**Experiment No 6**

**Name: Arya Manoj Madhavi**

**Div: D15B**

**Roll No.:31**

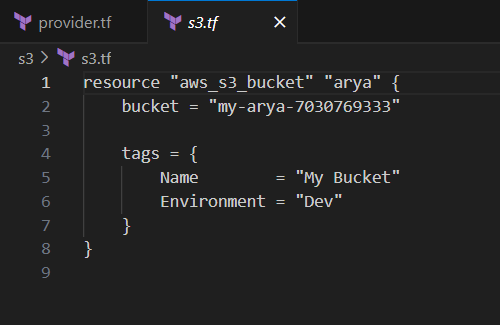
**Batch B**

**AIM: Creating S3 Bucket using terraform**

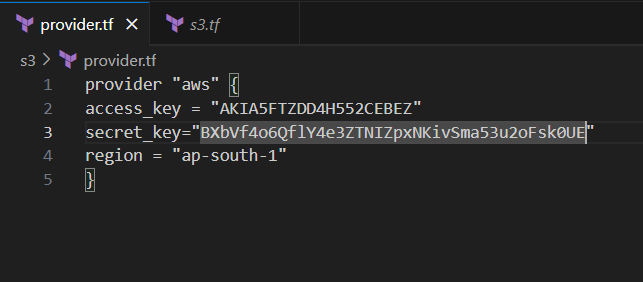
**Prerequisite:**

1. **Install Atom Editor for Writing the Scripts from** [**https://atom.io/**](https://atom.io/)
2. **Must have an AWS Access Key ID and Secret Access Key**

**Step 1: Write a Terraform Script in Atom for creating S3 Bucket on Amazon AWS**



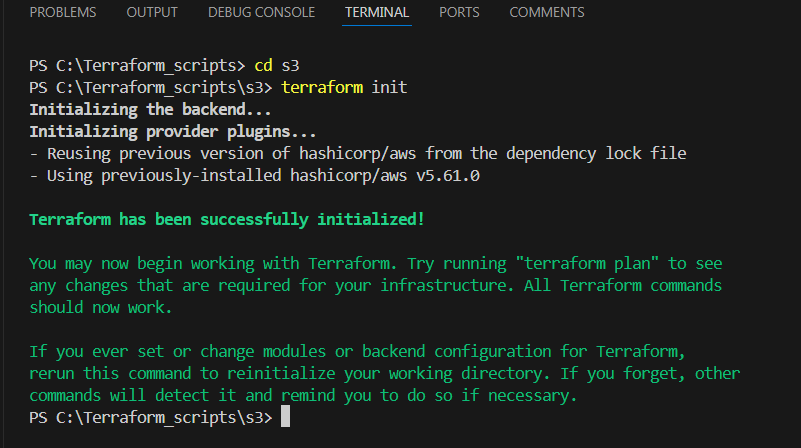
**Create a new provider.tf file and write the following contents into it.**



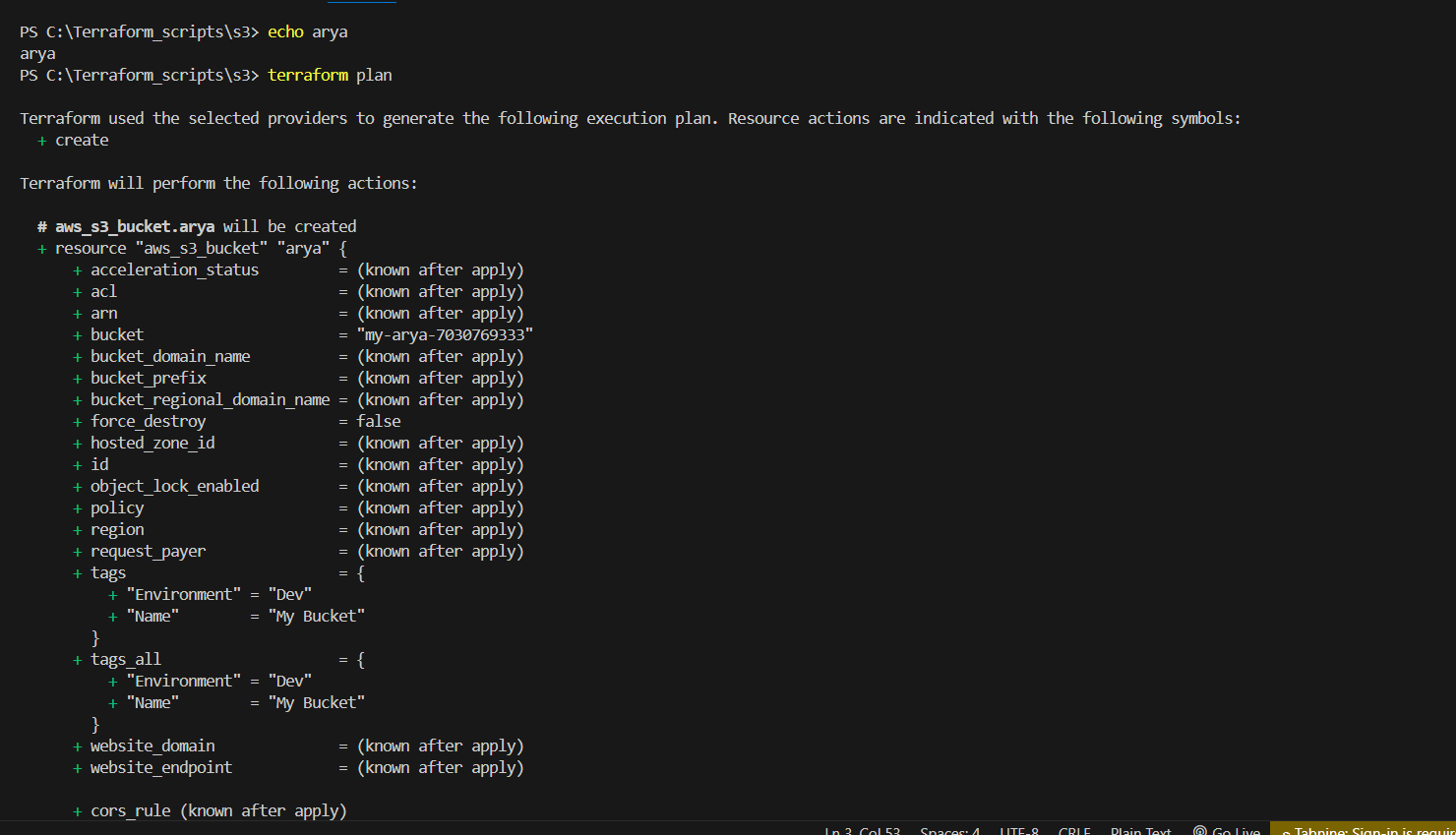
**Save both the files in same directory Terraform\_Scripts/S3**

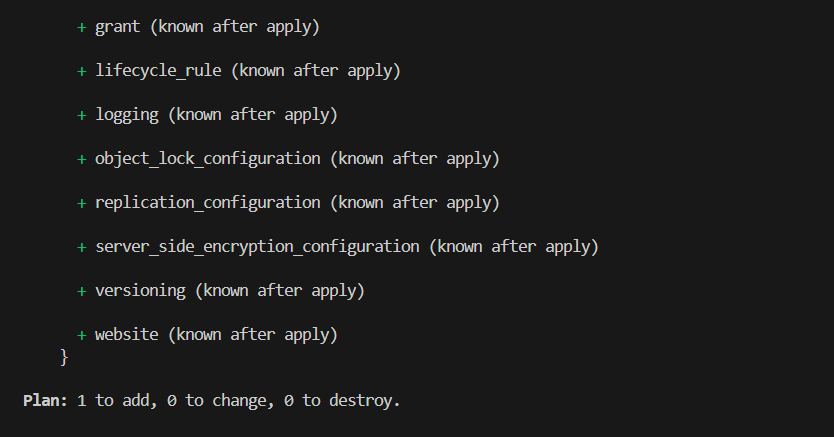
**Step 2: Open Command Prompt and go to Terraform\_Script\S3 directory where our .tf files are stored**

**Step 3: Execute Terraform Init command to initialize the resources**

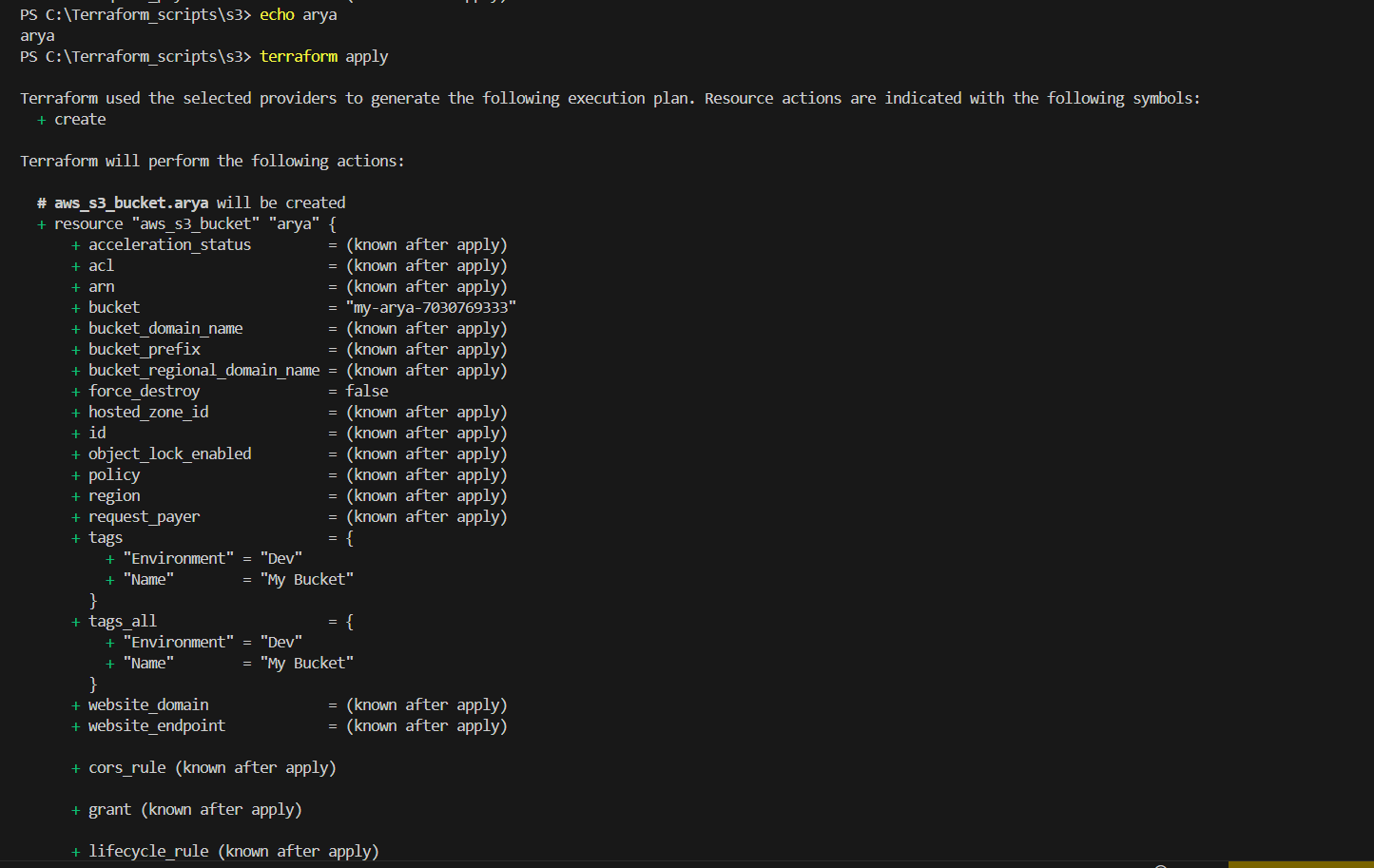


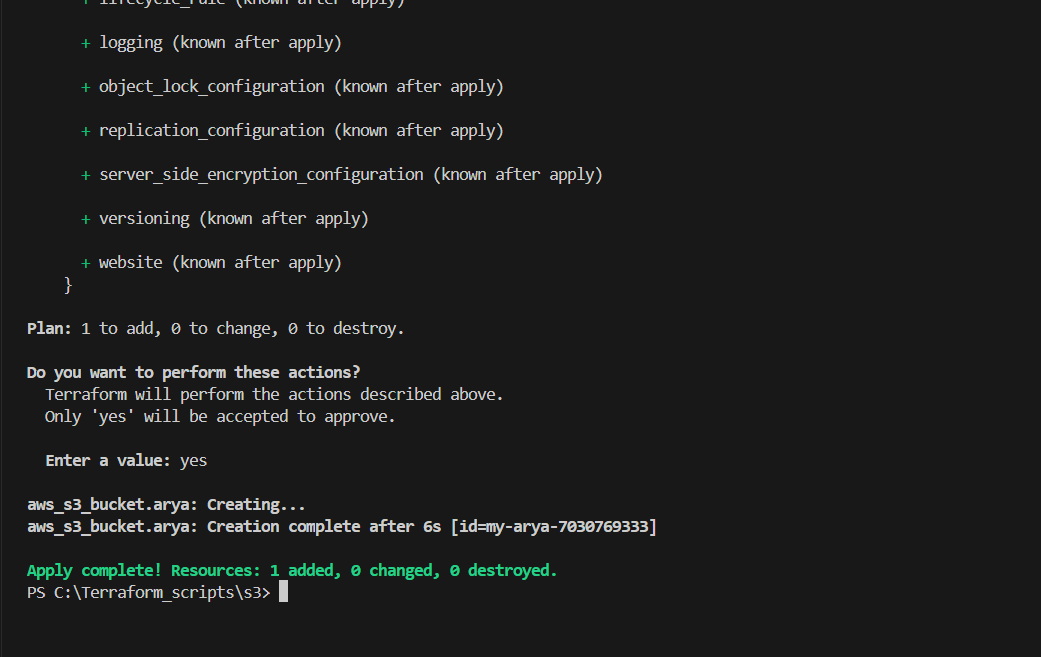
**Step 4: Execute Terraform plan to see the available resources**



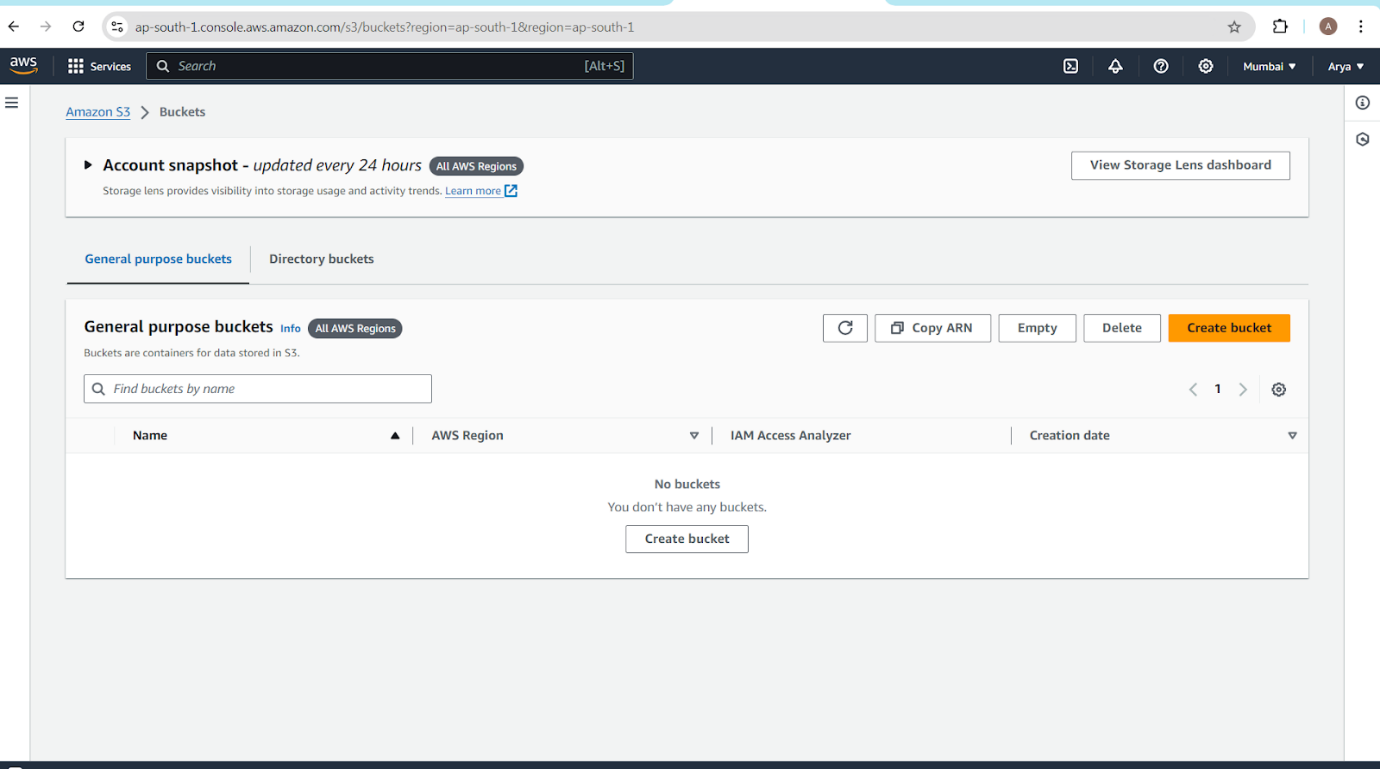


**Step 5: Execute Terraform apply to apply the configuration, which will automatically create an S3 bucket based on our configuration.**

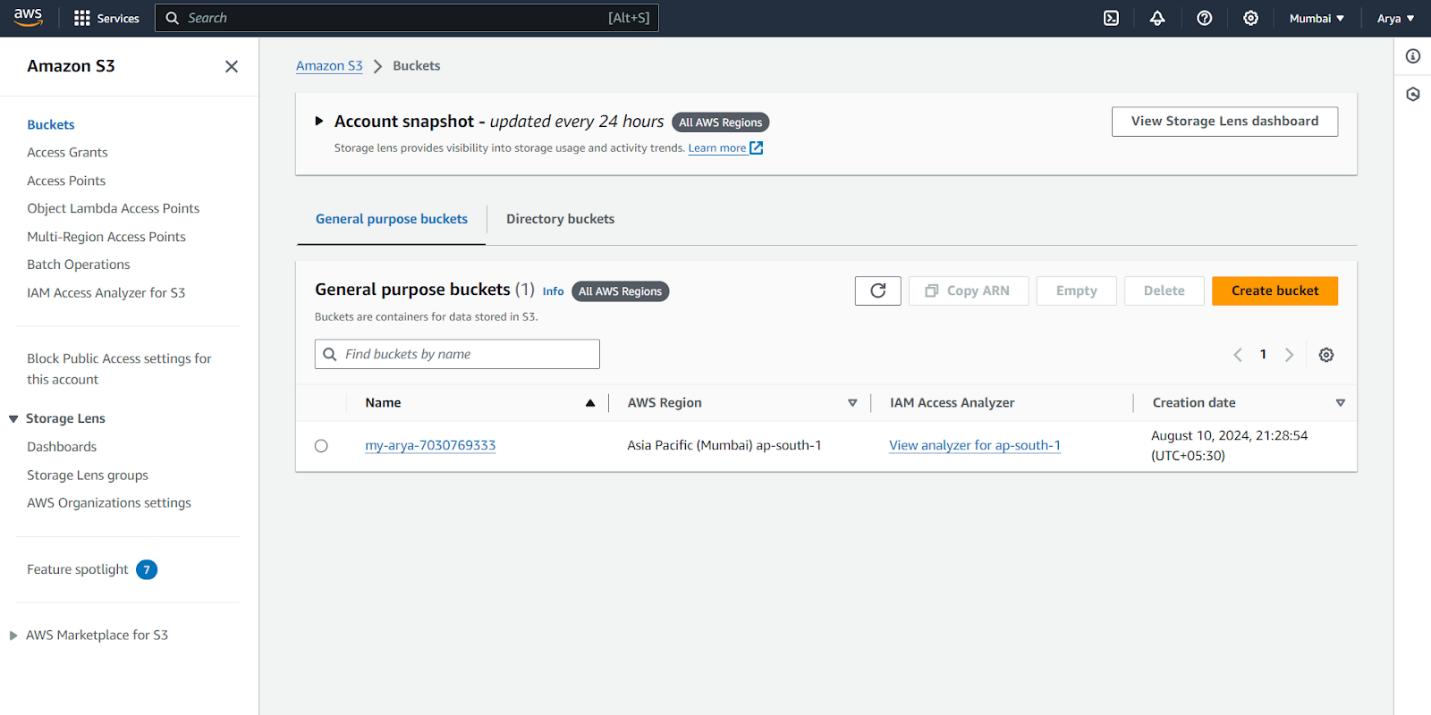




**AWS S3bucket  dashboard, Before Executing Apply command:**



**AWS S3 Bucket dashboard, After Executing Apply step:**



**Step 6: Execute Terraform destroy to delete the configuration, which will automatically delete an EC2 instance**

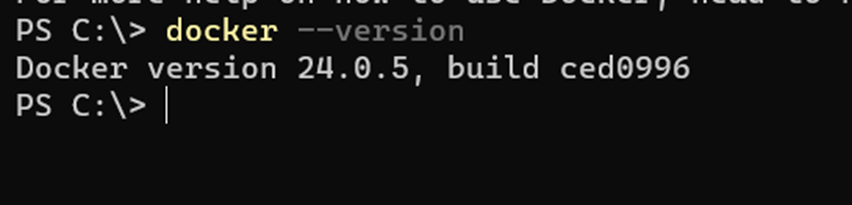
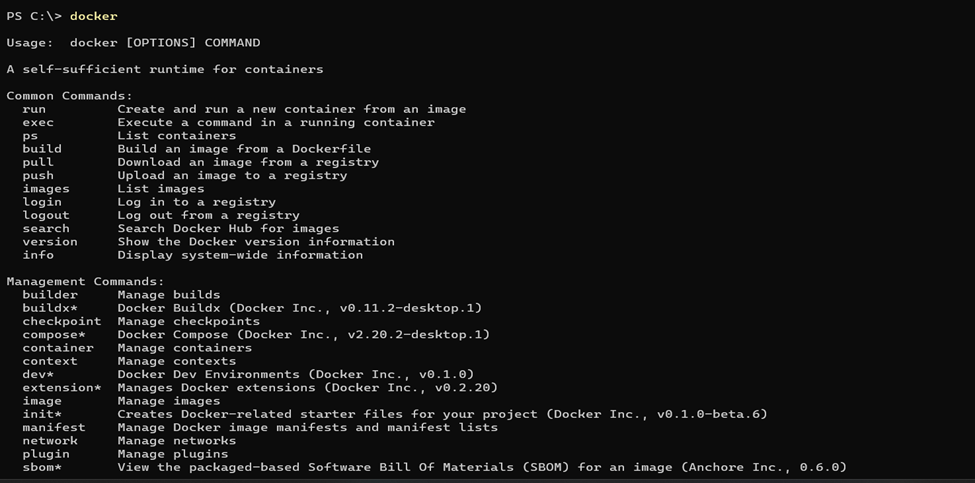
**AWS EC2 dashboard, After Executing Destroy step**

**AIM: Creating docker image using terraform**

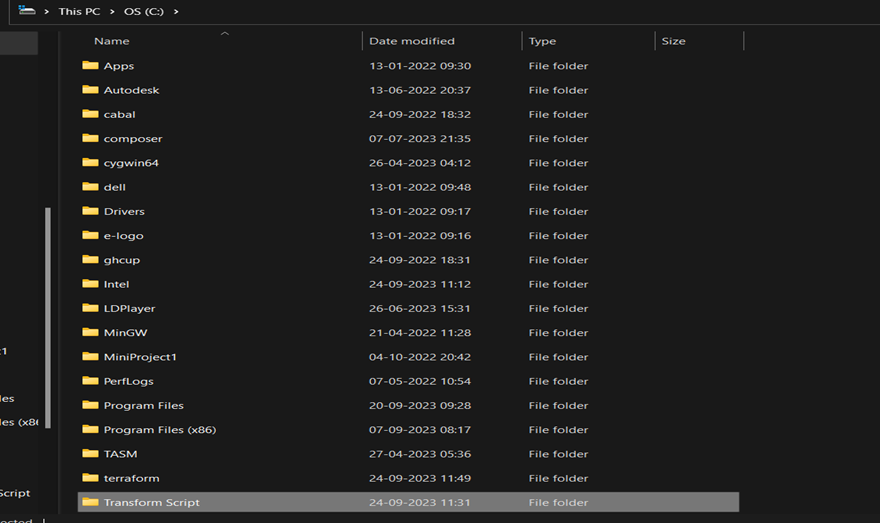
**Prerequisite:**

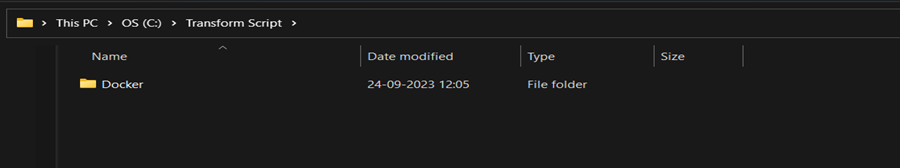
**1) Download and Install Docker Desktop from https://www.docker.com/**

**Step 1: Check the docker functionality**

**Now, create a folder named ‘Terraform Scripts’ in which we save our different types of scripts which will be further used in this experiment.**

**Step 2: Firstly create a new folder named ‘Docker’ in the ‘TerraformScripts’ folder.**

****

****

**Then create a new docker.tf file using vs code editor and write the following contents into it to create a Ubuntu Linux container.**

**Script:**

**terraform{**

**required\_providers{**

**docker = {**

**source = "kreuzwerker/docker"**

**version = "2.21.0"**

**}**

**}**

**}**

**provider "docker" {**

**host = "npipe:////.//pipe//docker\_engine"**

**}**

**# Pulls the image**

**resource "docker\_image" "nginx" {**

**name = "nginx:latest"**

**keep\_locally =false**

**}**

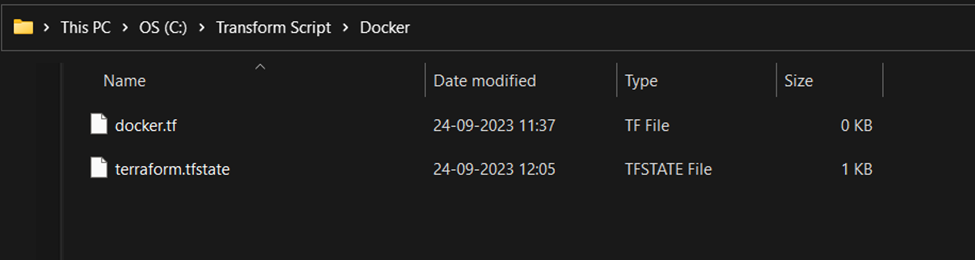
**# Create a container**

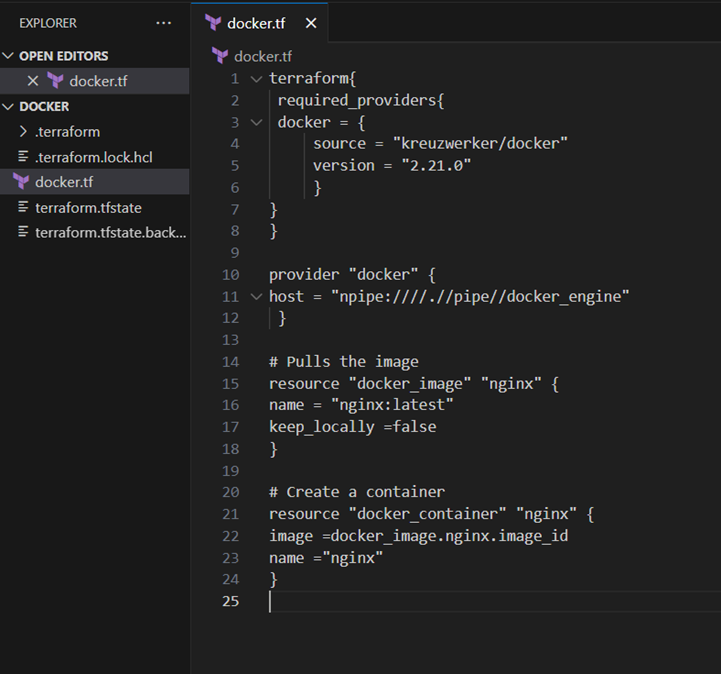
**resource "docker\_container" "nginx" {**

**image =docker\_image.nginx.image\_id**

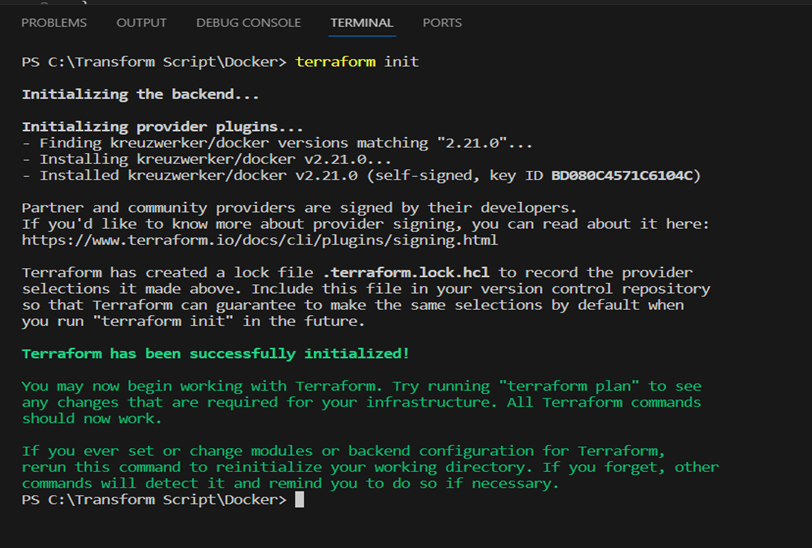
**name ="nginx"**

**}**

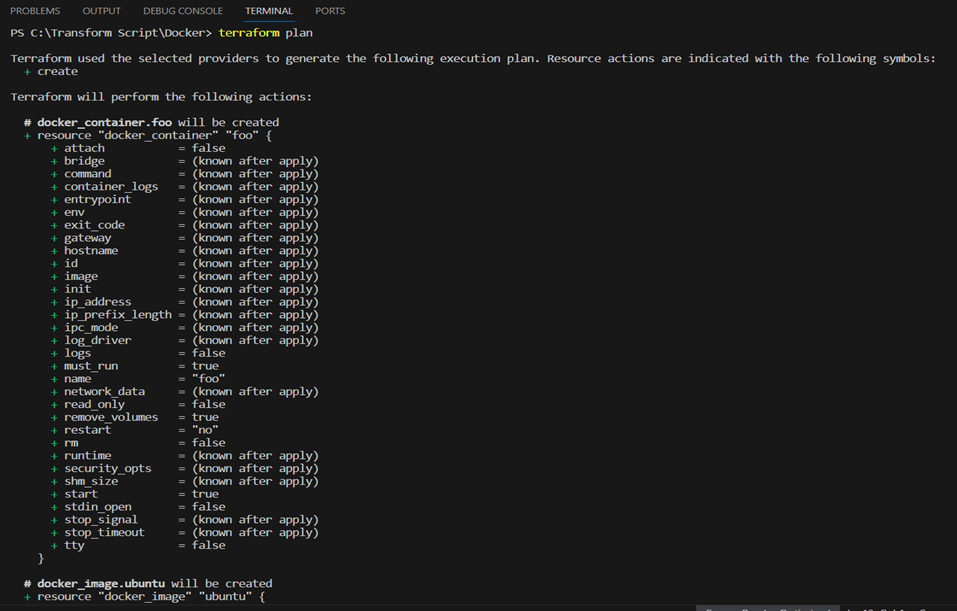
****

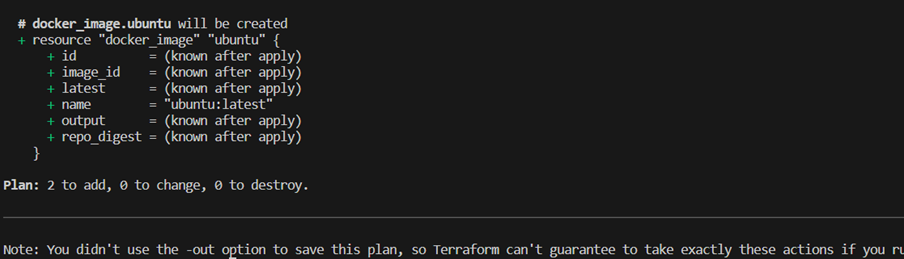
****

**Step 3: Execute Terraform Init command to initialize the resources**

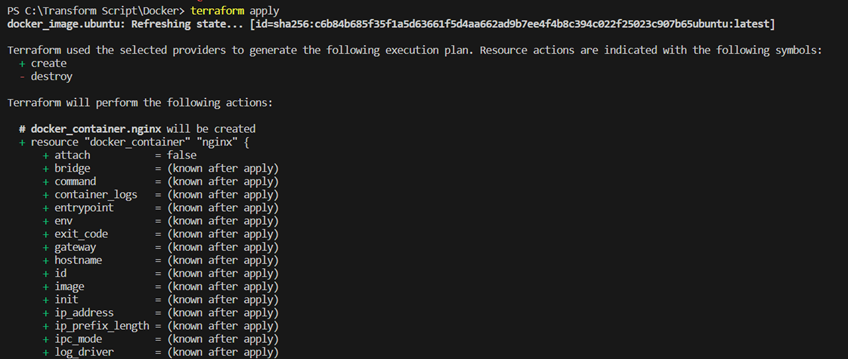
****

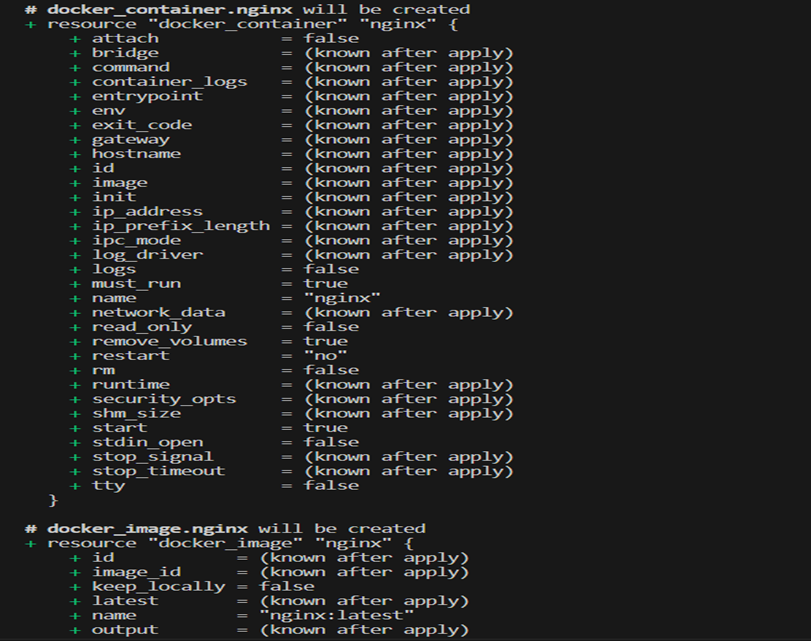
**Step 4: Execute Terraform plan to see the available resources**

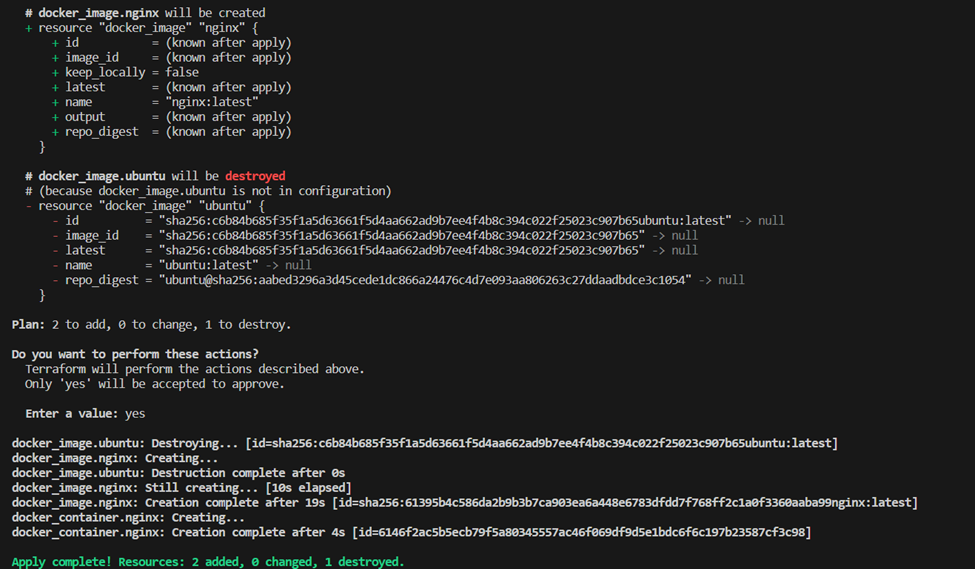
****

****

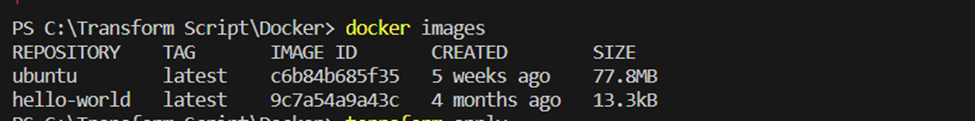
**Step 5: Execute Terraform apply to apply the configuration, which will automatically create and run the Ubuntu Linux container based on our configuration. Using command : “terraform apply”**

****

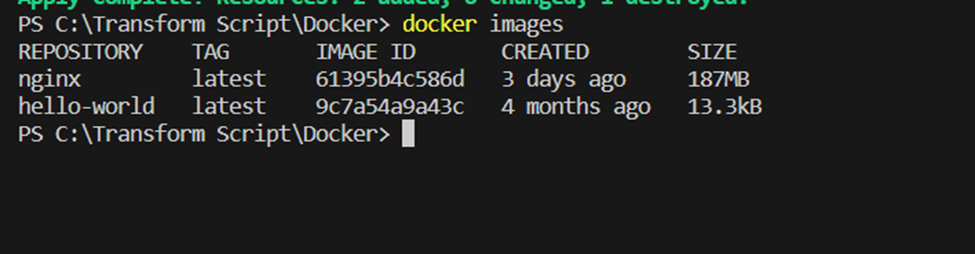
****

****

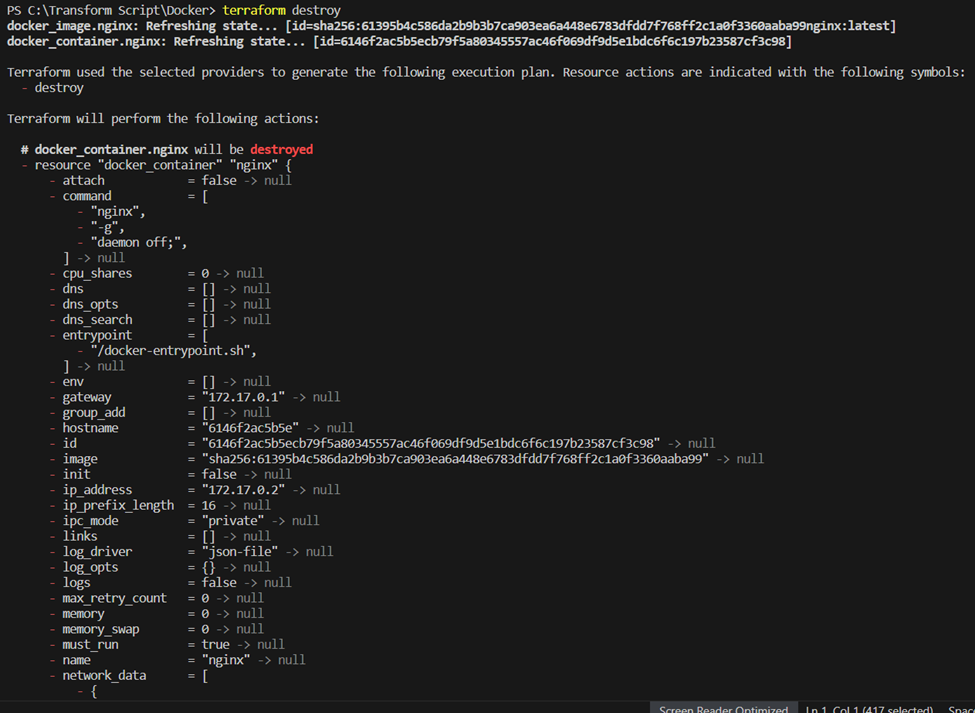
**Docker images, Before Executing Apply step:**

****

**Docker images, After Executing Apply step:**

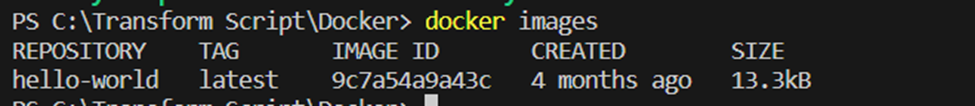
****

**Step 6: Execute Terraform destroy to delete the configuration, which will automatically delete the Ubuntu Container.**

****

****

**Docker images After Executing Destroy step**

****

**Conclusion:**

Thus, we have successfully created a Docker image using Terraform, demonstrating the power of Infrastructure as Code (IaC). By automating the creation and management of cloud resources, Terraform simplifies deployment processes, enhances scalability, and ensures consistency across environments. The integration with Docker further improves portability and efficiency, allowing for seamless application deployment across multiple platforms.