

# Assignment No 5

Name: Rohan N

Roll no: 333047

PRN: 22010064

DIV: C Batch: C2

**Write IaC using terraform to create EC2 machine on AWS or azure or google cloud. (Compulsory to use Input and output variable files)**

## AIM

→ Use terraform to create an EC2 instance

## Theory

→ What is terraform?

→ Terraform Cloud enables infrastructure automation for provisioning, compliance, and management of any cloud, datacenter, and service.

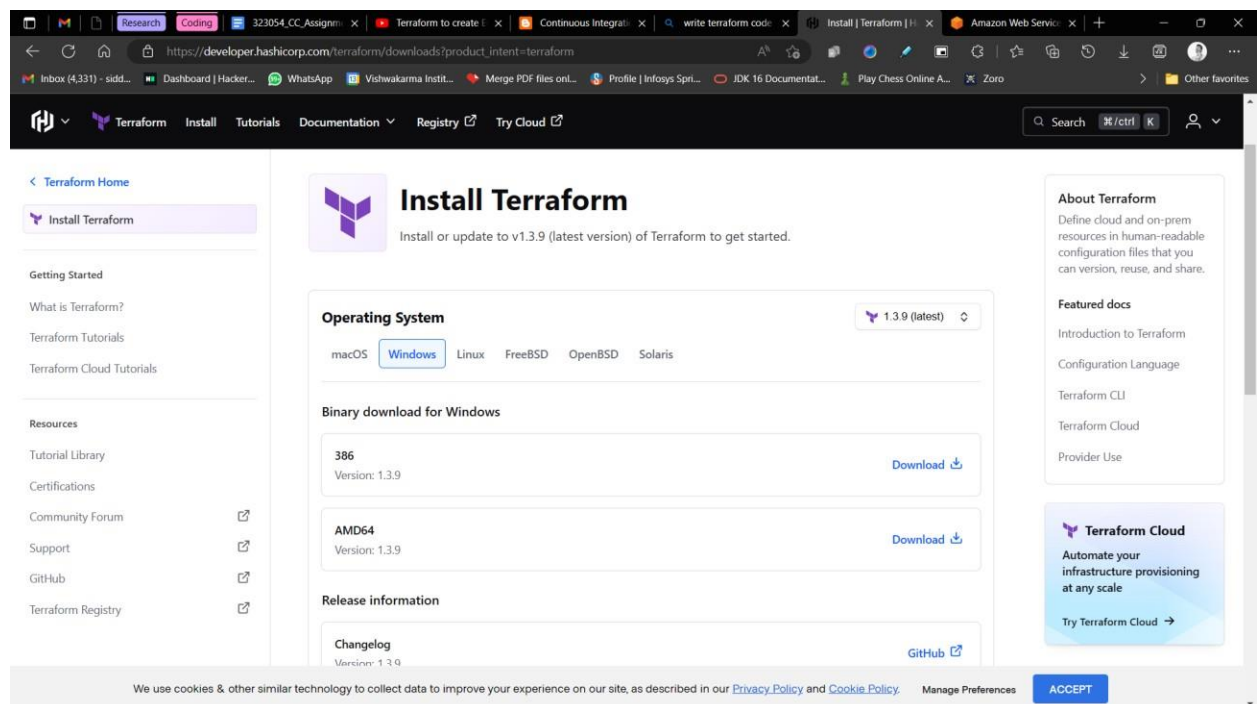
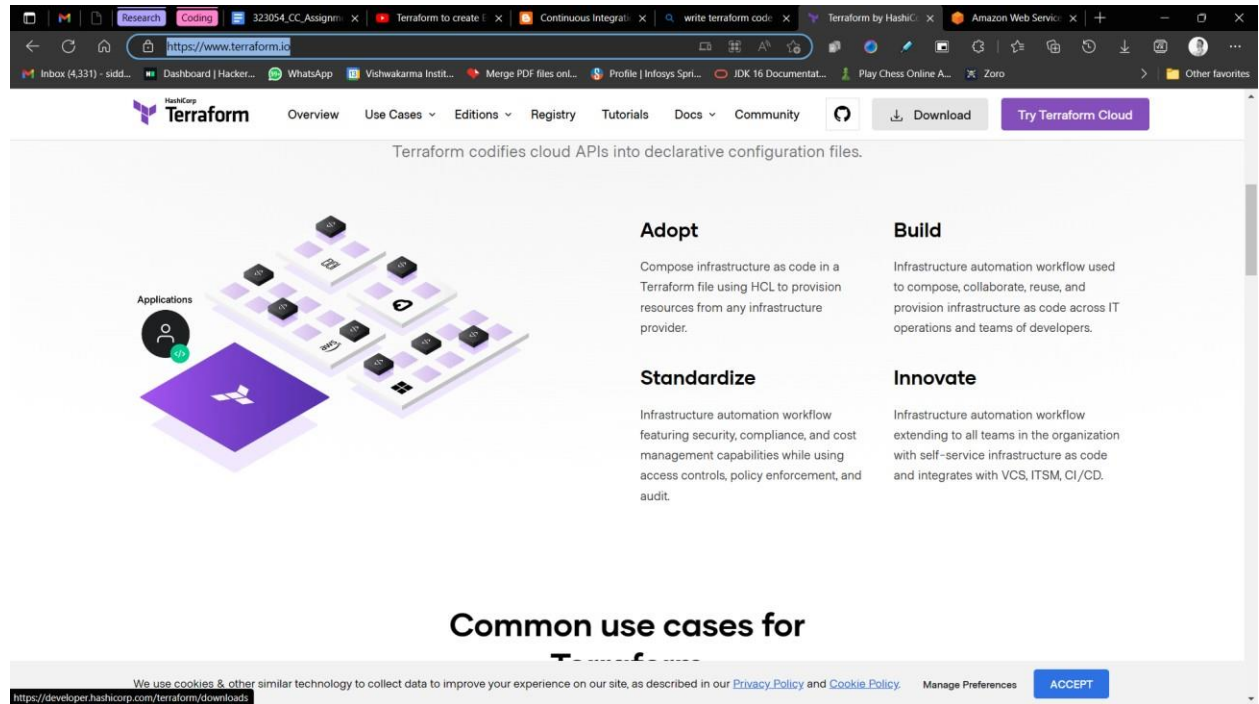
→ It is an open-source tool for provisioning and managing cloud infrastructure. Terraform can provision resources on any cloud platform.

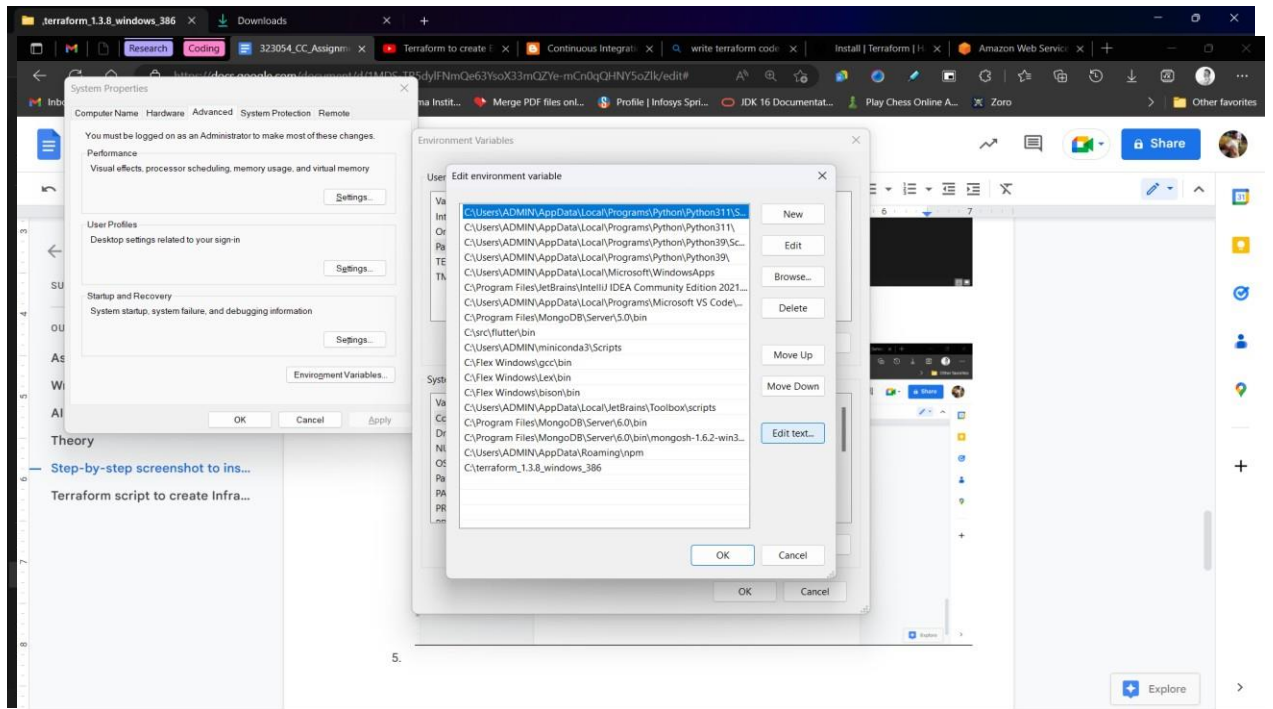
→ Terraform allows you to create infrastructure in configuration files(tf files) that describe the topology of cloud resources.

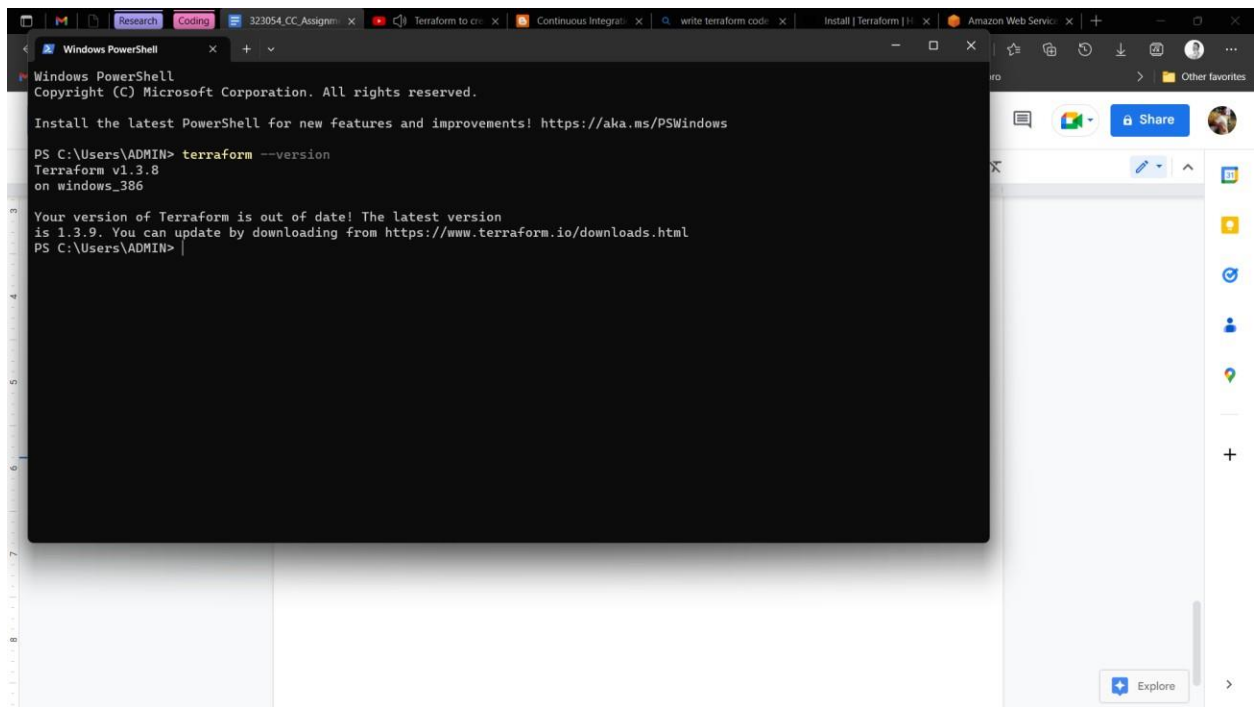
→ These resources include virtual machines, storage accounts, and networking interfaces or virtually any resource you want

# Step-by-step screenshot to install and configure Terraform + script (terraform)

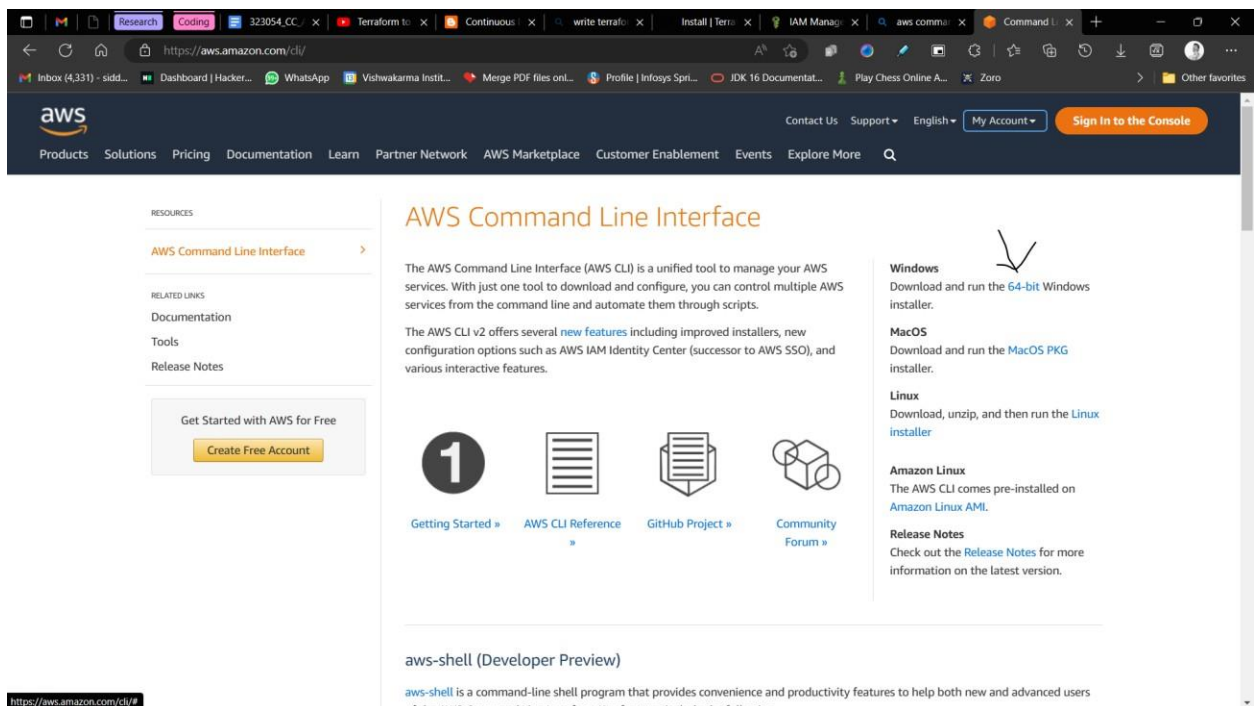
Download terraform from the website







Download Aws command line tool and configure it

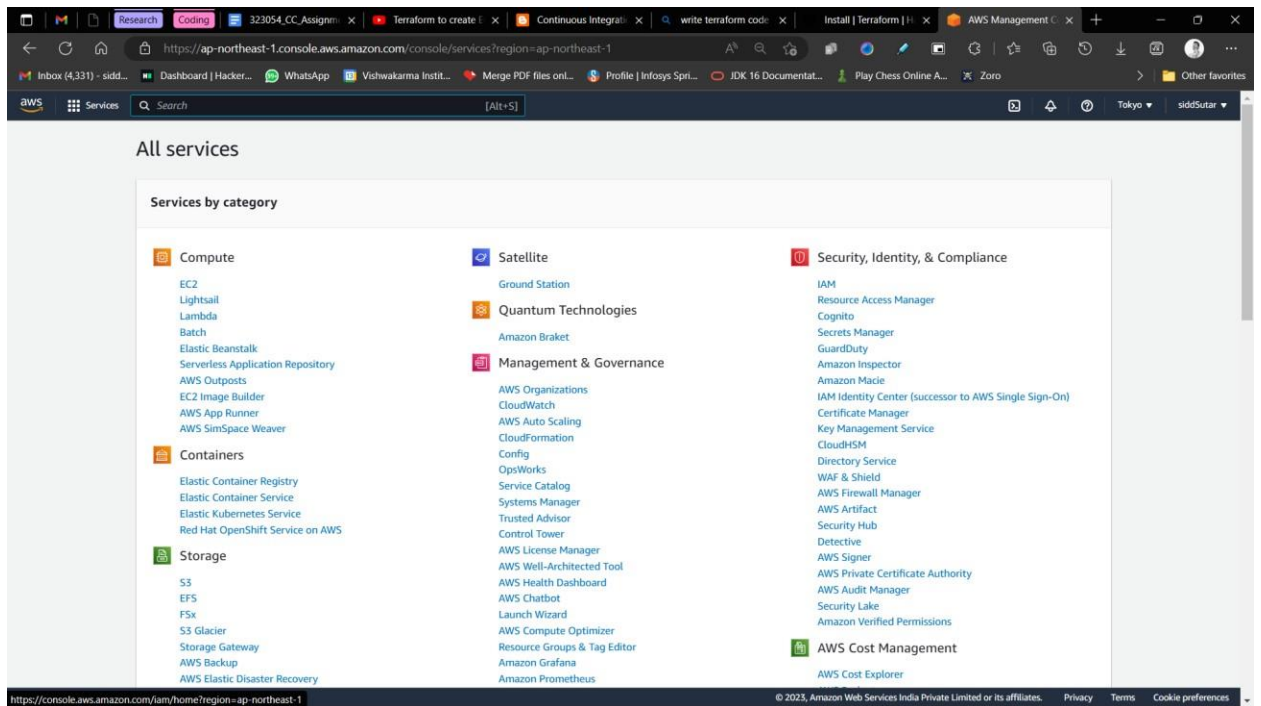
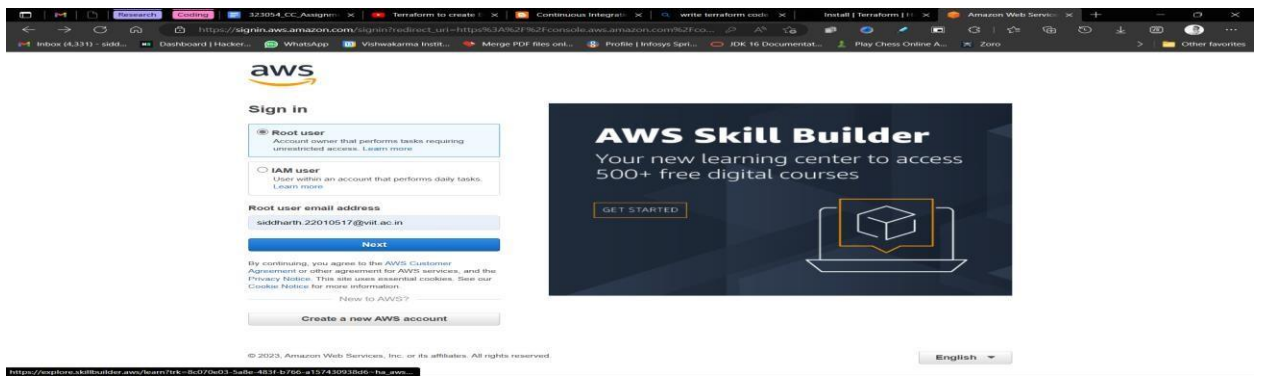


```

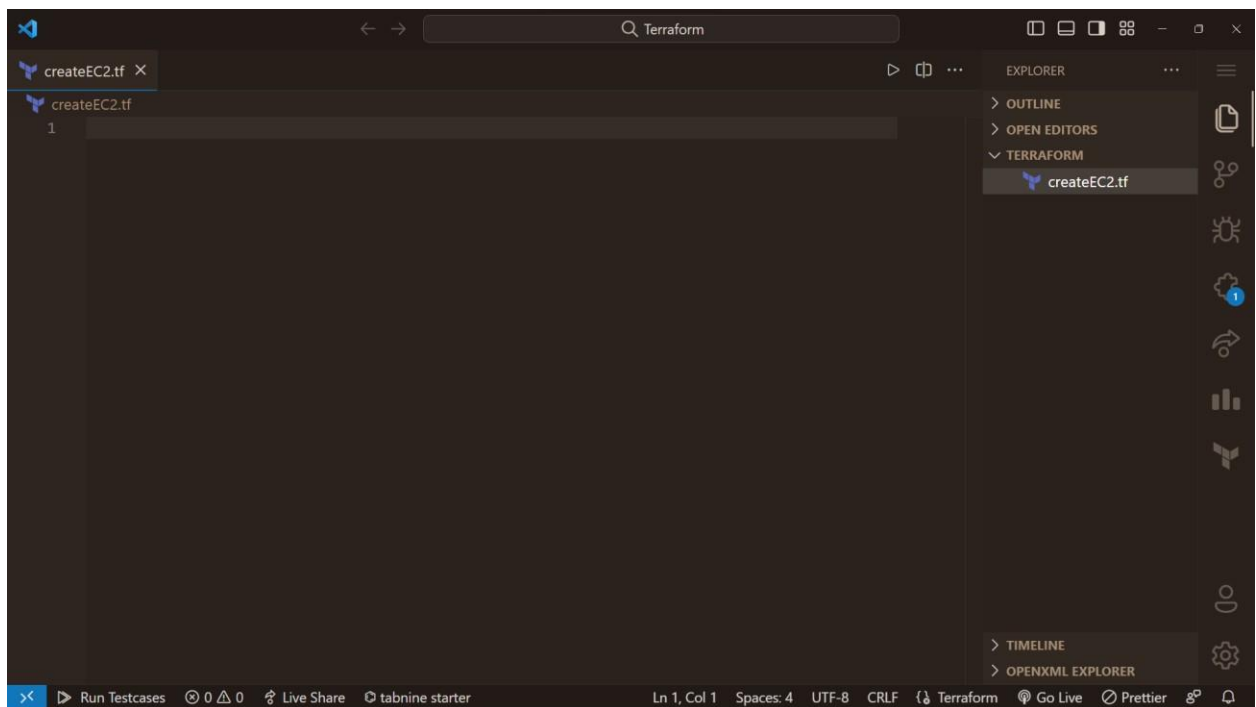
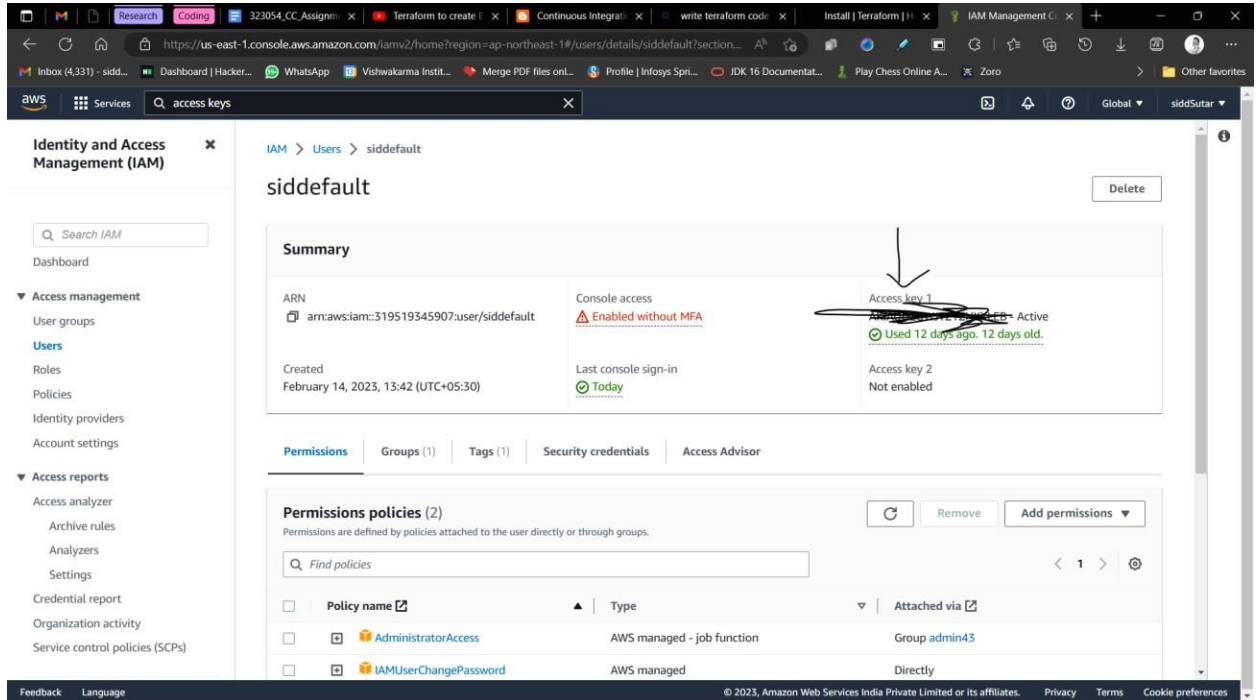
PS C:\Users\ADMIN> aws configure
AWS Access Key ID [*****XLEB]: AKIAUUGWJTYZHXLEB
AWS Secret Access Key [*****4Ga0]: F7PMtPTnB6aNF1cESy/OvB1P0cPIpm8Jz4GH4Ga0
Default region name [ap-south-1]: ap-south-1
Default output format [json]:
PS C:\Users\ADMIN> |

```

Login to aws and go to IAM service



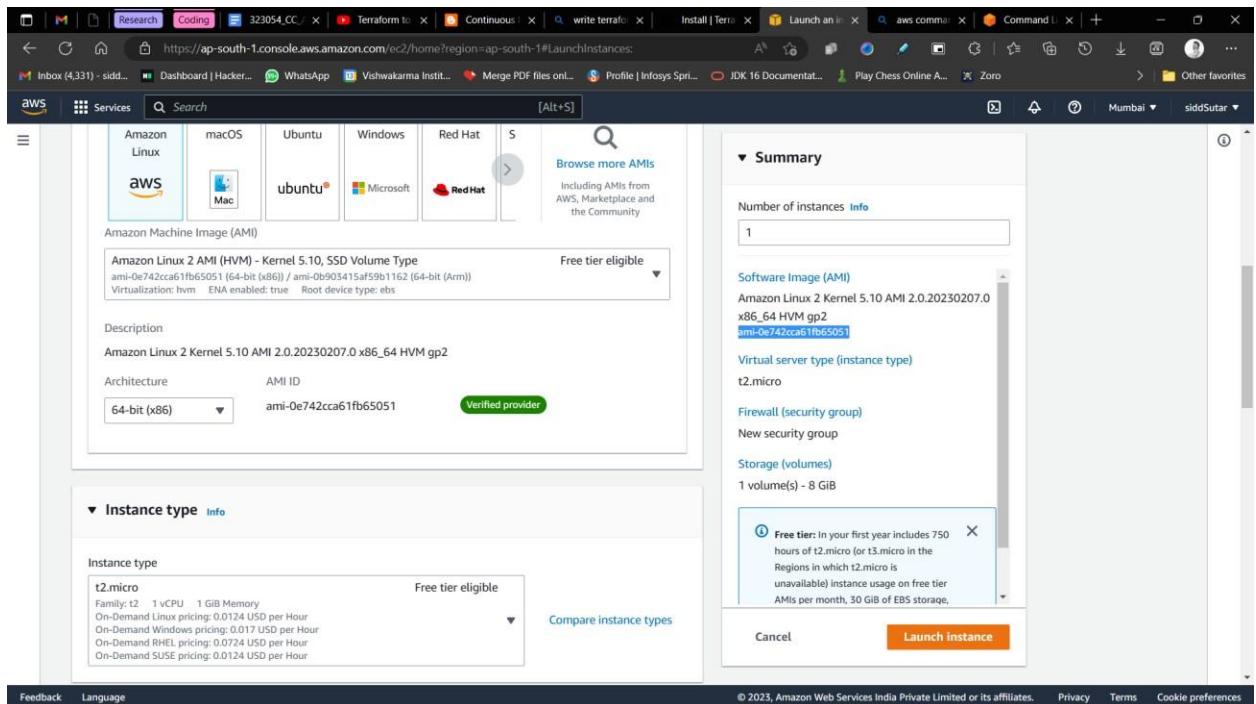
Create an user if you don't have one. In my case I have a user so I will be copying the access keys for later use



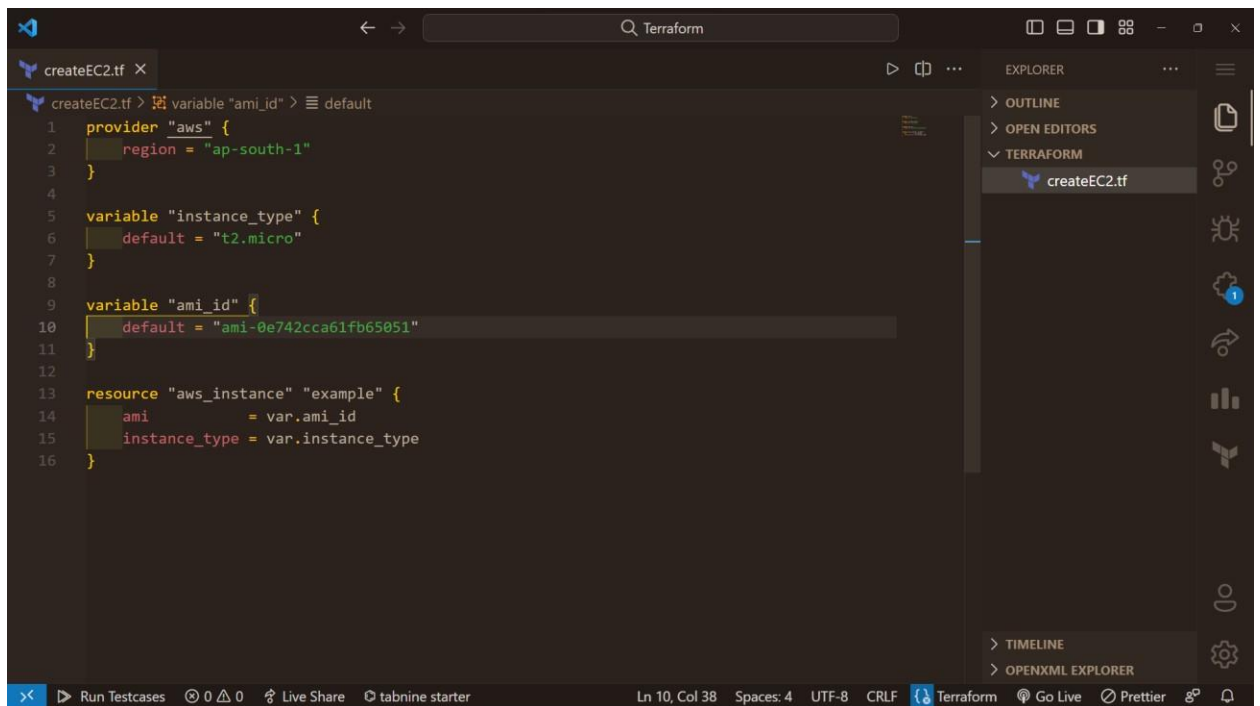
Write json code to create an EC2 instance & select the AMI ID for the machine



```
1 provider "aws" {
2   region = "ap-south-1"
3 }
4
5 variable "instance_type" {
6   default = "t2.micro"
7 }
8
9 variable "ami_id" {
10  default = ""
11 }
12
13 resource "aws_instance" "example" {
14   ami           = var.ami_id
15   instance_type = var.instance_type
16 }
```



Copy the ami Id to our Jason file



The screenshot shows a code editor with a dark theme. The main editor window displays a Terraform script named `createEC2.tf`. The script is as follows:

```
1 provider "aws" {
2   region = "ap-south-1"
3 }
4
5 variable "instance_type" {
6   default = "t2.micro"
7 }
8
9 variable "ami_id" {
10  default = "ami-0e742cca61fb65051"
11 }
12
13 resource "aws_instance" "example" {
14   ami           = var.ami_id
15   instance_type = var.instance_type
16 }
```

The right sidebar contains the Explorer panel with the following sections:

- OUTLINE
- OPEN EDITORS
- TERRAFORM
  - `createEC2.tf`
- TIMELINE
- OPENXML EXPLORER

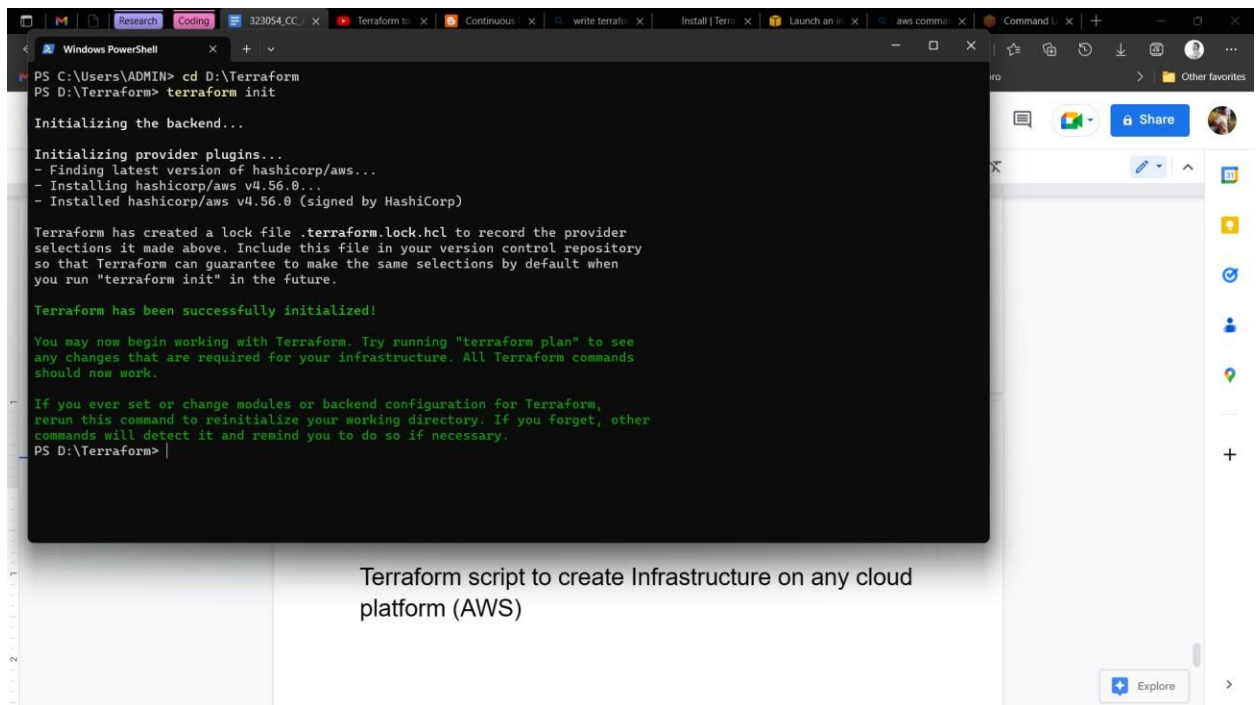
The bottom status bar shows the following information:

- Ln 10, Col 38
- Spaces: 4
- UTF-8
- CRLF
- Terraform
- Go Live
- Prettier

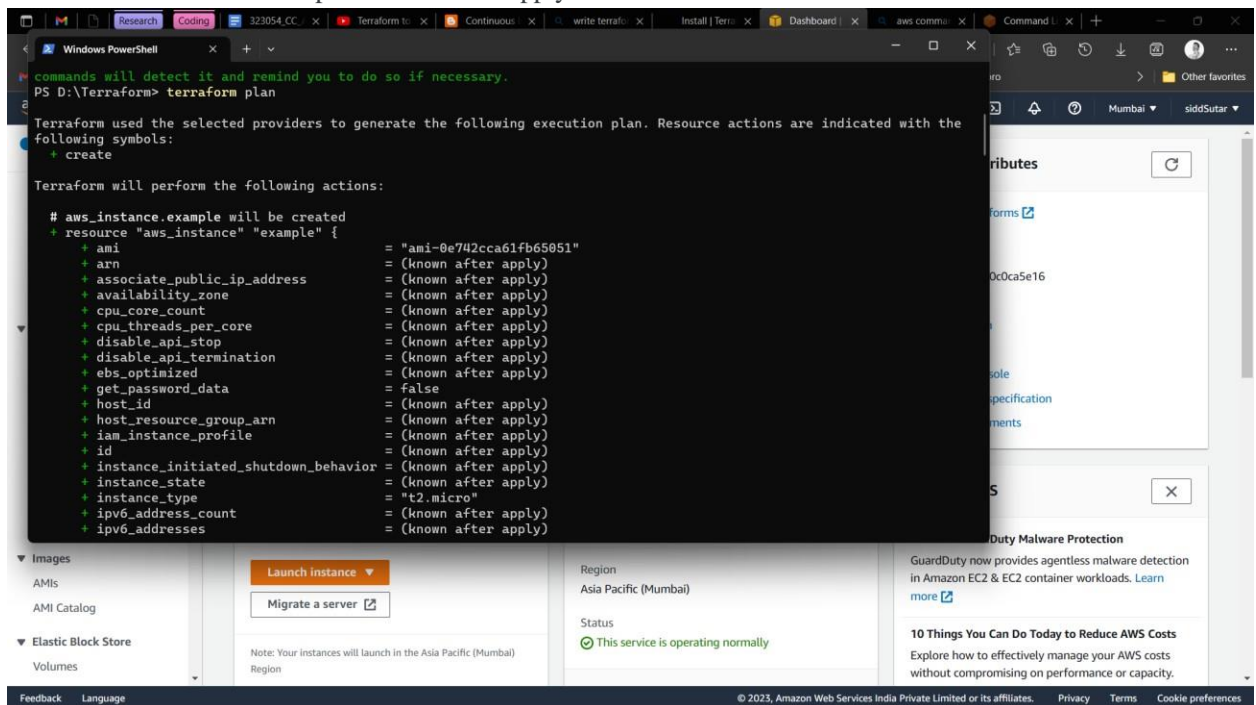
# Terraform Script

Change the directory and enter command terraform init





Put the command terraform plan & terraform apply



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

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if
you run "terraform apply" now.
PS D:\Terraform> terraform apply

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.example will be created
+ resource "aws_instance" "example" {
  + ami              = "ami-0e742cca61fb65051"
  + 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)
}
```

```
+ root_block_device {
  + delete_on_termination = (known after apply)
  + device_name           = (known after apply)
  + encrypted              = (known after apply)
  + iops                   = (known after apply)
  + kms_key_id             = (known after apply)
  + tags                   = (known after apply)
  + throughput             = (known after apply)
  + volume_id              = (known after apply)
  + volume_size            = (known after apply)
  + volume_type            = (known after apply)
}

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

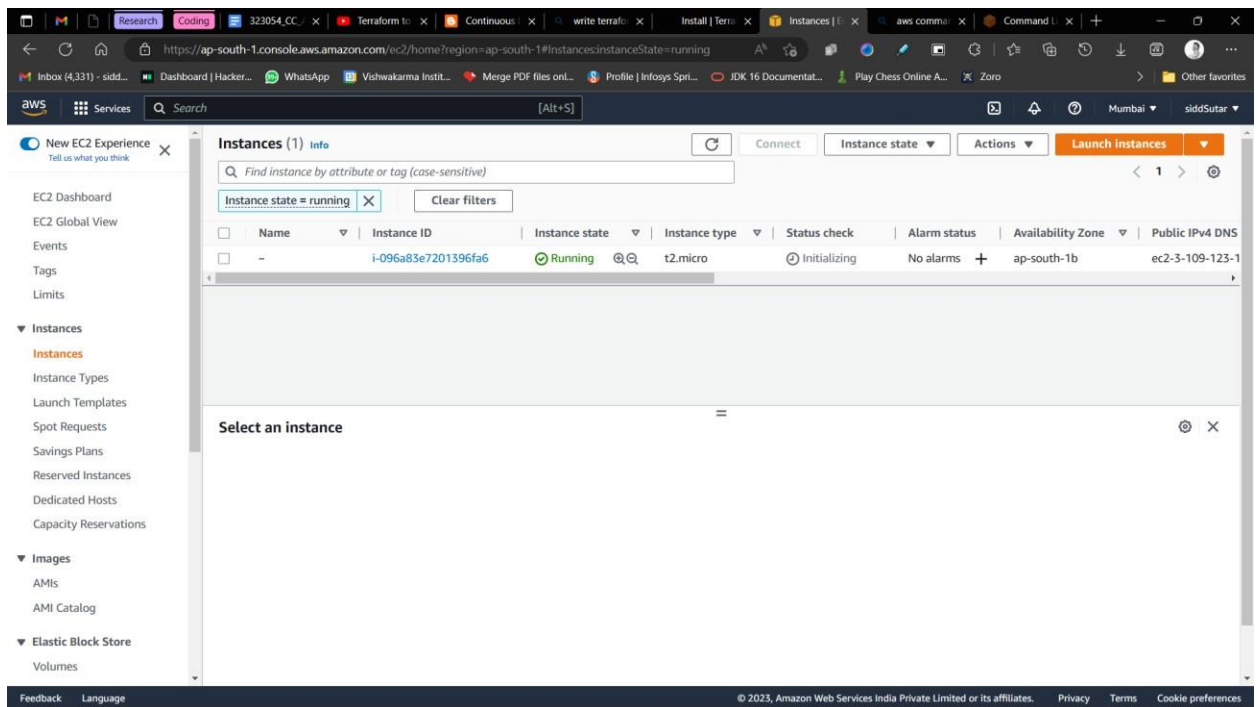
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

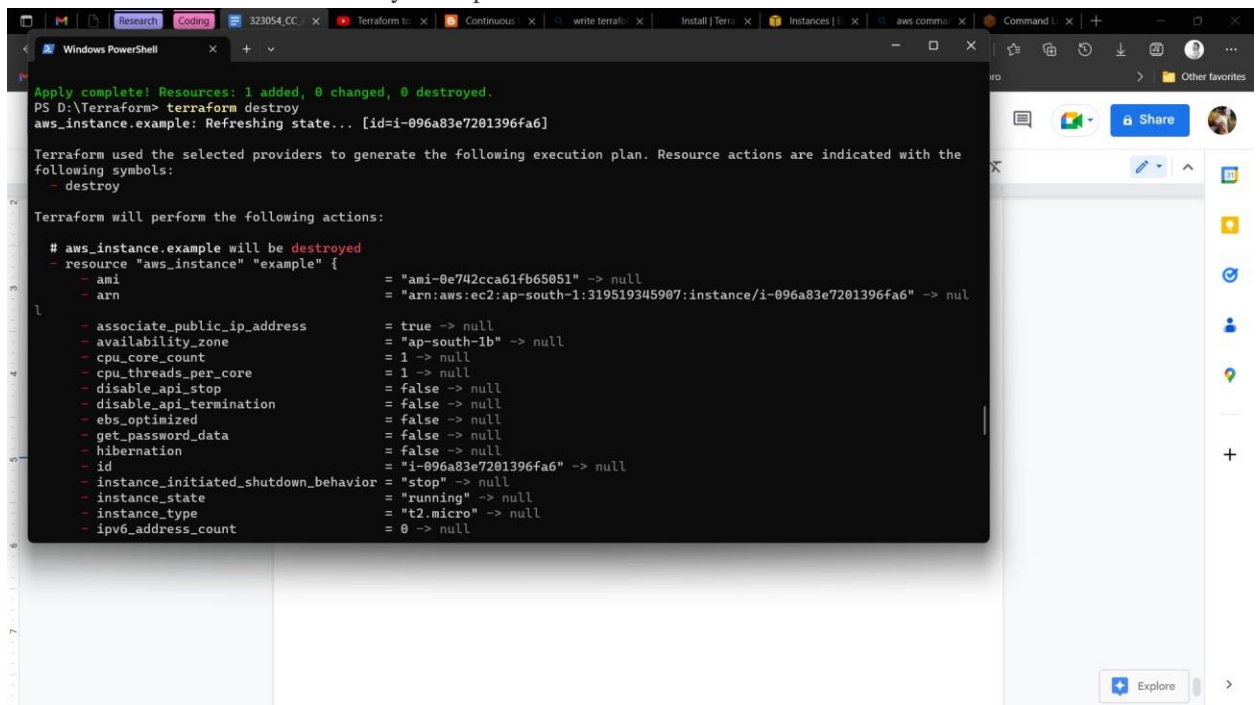
aws_instance.example: Creating...
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Still creating... [20s elapsed]
aws_instance.example: Still creating... [30s elapsed]
aws_instance.example: Creation complete after 32s [id=i-096a83e7201396fa6]

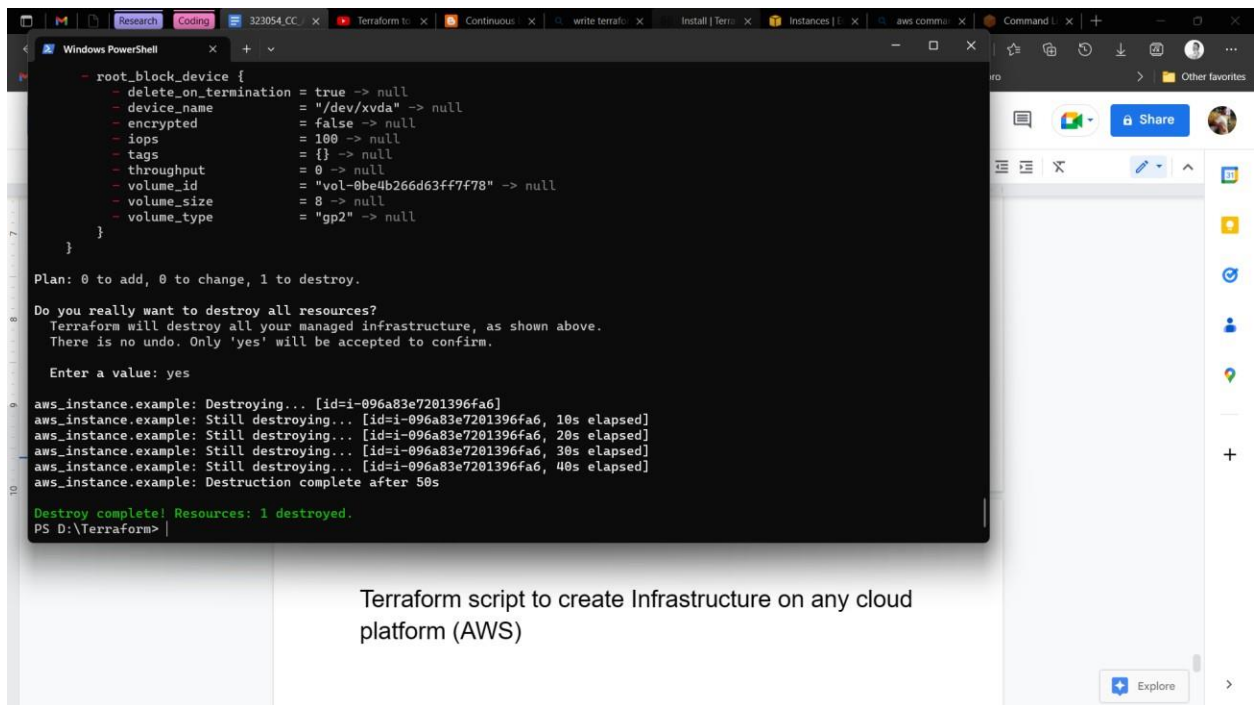
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
PS D:\Terraform>
```

See the create AWS “example” instance

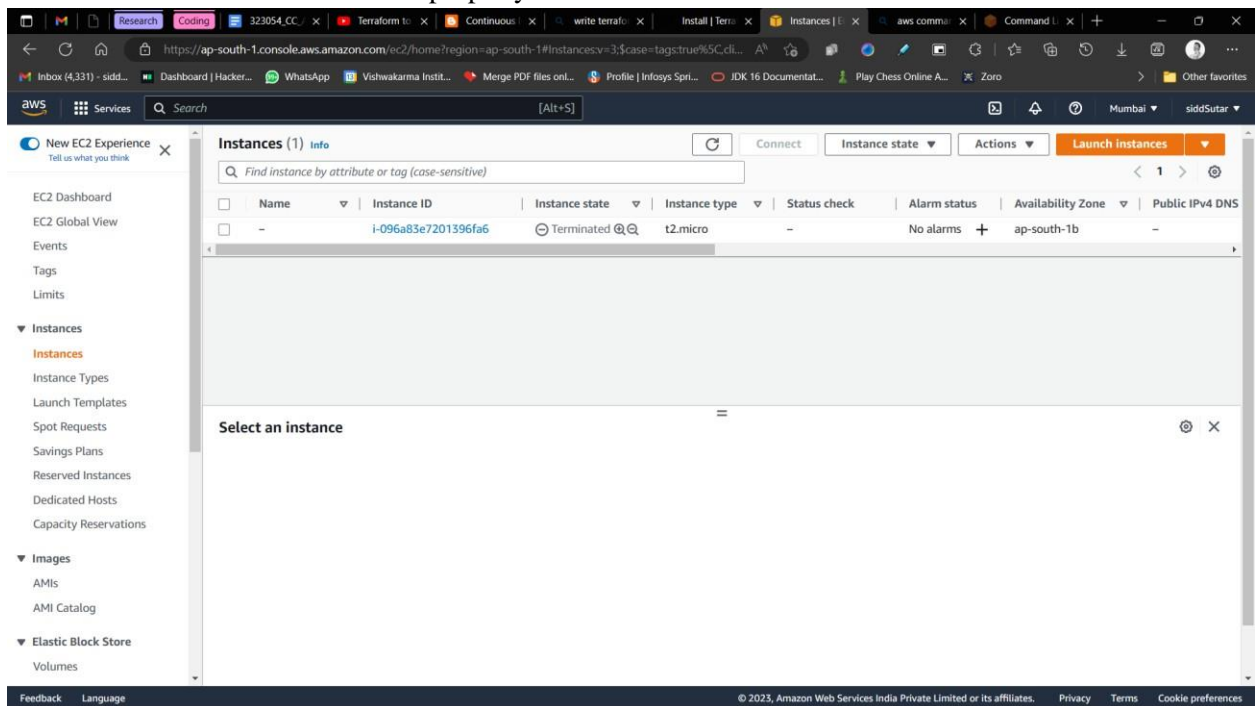


Put the command terraform destroy to stop the instance

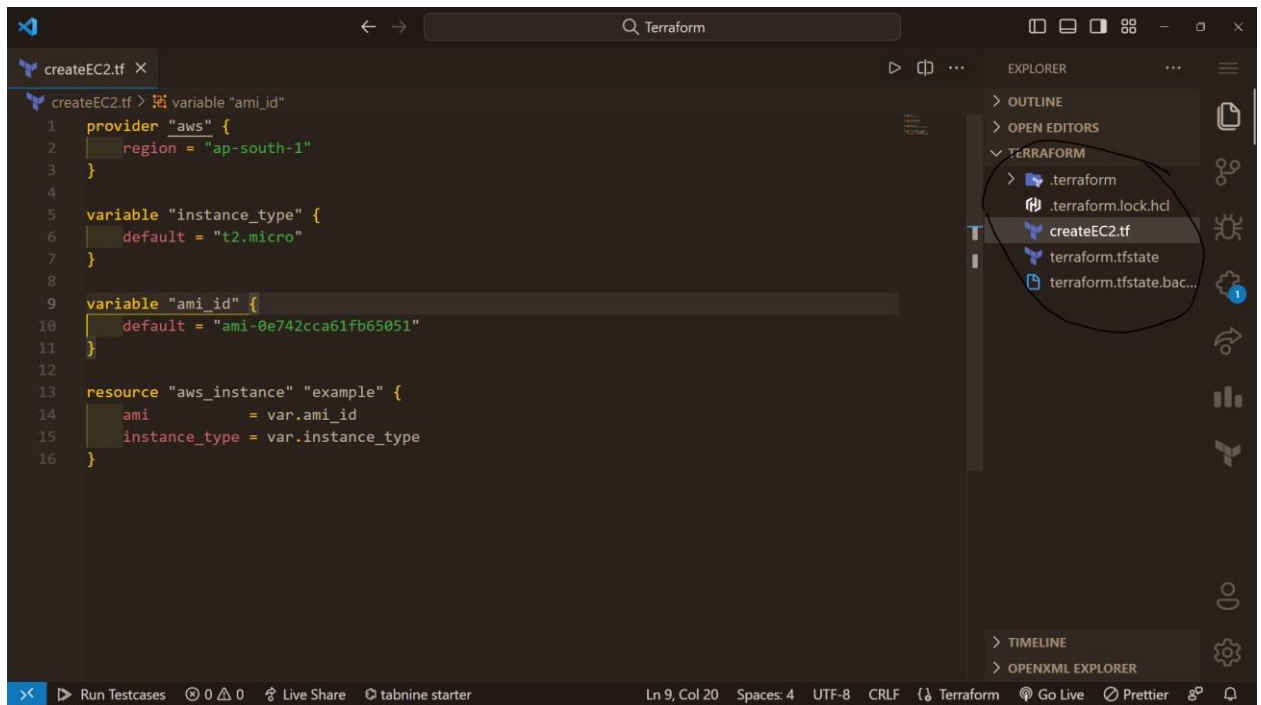




Let's check whether the instance is properly terminated or not



The final file should look like this



## Terraform json code

```
provider "aws" {
  region = "ap-south-1"
}

variable "instance_type" {
  default = "t2.micro"
}

variable "ami_id" {
  default = "ami-0e742cca61fb65051"
}

resource "aws_instance" "example" {
  ami           = var.ami_id
  instance_type = var.instance_type
}
```

# Conclusion

→ Terraform is understood alongside its basic commands.