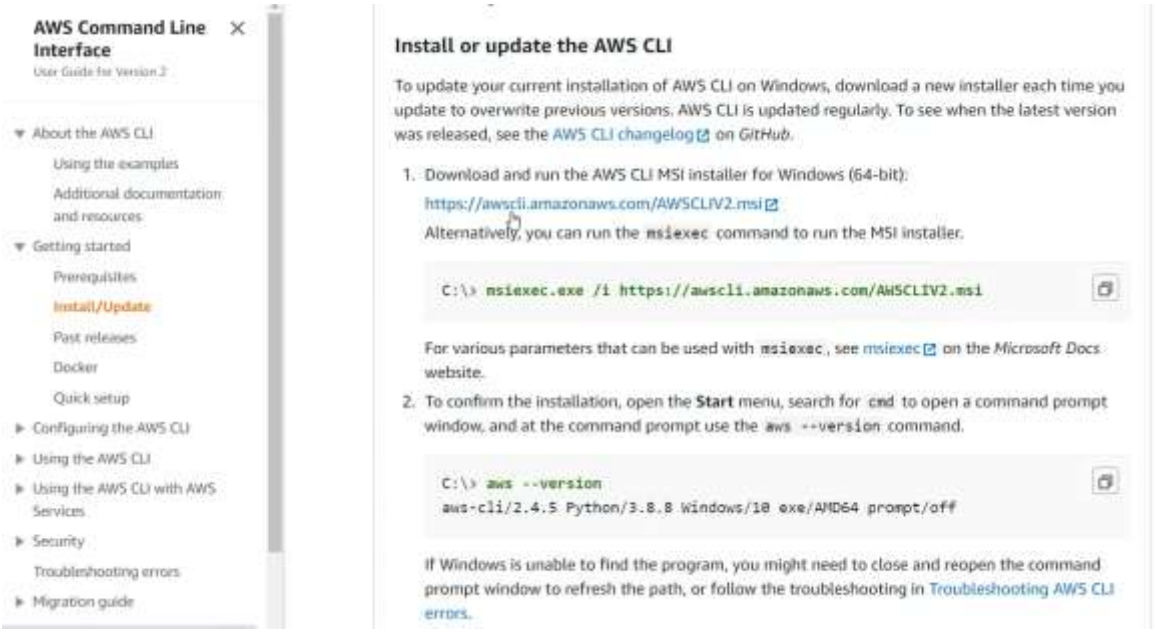
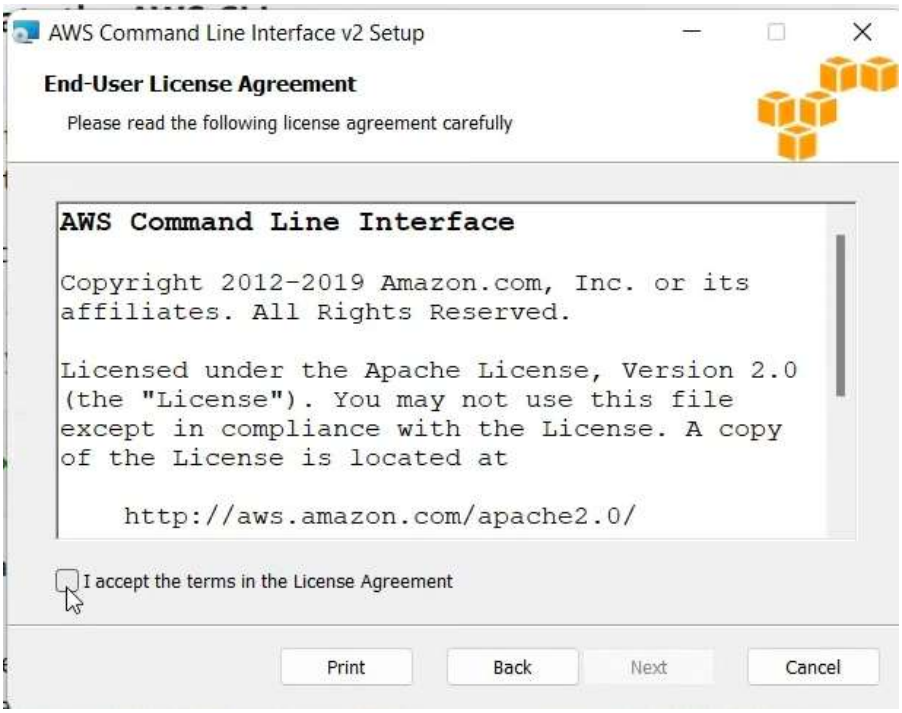
Step 1: Install and configure aws CLI

Search for **aws cli download** and click on the link

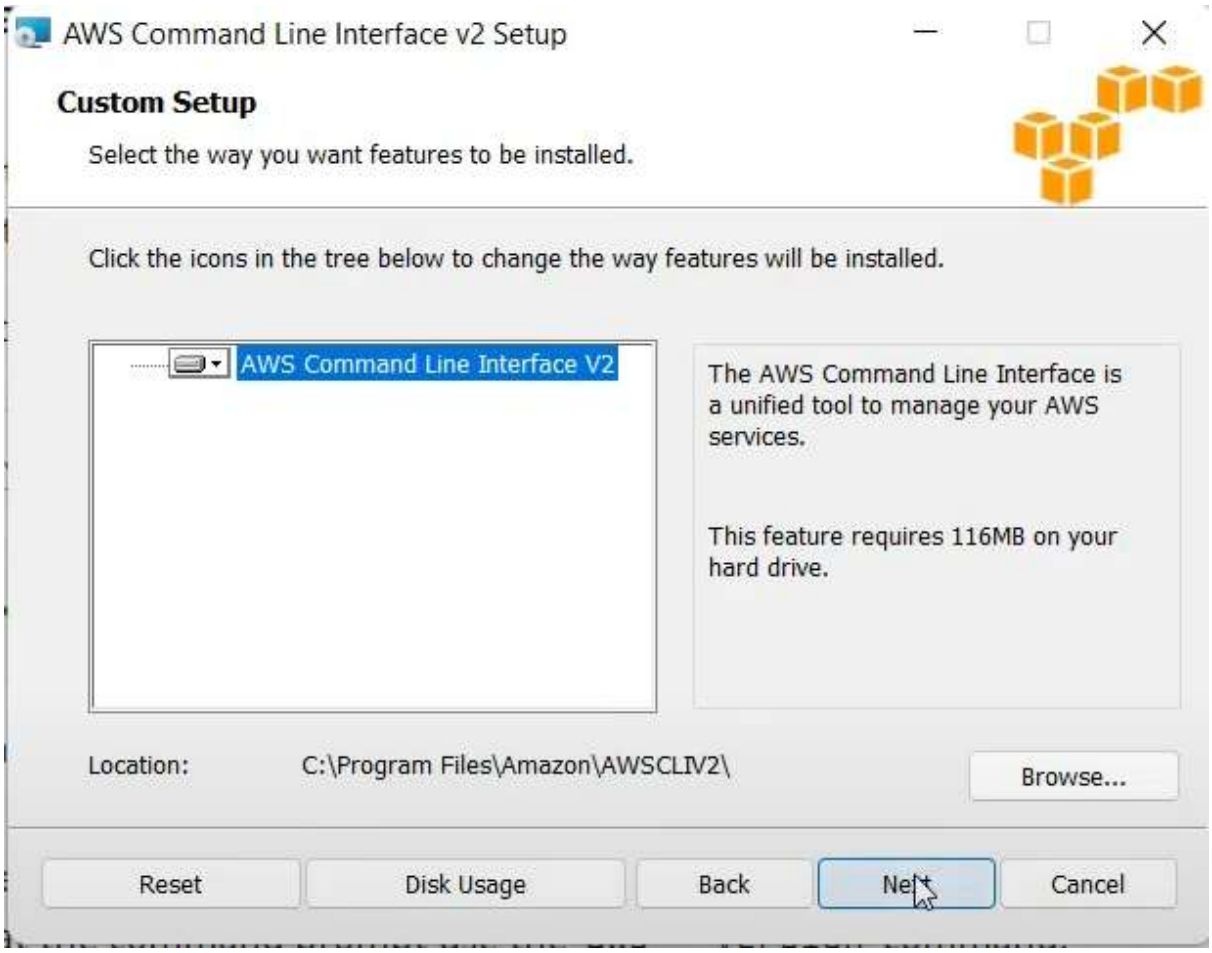
Click on the link for the **msi(Micro Star International) file** the download process of the msi file will start automatically



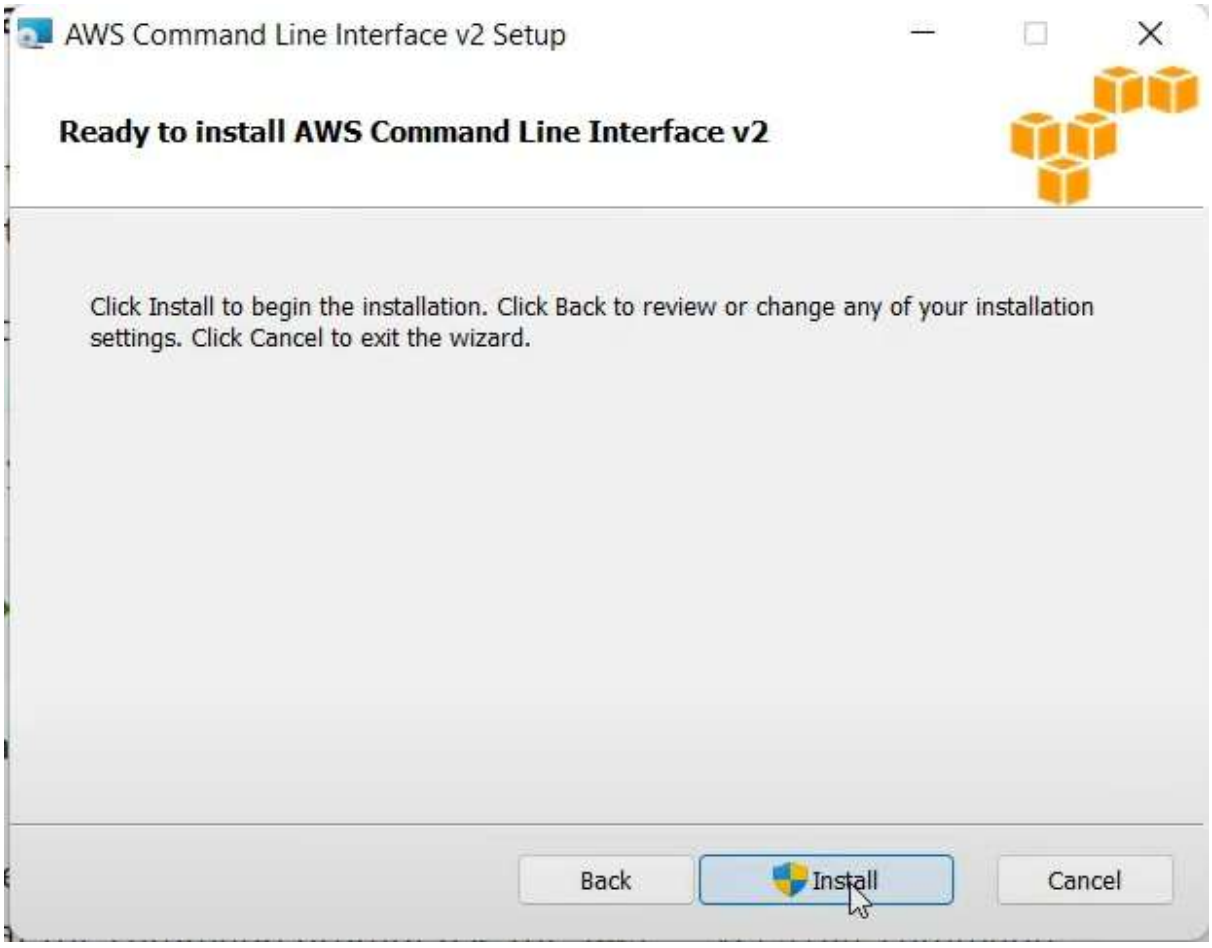
as you click on the **installed msi** file the following will get displayed click on **accept** the terms in the License Agreement



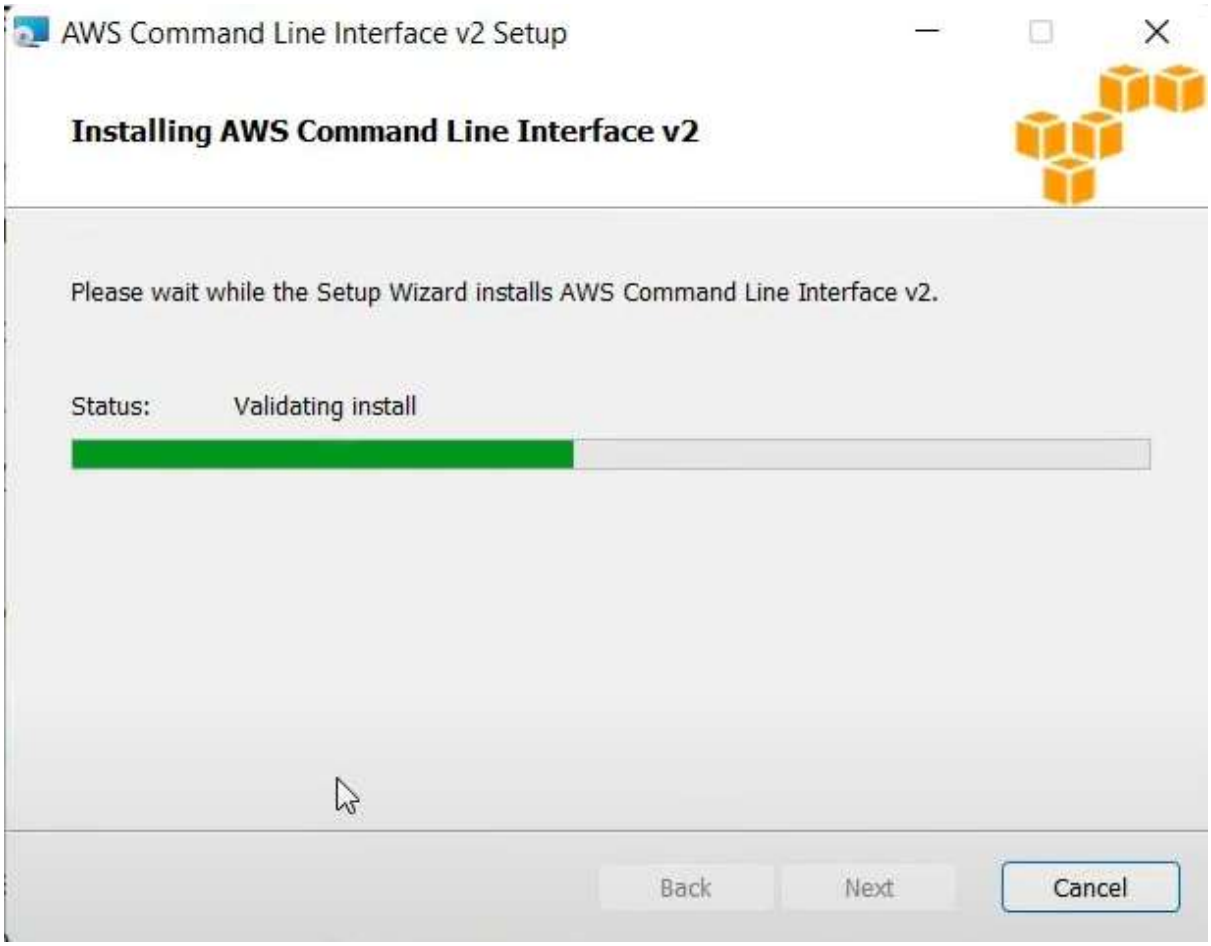
then cick on next



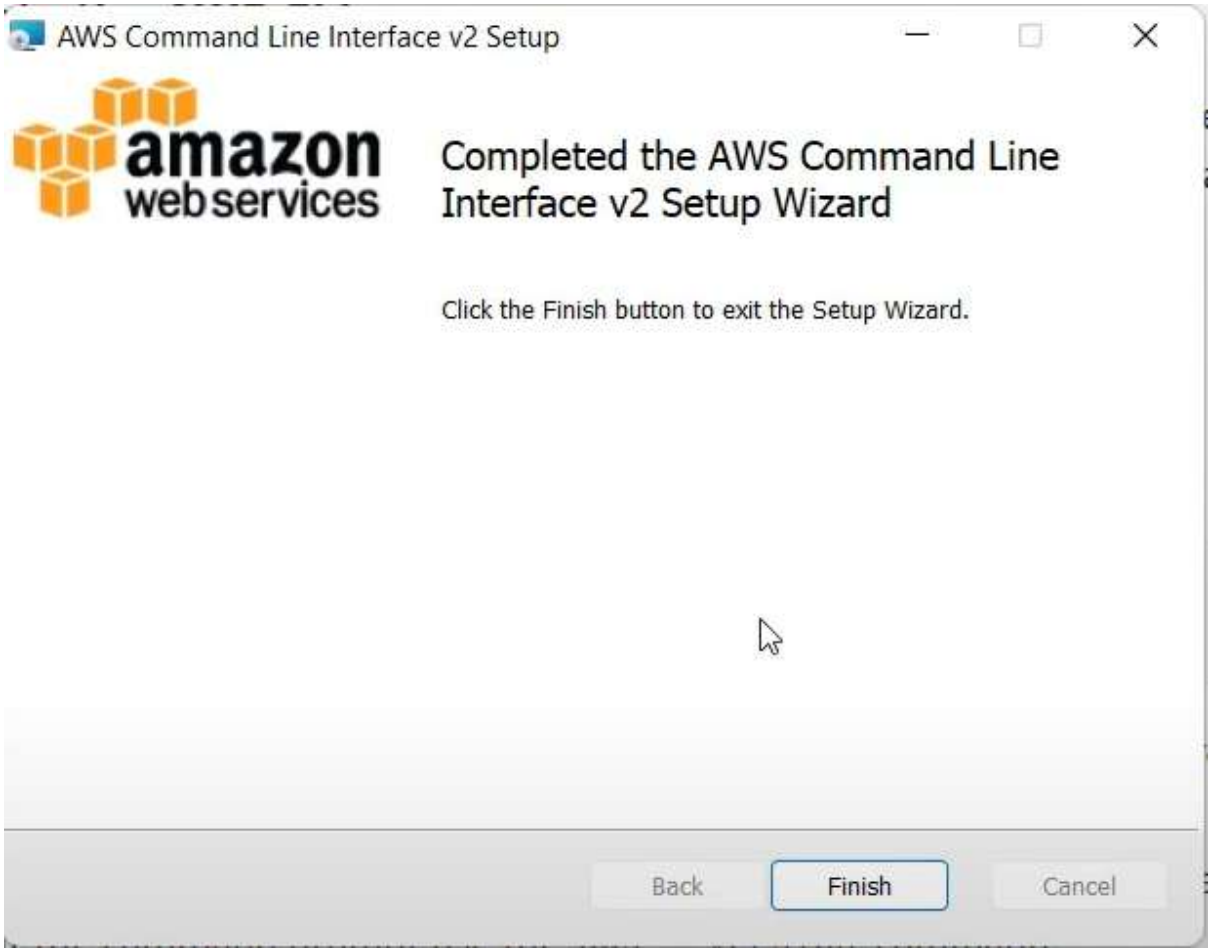
then click on **install**



finally it will start installing **aws cli** on the local machine



After installation is finished click on finish



Go to local machine cmd and type command **aws –version** the output should be **aws-cli/2.18.8 Python/3.12.6 Windows/11 exe/AMD64** the versions can be different

```
C:\Users\ganes>aws --version
aws-cli/2.18.8 Python/3.12.6 Windows/11 exe/AMD64
```

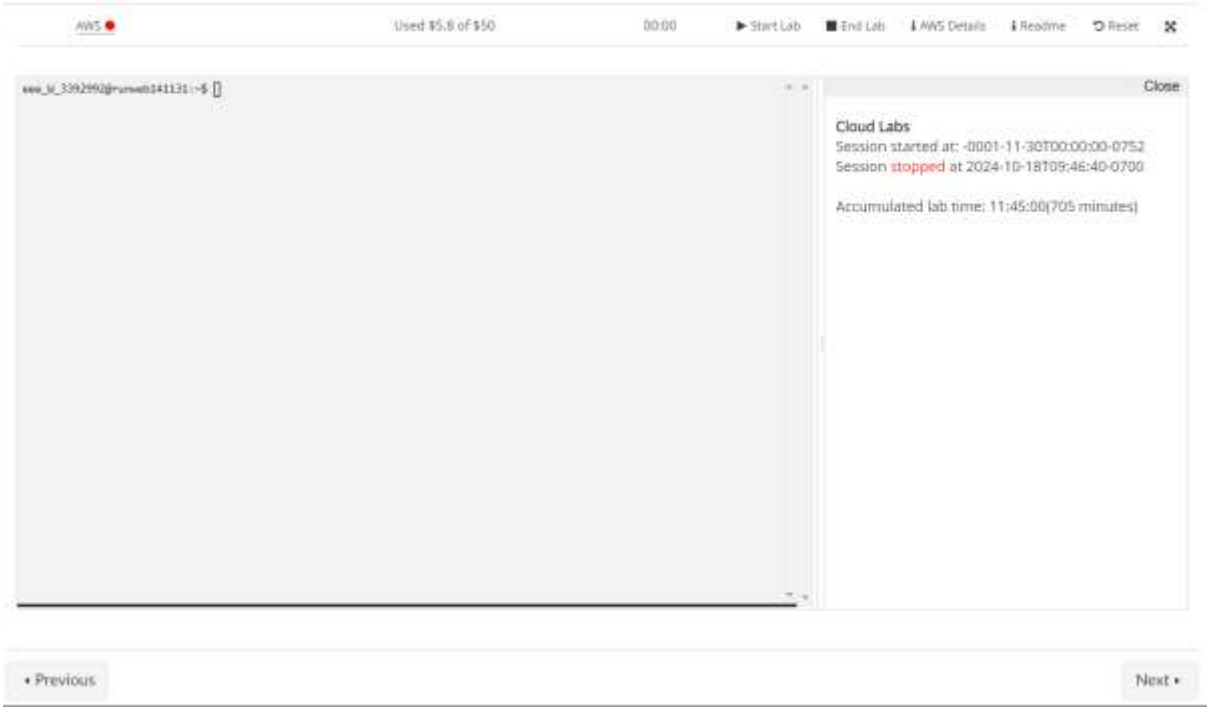
In order to **configure aws cli** there are **2 methods** to do so **:**
go on the aws academy site, provided by the college but remember it has only the student access and student doesn't has the access to create new user in IAM if you try to create it the following error will pop up

Method 1(create new user in IAM):

The second method(configure using temporary credentials, note it will be different for each lab session):

a)This is how you can get the credentials

when the lab is not started there are no credentials allotted to the user



b)as we can see there no credentials available here so in order to get the credentials we have to click on start the lab

c)Go to cmd and type the command “aws configure” and enter the details from the aws cli details above

```
C:\Users\ganes>aws configure
AWS Access Key ID [****************LK73]:
AWS Secret Access Key [****************mXc7]:
Default region name [us-west-2]:
Default output format [json]:
```

d)As we know this is session details which is temporary we also have to enter the additional command “aws configure set aws_session_token <Your_Session_Token>”

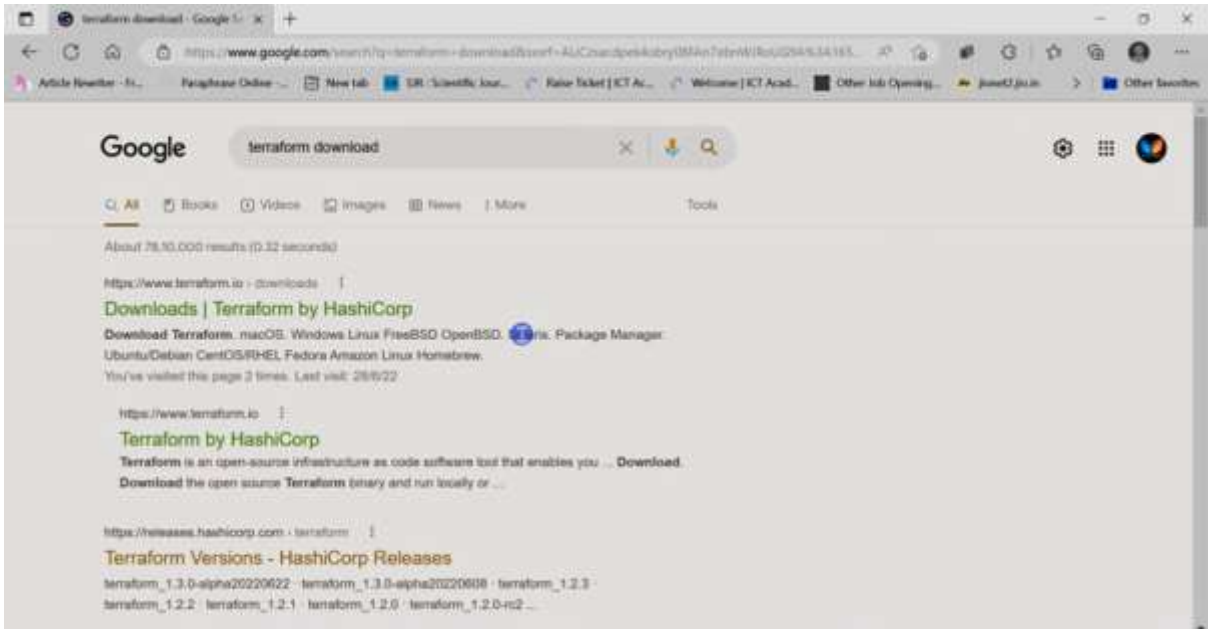
```
C:\Users\ganes>aws configure set aws_session_token IQoJb3JpZ2luX2VjEO3//////////wEaCXVzLXdlc3QtMiJIMEYCIQCyGhckwi0r0GeJT
7gKGWeq0C83IBa6W1ZcihnPpGTa9QIhAJ+Jz56NZv0JHpsoHxiXqMBZ+n5GVngZ/jNM613ar4ZRKrACCFYQAhoMMzM2MzI2MTkwNDUyIgxVZLlD9MD5r62OG
5EqjQJ0R8LLIn95JZrhy+Gq+nnTeLlrh8qJ2dkCIAiMIdz5JfAA/bTQED6Rj+03imJpA4UlUFh8sP/5EEvV5dg5Qhw/dt3fPd6xsY7IsdjzFYwkaViE5dJ9Z
AFIi8FXIpmqOMn81PjXe49DkF9cz8wWSdmmFgMGnJ8ZoGzIJYMgrf7vkMFFIbUIGl00mb8mB9apbEmm9yWLgnpplTREPoF7wK6A31zwWqhXx3p6uZgsSs3Xl
B64GsOnQnC5UoEFPZT4CHeabULXyqTOlgNk9yUD+hkAb+UU/vDtAV7VGkdkCLhV/ZIXYEQCfWRN+6Dh0nKd6eNjn7gmsae2GCPfoiDDm2AsluhL9jkncm/XT
o+8ATDe8sy4BjqcAXGE68jP1YT6q4KOhoN0LjFmgk136PzcOr62YTiwedPLtVr62FSP2TjcT0NL80cEEhOIWDAi0Q/z/gkKN3TVFl0MpBSRogAH7rDpKwm+0
MPcHyDsraJ/X//IH83mOhclMpAdZNNYRF8K7eO+QOvWbByl7ZhoY9Wu03RLIhZk1mJxi5GIq99yXCIGSRtPFmujjrkb59qW87Cl8ugZXg==
```

Cli will finaly get configured to our aws academy account

And you can confirm that by using command “aws ec2 list-instances”

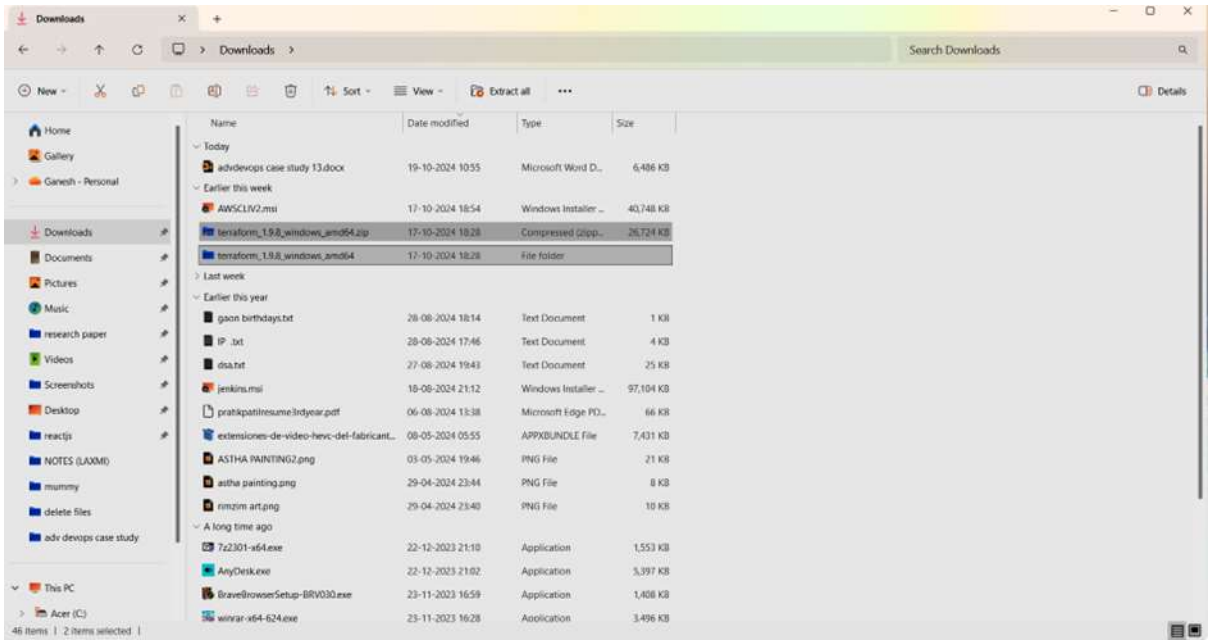
Step 2: **Install and configure Terraform**

Download the zip file of the terraform for the windows from the official website of the hashicorp

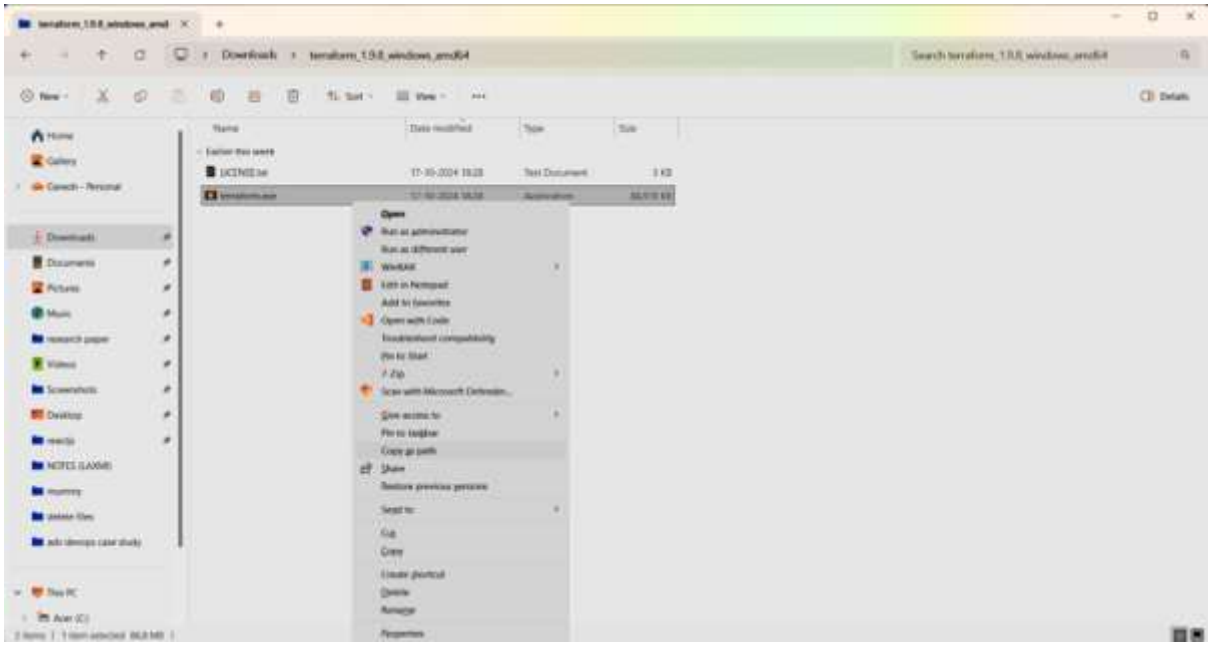


in my case I have downloaded the amd64 you can download any on them whichever supports your machine

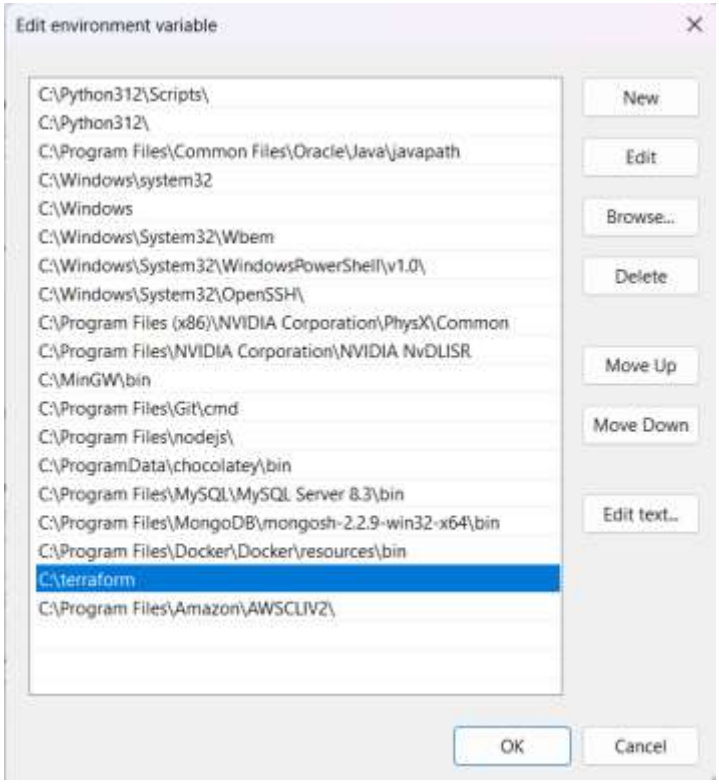
after downloading it you have to **unzip** it



Navigate inside of the unzip file and **copy it's path**



And add that path to environment variables



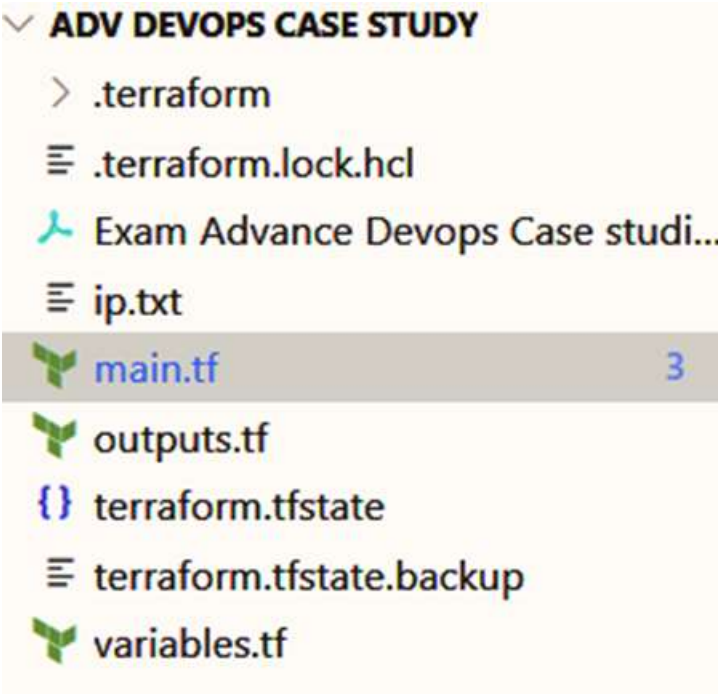
Check the terraform through cmd using **terraform –version**

```
C:\Users\ganes>terraform --version
Terraform v1.9.8
on windows_amd64
```

Step 3: **write the terraform code**

make the following directory structure in that new working folder for ex: adv devops case study

```
terraform file structure :


adv devops case study/
|
├── main.tf              Main Terraform configuration file
├── variables.tf         Variable definitions
├── outputs.tf           Outputs from Terraform configuration
└── ip.txt               This file will be created by Terraform during execution
```



main.tf terraform code :

```hcl
provider "aws" {
  region = "us-east-1" # desired region
}

resource "aws_instance" "example" {
  ami           = "ami-0ddc798b3f1a5117e" # valid AMI ID for your region
  instance_type = "t2.micro"
}

resource "aws_s3_bucket" "example" {
  bucket = "my-terraform-bucket-example" # Ensure this is a unique bucket name
  acl    = "private"
}

# Create a local file with the EC2 instance's public IP
resource "null_resource" "create_ip_file" {
  provisioner "local-exec" {
    command = "echo ${aws_instance.example.public_ip} > ip.txt"
  }

  depends_on = [aws_instance.example]
}

# Upload the IP file to the S3 bucket
resource "aws_s3_bucket_object" "ip_object" {
  bucket = aws_s3_bucket.example.bucket
  key    = "ec2-ip.txt"
  source = "ip.txt"
  acl    = "private"
}
```

output.tf code :

outputs.tf

```
output "s3_bucket_url" {
  value = aws_s3_bucket.example.bucket
}

output "ec2_public_ip" {
  value = aws_instance.example.public_ip
}
```

variables.tf

```
variable "ami" {
  description = "The AMI ID for the instance"
  type        = string
}

variable "instance_type" {
  description = "The instance type"
  type        = string
  default     = "t2.micro"
}
```

Step 4: **run terraform commands**

a) "terraform init" command

```
C:\Users\ganes\Desktop\adv devops case study>terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/null from the dependency lock file
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/null v3.2.3
- Using previously-installed hashicorp/aws v5.72.1

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.

C:\Users\ganes\Desktop\adv devops case study>
```

b) " terraform fmt" command:

```
C:\Users\ganes\Desktop\adv devops case study>terraform fmt
```

c) "terraform validate" command:

```
C:\Users\ganes\Desktop\adv devops case study>terraform validate

Warning: Argument is deprecated

  with aws_s3_bucket.example,
  on main.tf line 12, in resource "aws_s3_bucket" "example":
  12:   acl    = "private"

Use the aws_s3_bucket_acl resource instead

(and one more similar warning elsewhere)

Warning: Deprecated Resource

  with aws_s3_bucket_object.ip_object,
  on main.tf line 25, in resource "aws_s3_bucket_object" "ip_object":
  25: resource "aws_s3_bucket_object" "ip_object" {

use the aws_s3_object resource instead

Success! The configuration is valid, but there were some validation warnings as shown above.
```

d) "terraform plan" command:

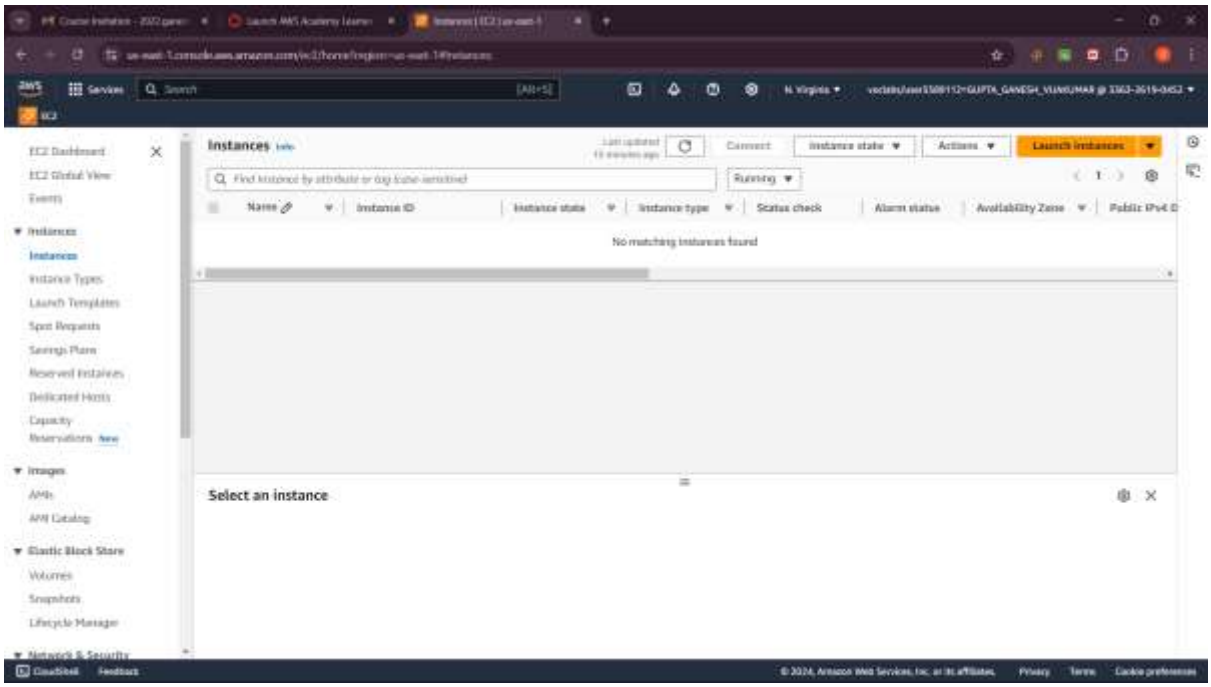
```
C:\Users\ganes\Desktop\adv devops case study>terraform plan
var.ami
  The AMI ID for the instance

  Enter a value: yes
```

enter yes and enter to see the plan that is going to execute on aws

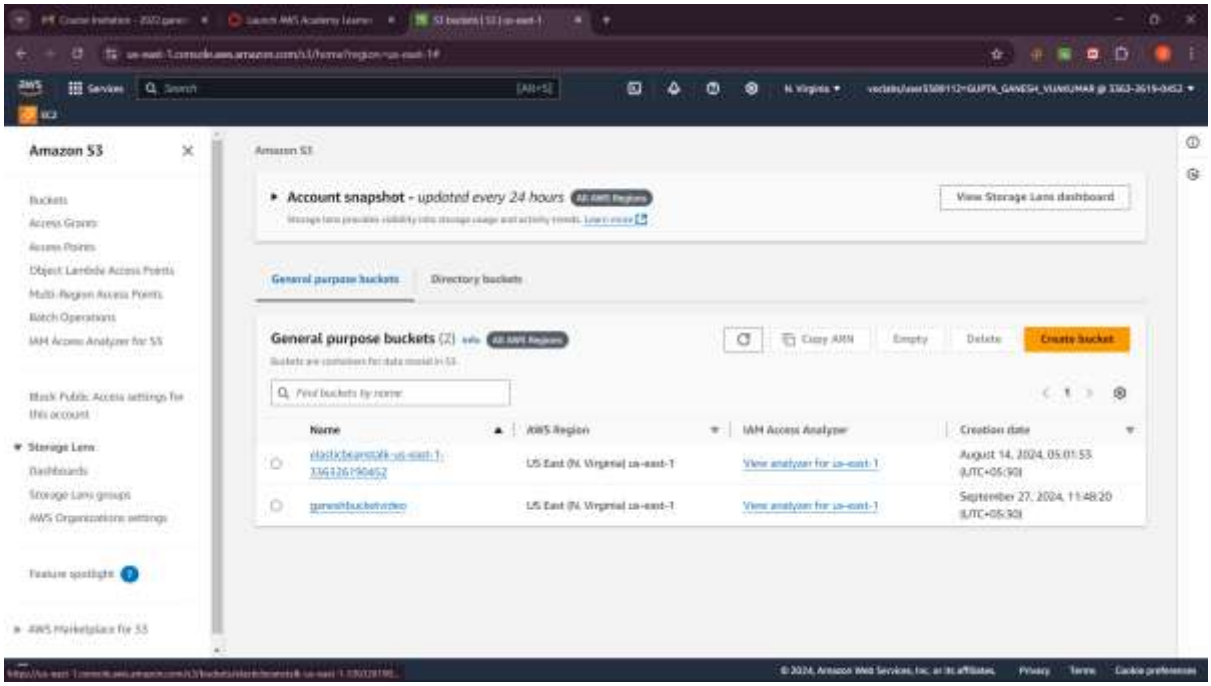
```
C:\Users\ganes\Desktop\adv devops case study>terraform plan
var.ami
  The AMI ID for the instance

  Enter a value: yes


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-0ddc798b3f1a5117e"
      + 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)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
```

```
  # aws_s3_bucket.example will be created
  + resource "aws_s3_bucket" "example" {
      + acceleration_status         = (known after apply)
      + acl                         = "private"
      + arn                         = (known after apply)
      + bucket                      = "my-terraform-bucket-example"
      + bucket_domain_name          = (known after apply)
      + bucket_prefix               = (known after apply)
      + bucket_regional_domain_name = (known after apply)
      + force_destroy               = false
      + hosted_zone_id              = (known after apply)
      + id                          = (known after apply)
      + object_lock_enabled         = (known after apply)
      + policy                      = (known after apply)
      + region                      = (known after apply)
      + request_payer               = (known after apply)
      + tags_all                    = (known after apply)
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)

      + cors_rule (known after apply)

      + grant (known after apply)

      + lifecycle_rule (known after apply)

      + logging (known after apply)

      + object_lock_configuration (known after apply)
```

Before executing “terraform apply” command there is no ec2 instance running



Before executing “terraform apply” command there are only 2 s3 bucket instances and note that every s3 bucket has unique name



e) Executing “terraform apply” command

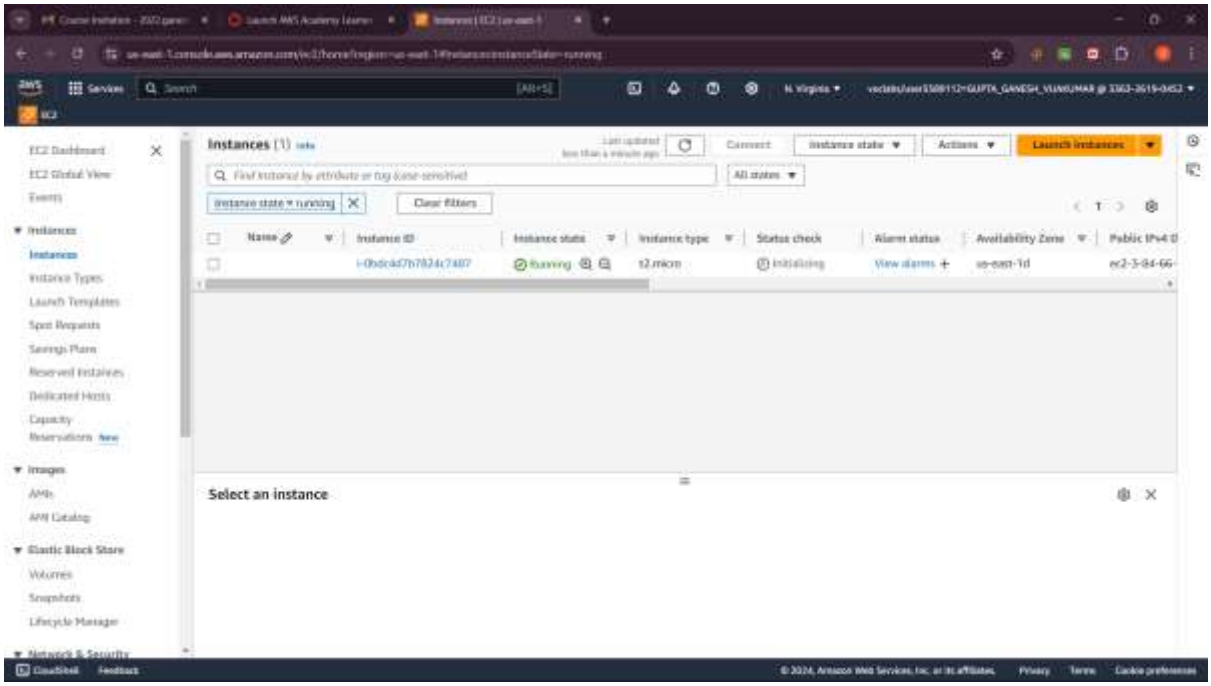
```
C:\Users\ganes\Desktop\adv devops case study>terraform apply
var.ami
  The AMI ID for the instance

  Enter a value:
```

```
C:\Users\ganes\Desktop\adv devops case study>terraform apply
var.ami
  The AMI ID for the instance

  Enter a value: yes


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:
```

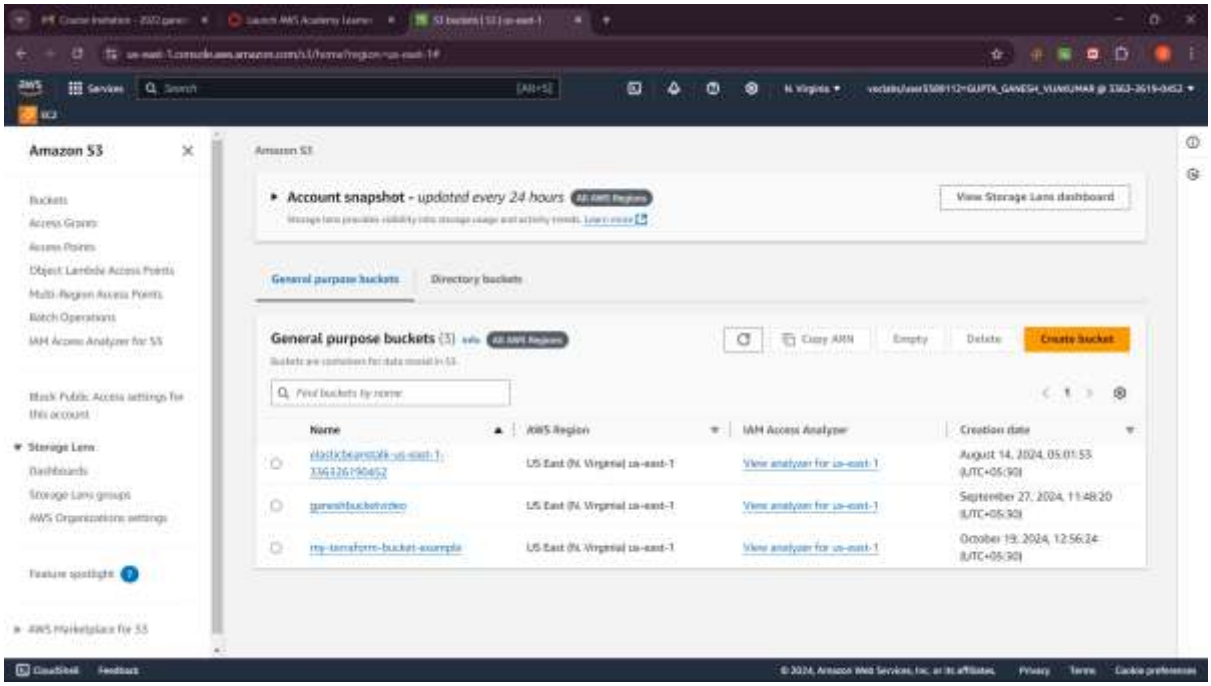
```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_s3_bucket.example: Creating...
aws_instance.example: Creating...
aws_s3_bucket.example: Creation complete after 5s [id=my-terraform-bucket-example]
aws_s3_bucket_object.ip_object: Creating...
aws_s3_bucket_object.ip_object: Creation complete after 0s [id=ec2-ip.txt]
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Creation complete after 16s [id=i-0bdc4d7b7824c7487]
null_resource.create_ip_file: Creating...
null_resource.create_ip_file: Provisioning with 'local-exec'...
null_resource.create_ip_file (local-exec): Executing: ["cmd" "/C" "echo 3.84.66.70 > ip.txt"]
null_resource.create_ip_file: Creation complete after 0s [id=8958148951290668 68]

Warning: Argument is deprecated

  with aws_s3_bucket.example,
  on main.tf line 12, in resource "aws_s3_bucket" "example":
  12:   acl    = "private"

Use the aws_s3_bucket_acl resource instead

(and 2 more similar warnings elsewhere)

Warning: Deprecated Resource

  with aws_s3_bucket_object.ip_object,
  on main.tf line 25, in resource "aws_s3_bucket_object" "ip_object":
  25: resource "aws_s3_bucket_object" "ip_object" {

use the aws_s3_object resource instead

Apply complete! Resources: 4 added, 0 changed, 0 destroyed.

Outputs:

ec2_public_ip = "3.84.66.70"
s3_bucket_url = "my-terraform-bucket-example"

C:\Users\ganes\Desktop\adv devops case study>
```

Now check the aws ec2 instances and s3 bucket at academy

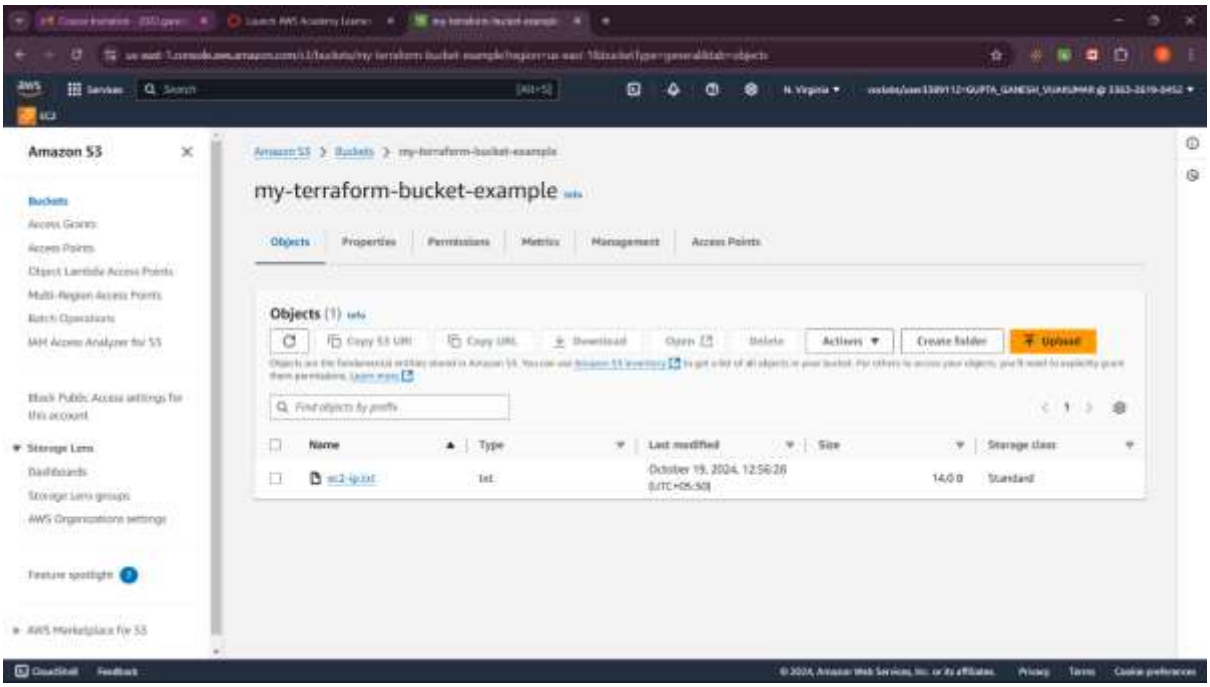
1 ec2 instance is created



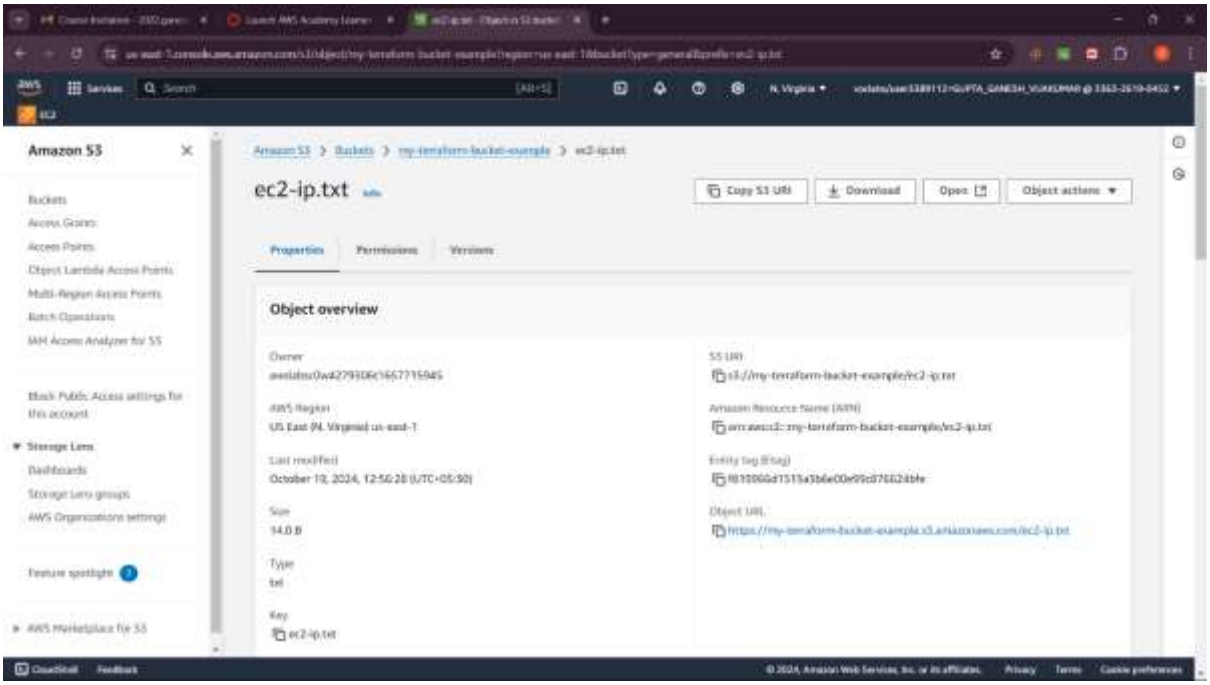
1 s3 bucket is also created



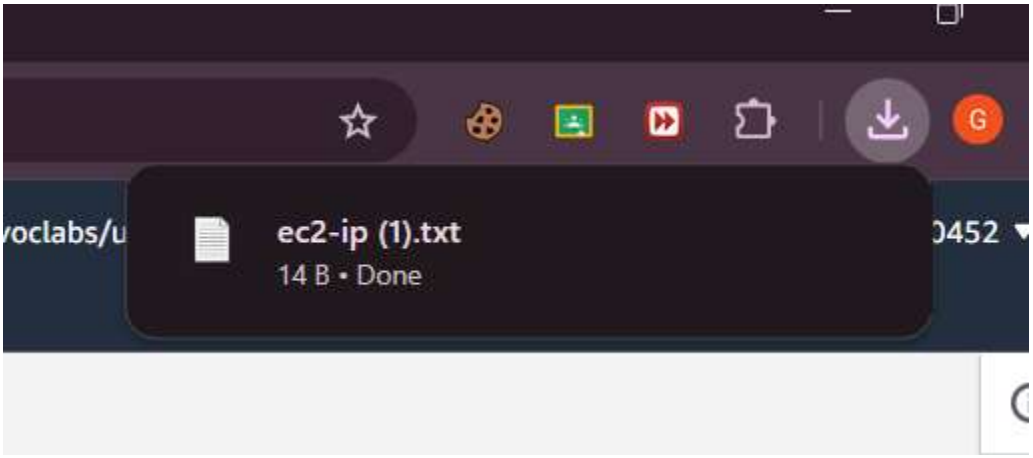
Now click on s3 bucket that is newly created



Now click on ec2-ip.txt



Click on download

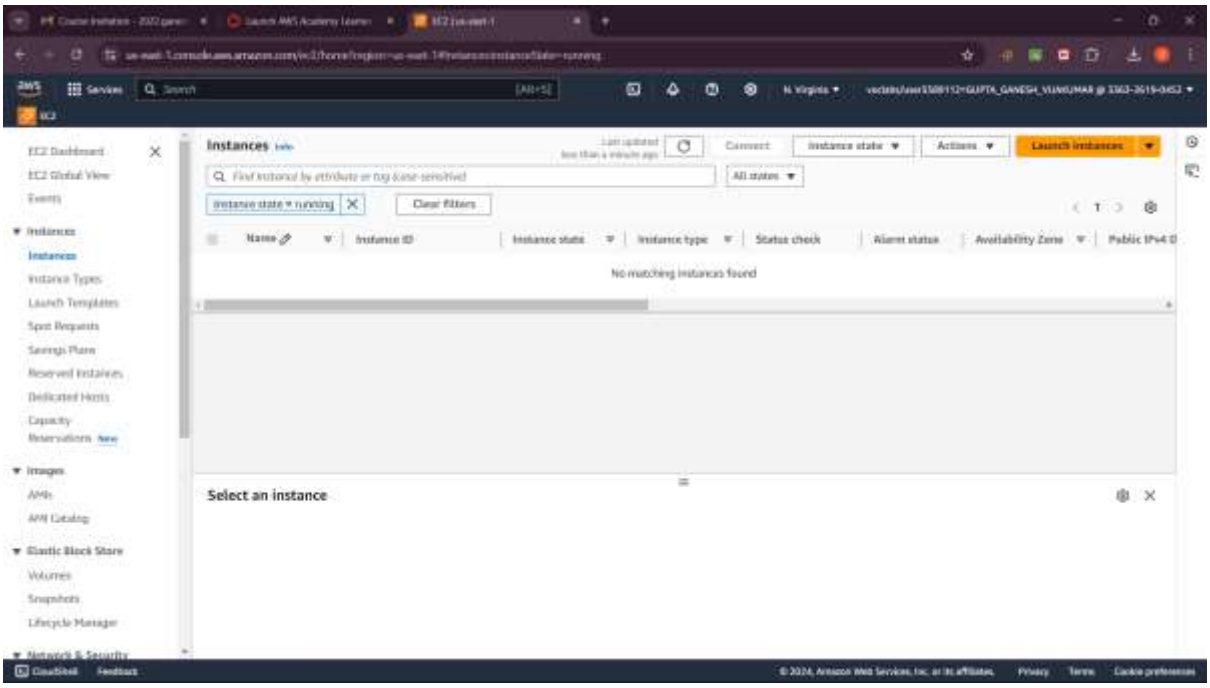


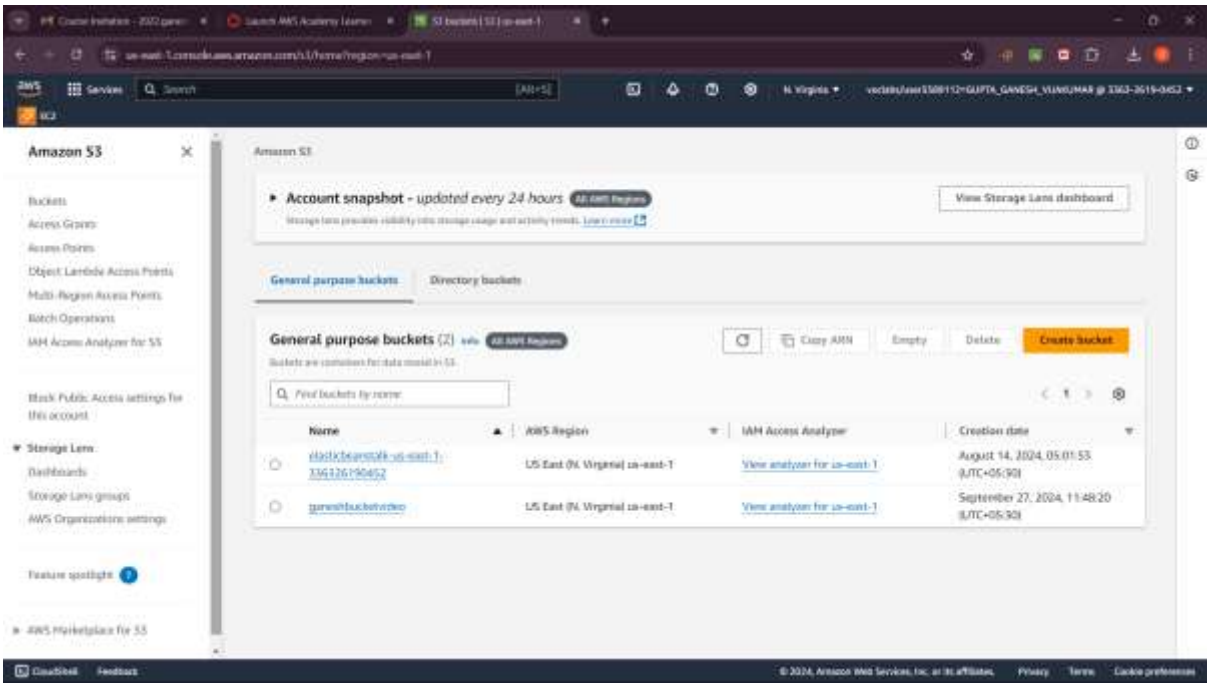
Open it

The ip address is stored inside the s3 bucket is finally proved

And then use terraform destroy command to destroy all the instances that are made

```
C:\Users\ganes\Desktop\adv devops case study>terraform destroy
var.ami
  The AMI ID for the instance

  Enter a value: yes

aws_s3_bucket.example: Refreshing state... [id=my-terraform-bucket-example]
aws_instance.example: Refreshing state... [id=i-0bdc4d7b7824c7487]
aws_s3_bucket_object.ip_object: Refreshing state... [id=ec2-ip.txt]
null_resource.create_ip_file: Refreshing state... [id=8958148951290668 68]

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

Terraform will perform the following actions:

  # aws_instance.example will be destroyed
  - resource "aws_instance" "example" {
      - ami                          = "ami-0ddc798b3f1a5117e" -> null
      - arn                          = "arn:aws:ec2:us-east-1:336326190452:instance/i-0bdc4d7b7824c7487" -> null
      - associate_public_ip_address  = true -> null
```





EC2 instance and S3 bucket is destroyed.