Advance DevOps Practical Examination Case Study Assignment

NAME-SPANDAN DEB

CLASS-D15A

ROLL NO-13

13. Basic Infrastructure Management with Terraform

● **Concepts Used:** Terraform, AWS EC2, and S3.

● **Problem Statement**: "Use Terraform to create an EC2 instance and an S3 bucket. Store

the EC2 instance's IP address in the S3 bucket."

● **Tasks**:

○ Write a simple Terraform script to provision an EC2 instance and an S3 bucket.

○ Use Terraform outputs to extract the EC2 instance's IP address.

○ Store the IP address in a text file in the S3 bucket.

1.**Introduction**

Case Study Overview: This case study focuses on automating cloud infrastructure provisioning using Terraform, specifically for AWS EC2 and S3 services. The primary goal is to efficiently manage cloud resources with Infrastructure as Code (IaC) principles. By using Terraform, we can deploy and configure an EC2 instance and an S3 bucket in AWS, ensuring consistency and automation in resource management.

Key Feature and Application: The unique feature of this case study is the seamless automation of infrastructure provisioning and configuration using Terraform. This automation eliminates the need for manual cloud resource management, allowing users to define and manage infrastructure as code. In practical terms, this enables developers and cloud engineers to create scalable, repeatable cloud environments. The EC2 instance's public IP address is extracted and stored in an S3 bucket, showcasing how cloud resources can be dynamically interconnected through automation.

Third-Year Project Integration (Optional):

1)Automated Provisioning of OCR Processing Environment: Terraform can be used to automatically create and manage the EC2 instances required to run OCR tasks. This removes the need for manual configuration, allowing the environment to be set up quickly when new OCR jobs are required.

2)Scalability: As OCR tasks can be resource-intensive, Terraform allows the infrastructure to scale automatically. More EC2 instances can be provisioned based on the load, ensuring the OCR solution can handle large batches of images efficiently.

3) Data Storage and Retrieval with S3: AWS S3 can be utilized to store images for OCR processing and the extracted text. Terraform can automate the creation of S3 buckets to manage this data flow seamlessly, storing both input and output data in an organized and secure manner.

4) Cost Optimization: By using Terraform, resources such as EC2 instances can be spun up only when needed and terminated after use. This ensures that OCR tasks are performed without incurring unnecessary costs, as the infrastructure is dynamically managed.

5)Consistency and Repeatability: Terraforms Infrastructure as Code (IaC) approach ensures that the OCR environment is consistently set up each time, with the same configurations and dependencies. This is critical in OCR projects, where consistent results depend on identical environments.

2.**Step-by-Step Explanation**

Step 1: Provide a detailed explanation of the initial setup or configuration.

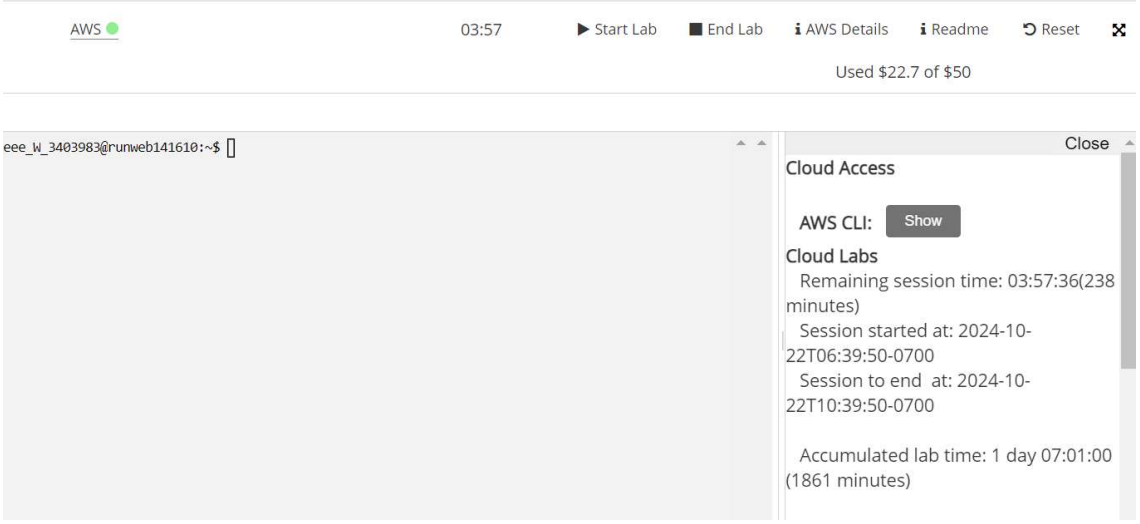
1.Install Terraform: The first step is to install Terraform on your local machine. Terraform is the tool that helps automate the process of creating and managing cloud resources. Confirm installation with terraform -v command, this should display the version of Terraform installed, confirming that it is ready for use.

2.Create AWS Account: AWS CLI is required by Terraform to communicate with your AWS account so having an AWS account is a necessity.

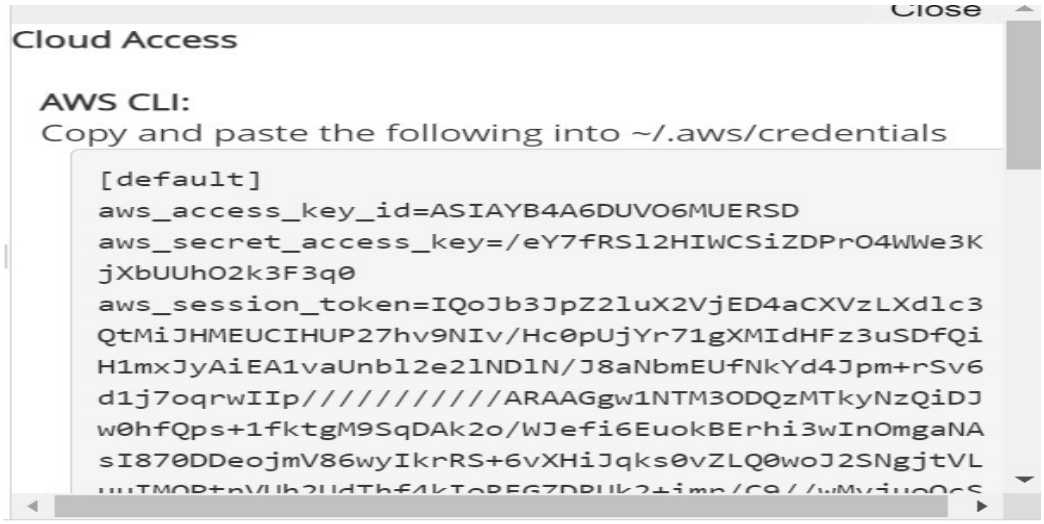
3.Create a Project Folder and Configure AWS CLI: On your local machine, create a new folder and open it on VS Code, this folder will hold the Terraform configuration files .Get the access key id, secret access key and session token from AWS CLI and paste it in the terminal of AWS .

Step 2: Continue with subsequent steps, including any code snippets or commands.

Go to AWS Learners Lab click start and then click on AWS details

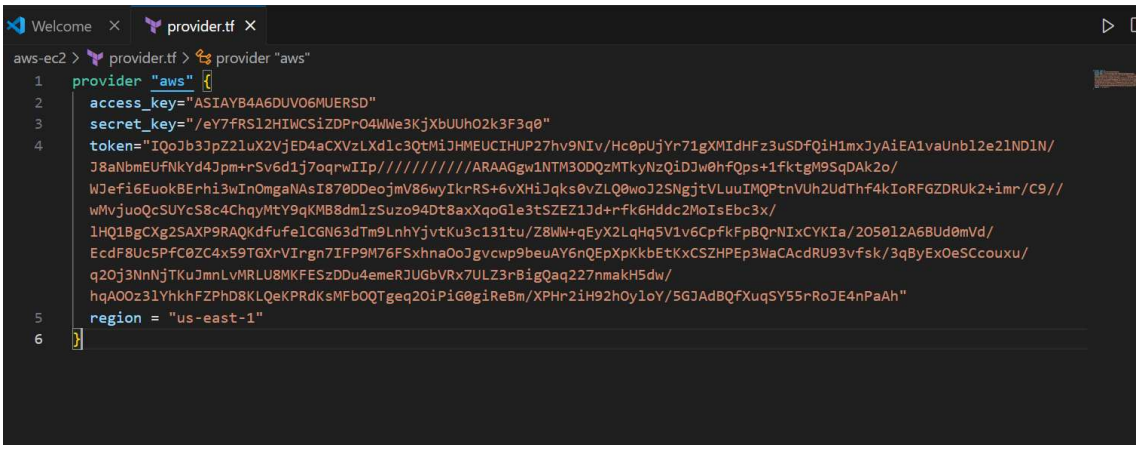


We can see AWS CLI click show.

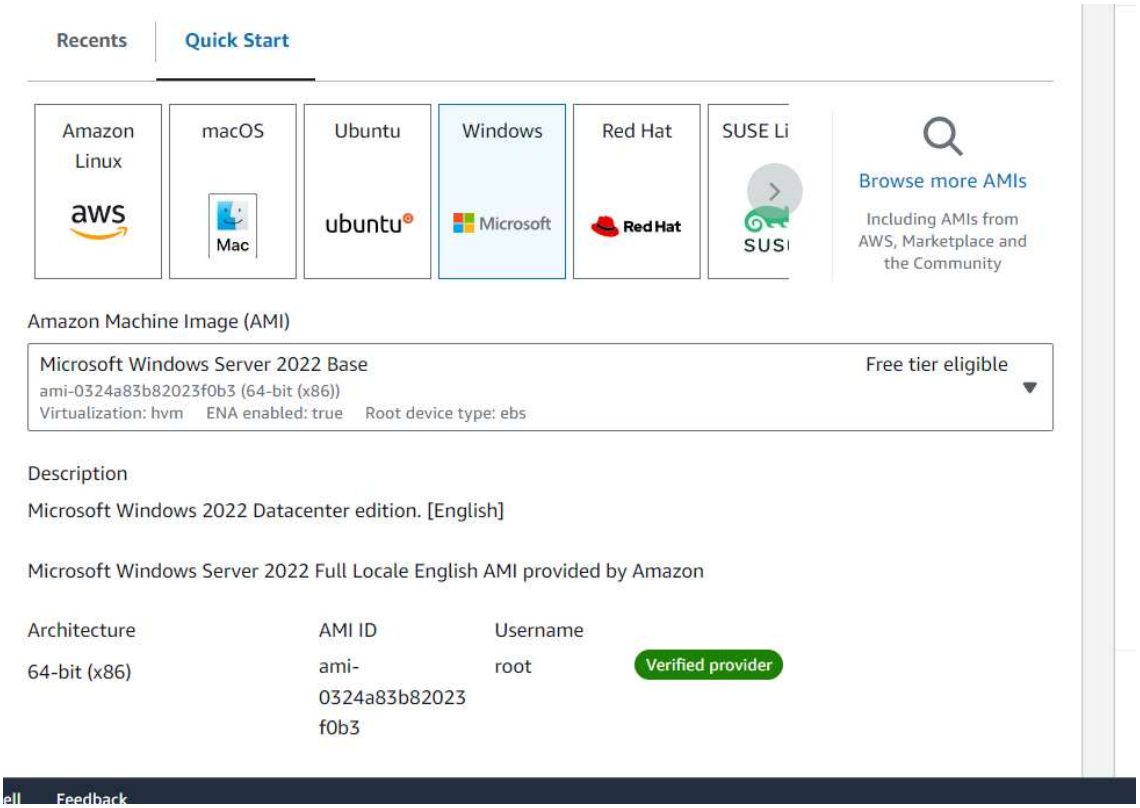


These are the credentials copy paste these in a file in VS code. Add export keyword put the values in quotes and copy paste it in the Learner Lab.

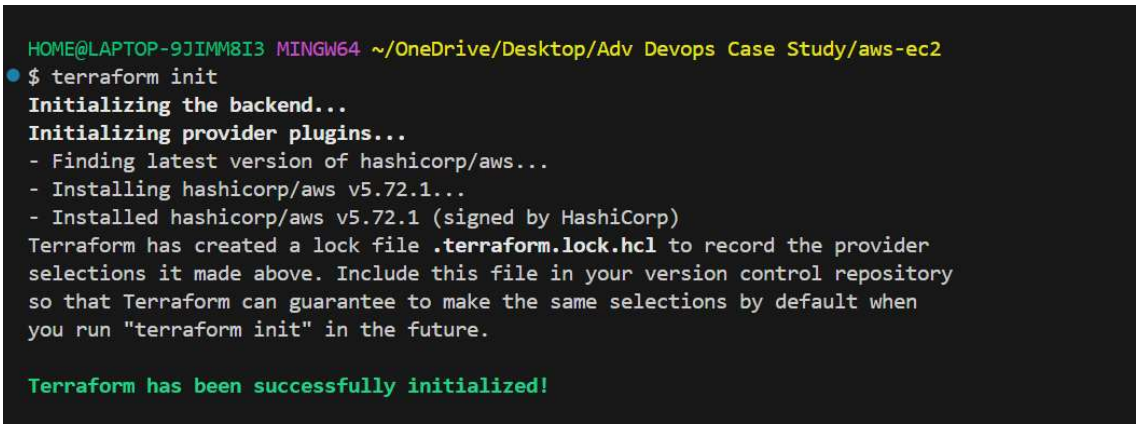
Create a provider.tf file and add this code. This Terraform code block defines the AWS provider with specific credentials (access key, secret key, and token) to authenticate and interact with the AWS services.



Now Create a new file main.tf in the folder. Click on AWS in learner lab it will route you to the dashboard, search EC2, click on launch instance. Select windows AMI and copy its AMI ID.

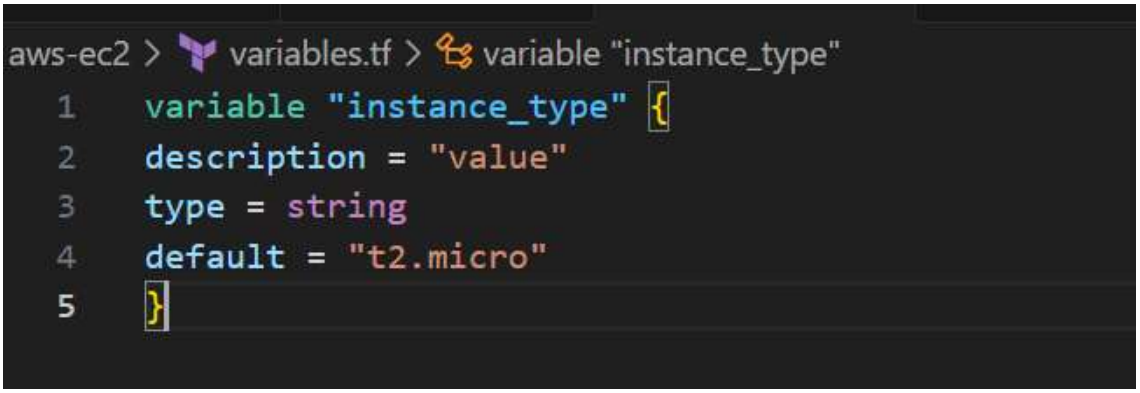


Go to the terminal check whether its in current directory if not cd over it. In terminal initialize terraform.



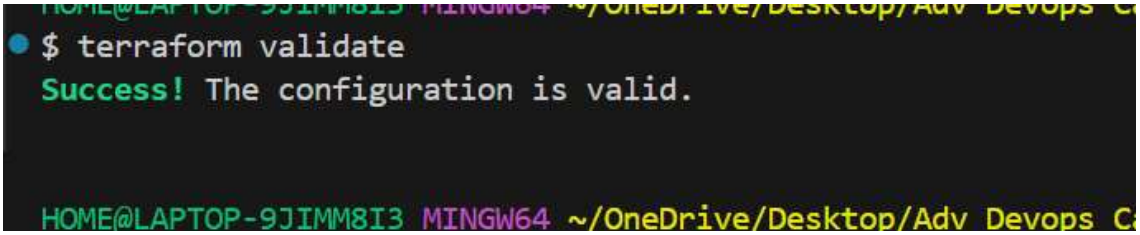
Create two files output.tf and variable.tf

Use this code in variables.tf. It is used to set the EC2 instance type



In main.tf change the value of instance_type to var.instance_type

Validate the configurations by using terraform validate



Use this code in output.tf. This is used to see key information of the infrastructure the public Ip of ec2 instance and the instance type.

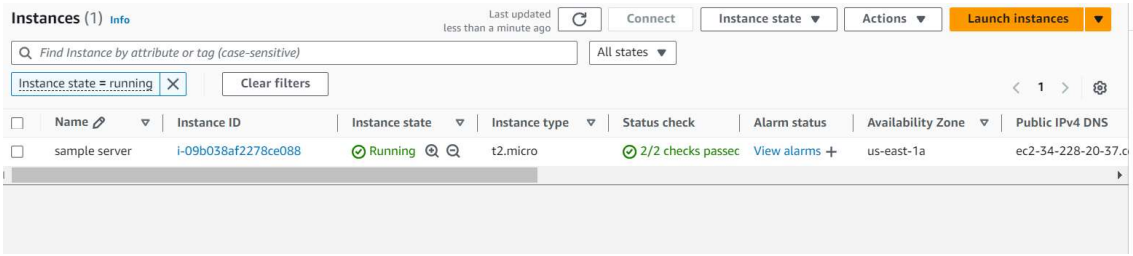
In terminal write terraform plan and apply. You will be able to see the IP address of the EC2 instance Copy it as it will be stored in a text file and the file will be stored in S3 bucket.

```
$ terraform plan

Terraform used the selected providers to generate the following execution plan.
  + create

Terraform will perform the following actions:

  # aws_instance.myserver will be created
  + resource "aws_instance" "myserver" {
      + ami                                  = "ami-0324a83b82023f0b3"
      + arn                                  = (known after apply)
```

```
HOME@LAPTOP-9JIMM8I3 MINGW64 ~/OneDrive/Desktop/Adv Devops Case Study/aws-ec2
$ terraform apply

Terraform used the selected providers to generate the following execution plan. Re
  + create

Terraform will perform the following actions:

  # aws_instance.myserver will be created
  + resource "aws_instance" "myserver" {
      + ami                                  = "ami-0324a83b82023f0b3"
```

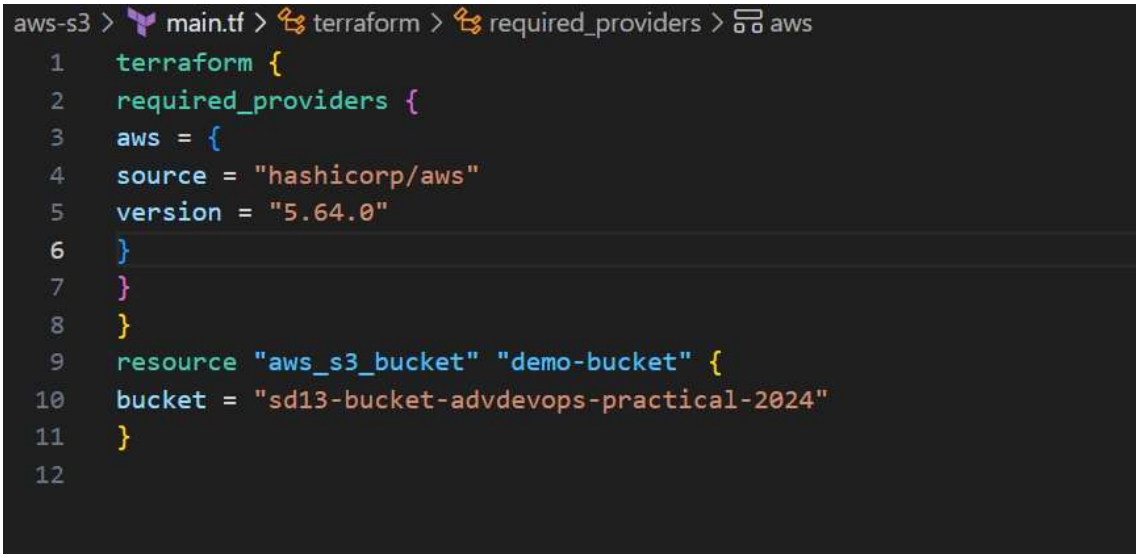
```
aws_instance.myserver: Creating...
aws_instance.myserver: Still creating... [10s elapsed]
aws_instance.myserver: Creation complete after 17s [id=i-09b038af2278ce088]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Outputs:

ec2_instance_type = "t2.micro"
ec2_public_ip = "34.228.20.37"
```

To check if the instance is being created go to AWS



Now we have to create an S3 bucket using terraform create a new folder aws-s3 and copy the provider.tf file as the credentials is going to remain the same.

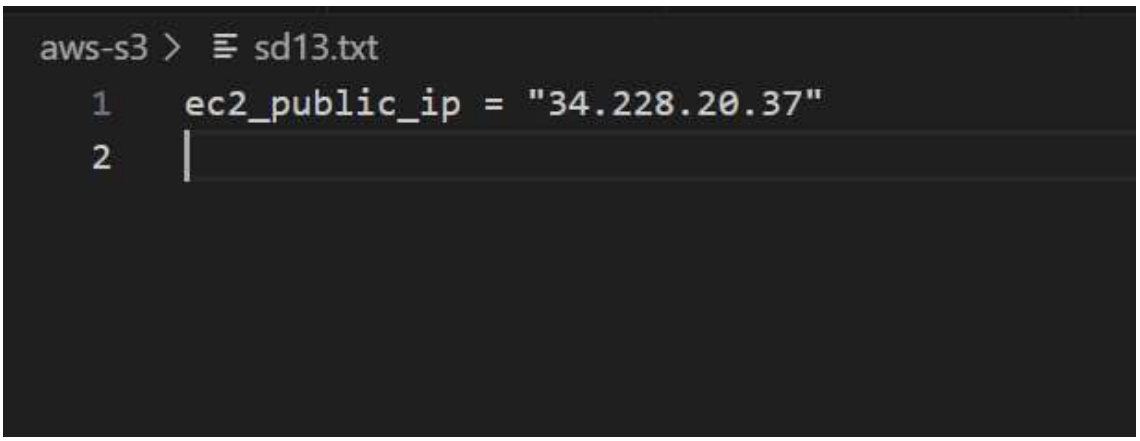
Create main.tf use this code to create bucket



Initialize, plan and apply terraform

```
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_s3_bucket.demo-bucket: Creating...
aws_s3_bucket.demo-bucket: Creation complete after 6s [id=sd13-bucket-advdevops-practical-2024]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Create a new text file and store the ec2 instance public IP address



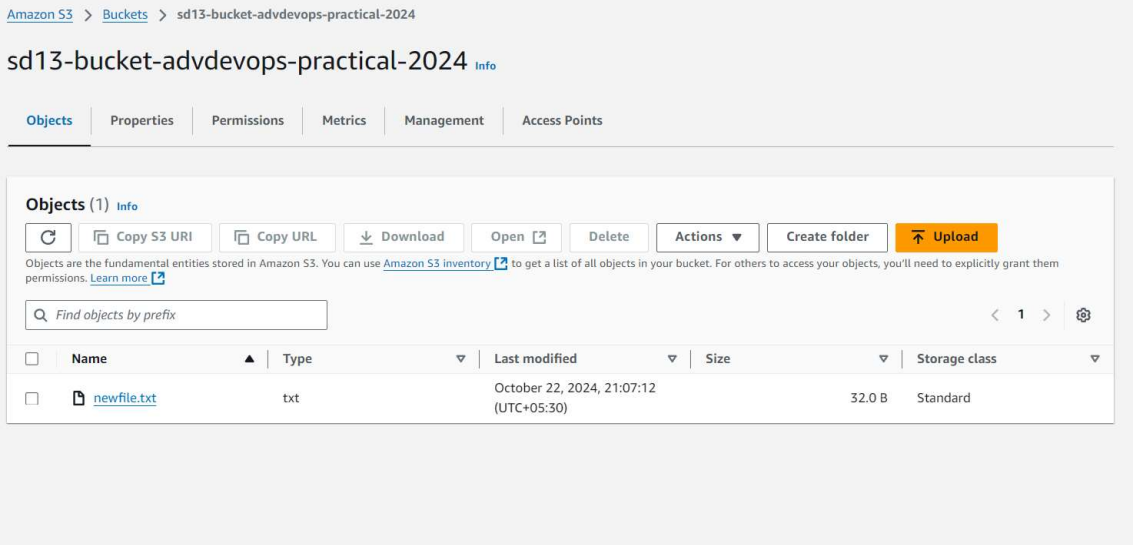
Add this code in the main.tf to upload the file in S3 bucket.

```
resource "aws_s3_object" "bucket-data" {
bucket = aws_s3_bucket.demo-bucket.bucket
source = "./sd13.txt"
key = "newfile.txt"
}
```

Terraform apply in the terminal

```
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_s3_object.bucket-data: Creating...
aws_s3_object.bucket-data: Creation complete after 2s [id=newfile.txt]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

HOME@LAPTOP-9JIMM8I3 MINGW64 ~/OneDrive/Desktop/Adv Devops Case Study/aws-s3
$
```

Bucket has been created in AWS



If we download the text file, we can see the IP address stored