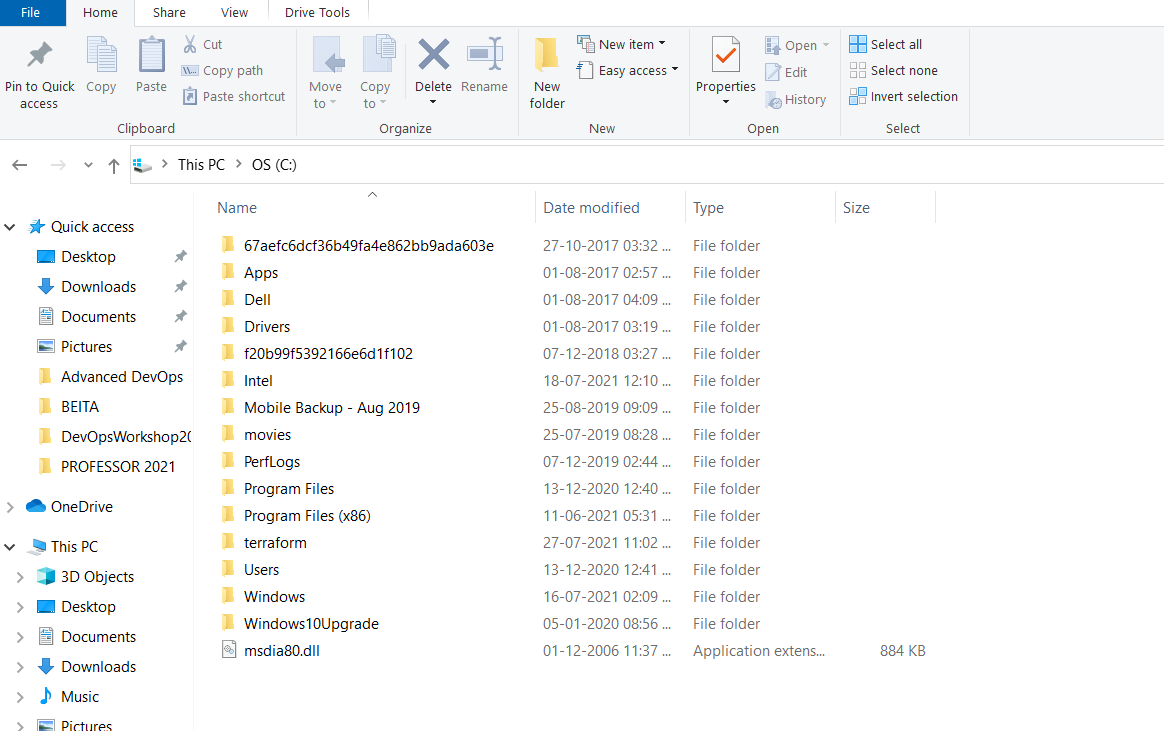
1. **Laboratory Exercise**

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

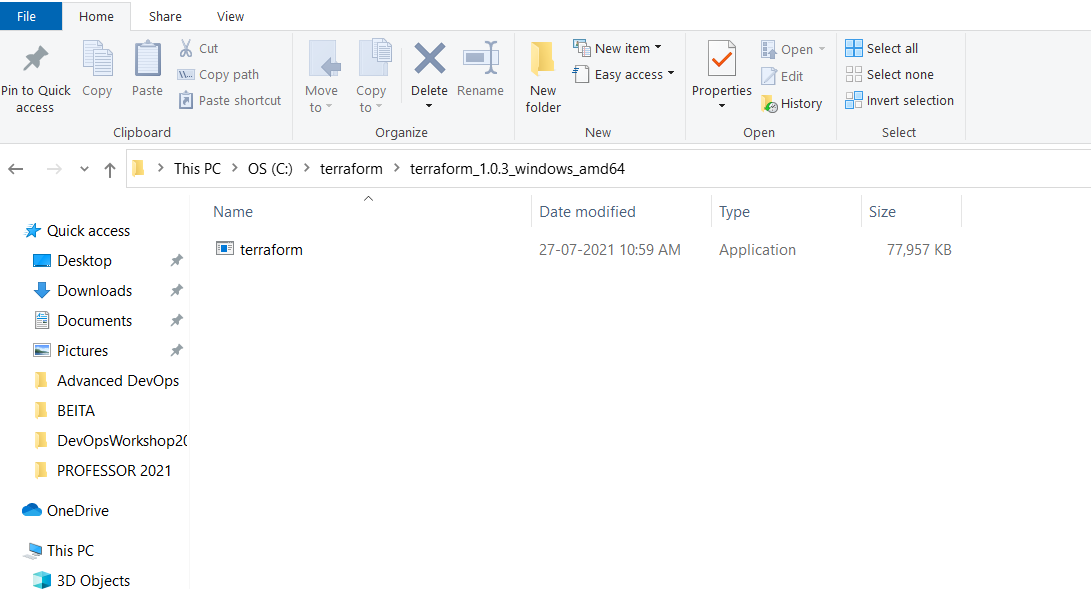


**Step 2: Download Terraform for Windows 64-bit / (32-bit).**

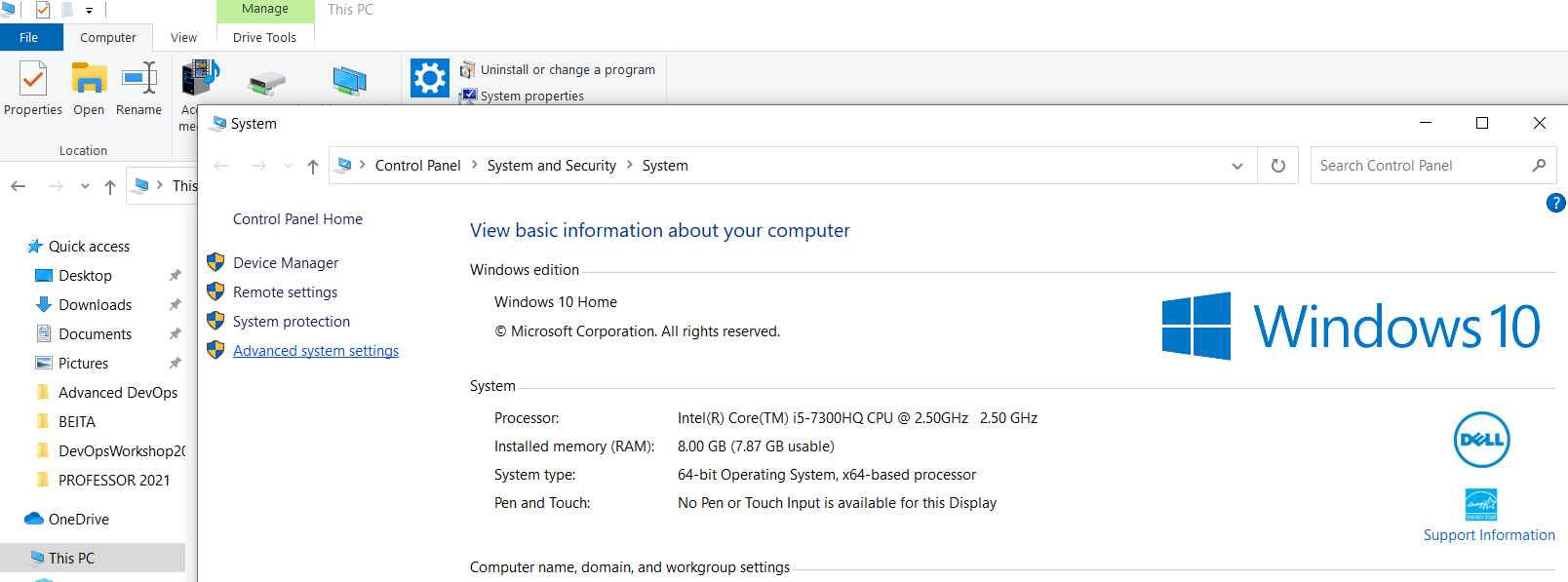


**Step 3: Create a folder ‘terraform’ in drive C .** 

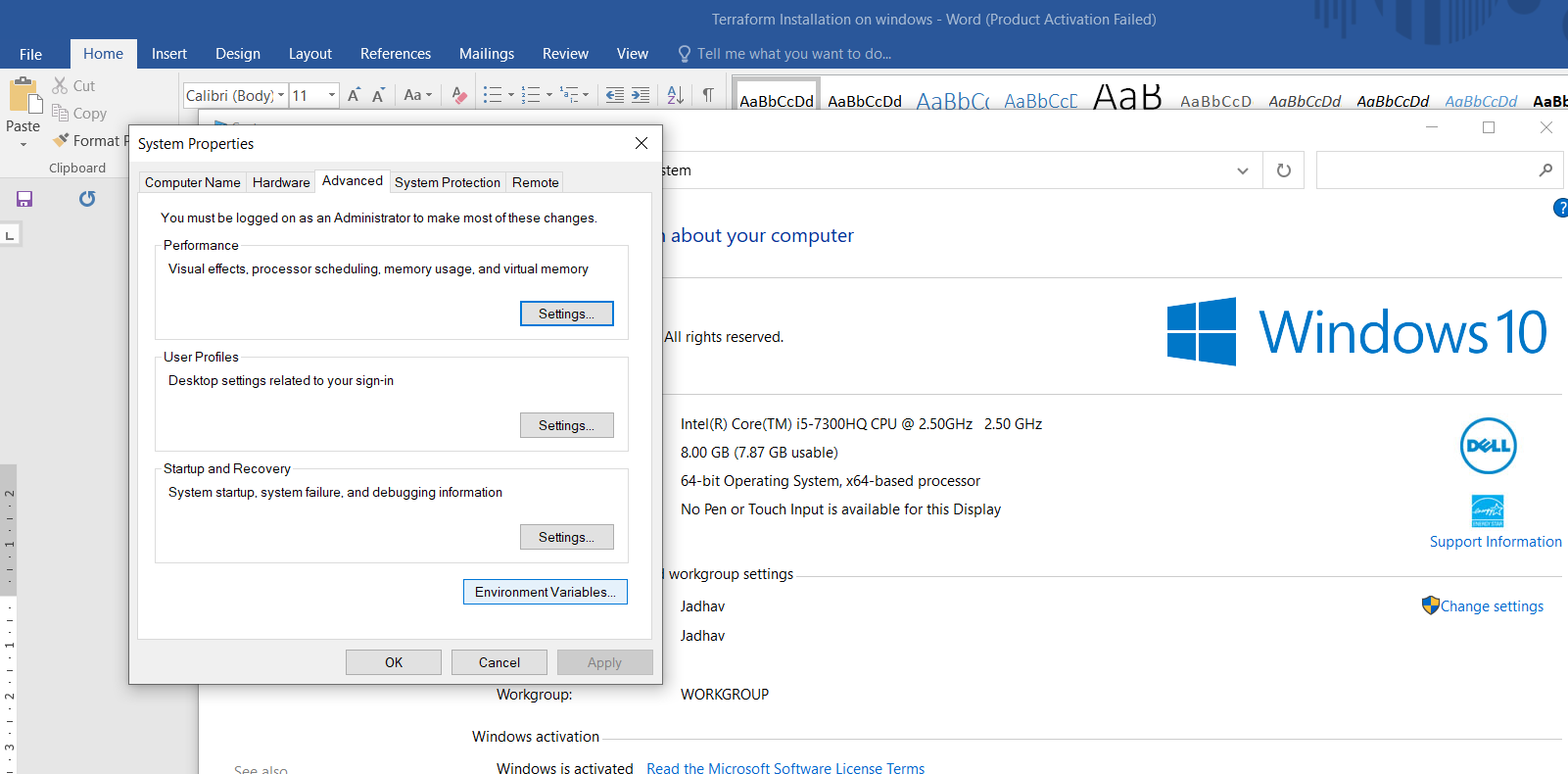
**Step 4 : Extract downloaded zip in to this c:/terraform folder**

****

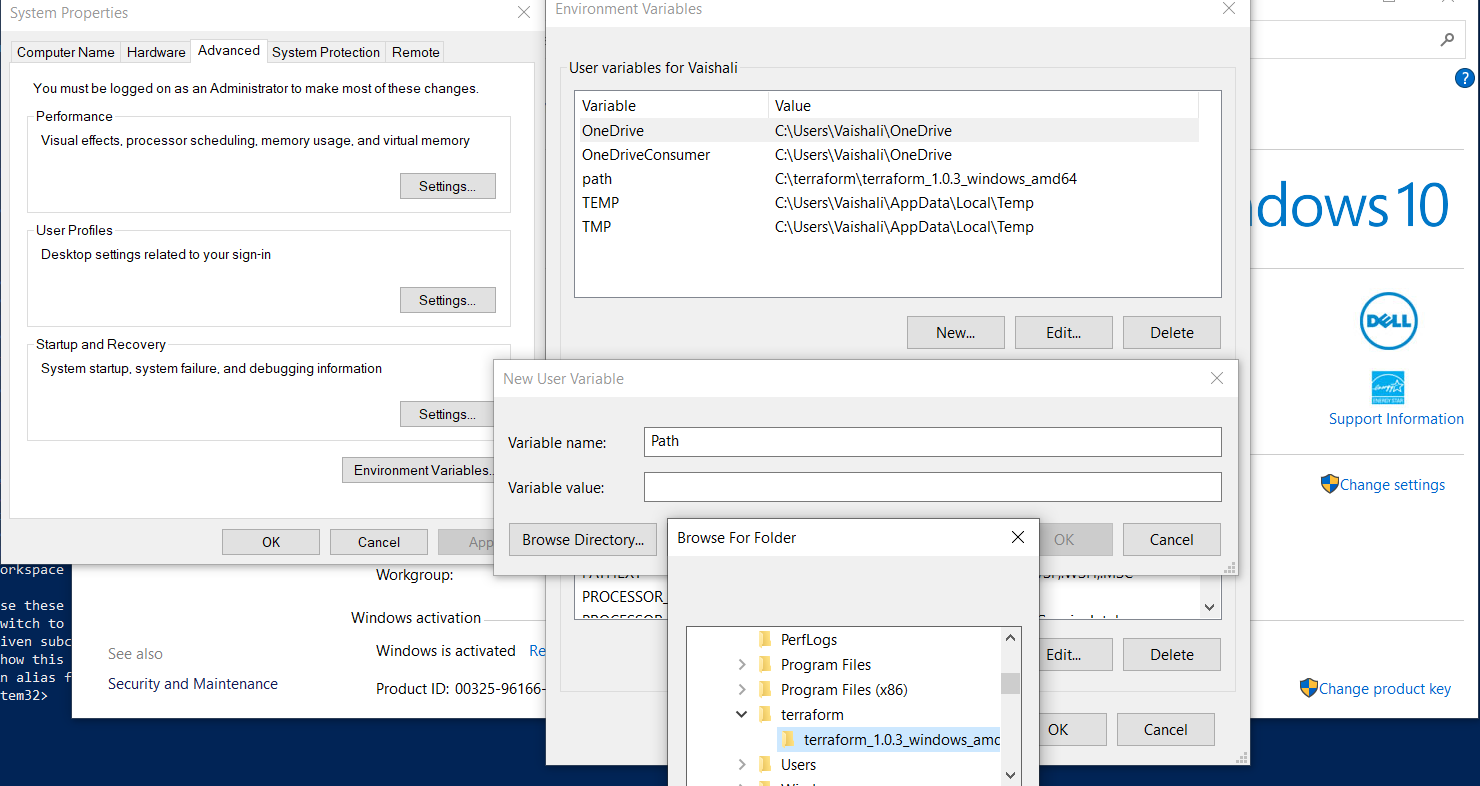
**Step 5: Now we need to set path for terraform. Go to My computer/ This PC, right click, select propoerties, go to advanced system settings.**



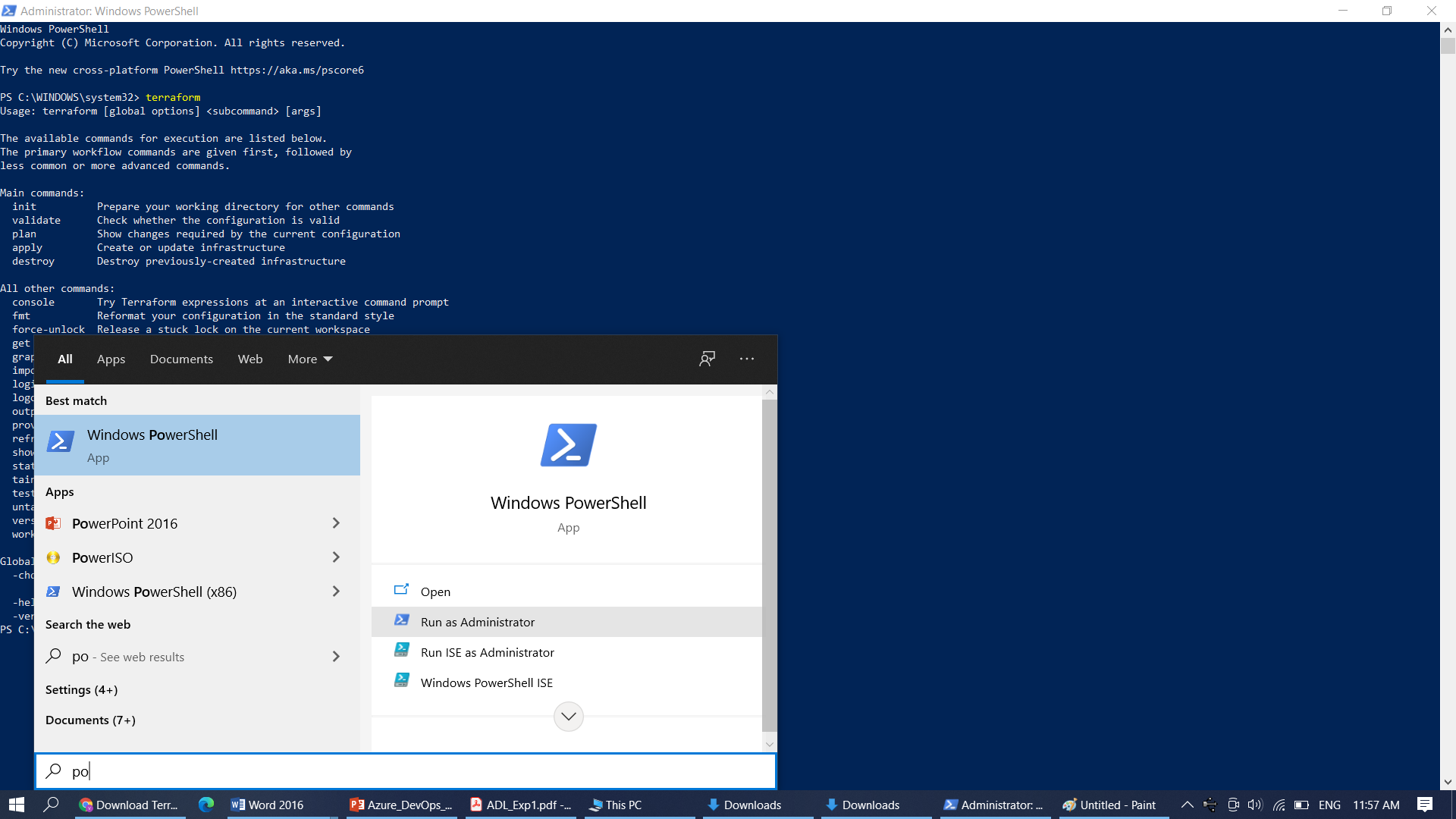
**Step 6: click on environment variable**



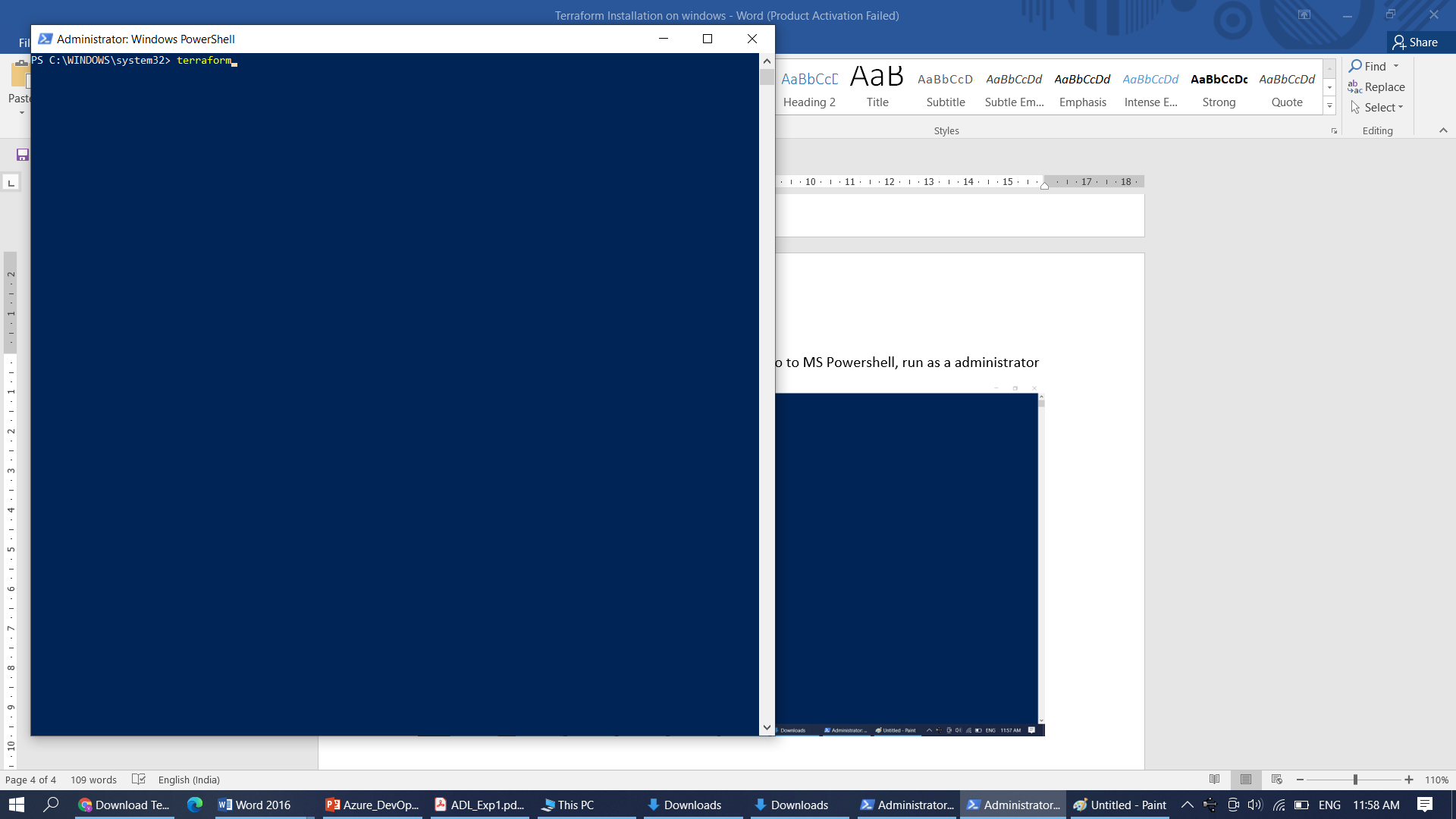
**Step 7: click on New, give variable name = Path, click on browse directory, select c:/terraforms/terra….exe, OK**



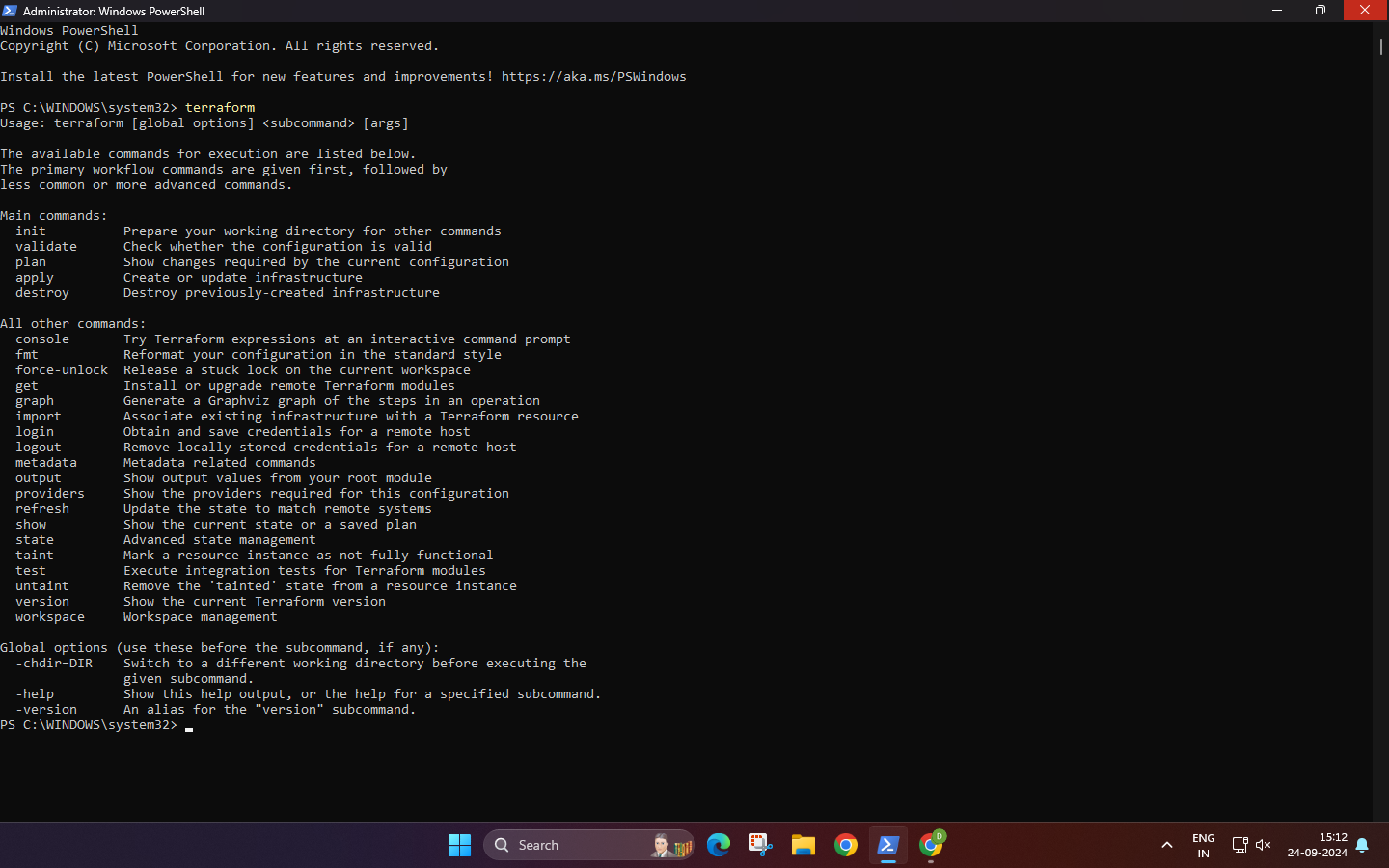
**Step 8: Cross verify terraform installed properly or not . go to MS Powershell, run as a administrator**



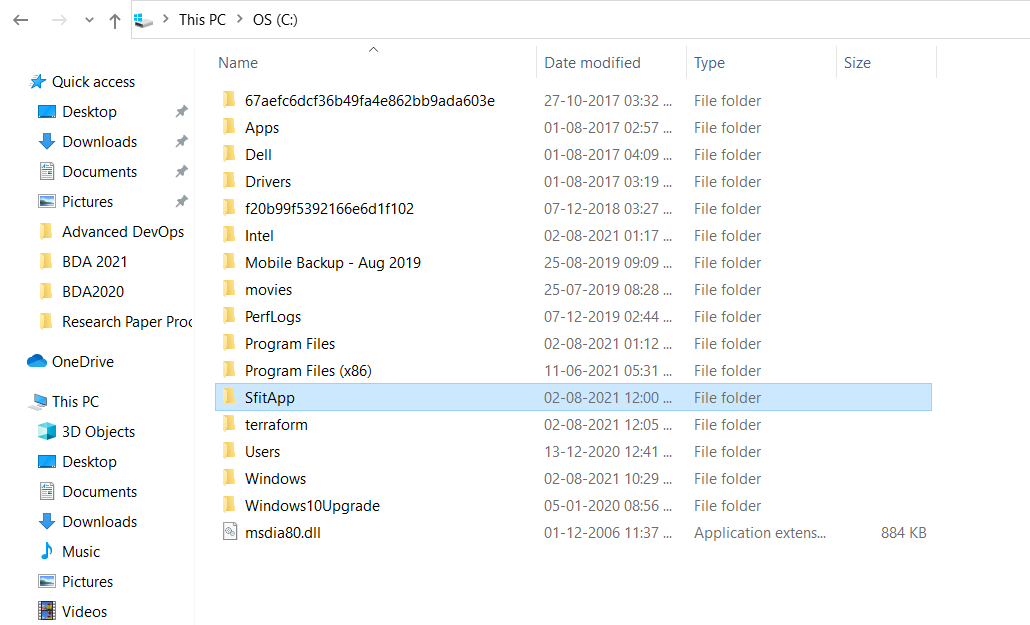
**Step 9: Type terraform**



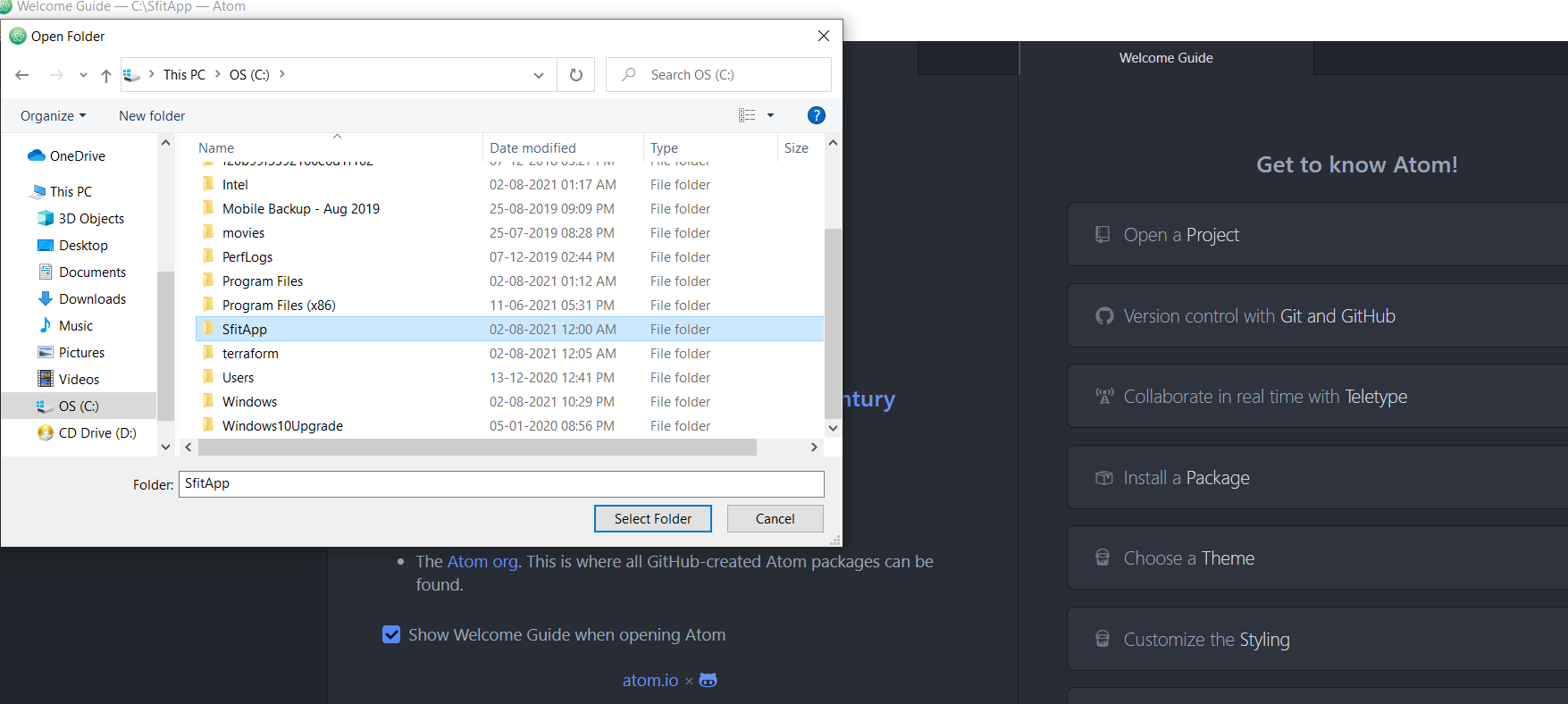
**Step 10: You will find init, validate, plan, apply and destroy options means you have installed terraform successfully.**



**Step 11: Create a folder c:\SfitApp**

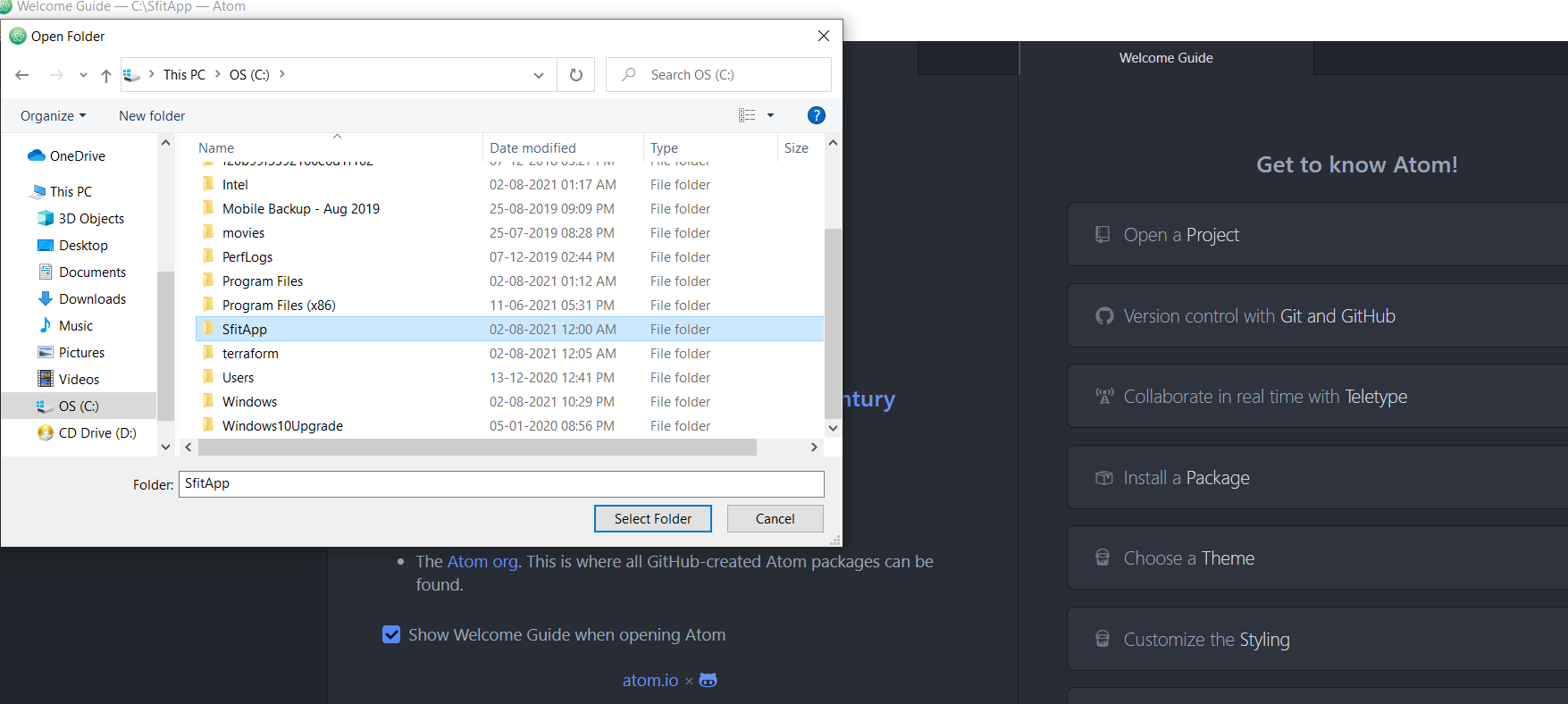


**Step 12 : Open Atom/VS CODE Editor …Open folder SfitApp**

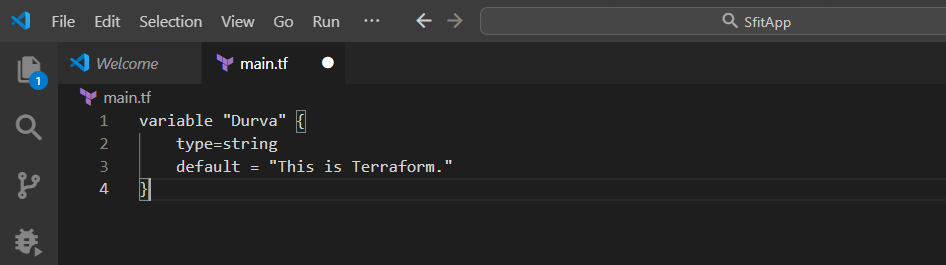


**Step 13: write main.tf file with input variables. The input variables, like the one above, use a couple of different types: string, list, map, and Boolean.**

Step 12 : Open Atom**/VS CODE** Editor …Open folder SfitApp

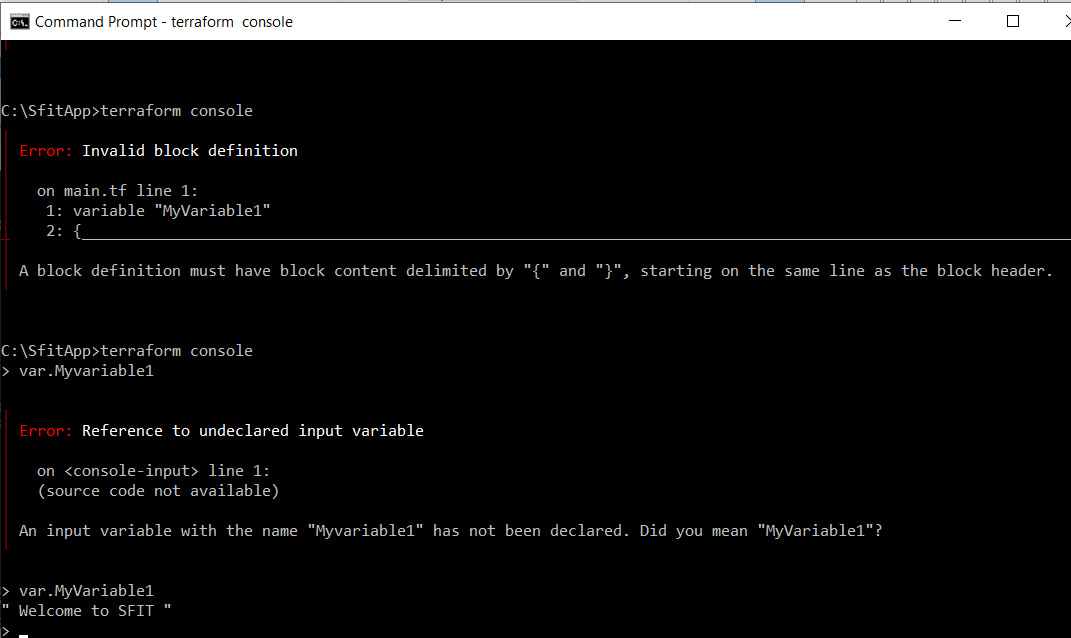


**Step 13: write main.tf file with input variables. The input variables, like the one above, use a couple of different types: string, list, map, and Boolean.**

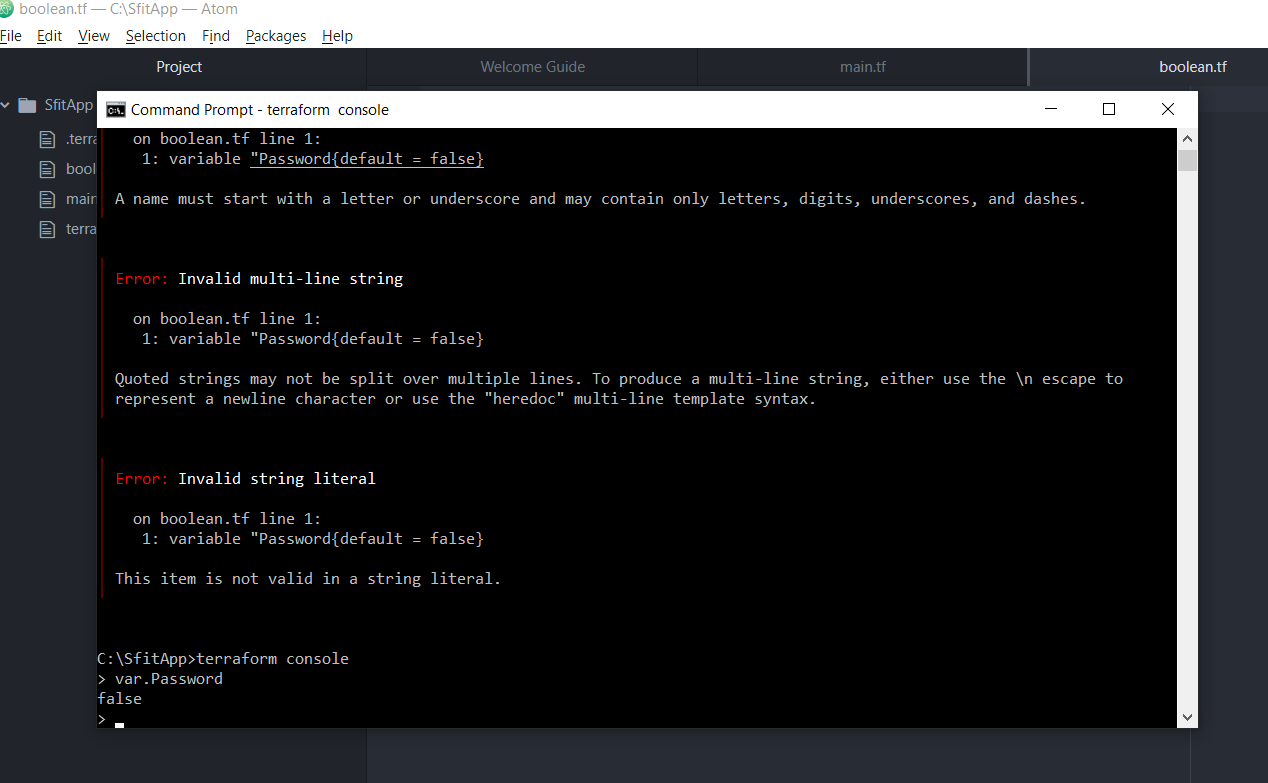
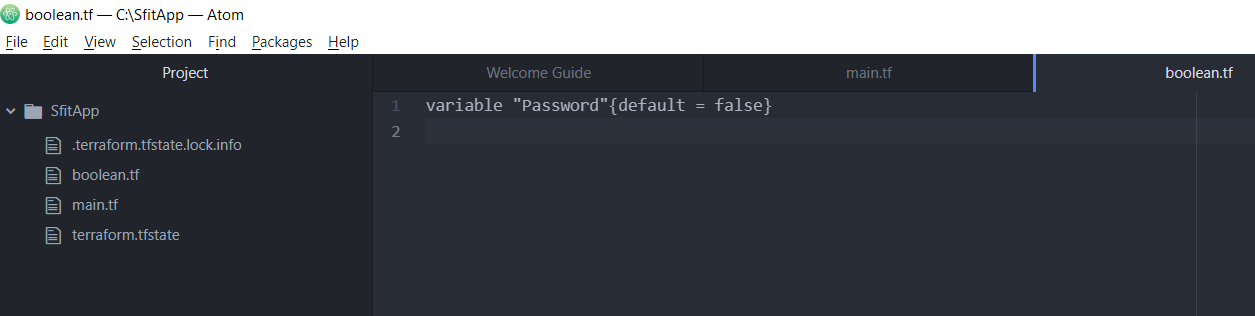


**Step 14: Check the output on command Prompt…Go to C:\SfitApp, Type Terraform Console**

You will get terraform prompt, run the .tf with var.MyVariable1, You will get welcome to SFIT msg.

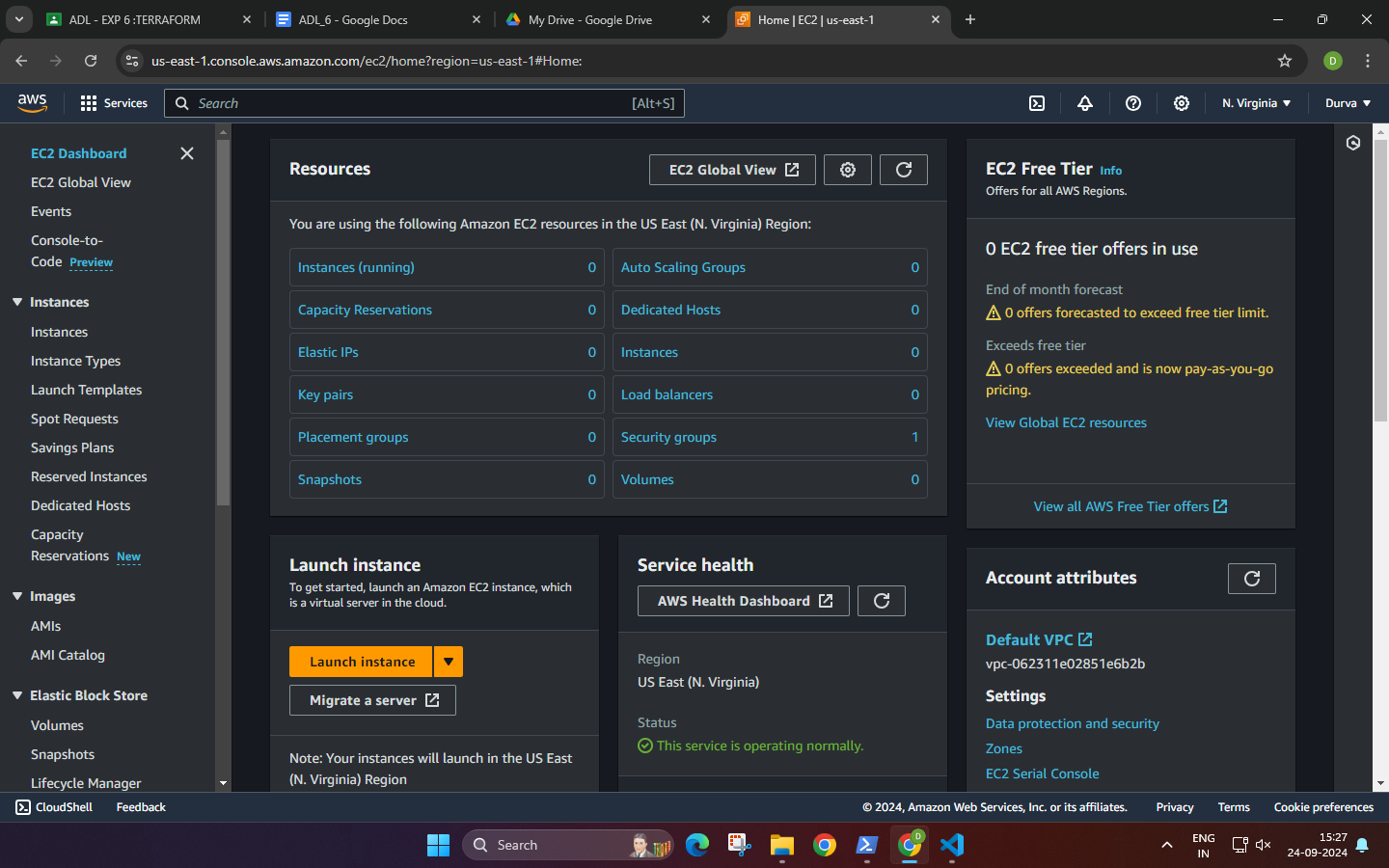


Step 15: try Boolean variable…Create Boolean.tf



**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

**IAM:**

**USERS:**

**ADD USER:**

Give console access

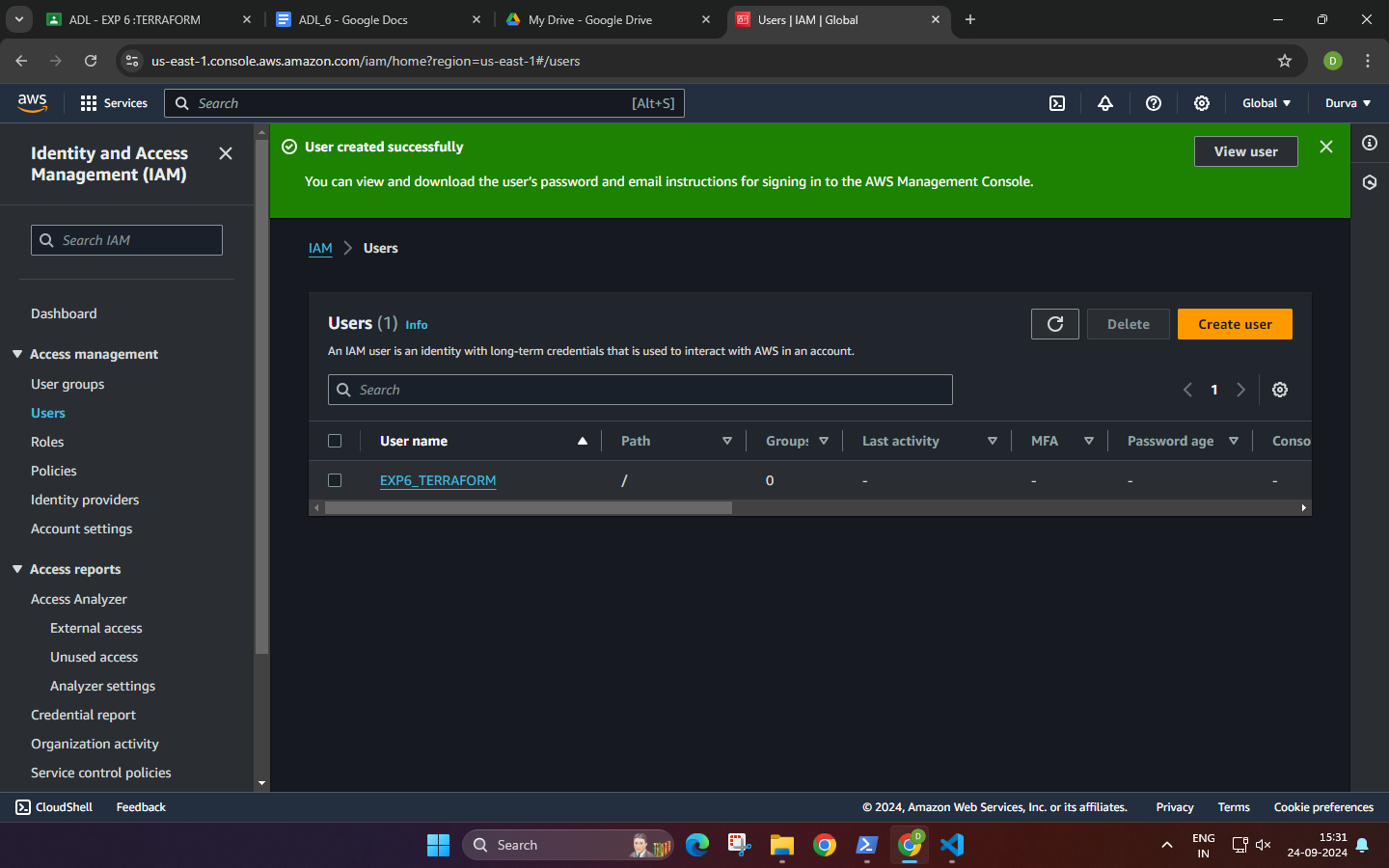
I want to create user

custom pwd

Attach policy directly : Administrator Access

Review:

Create user:



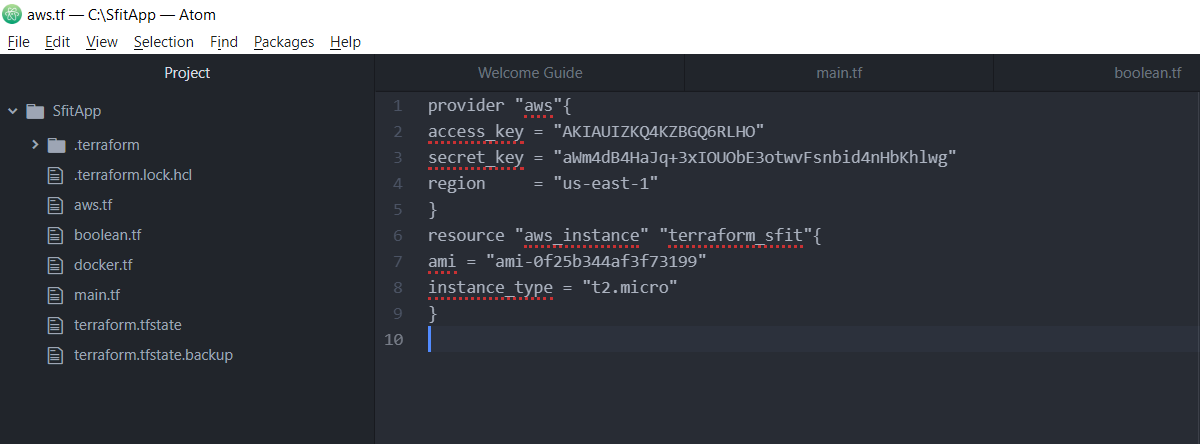
Clk on user name

Create access key

Command line interface

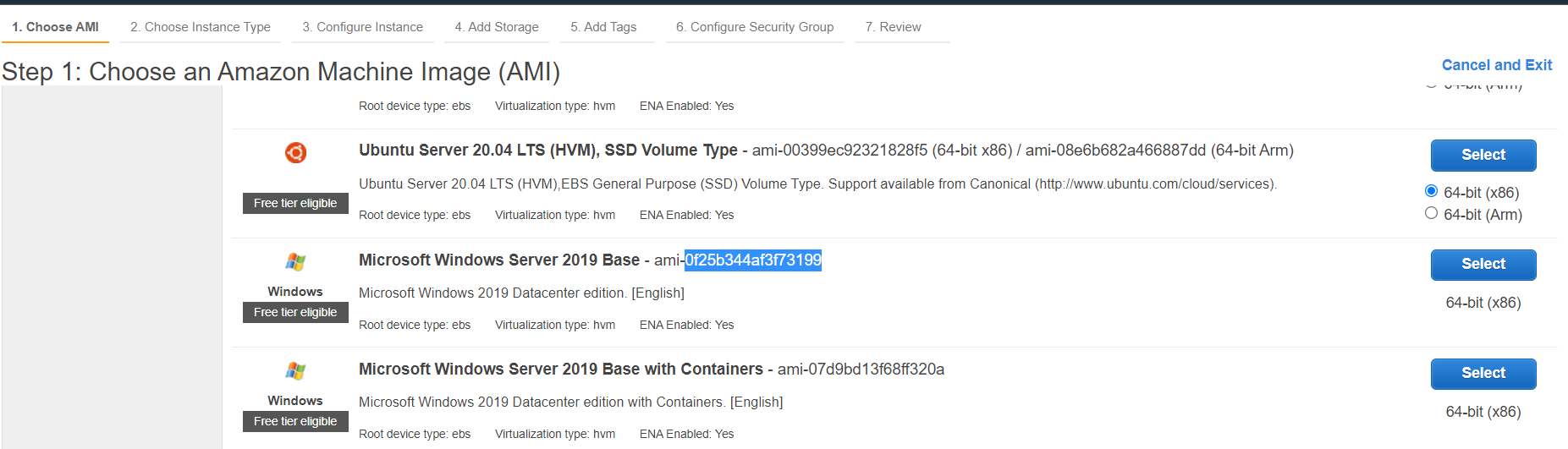
Create access key

**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-west-1"  }  resource "aws\_instance" "terraform sfit" {  ami = ""  instance\_type = "t2.micro"  } |
| --- |

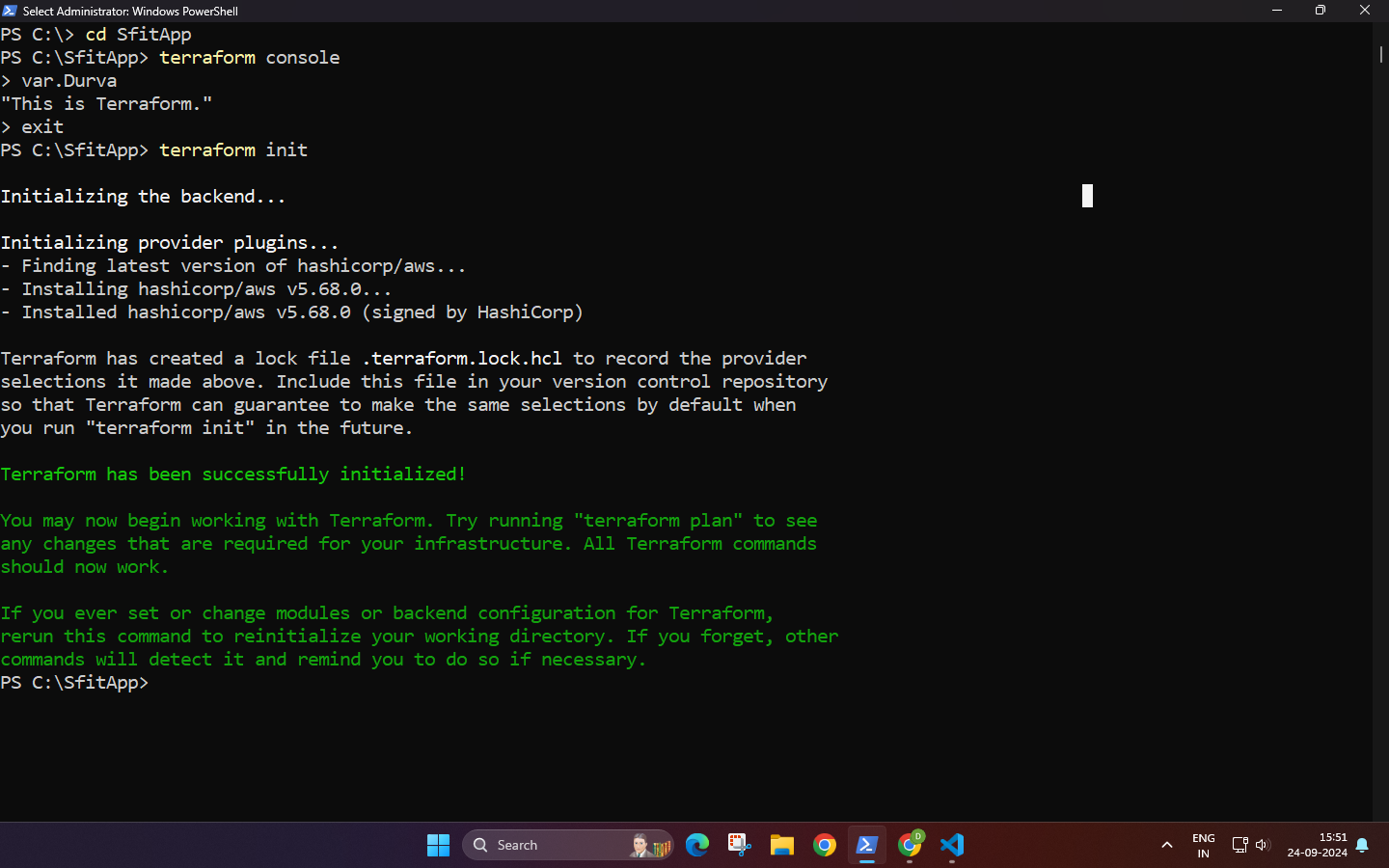
In Launch instance, you will get ami : amazon machine image



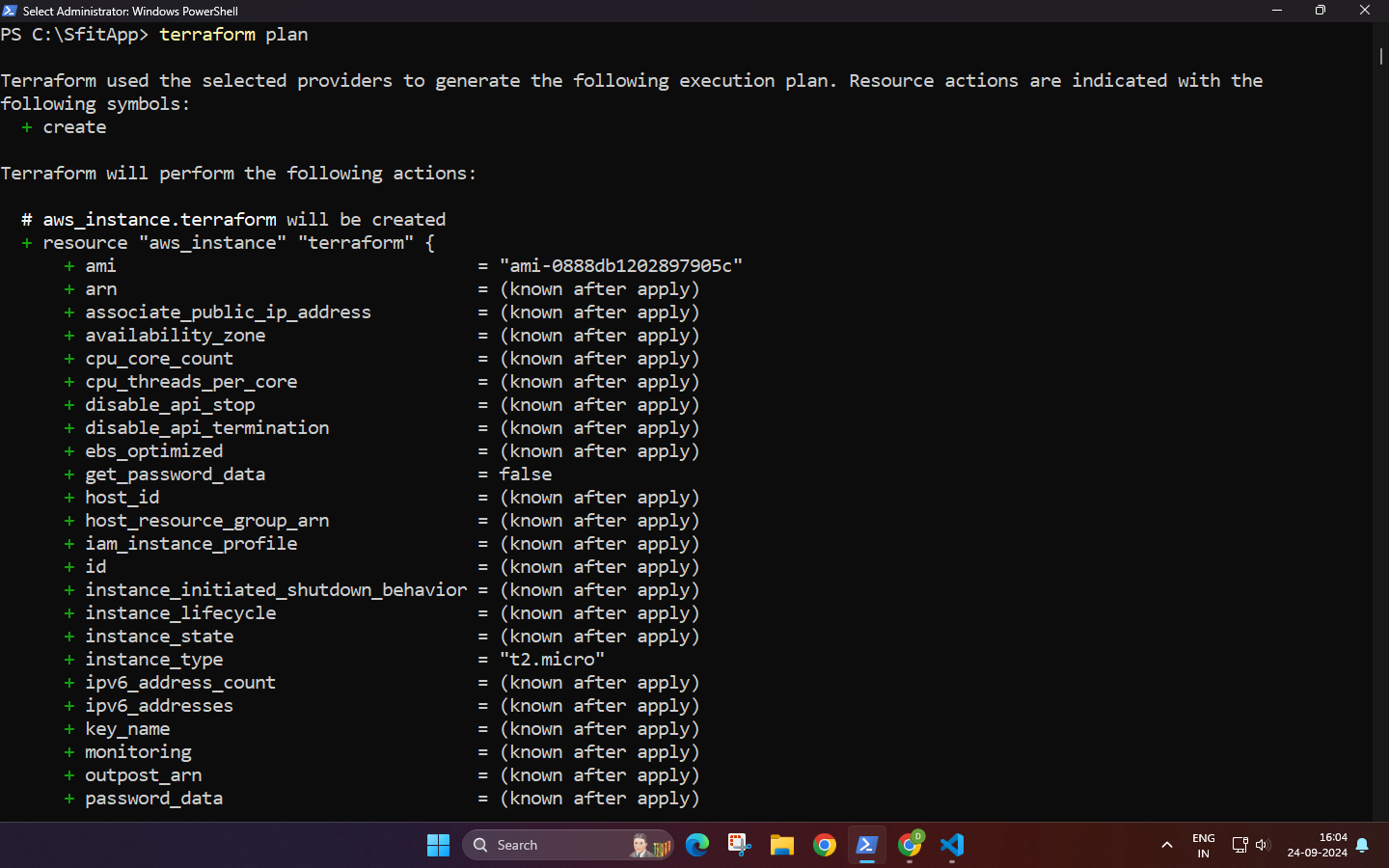
**For Instance type :** t2.micro is freely available

**Step 4:** Now initialize the terraform …type c:\SfitApp> terraform init

Terraform has been initialized successfully.



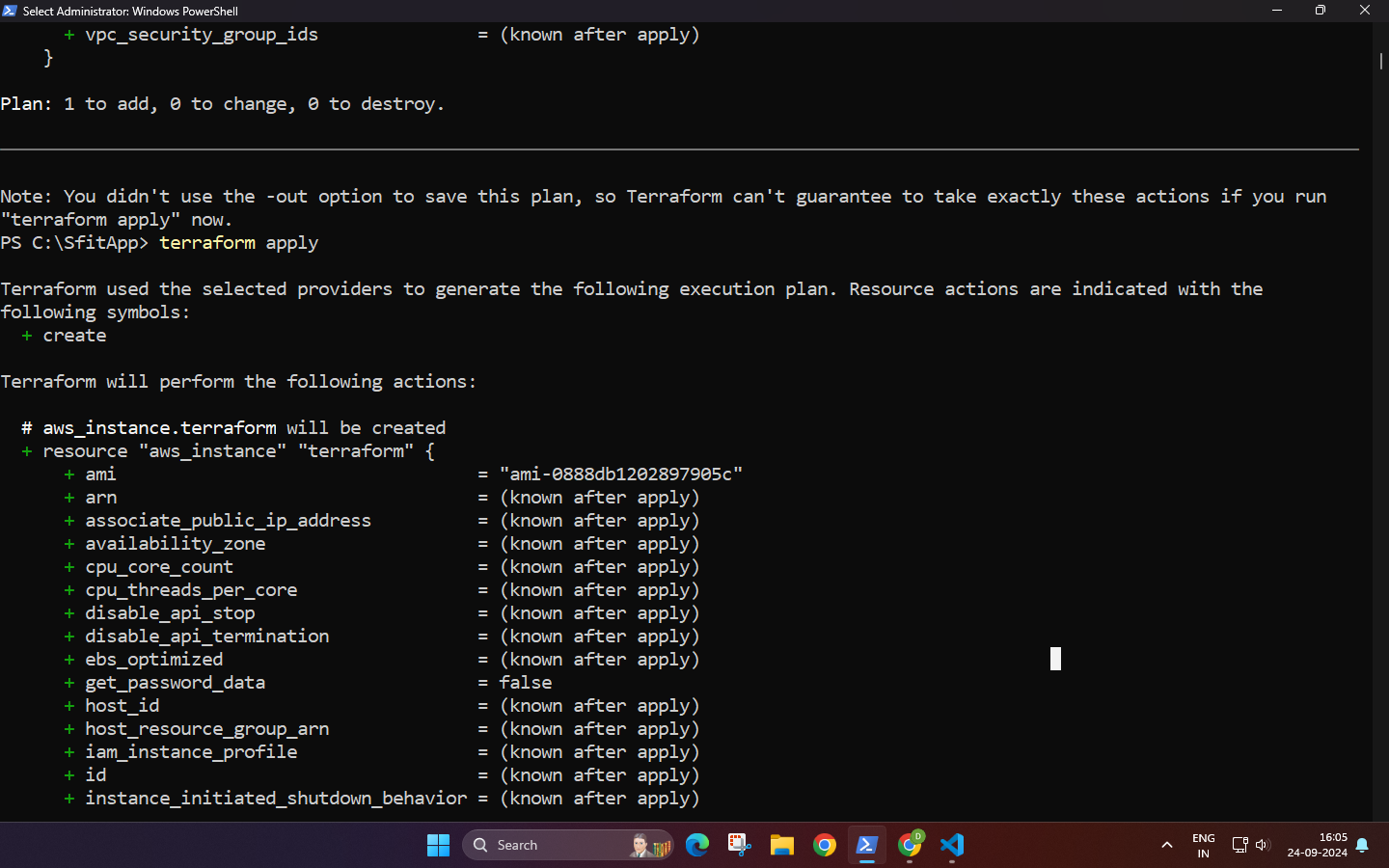
**Step 5:** c:\sfitApp>terraform plan



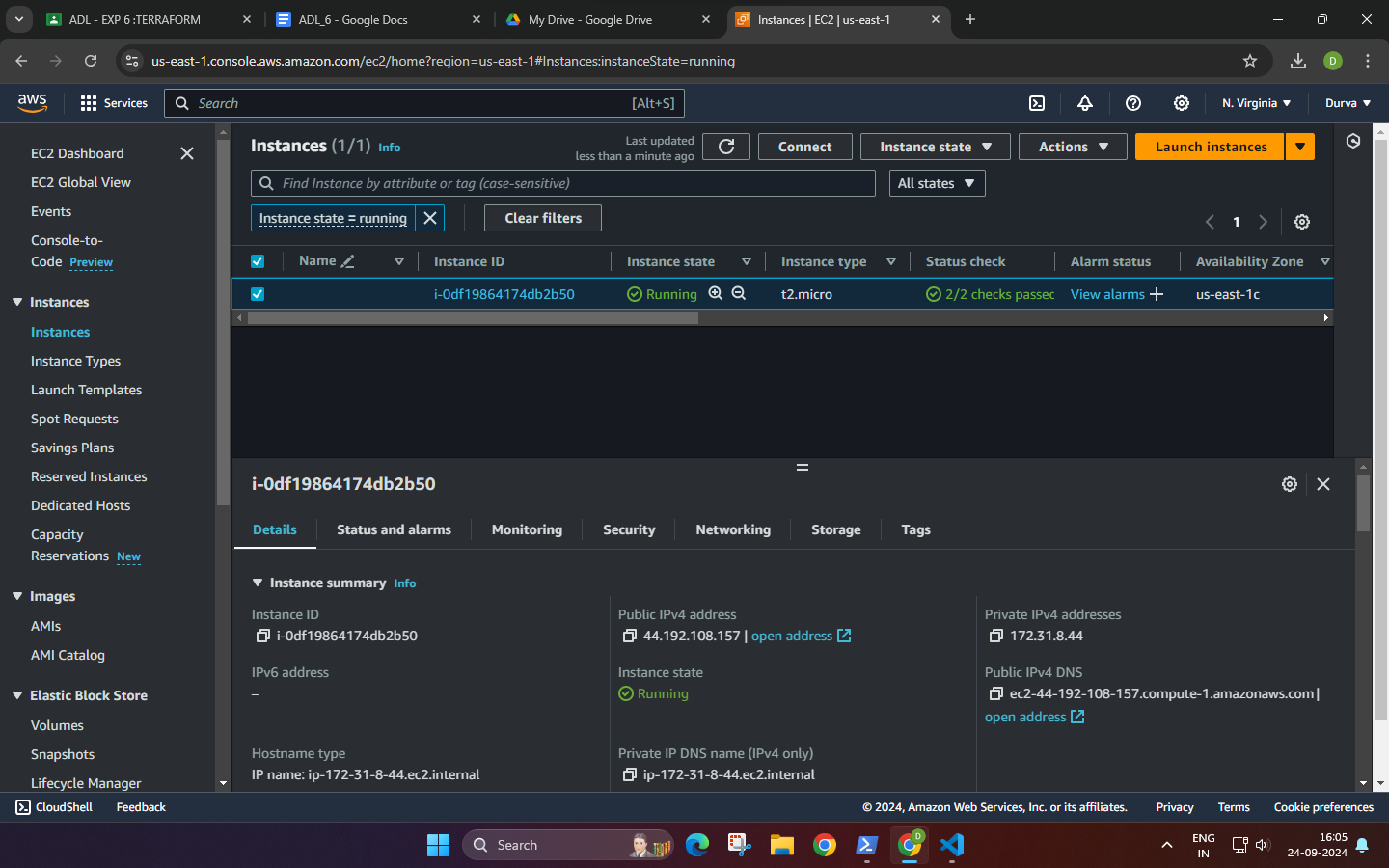
**Step 6:** Check the instance on Ec2 before terraform apply

Instance is not yet created.

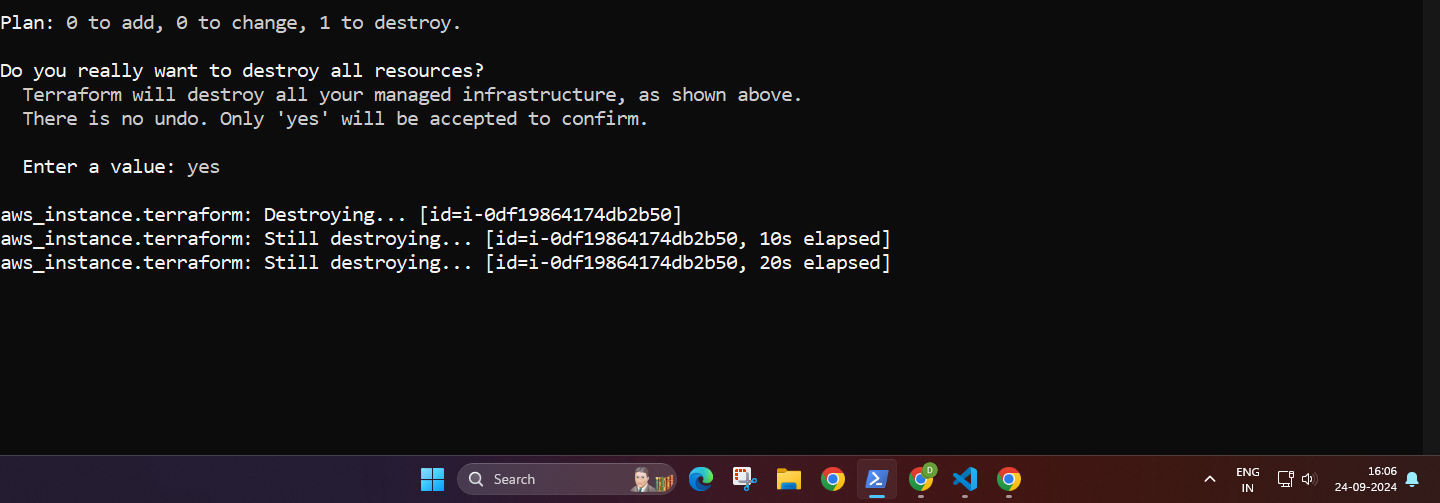
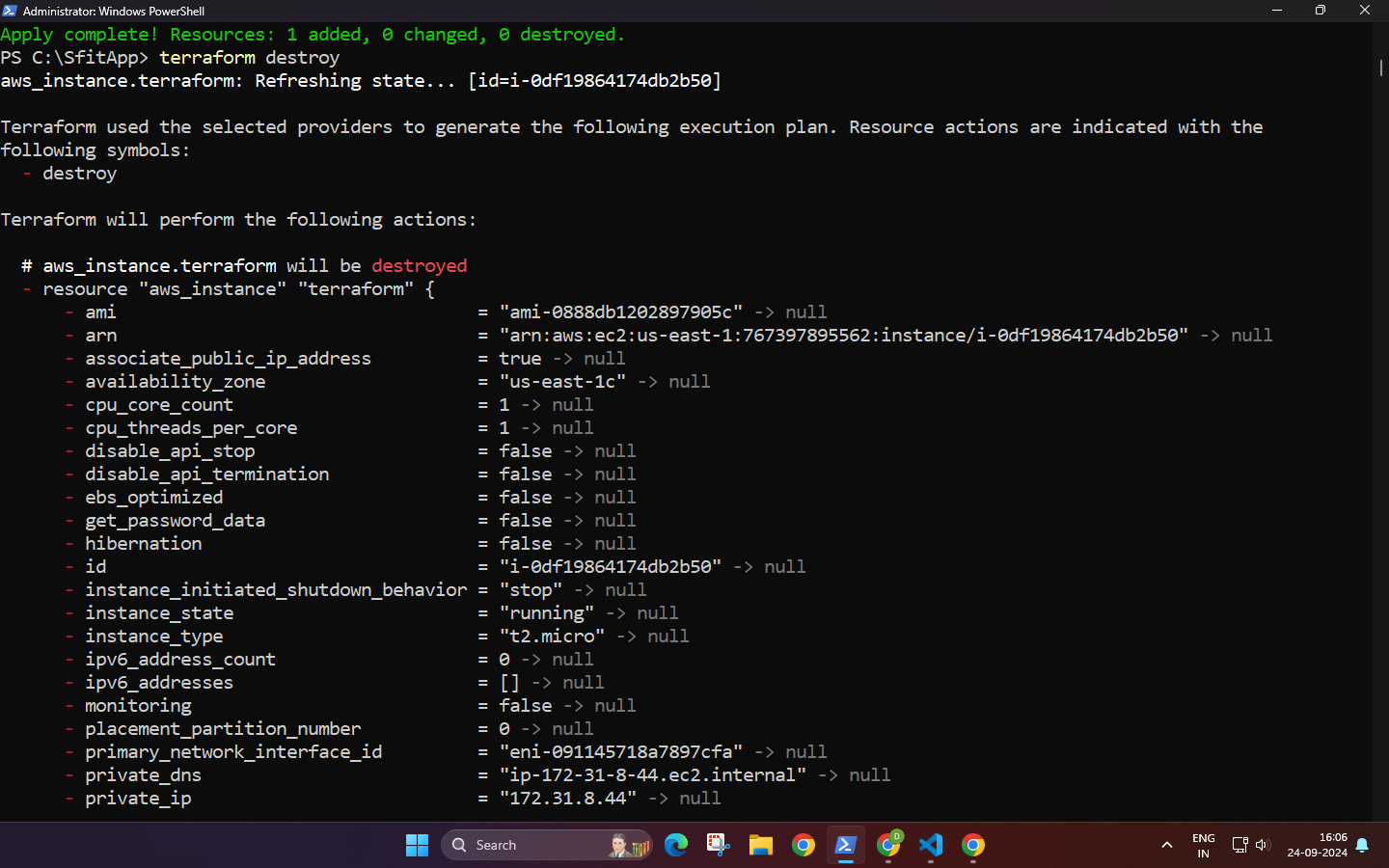
**Step 7:** Terraform apply



**Step 8:** Check terraform created instances on EC2…we have created 3 instances.



**Step 9:** Now destroy the instance from command prompt….c:\SfitApp> terraform destroy



1. **Post-Experiments Exercise**
2. **Extended Theory:**

* **Terraform Vs. Kubernetes (Soft copy)**

| **TERRAFORM** | **KUBERNETES** |
| --- | --- |
| Infrastructure provisioning and management | Container orchestration and management |
| Cloud infrastructure (servers, networks, etc.) | Application deployment and scaling in containers |
| Uses state files to track infrastructure | Tracks state of containerized applications |
| Declarative with HCL (HashiCorp Configuration Language) | Declarative with YAML or JSON for manifest files |
| Supports multiple cloud providers (AWS, Azure, GCP) | Platform-agnostic but designed for containerized apps |
| Manual or scripted scaling of infrastructure | Autoscaling for containers and services |
| Manages long-lived infrastructure | Typically manages short-lived, stateless containers |
| No built-in orchestration | Built-in orchestration for containers |
| Manages full infrastructure lifecycle (creation, deletion) | Manages lifecycle of containerized apps |
| Integrates with external services (DNS, monitoring, etc.) | Relies on external services but focuses on application management within clusters |

* **Terraform Vs. Ansible (Soft copy)**

| **TERRAFORM** | **ANSIBLE** |
| --- | --- |
| Focuses on infrastructure provisioning and management | Focuses on configuration management and application deployment |
| Uses declarative syntax (HCL) to define infrastructure | Uses procedural YAML playbooks for automation tasks |
| Ideal for provisioning cloud resources (VMs, networks) | Ideal for configuring servers and automating software deployments |
| Manages state through a state file | Stateless, no need for a state file |
| Primarily used for infrastructure as code (IaC) | Used for configuration management and automation |
| Has strong support for multi-cloud environments | Supports multi-cloud but better suited for configuration tasks |
| Requires explicit commands to apply changes | Can perform idempotent changes without tracking state |
| Best suited for infrastructure lifecycle management | Best suited for system configurations and task automation |
| More suitable for large-scale infrastructure deployments | Suitable for both infrastructure management and small-scale automation tasks |
| Slower to apply changes as it plans and tracks state | Faster for configuration changes and runtime execution |

* 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. **GitHub:** Terraform can pull modules and configuration files from GitHub repositories, both public and private, using OAuth tokens or SSH keys for authentication.
3. **GitLab:** Terraform integrates with GitLab to fetch configuration files and modules from repositories, supporting both self-managed and GitLab.com instances.
4. **Bitbucket:** Terraform can access repositories on Bitbucket Cloud or Bitbucket Server, allowing teams to store infrastructure configurations and collaborate on changes.
5. **Azure Repos:** Azure Repos, part of Azure DevOps, is supported by Terraform for fetching infrastructure code, enabling integration with Microsoft’s cloud-based version control system.
6. **AWS CodeCommit:** Terraform supports pulling code from AWS CodeCommit, which is a managed source control service in AWS, suitable for storing and versioning infrastructure code.
7. **Gitea:** Terraform can connect to Gitea, a lightweight self-hosted Git service, for accessing and managing Terraform modules.
8. **Generic Git Repositories:** Terraform supports generic Git repositories, allowing users to fetch configurations from any Git-compatible source, including self-hosted instances.
9. **Name some major competitors of Terraform.**

ANS**:** Some major competitors of **Terraform** in the infrastructure provisioning and management space:

1. **AWS CloudFormation:** A native Infrastructure as Code (IaC) tool for provisioning and managing AWS resources using JSON or YAML templates.
2. **Pulumi:** A modern IaC tool that allows infrastructure provisioning using familiar programming languages like Python, TypeScript, and Go.
3. **Ansible:** A configuration management and automation tool that can also handle infrastructure provisioning using playbooks in YAML.
4. **Chef:** A configuration management tool that automates the deployment and management of infrastructure, commonly used for provisioning.
5. **SaltStack:** A powerful automation tool that manages infrastructure at scale, offering features for provisioning, configuration management, and orchestration.
6. **Google Cloud Deployment Manager:** Google Cloud’s native tool for provisioning resources using YAML, Python, or Jinja2 templates.
7. **Red Hat Ansible Tower:** The enterprise version of Ansible with additional features for automating infrastructure provisioning and managing complex workflows.

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

1. **Multi-Cloud Flexibility**: Supports various cloud providers, avoiding vendor lock-in.
2. **Infrastructure as Code (IaC)**: Defines infrastructure using code for easy management and versioning.
3. **Declarative Syntax**: Describes desired infrastructure state without detailing the steps to achieve it.
4. **Strong Community Support**: Large community with extensive resources and pre-built modules.
5. **Automation**: Automates provisioning, speeding up processes and reducing manual errors.
6. **State Management**: Tracks the current state of infrastructure for incremental changes.
7. **Collaboration-Friendly**: Facilitates team collaboration through version control for changes.
8. **Reusable Modules**: Promotes best practices with reusable configurations.
9. **Easy Integration**: Works well with other DevOps tools and CI/CD pipelines.
10. **Cost Efficiency**: Helps avoid over-provisioning, leading to better cost management.

**C. Conclusion: in hand**

1. Write what was performed in the experiment
2. Mention a few applications of what was studied.
3. Write the significance of the studied topic
4. **References:**
5. <https://www.ibm.com/cloud/learn/terraform#toc-terraform--OoC-5III>
6. <https://www.simplilearn.com/terraform-interview-questions-and-answers-article>
7. <https://aws.amazon.com/microservices/>
8. <https://www.monkeyvault.net/docker-vs-virtualization/>