TERRAFORM (IaC)

1) What is IaC and its advantages?

A) Infrastructure as Code (IaC) is the managing and provisioning of infrastructure through code instead of through manual processes. With IaC, configuration files are created that contain your infrastructure specifications, which makes it easier to edit and distribute configurations.

Advantages of IaC

- Consistency and Reproducibility.
- Version Control.
- Automation and Efficiency.
- Scalability and Flexibility.
- Documentation and Self-Documenting Code.
- Testing and Validation.
- Cross-platform and Cloud Agnostic.

2) Types of services can be done by using Iac?

A) 1) Infrastructure management.
'2) Configuration Management.

3) What is terraform and its alternatives?

A) Terraform is an open-source Infrastructure as Code (IAC) tool developed by HashiCorp. It allows you to define and provision infrastructure resources (such as virtual machines, networks, databases, and more) in a declarative manner using configuration files. It is designed to work with various cloud providers and infrastructure platforms, making it a popular choice for managing infrastructure across different environments.

Key features of Terraform

- Declarative Configuration.
- Infrastructure Versioning.
- State Management.
- Modularity and Reusability.

Terraform alternative tools are AWS CloudFormation, Google Cloud Deployment Manager, Azure Resource Manager (ARM) Templates, Ansible, Chef and Puppet.

4) Terraform terminology?

A) i) **Terraform Configuration:** It refers to the set of files and directories containing the Terraform code that defines your infrastructure.

ii) **Provider**: It allow Terraform to manage resources in various environments, such as AWS, Azure, Google Cloud, etc.

iii) **Resource:** It represents a single infrastructure component that Terraform manages, such as virtual machines, networks, databases, etc.

iv) **Module**: Modules allow you to create custom abstractions and share configurations between different projects.

v) **Outputs:** Terraform outputs allow you to expose information about your infrastructure to other resources or to the user. You can use outputs to share data between resources, or to display important information to the user.

vi) **Terraform Cloud**: Terraform Cloud is a hosted service that provides additional features for managing your Terraform infrastructure, such as collaborative workspaces, automated runs, and version control.

vii) **Remote State**: Terraform's remote state feature allows you to store your infrastructure state in a remote location, such as an S3 bucket or a Terraform Cloud workspace. This can be useful for collaboration and for keeping track of changes to your infrastructure.

viii) **State File**: The state file is automatically created when apply command entered and it is managed by Terraform. By default, it is stored in a local file named terraform.tfstate.

ix) **Variables**: Terraform variables allow you to define values that can be used throughout your infrastructure configuration. You can use variables to make your configuration more flexible and reusable, and to make it easier to manage complex infrastructure.

5) Terraform life cycle?

A) The typical life cycle of Terraform involves the following steps

i) **Initialize**: Run '**terraform init**' in the working directory to initialize Terraform for your project. This command downloads the necessary provider plugins and sets up the backend configuration.

ii) **Plan**: Run '**terraform plan**' to create an execution plan. It doesn't make any changes during this step. It only shows you what actions thus Terraform will take.

iii) **Apply**: Run '**terraform apply**' to execute the changes proposed in the plan. It will create, modify, or delete resources as needed to reach the desired state specified in the configuration files.

iv) **Destroy**: Run '**terraform destroy**' to remove all the resources created by Terraform based on the configuration.

6) Terraform installation and configuring with AWS cloud for infrastructure provisioning Purpose?

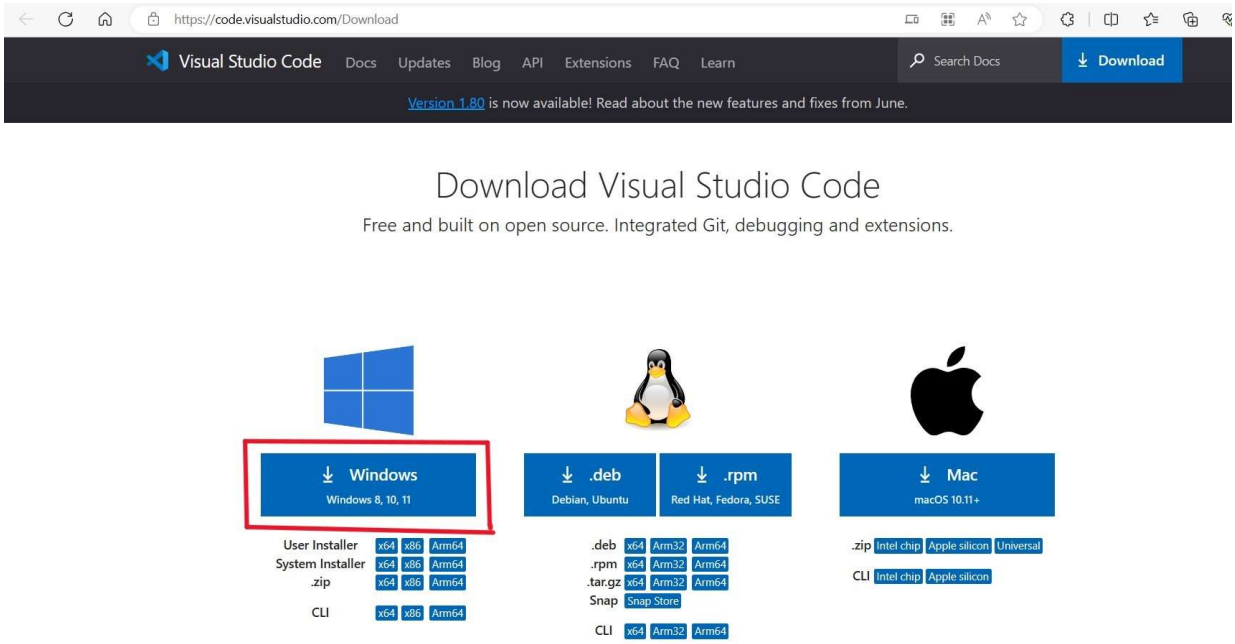
A) To install and configure terraform in windows machine

Pre-requisites are

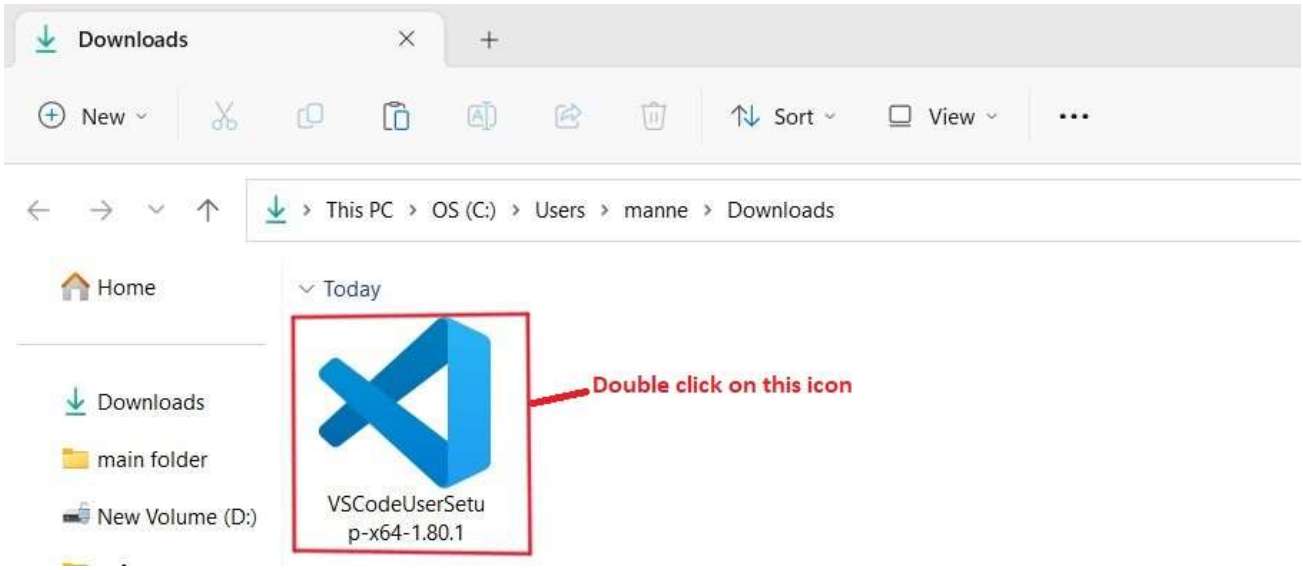
1) Visual Studio Code / VS code (Installation steps shown below)
2) AWS CLI (Installation steps shown below including screenshots)

To download: https://code.visualstudio.com/Download

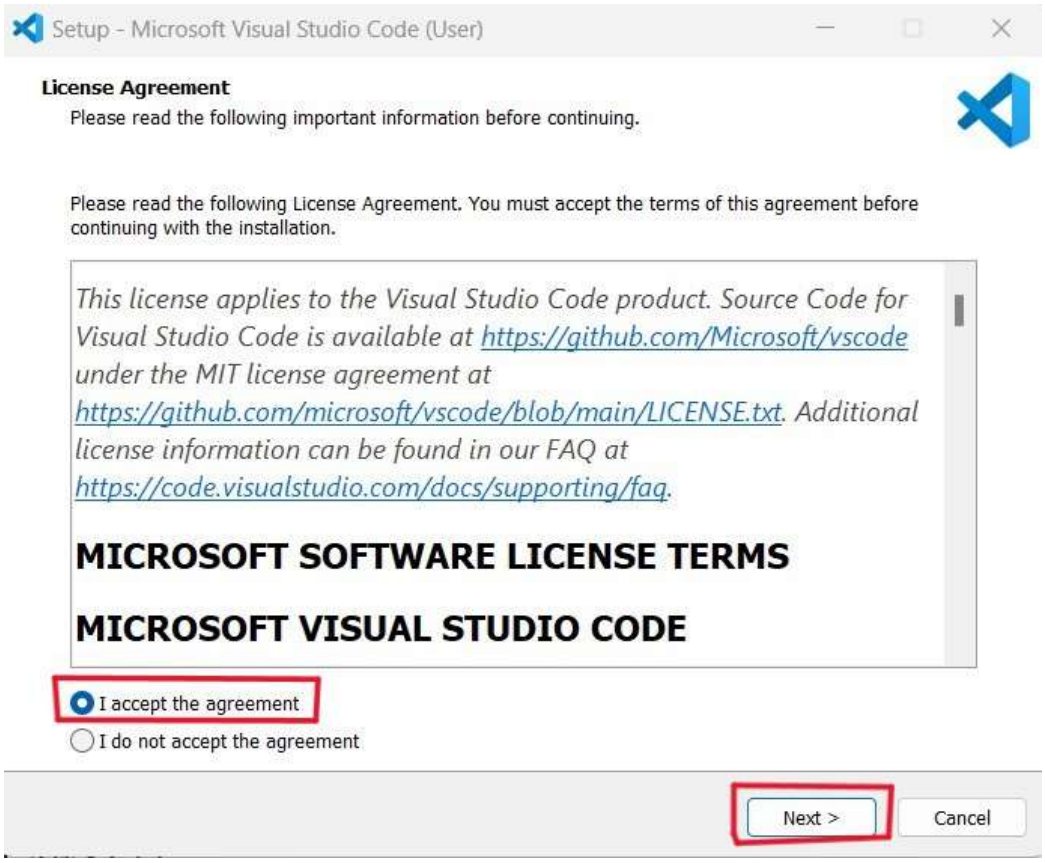
Steps as follow, select the highlighted box (Windows) shown in below image.



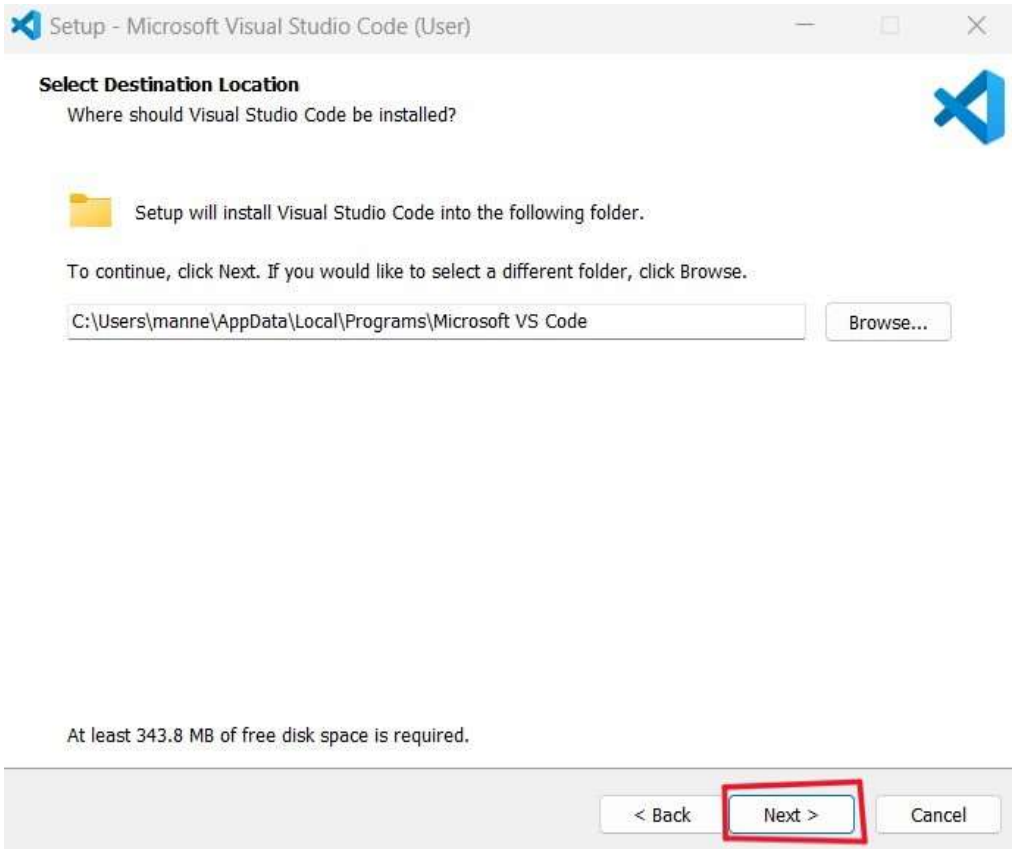
After completion of download, double click on that file



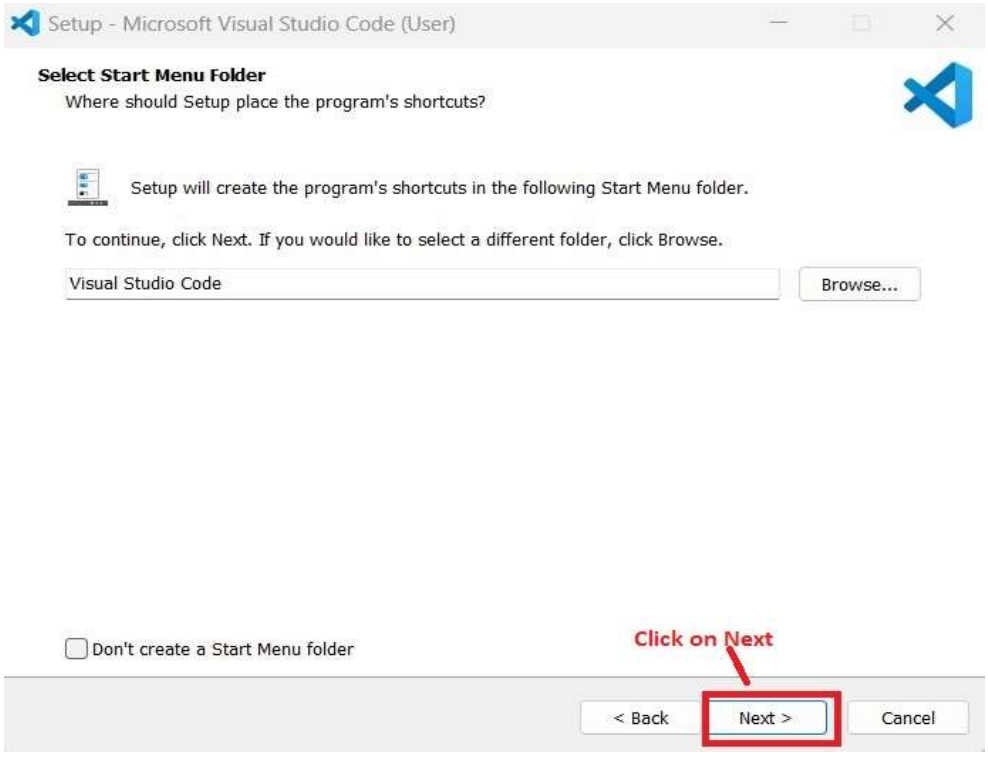
After double clicking on that icon, the following screen will appear.



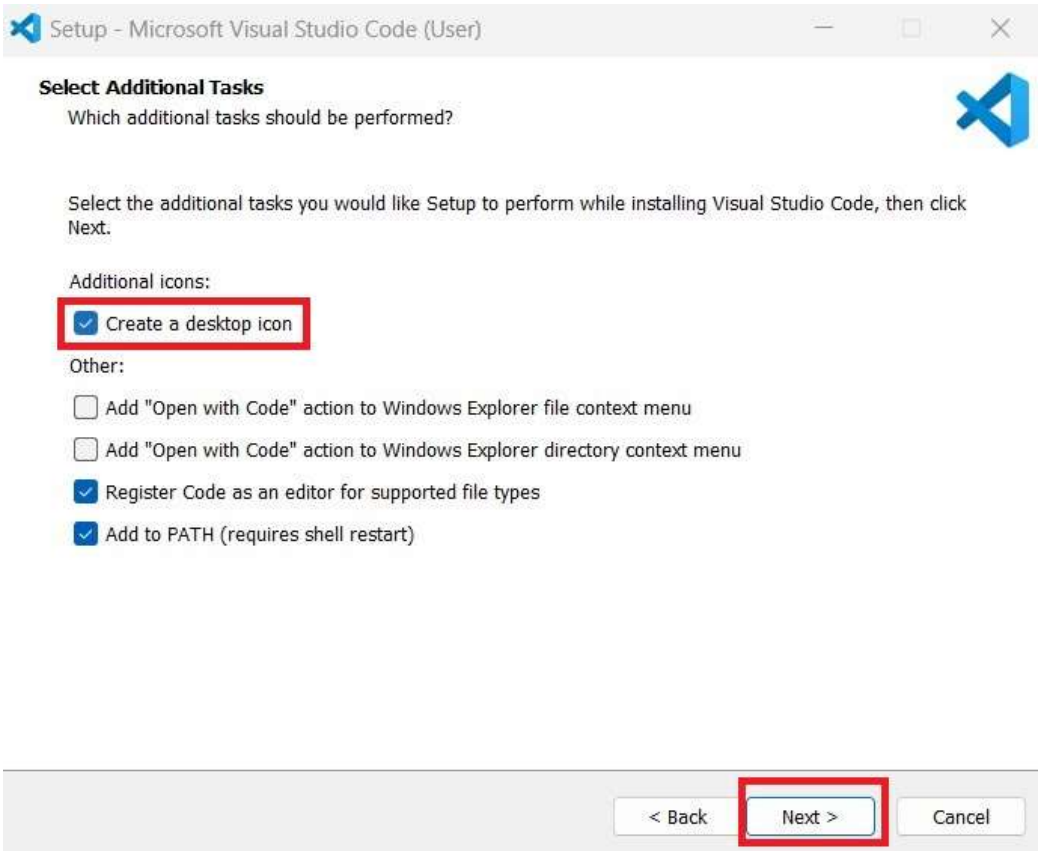
After clicking on Next button, the following screen will appear.



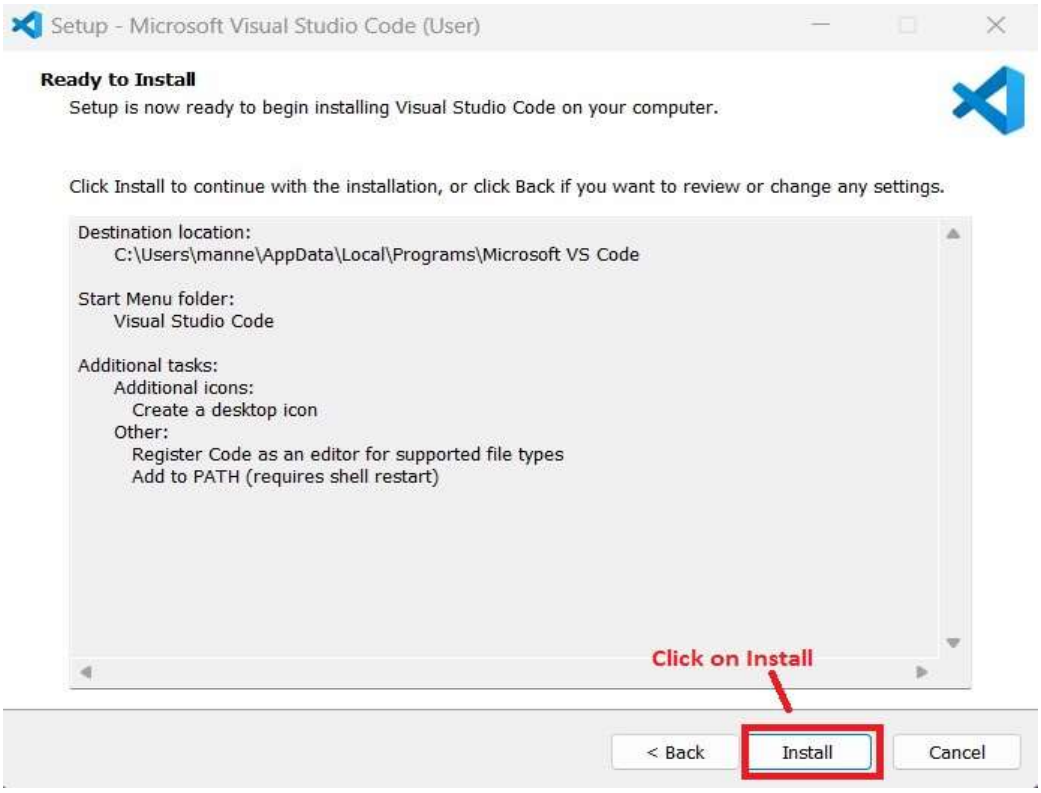
After clicking on Next button, the following screen will appear.



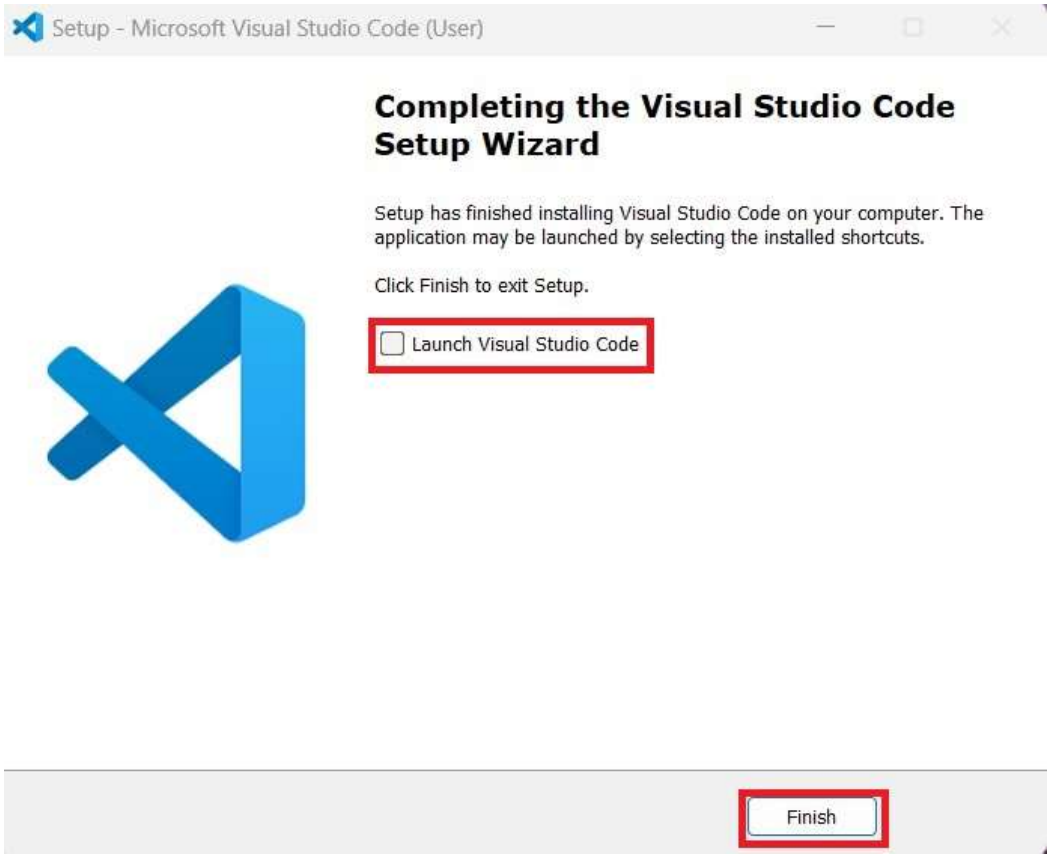
After clicking on Next button, the following screen will appear.



After clicking on Next button, the following screen will appear.



After clicking on Install button, the following screen will appear.

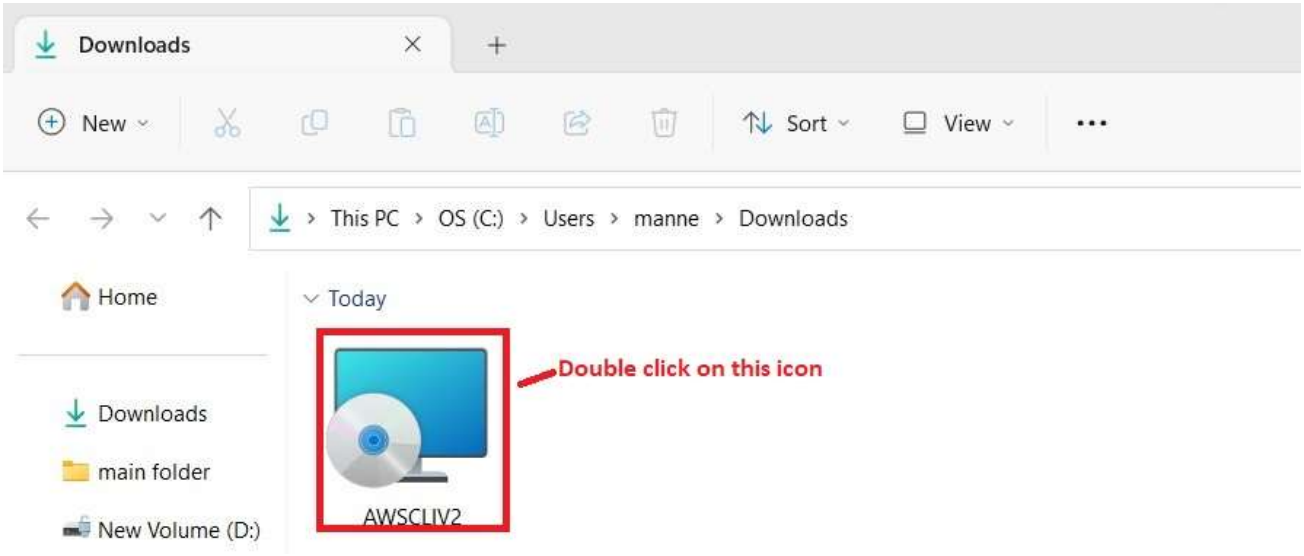


Finally click on that check box and finish icon. Visual studio code setup was finished.

AWS CLI INSTALLATION

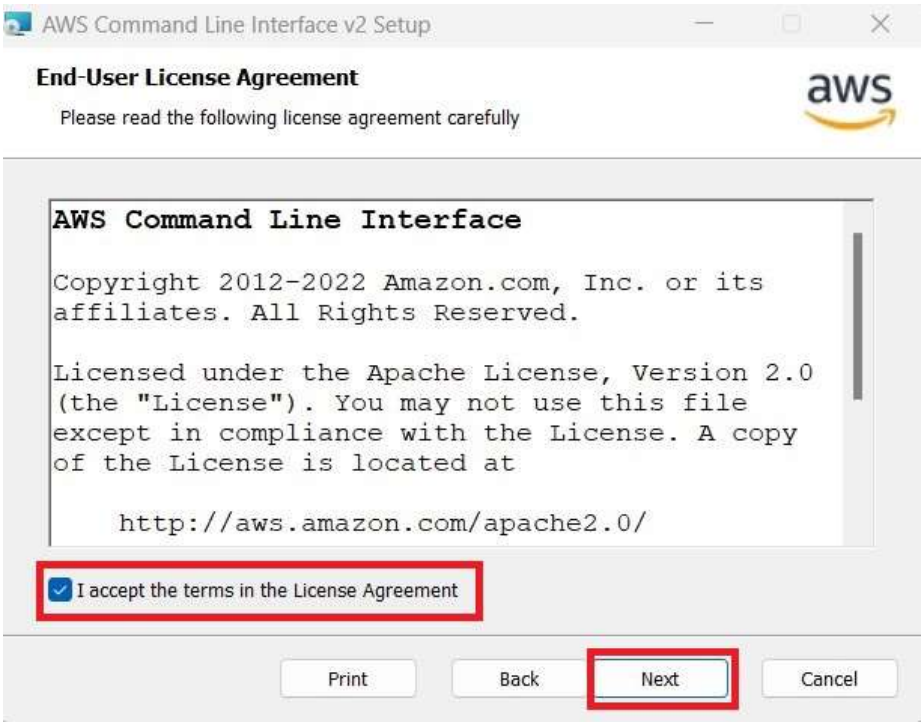
To download: https://awscli.amazonaws.com/AWSCLIV2.msi

After clicking on the above link, AWS-CLI software will be downloaded.

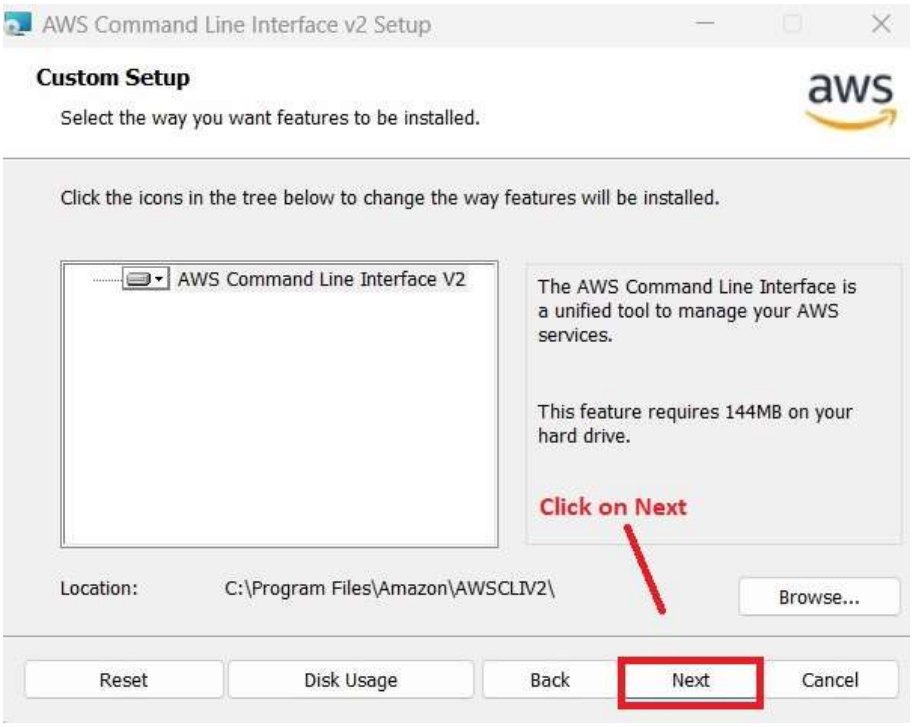


After double-clicking on that icon, the following screen will be displayed

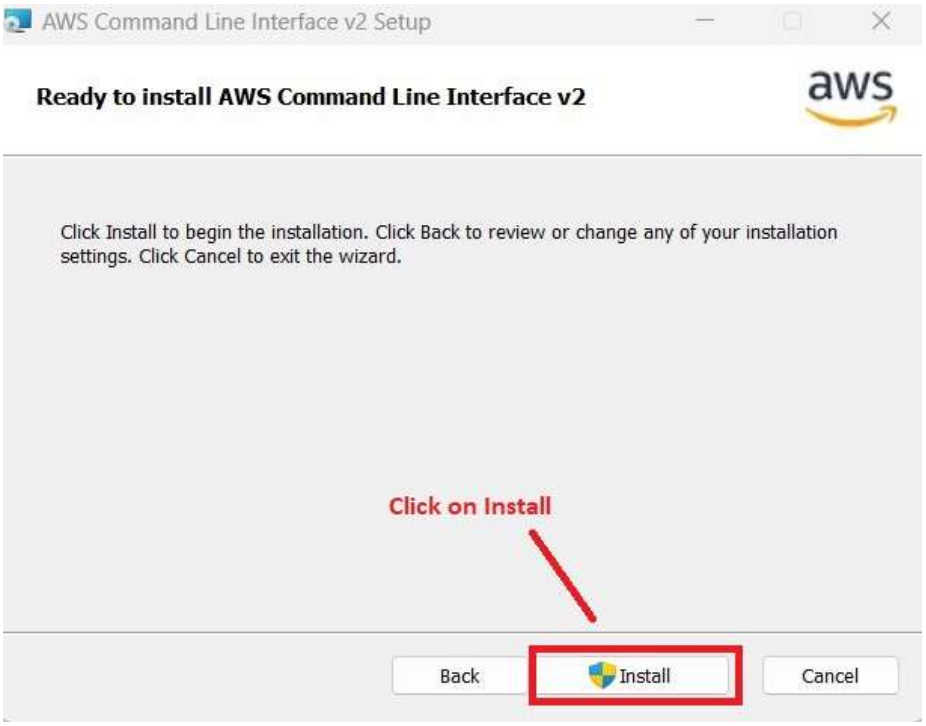
After clicking on next button, the following screen will be displayed.



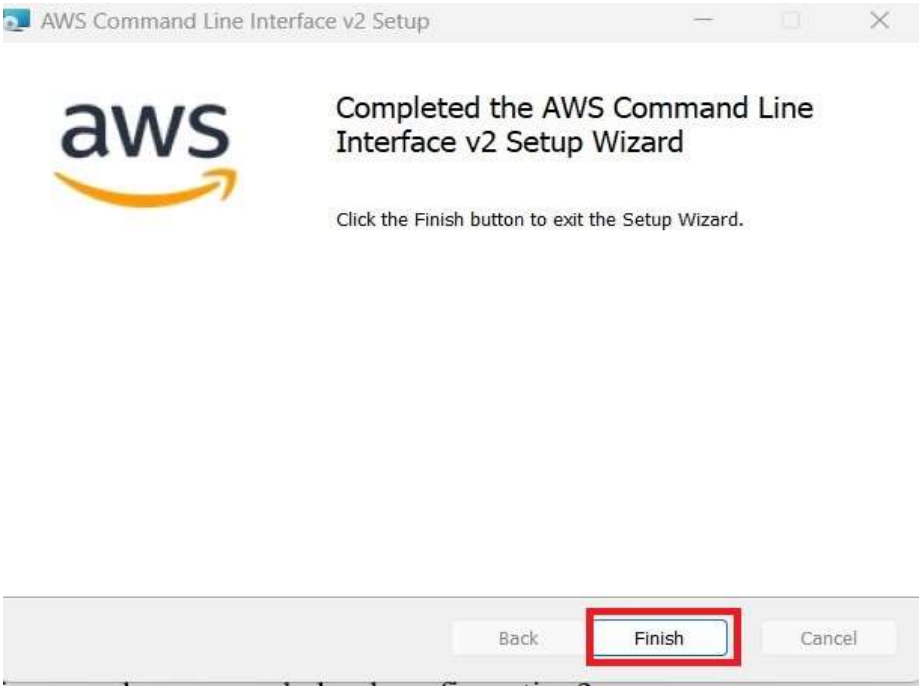
After clicking on next button, the following screen will be displayed.



After clicking on next button, the following screen will be displayed.

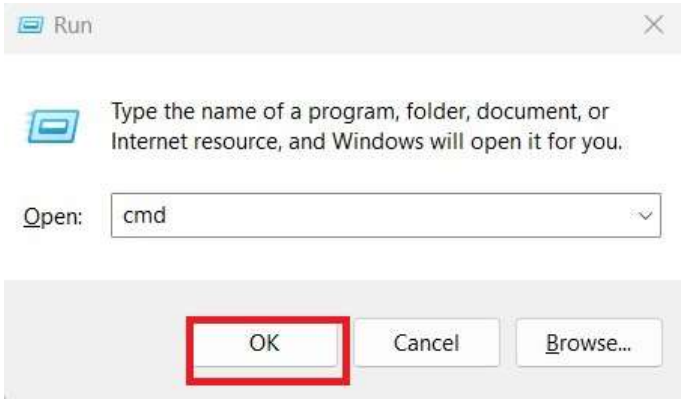


After clicking on 'Install' button, Alert will be displayed click on 'Yes' the following screen will be displayed.

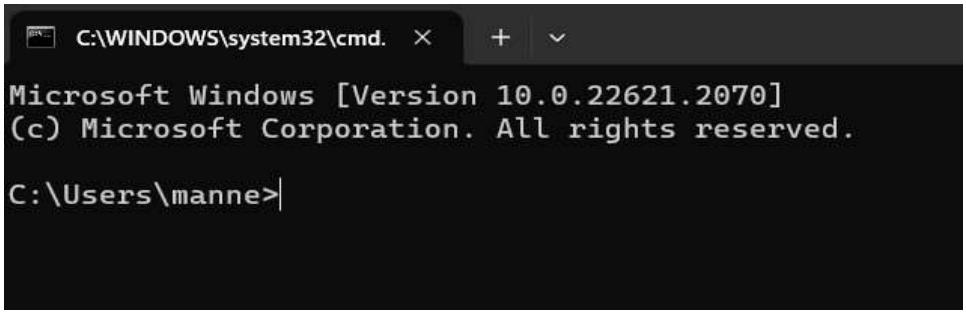


Finally click on finish button, AWS-CLI installation completed.

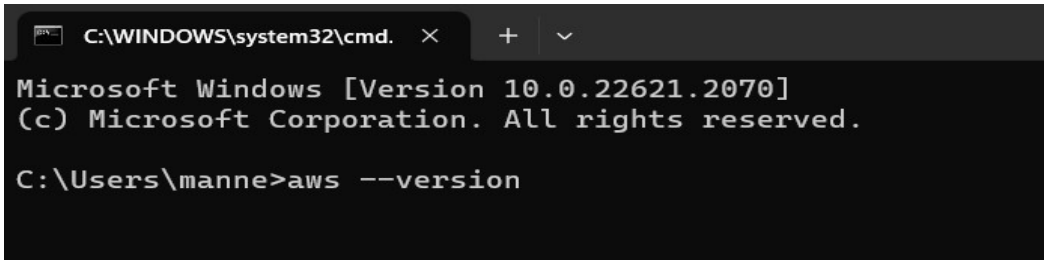
After that, we have to set the environment variables by opening command prompt. Using (Windows Key + R) the following screen will be displayed.



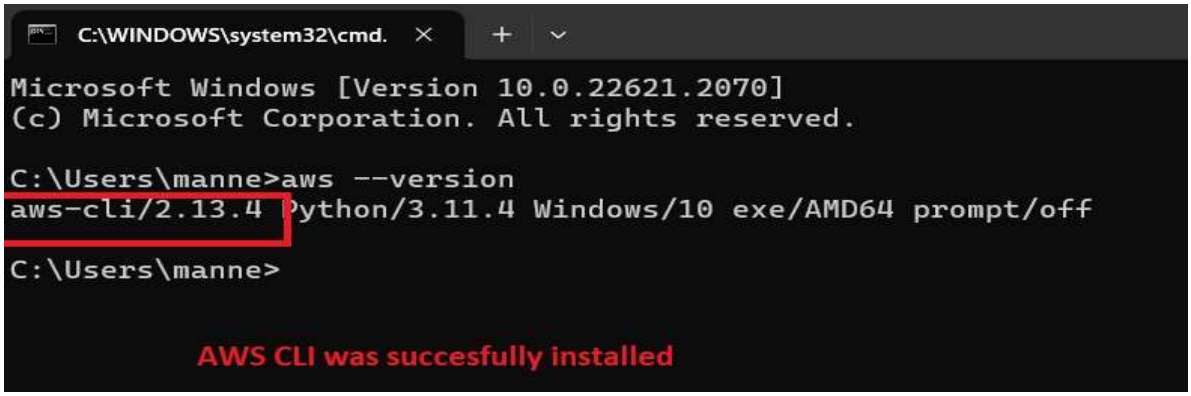
After clicking on that ok button, the following screen will be displayed.



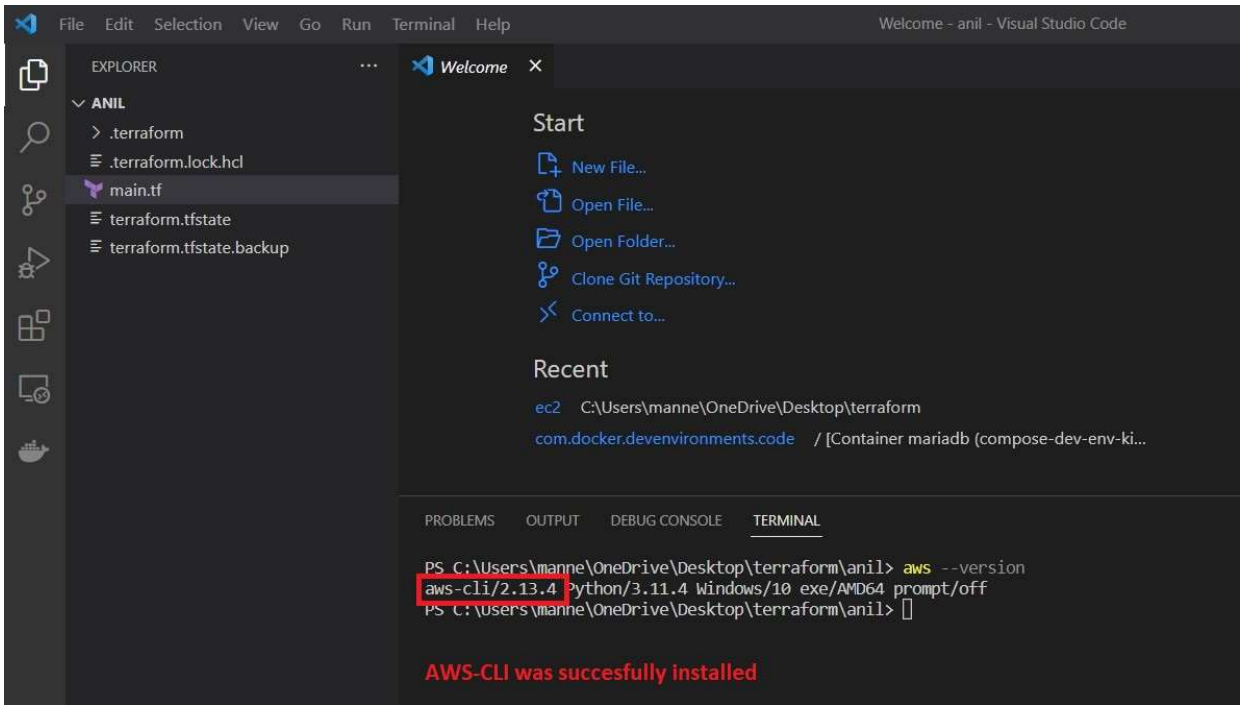
Then to view the AWS-CLI version, in command prompt enter ‘aws –version’ as shown in screenshot.



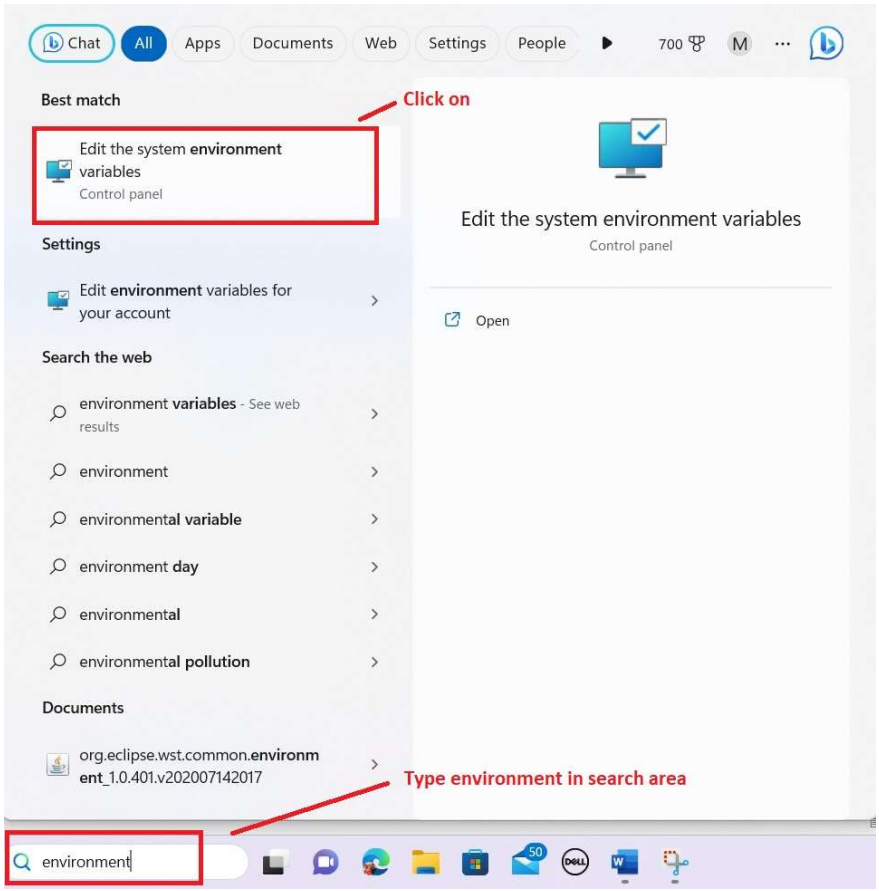
After clicking on Enter button, the following screen will be displayed.



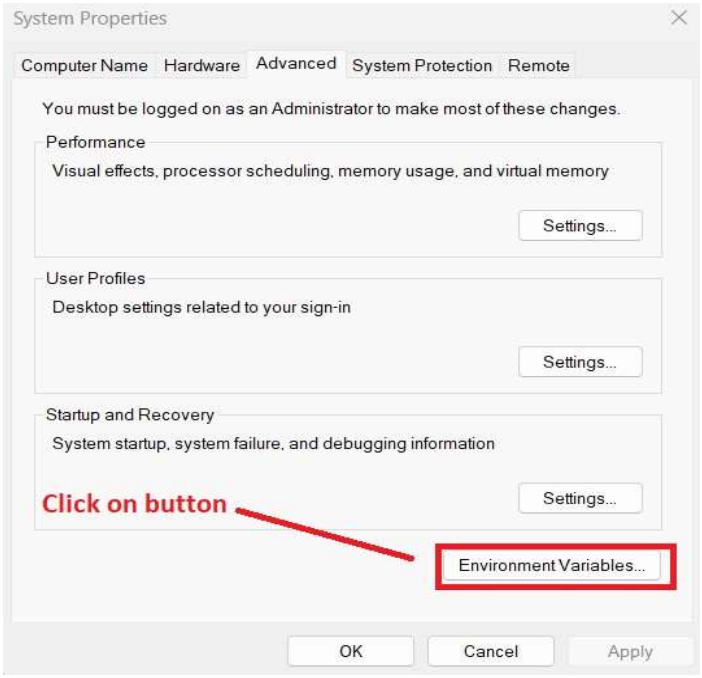
Same step has to be followed, by opening visual studio code and check for AWS-CLI version. scree looks like below.



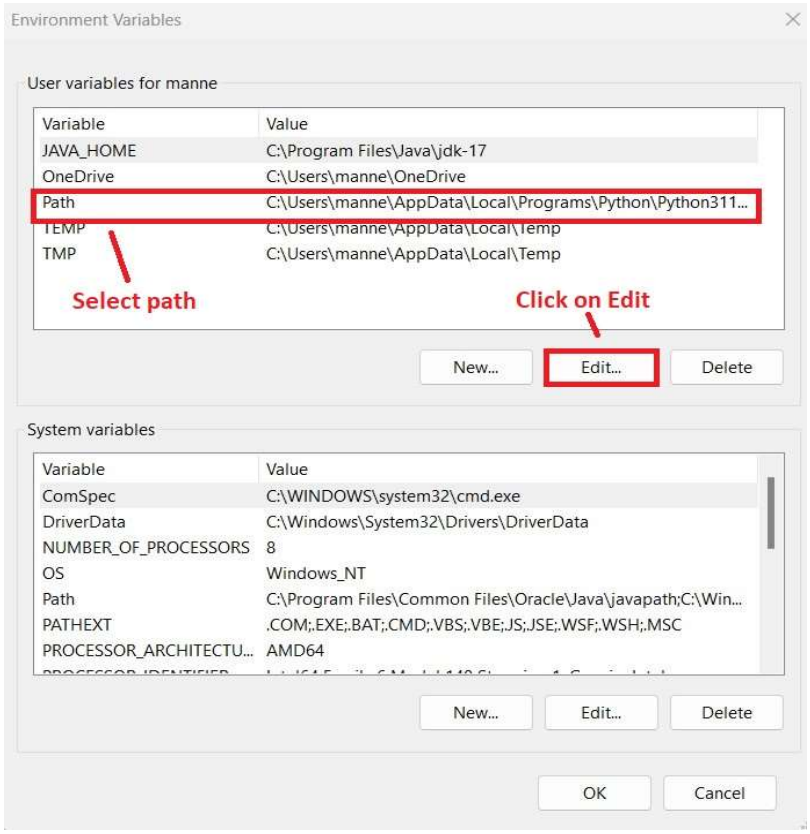
If the content in the above image is not displayed, we have to add AWS-CLI to Environment variables. First search for environment variables settings as follows.



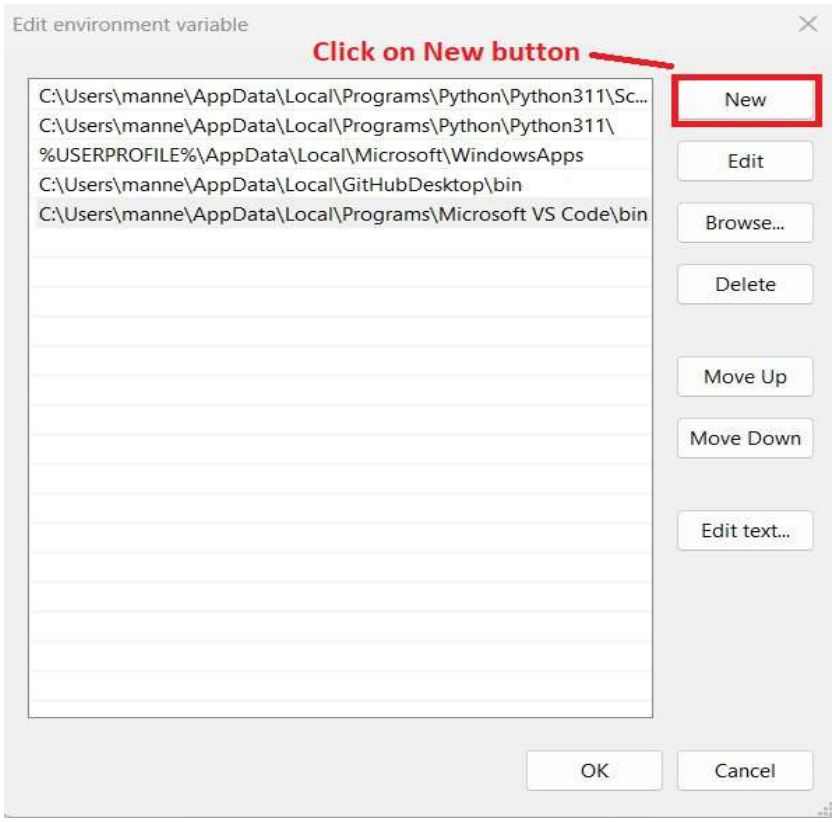
After clicking on the ‘Edit the system environment variables’, the following screen will be displayed.



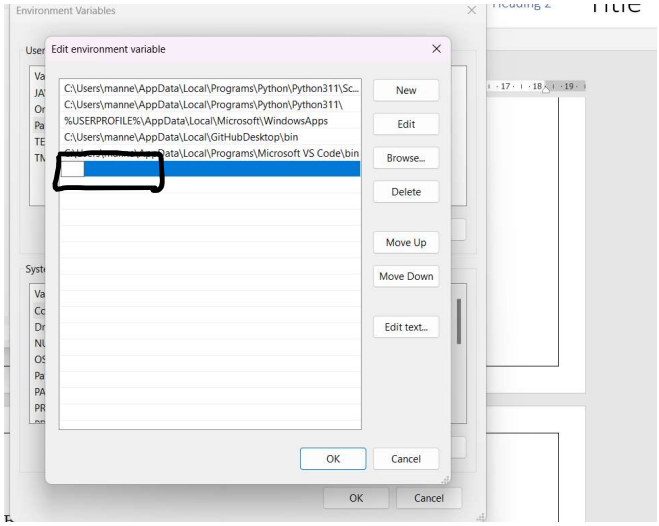
After clicking on that ‘Environment Variables’ button, the following screen will be displayed.



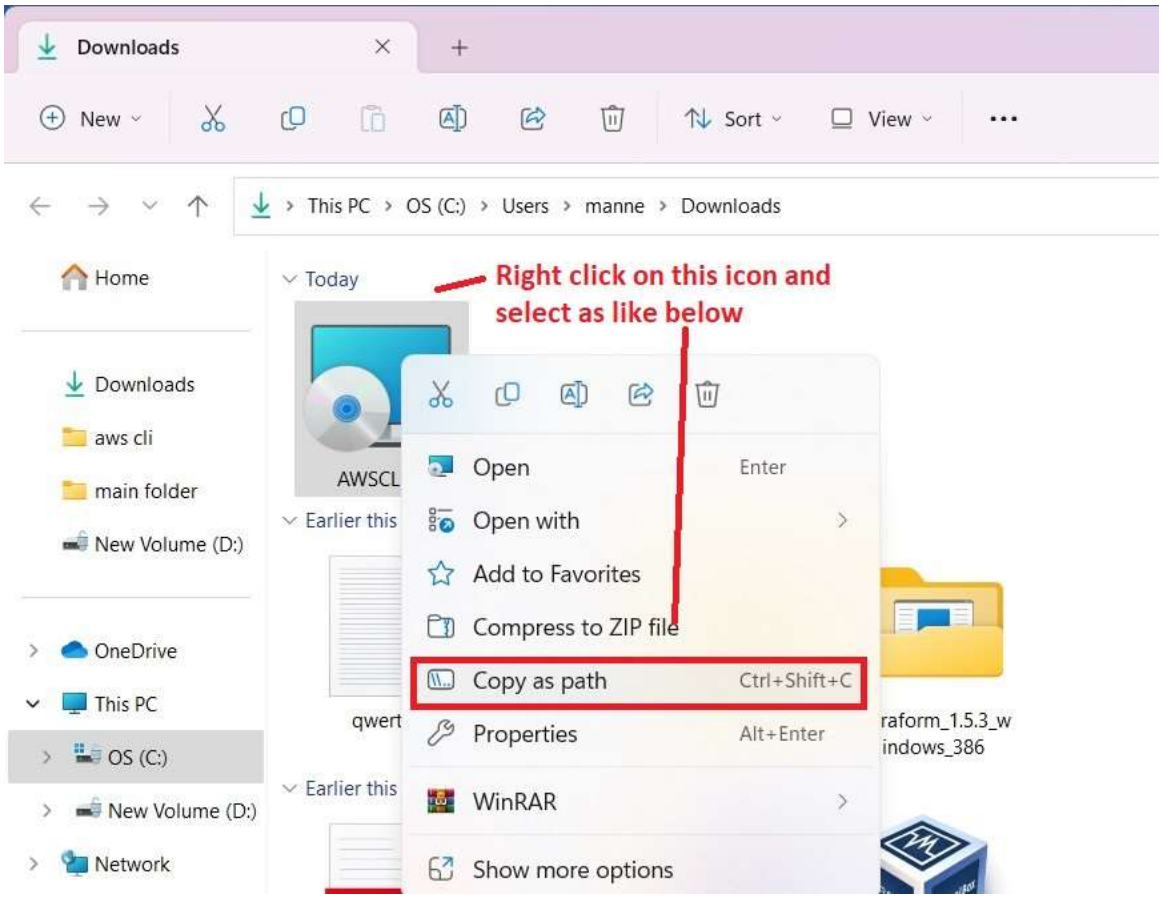
After clicking on that Edit button, the following screen will be displayed.



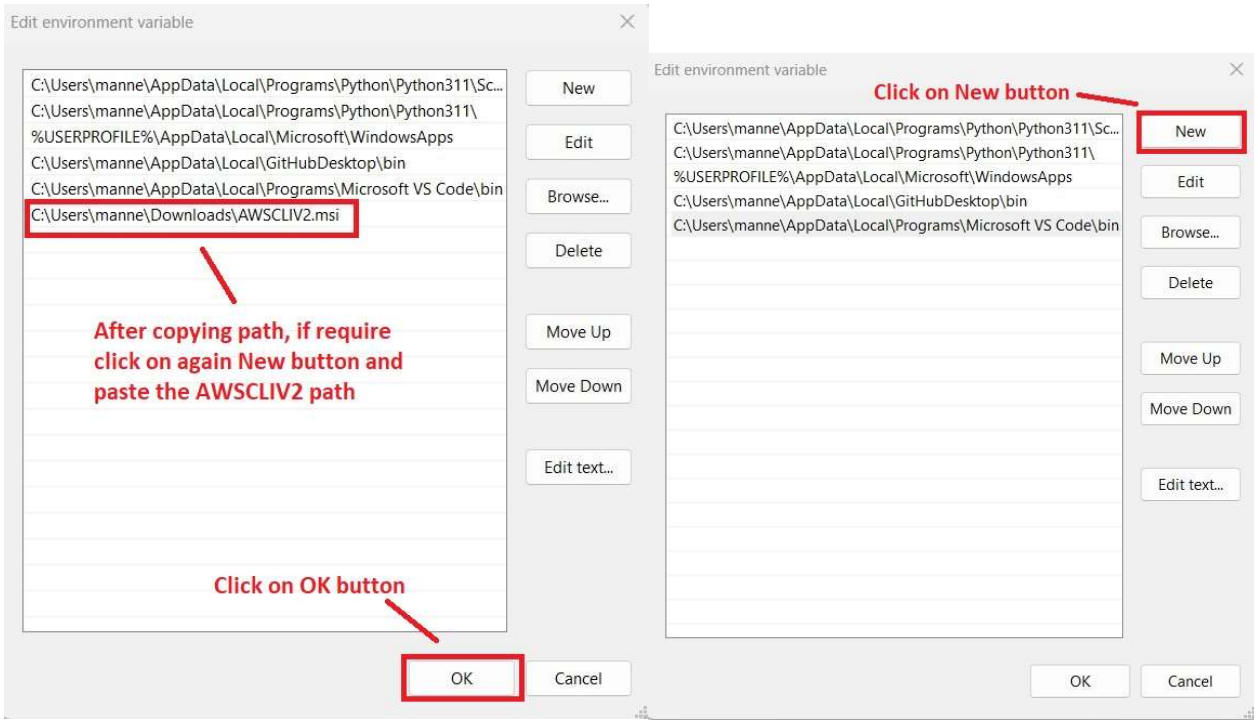
After clicking on new button, the following screen will be displayed.



In the empty field we have to paste the AWS-CLI.MSI path as follows. Goto downloads folder and select AWSCLIV2



After pasting the path, the screen looks like below



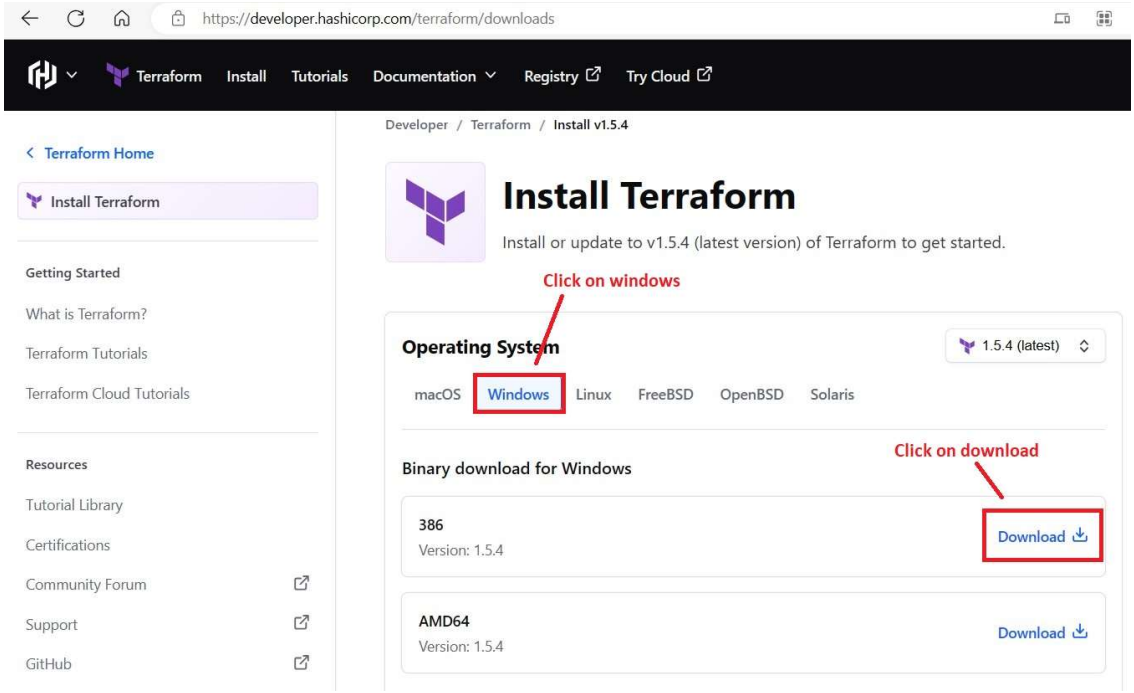
After that close all windows related to Environmental variables.

Then again check the aws –version.

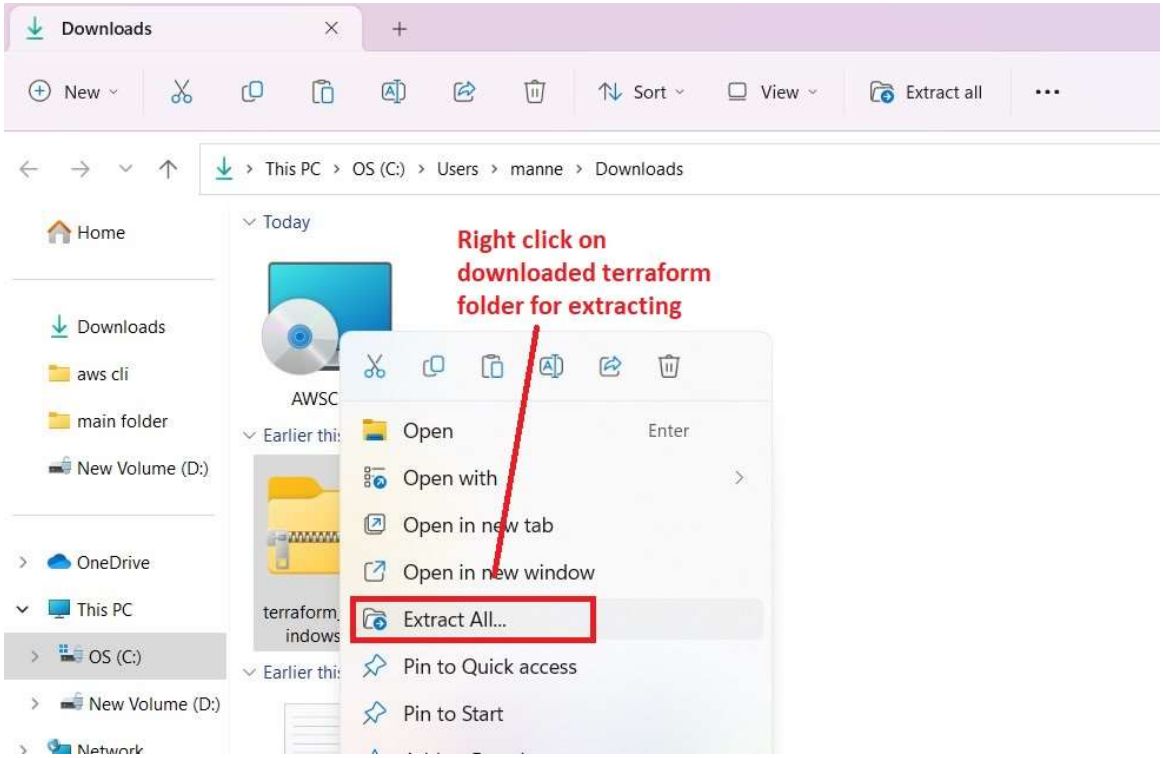
Now, terraform installation will start.

To download: https://developer.hashicorp.com/terraform/downloads

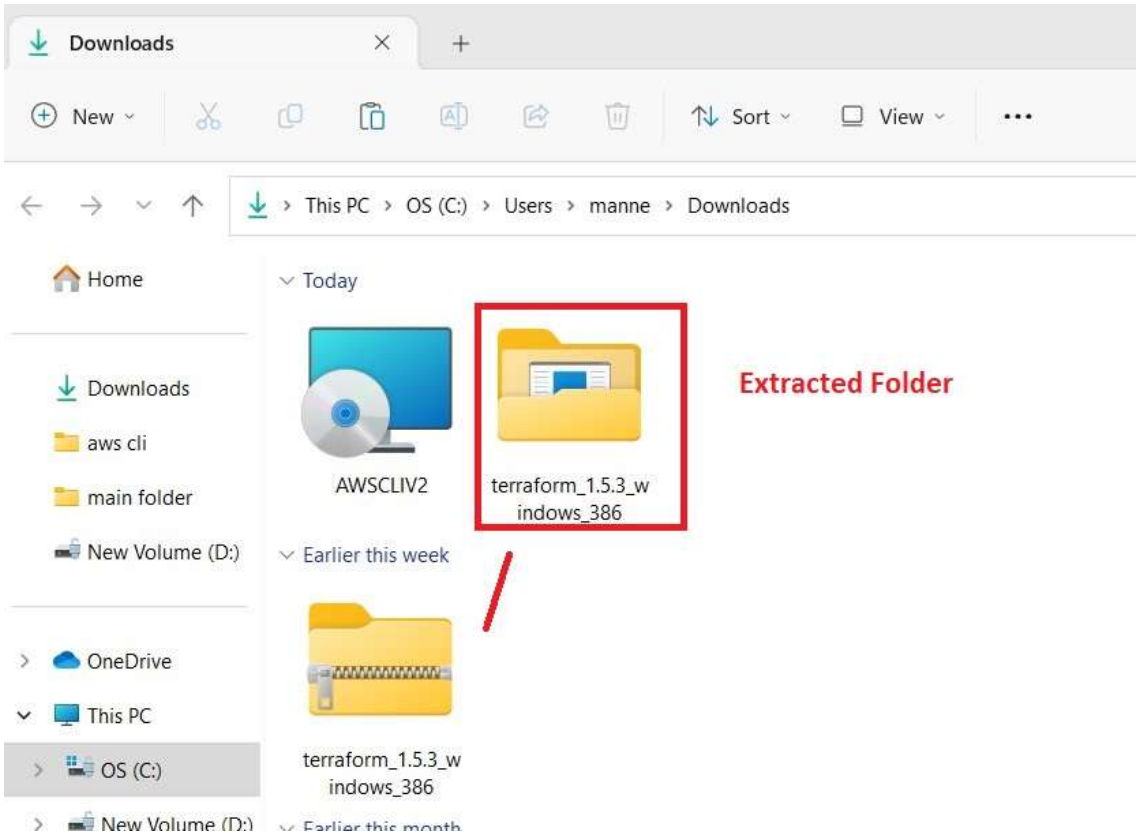
After clicking that link the following page will be displayed.



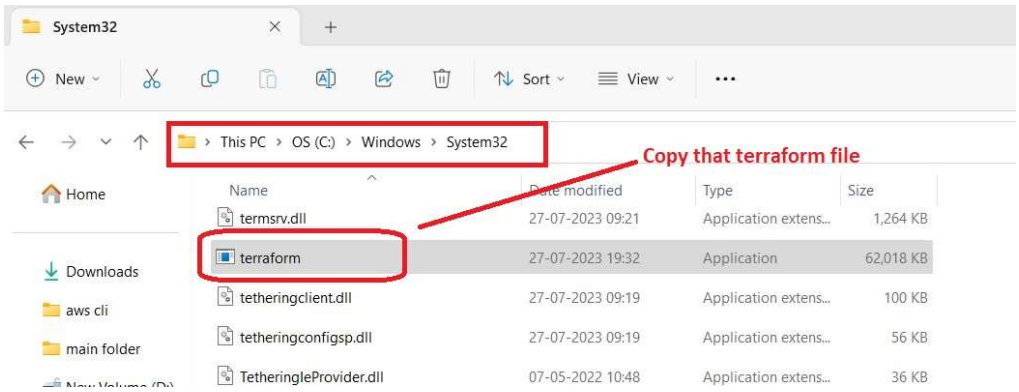
After downloading the terraform, extract the software as shown below.



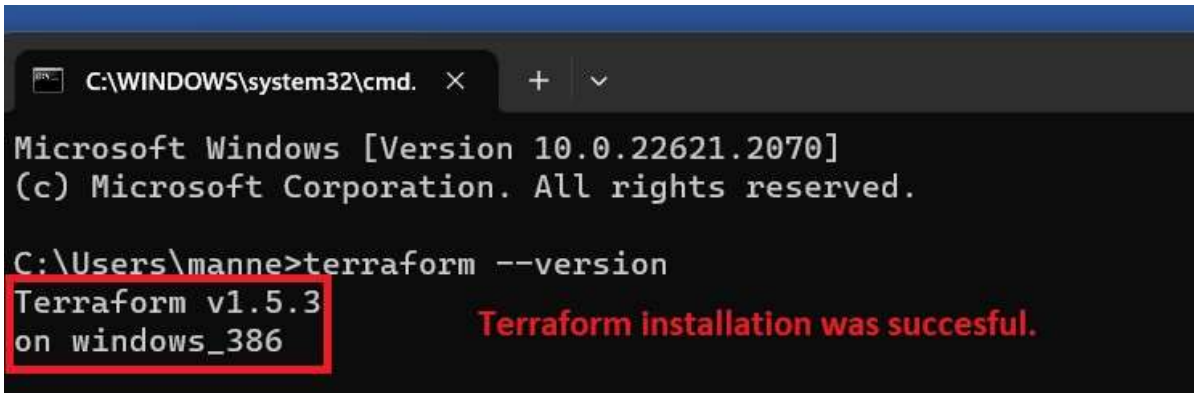
After extracting the folder, it seems like as shown in following screenshot.



After extracting that folder, open that folder and copy the terraform.exe file and paste in the path as shown in following screen. (C:\Windows\System32)



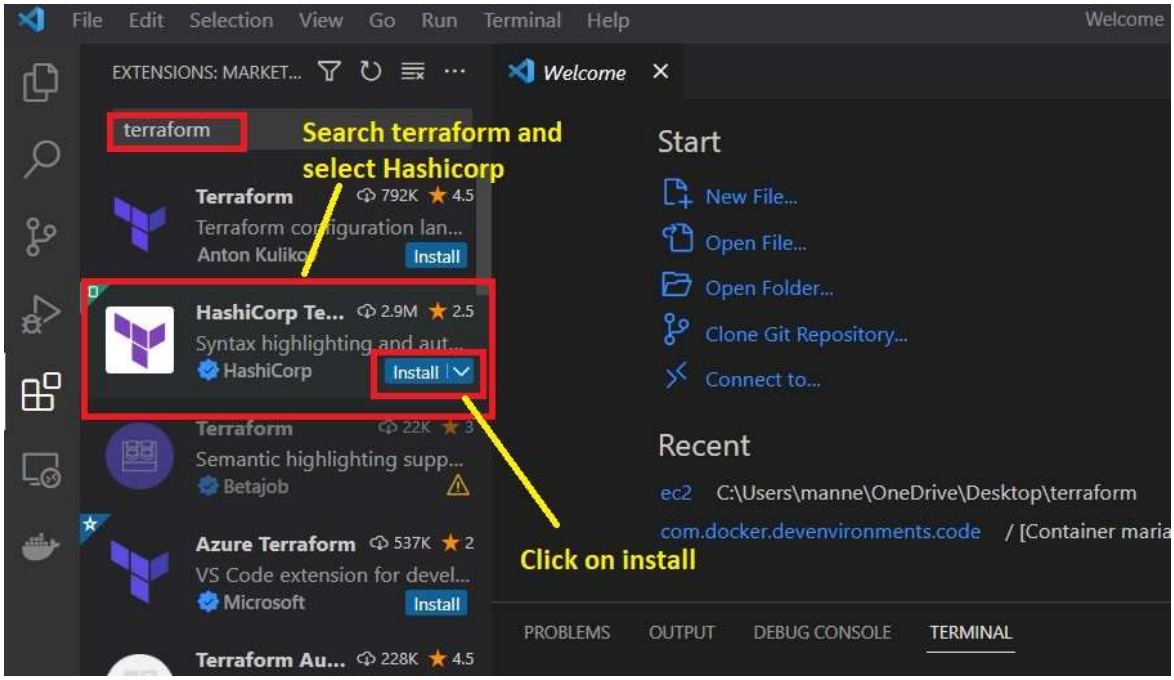
After pasting the terraform.exe file in the above said folder, open command prompt and check for terraform –version as below



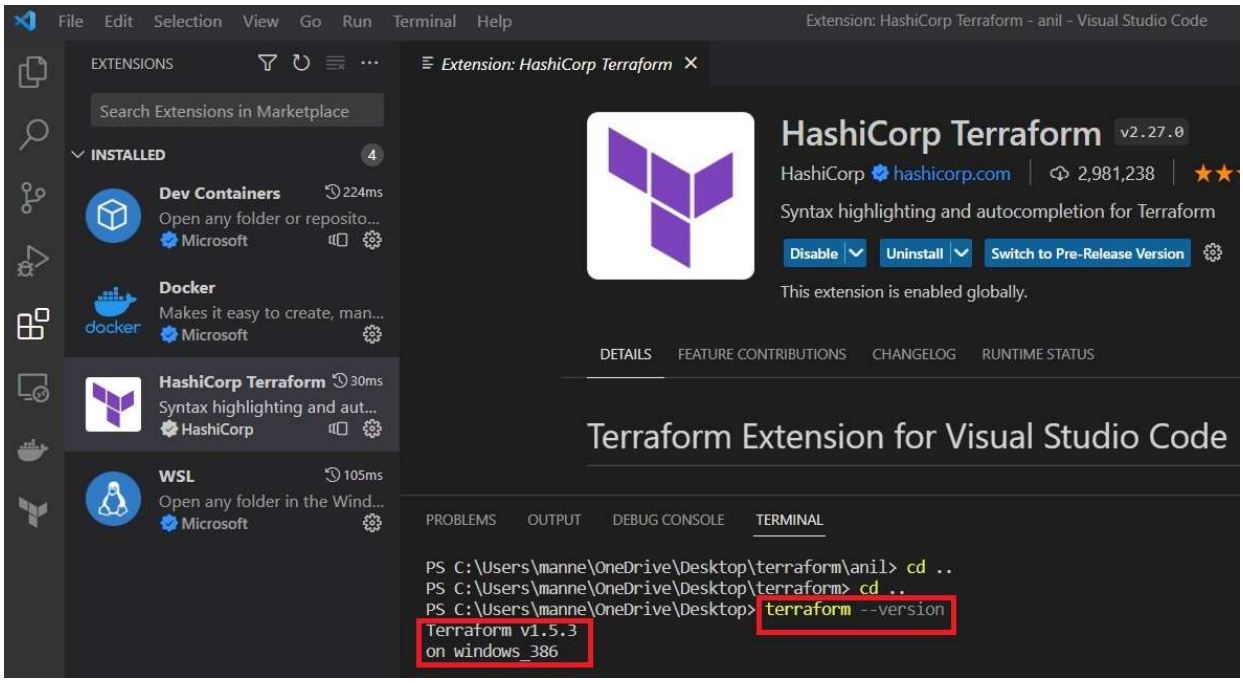
If terraform version found, the installation is successful.

After that, again open Visual studio code and install terraform plugin as shown below.

(Plugin is required to get terraform script suggestions)



After that again check for terraform –version in visual studio and gets validate about successful installation. Screen looks like in next page.

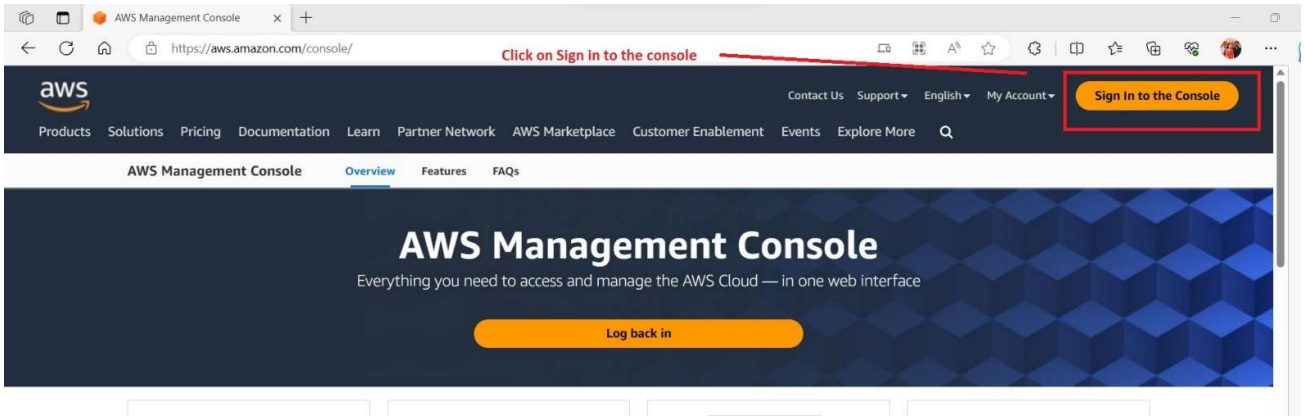


After that we have to integrate AWS (Using Access key and Secret access key) and terraform.

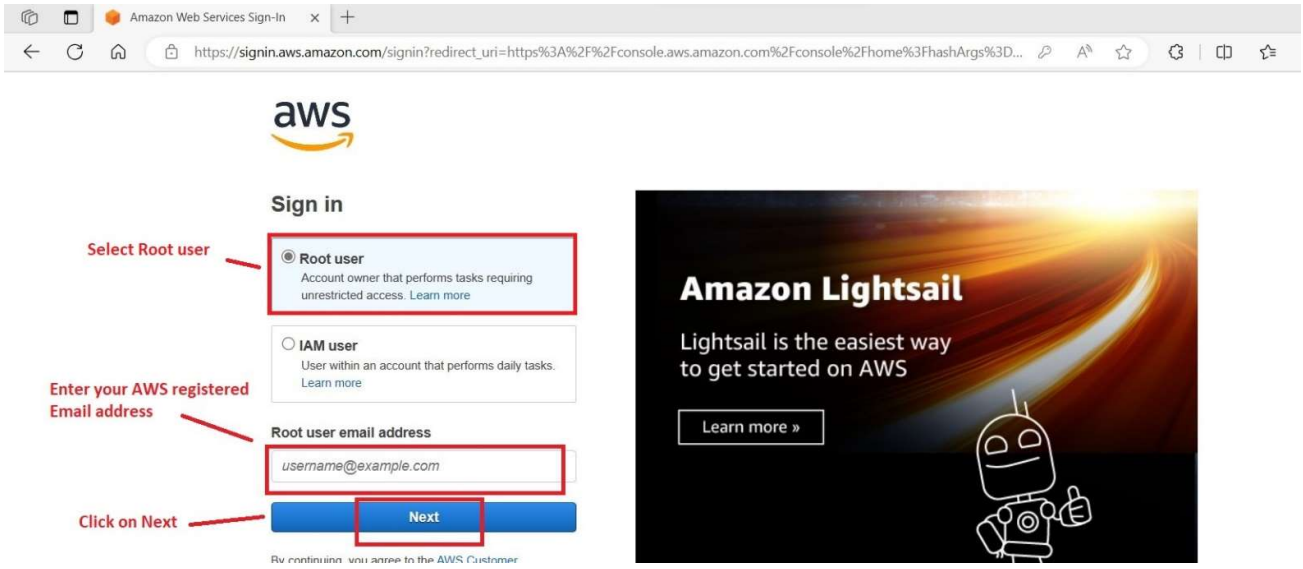
So, to get access key and secret access key from AWS console. (Follow the steps)

URL: https://aws.amazon.com/console/

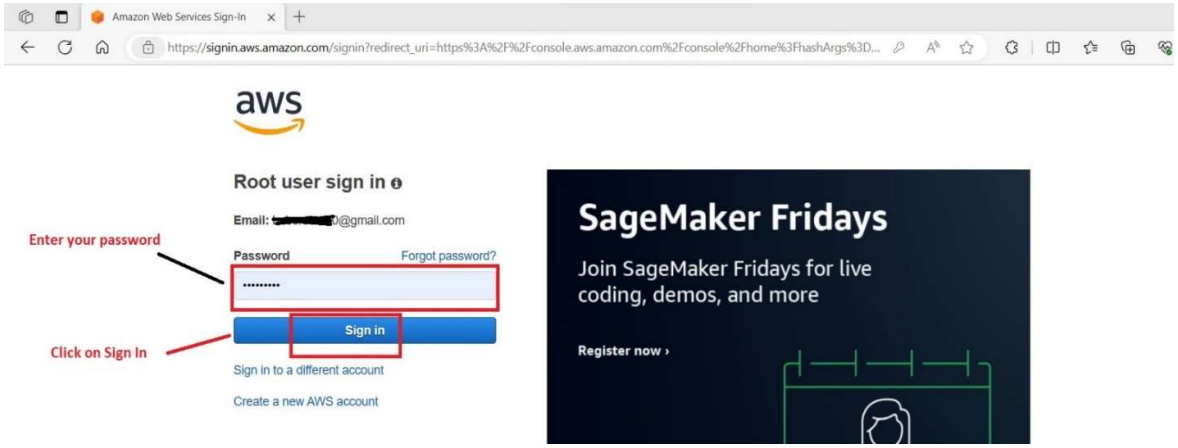
After opening that link the following page will be displayed.



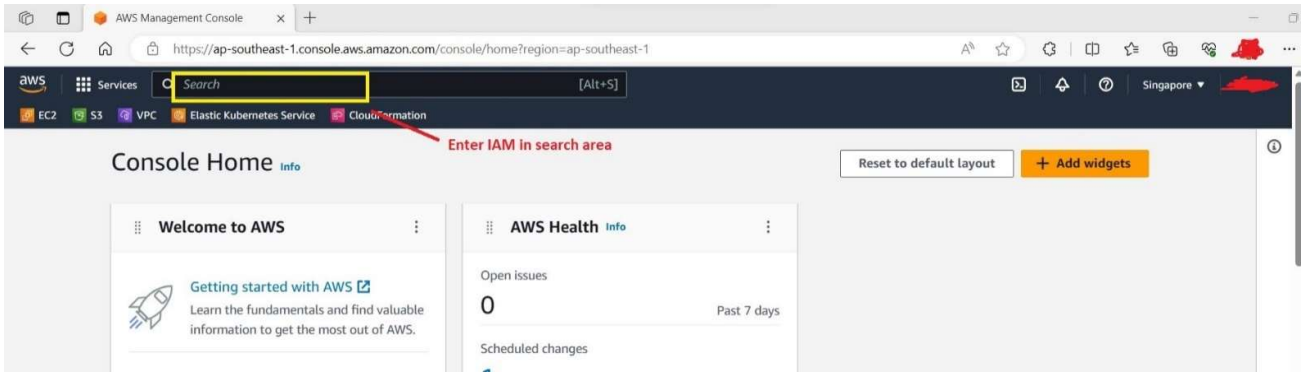
After clicking on ‘sign in to the console’ button, the following page will be displayed.



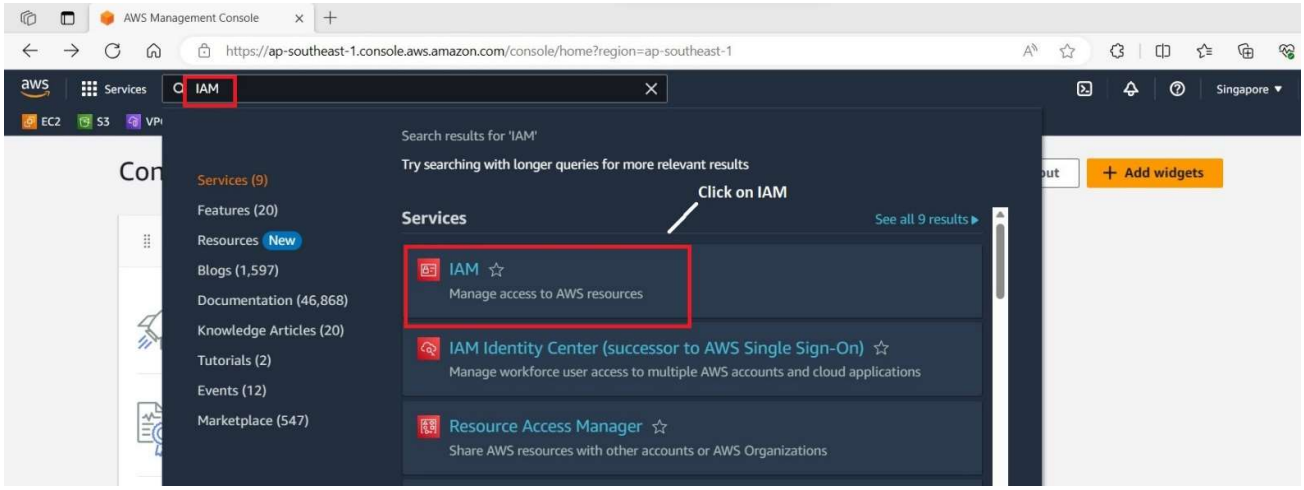
After clicking on ‘Next’ button, the following page will be displayed



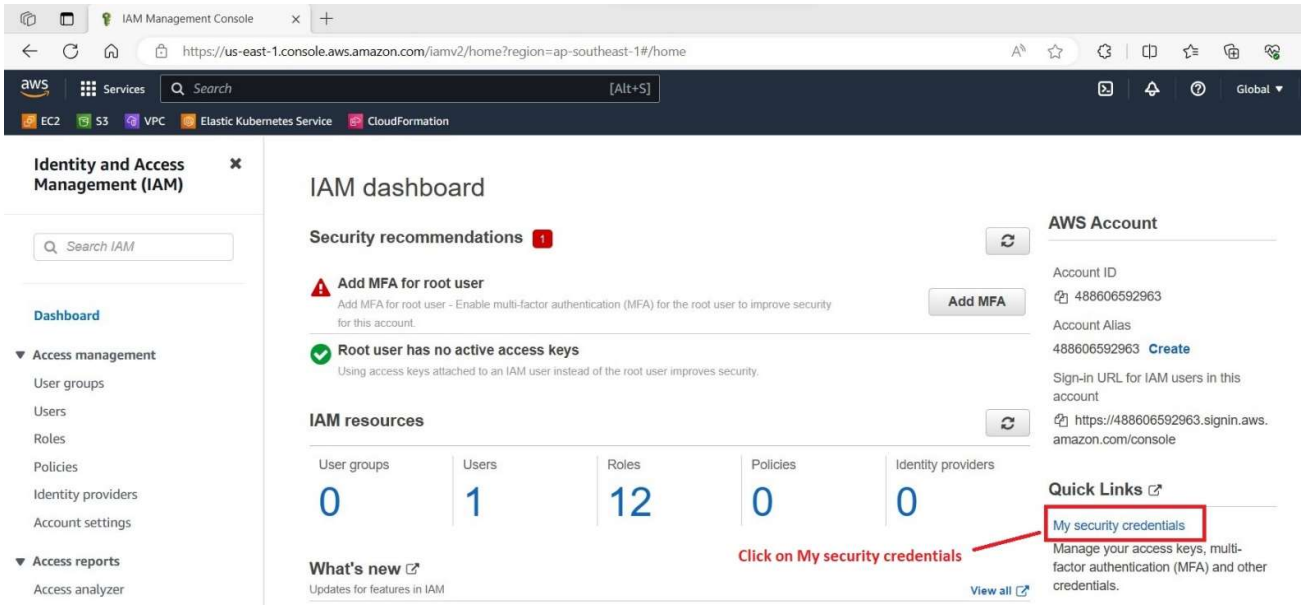
After clicking on ‘Sign In’ button, the following page will be displayed.



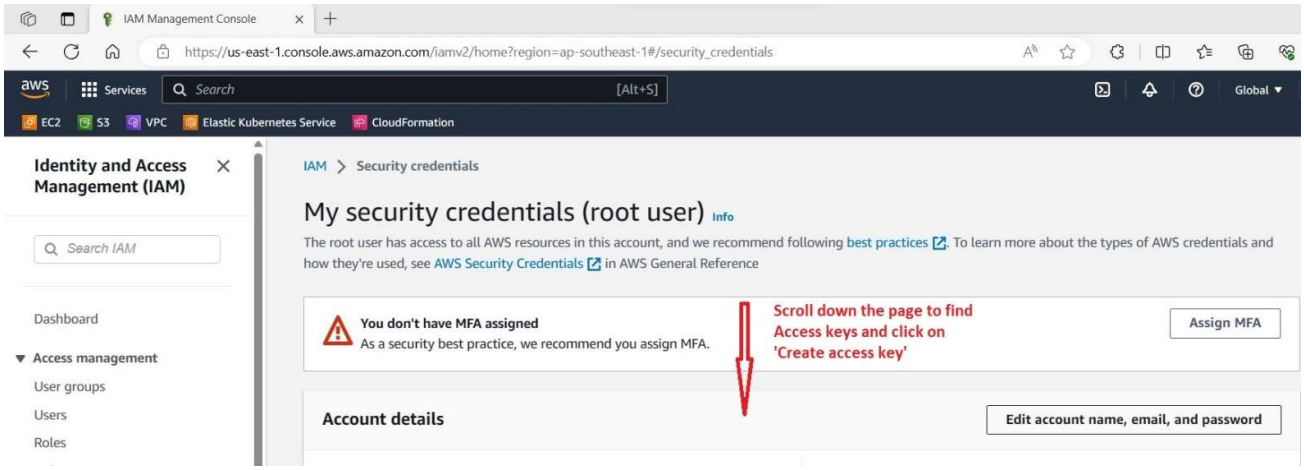
After entering IAM in search area, the following page will be displayed.

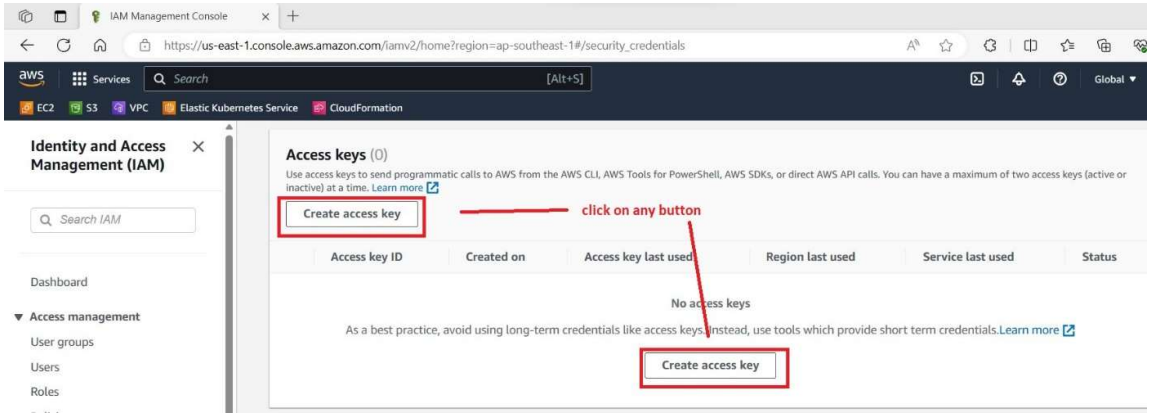


After clicking on IAM, the following page will be displayed.

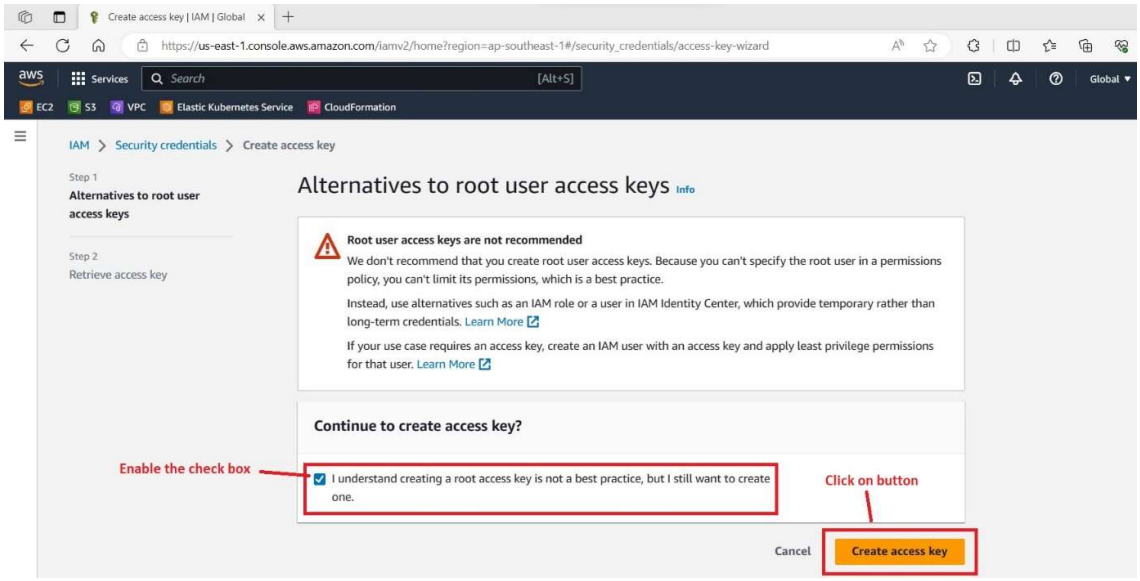


After clicking on ‘My security credentials’ button, page displayed as shown below.

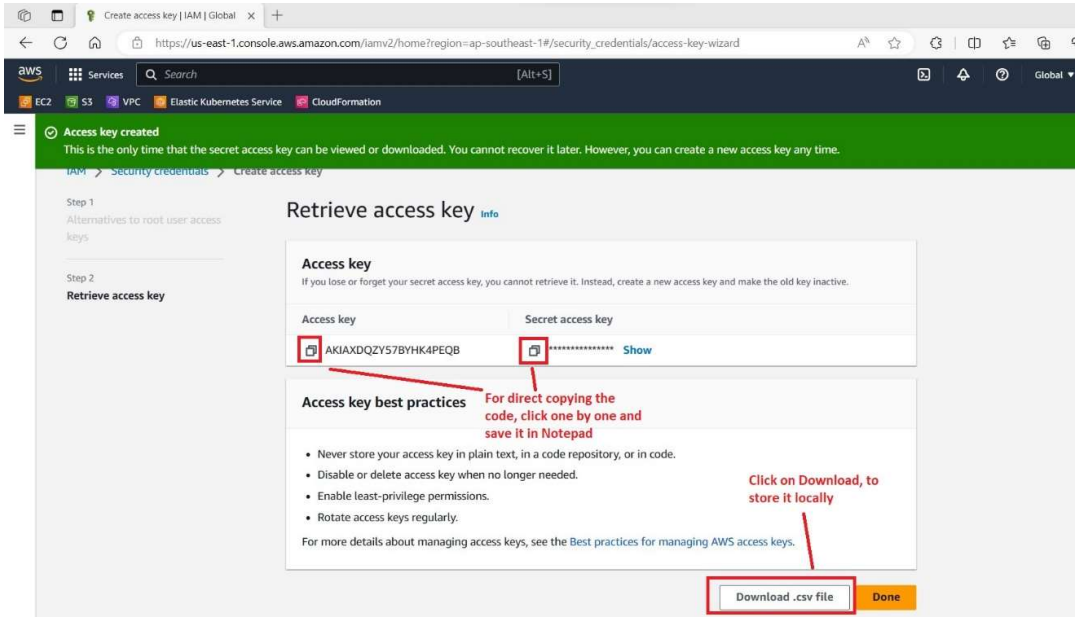




After clicking on ‘Create access key’ button the following page will be displayed.



After enabling check box and clicking on ‘Create access key’, the following page will be displayed.

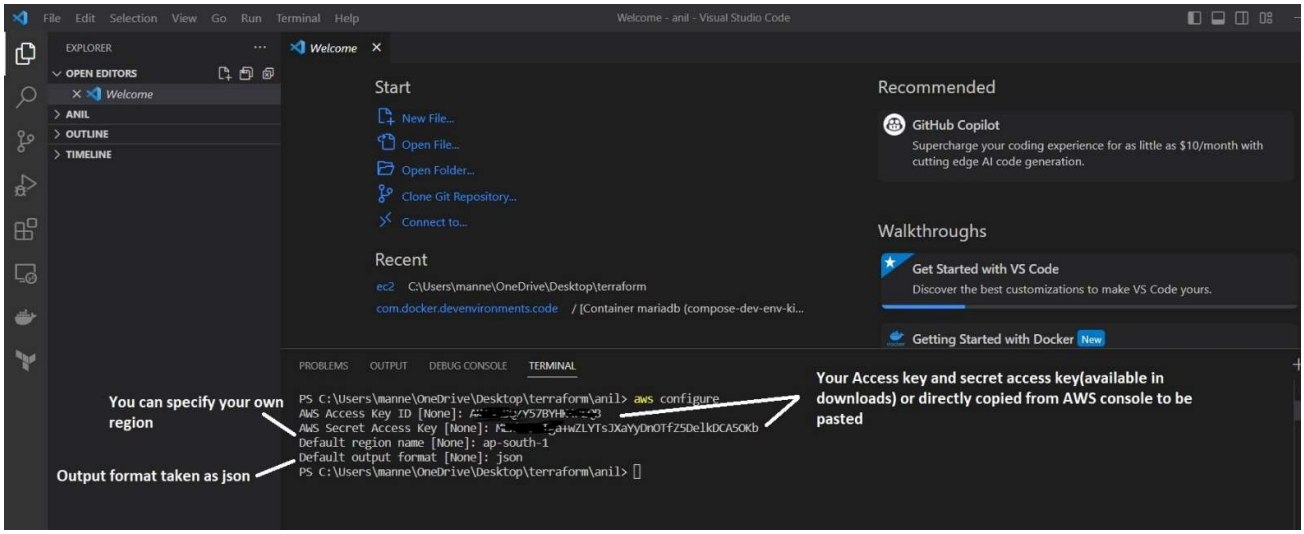


Finally, we had generated the Access key and secret access key. Alternatively, we had downloaded the file and it will be stored in local.

Terraform and AWS CLI integration

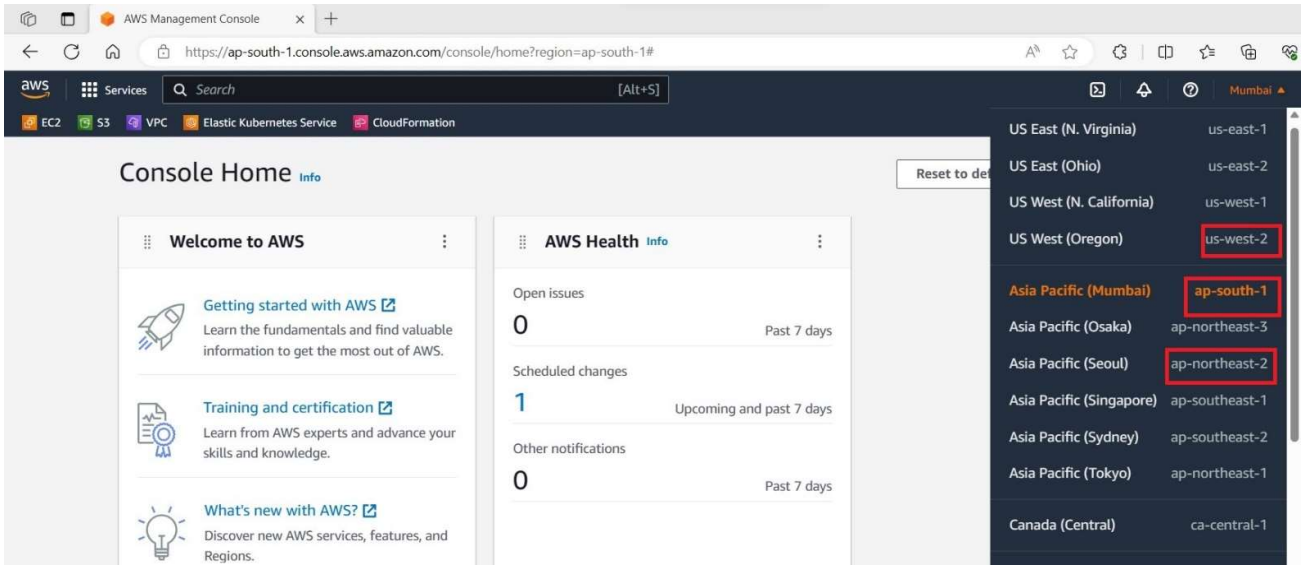
Open visual studio code and in terminal, by using the following commands

aws configure -- Click on enter button



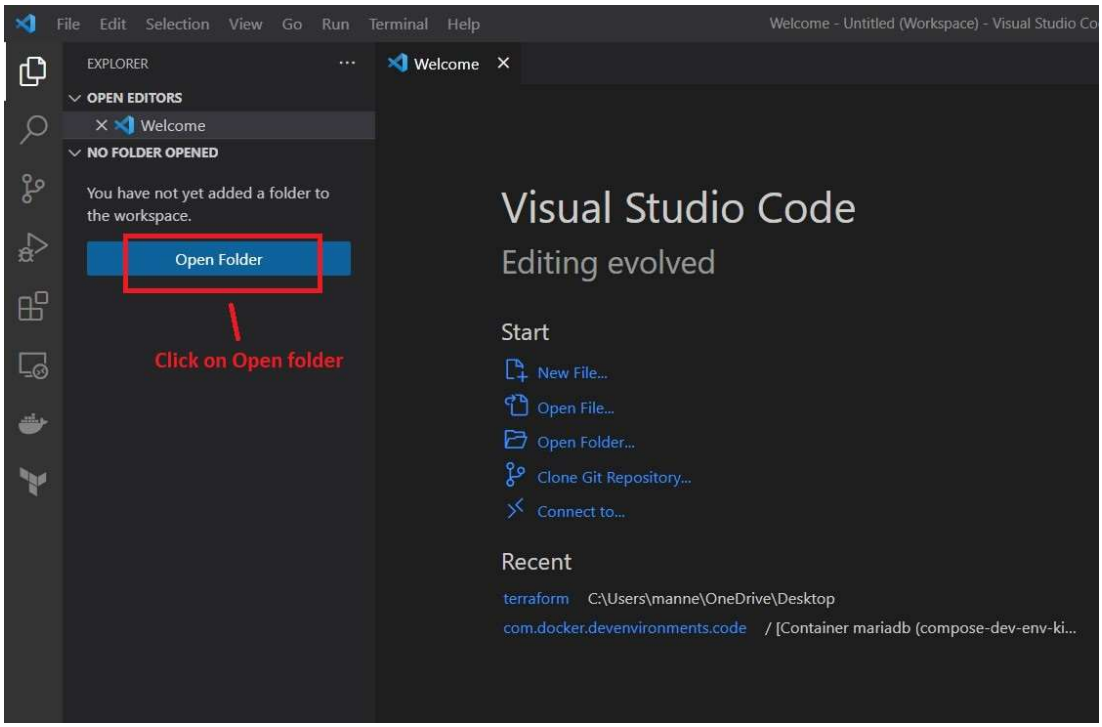
Now you are ready to launch a service in AWS through terraform script.

To select region in AWS console.

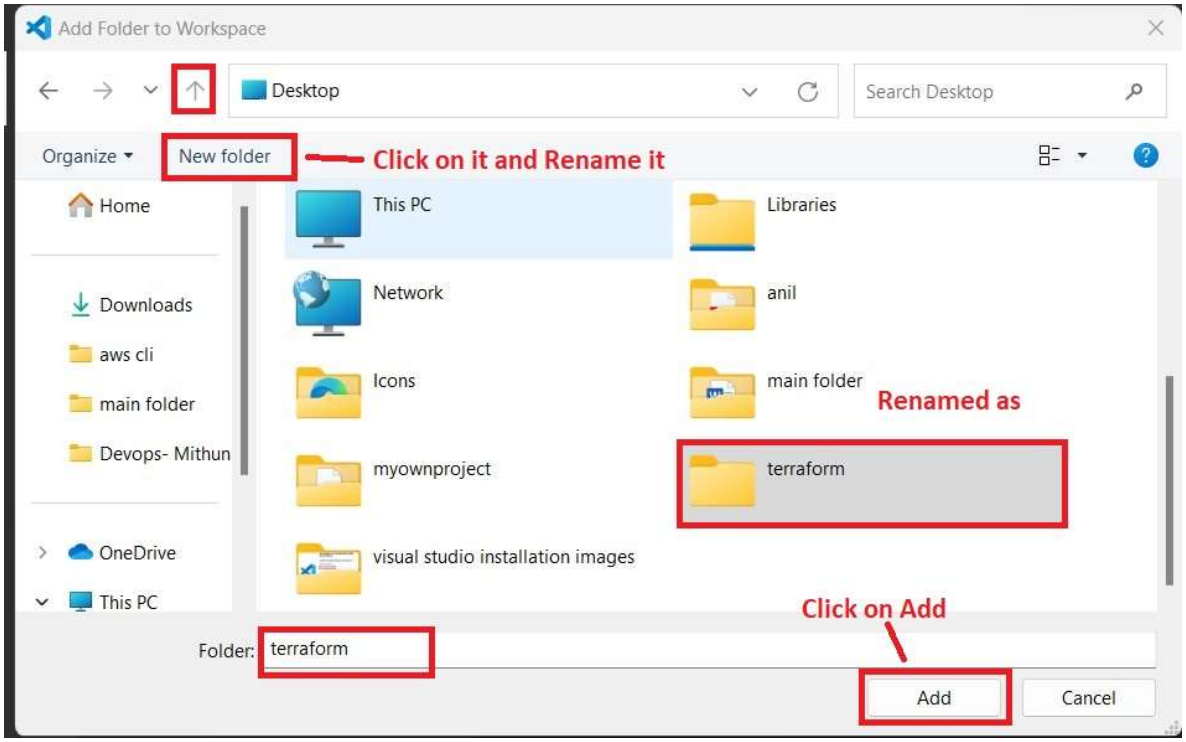


7) Basic services like EC2, VPC and S3 bucket creation with terraform?

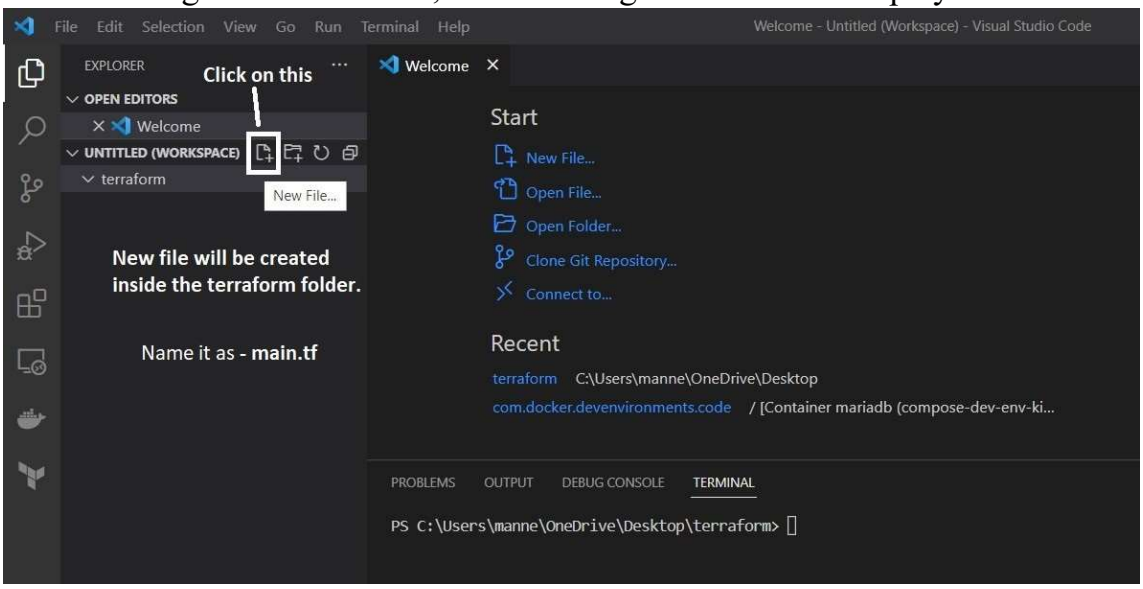
A) To create any service using terraform, first we need to create an empty folder (in any location) and in that main.tf file have to be created in visual studio code (for script suggestions) Follow step by step. Individually each service will be shown. For each service we have to create new folder and new file i.e., main.tf.



After clicking on open folder, you have to select your own folder location. Shown below.



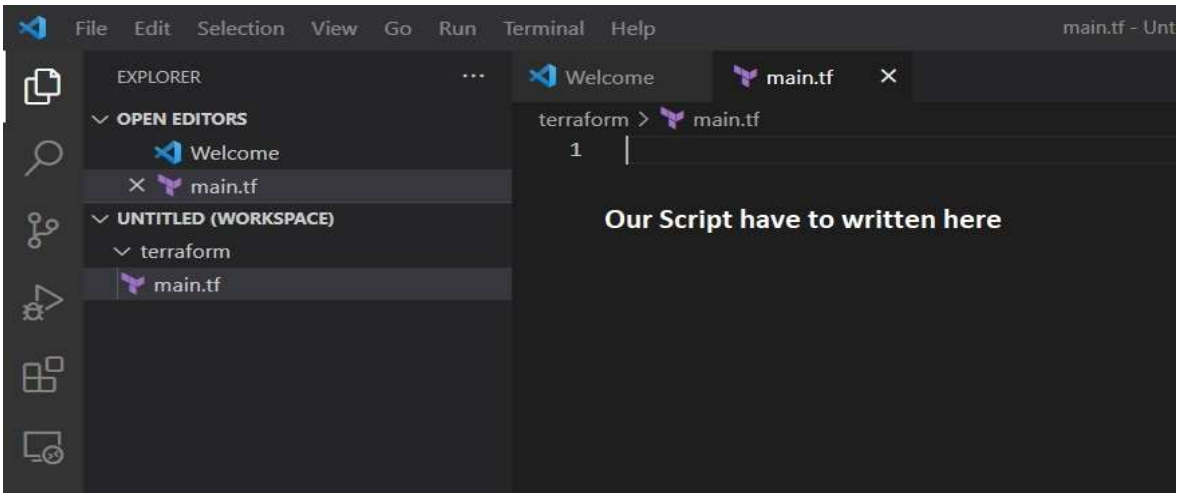
After clicking on 'Add' button, the following screen will be displayed.



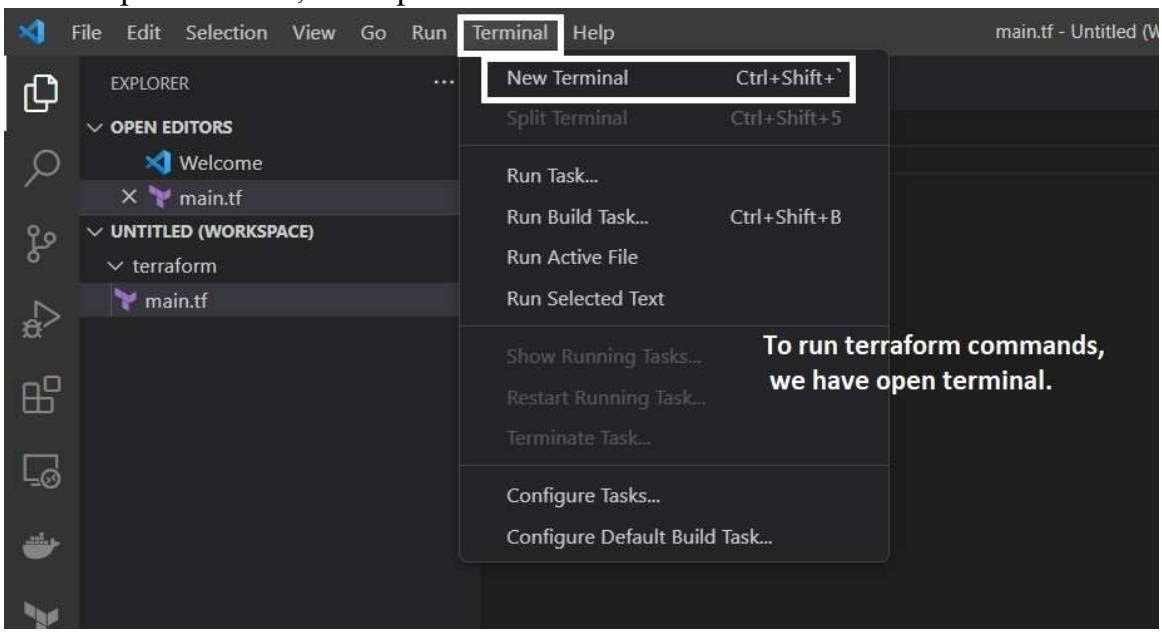
New file is renamed as main.tf and script have to be written in that file.

Main.tf(Script)

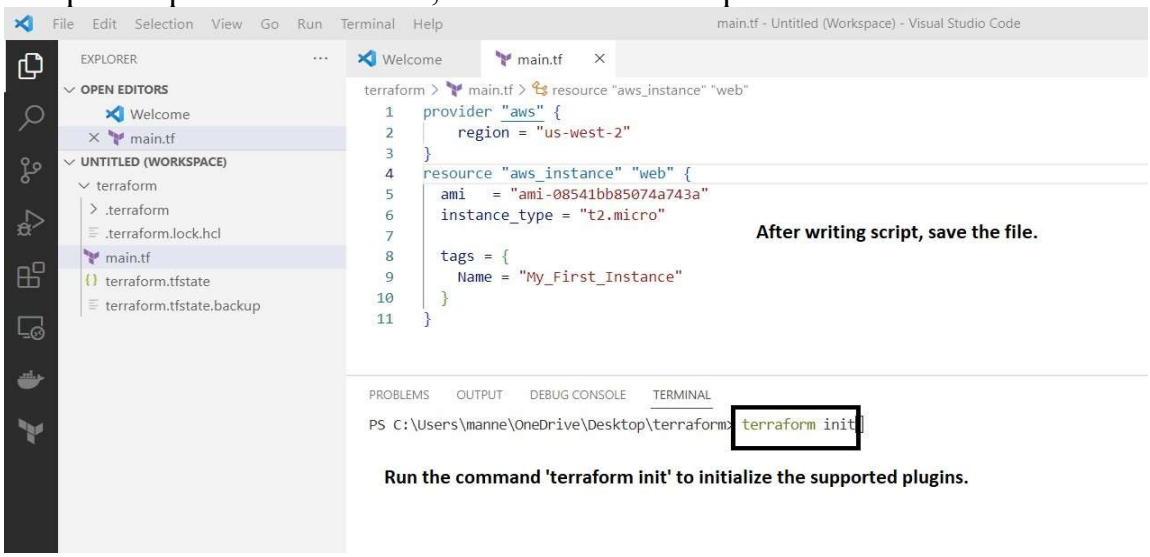
```
provider "aws" {
   region = "us-west-2"
}
resource "aws_instance" "web" {
  ami           = "ami-08541bb85074a743a"
  instance_type = "t2.micro"

  tags = {
    Name = "My_First_Instance"
  }
   }
```



To open terminal, do steps as follows.



Sample script for Ec2 creation, based on the OS requirement. AMI id will be selected.



After saving the run the command, **'terraform init',** screen looks like shown in next page.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

PS C:\Users\manne\OneDrive\Desktop\terraform> terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.10.0

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.
PS C:\Users\manne\OneDrive\Desktop\terraform>
```

After successful initialization of terraform, run '**terraform plan'** in terminal.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

      + secondary_private_ips               = (known after apply)
      + security_groups                     = (known after apply)
      + source_dest_check                   = true
      + spot_instance_request_id            = (known after apply)
      + subnet_id                           = (known after apply)
      + tags                                = {
          + "Name" = "My_First_Instance"
        }
      + tags_all                            = {
          + "Name" = "My_First_Instance"
        }
      + tenancy                             = (known after apply)
      + user_data                           = (known after apply)
      + user_data_base64                    = (known after apply)
      + user_data_replace_on_change         = false
      + vpc_security_group_ids              = (known after apply)
    }

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

It is just a dry run, Changes will be shown here

After that, you have to run the command '**terraform apply --auto-approve=true'** in terminal for infrastructure creation in AWS.

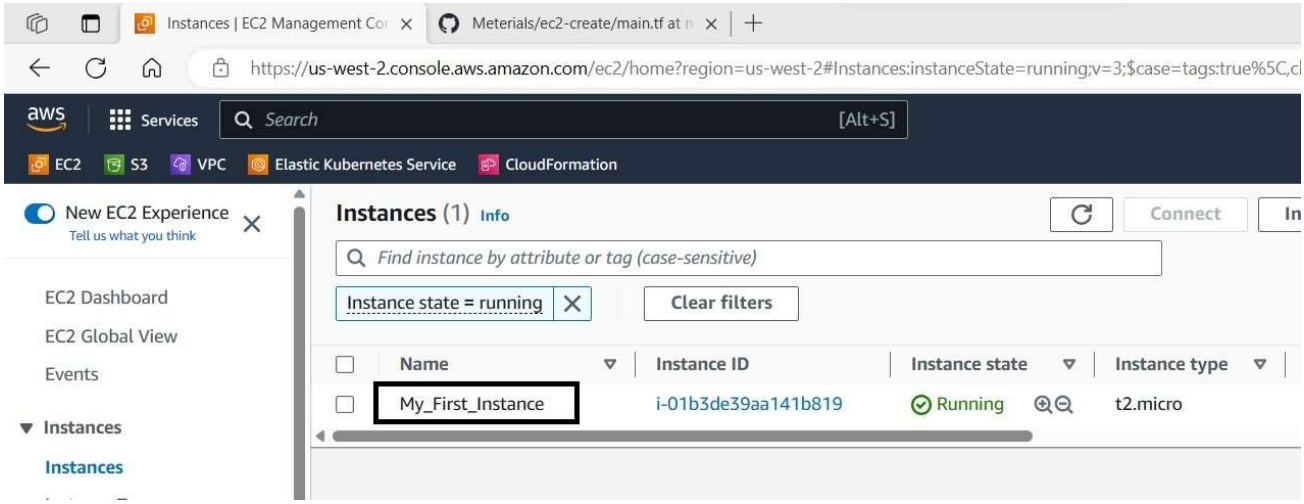
```
        }
      + tenancy                             = (known after apply)
      + user_data                           = (known after apply)
      + user_data_base64                    = (known after apply)
      + user_data_replace_on_change         = false
      + vpc_security_group_ids              = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
aws_instance.web: Creating...
aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Still creating... [20s elapsed]
```

Instance creation under progress

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

      + security_groups                      = (known after apply)
      + source_dest_check                    = true
      + spot_instance_request_id             = (known after apply)
      + subnet_id                            = (known after apply)
      + tags                                 = {
          + "Name" = "My_First_Instance"
        }
      + tags_all                             = {
          + "Name" = "My_First_Instance"
        }
      + tenancy                              = (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      + user_data_replace_on_change          = false
      + vpc_security_group_ids               = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
aws_instance.web: Creating...
aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Still creating... [20s elapsed]
aws_instance.web: Still creating... [30s elapsed]
aws_instance.web: Creation complete after 36s [id=i-0731740d23b42c6bd]

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

Instance created succesfully in AWS

View the instance in AWS console.

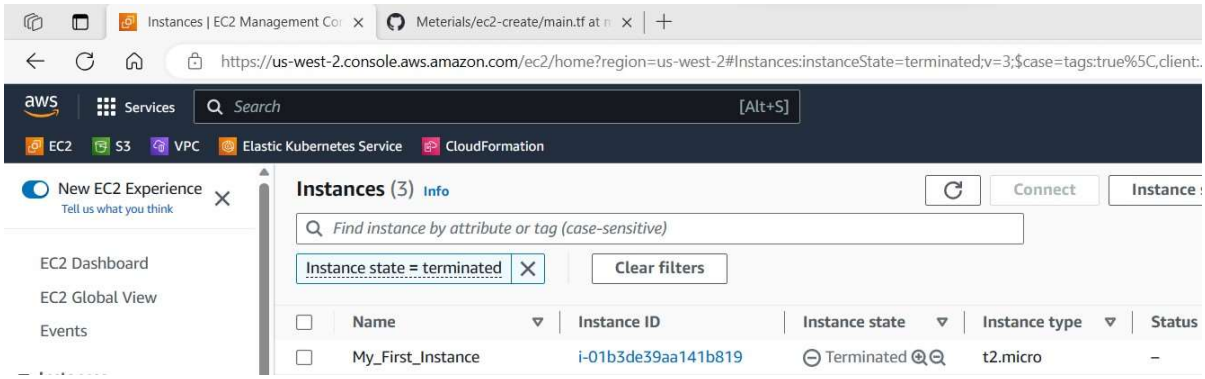


To terminate the instance, run the command '**terraform destroy --auto-approve=true'** in terminal for infrastructure creation in AWS.

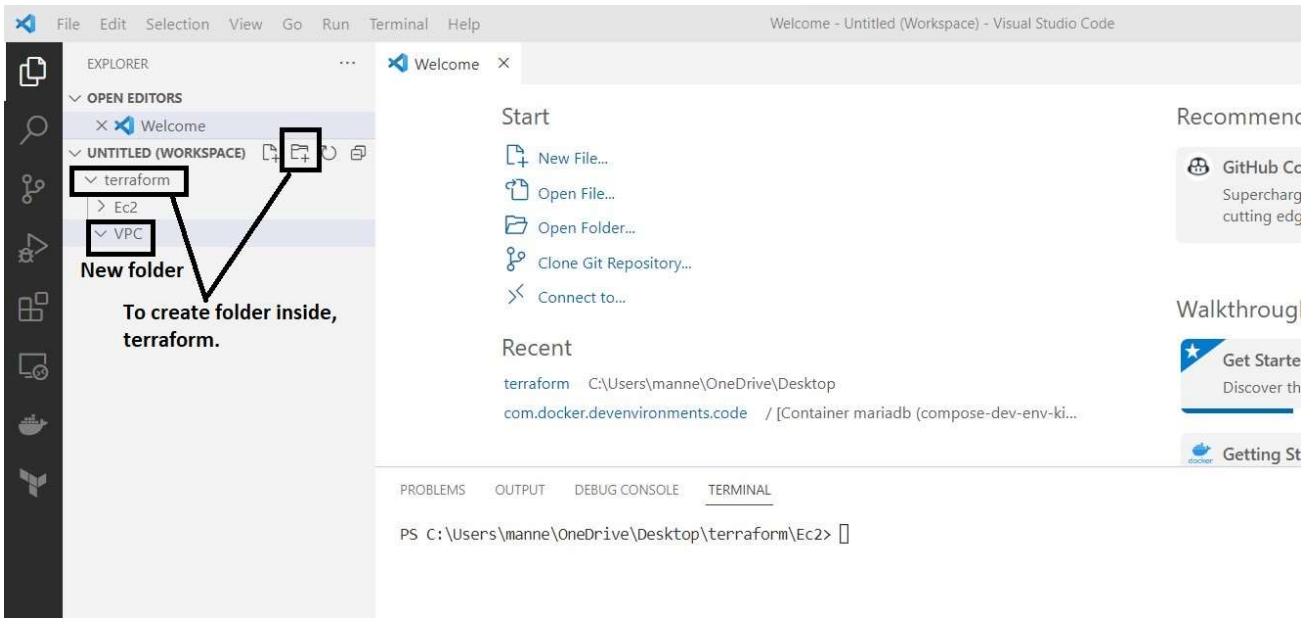
```
        - tags                 = {} -> null
        - throughput           = 125 -> null
        - volume_id            = "vol-02e5b775af393fad1" -> null
        - volume_size          = 8 -> null
        - volume_type          = "gp3" -> null
      }
  }

Plan: 0 to add, 0 to change, 1 to destroy.
aws_instance.web: Destroying... [id=i-01b3de39aa141b819]
aws_instance.web: Still destroying... [id=i-01b3de39aa141b819, 10s elapsed]
aws_instance.web: Still destroying... [id=i-01b3de39aa141b819, 20s elapsed]
aws_instance.web: Still destroying... [id=i-01b3de39aa141b819, 30s elapsed]
aws_instance.web: Destruction complete after 32s

Destroy complete! Resources: 1 destroyed.
```

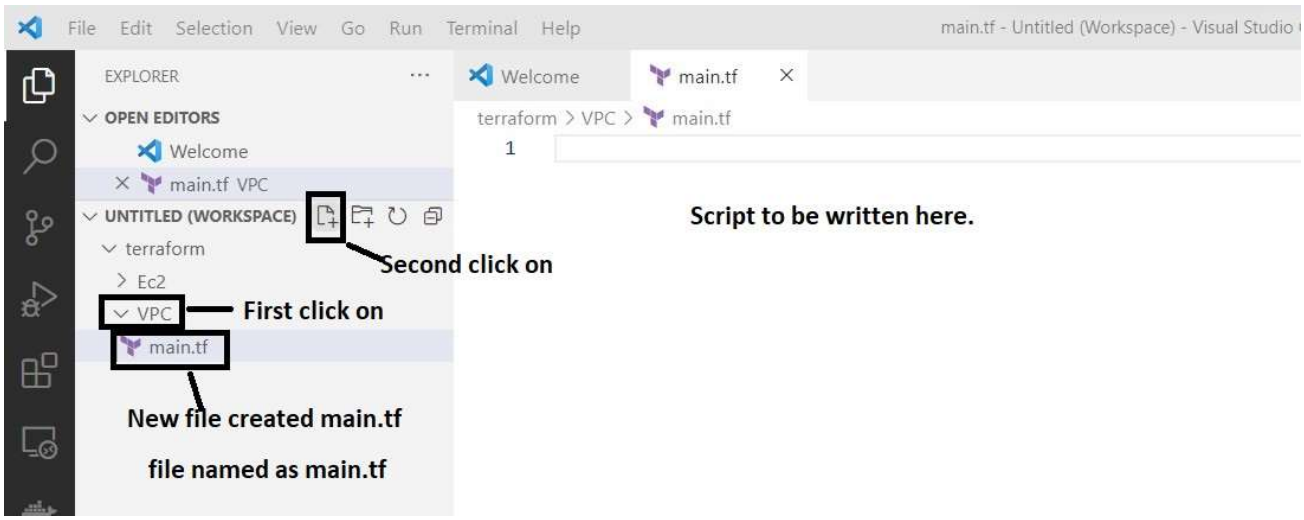
Instance terminated

View the termination status in AWS.



To create VPC, again we have to create a new folder inside terraform folder and named it as VPC. Again, create a file with name ‘main.tf’,





Main.tf script

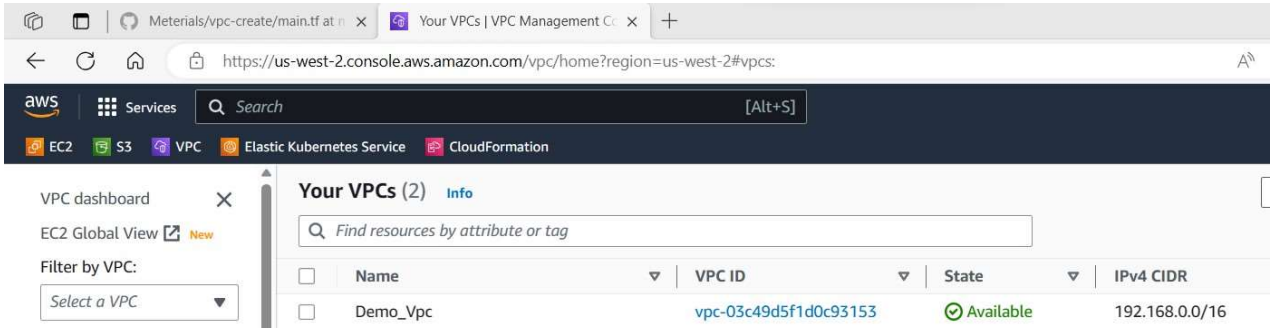
```
provider "aws" {
 region = var.aws_region
}
resource "aws_vpc" "main" {
   cidr_block = var.cidr_block[0]
   tags = {
     Name = var.vpc_name
   }
}
resource "aws_subnet" "subnet1" {
 vpc_id = aws_vpc.main.id
 cidr_block = var.cidr_block[1]
 tags = {
   Name = var.subnet_name[0]
 }
}
resource "aws_subnet" "subnet2" {
  vpc_id = aws_vpc.main.id
  cidr_block = var.cidr_block[2]
  tags = {
    Name = var.subnet_name[1]
  }
}
```

After saving the file, we have to run the command '**terraform init**' in terminal.

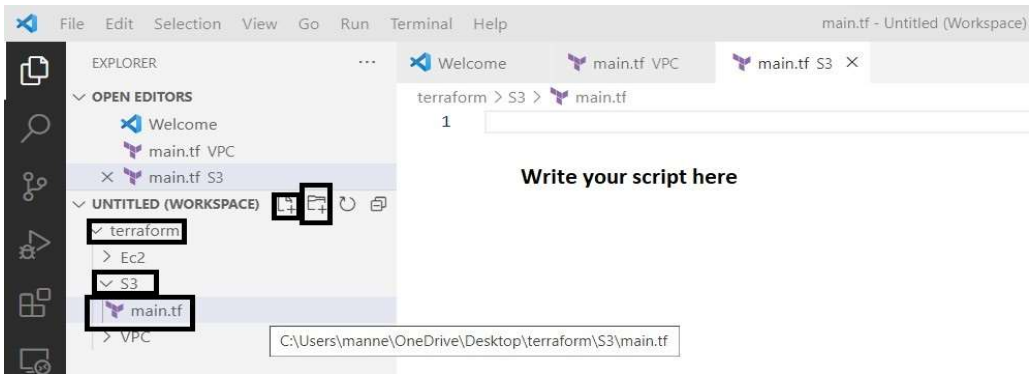
After that, successful initialization run the command '**terraform plan**' in terminal.

After that, run the command '**terraform apply –auto-approve=true**' in terminal.

After that, run the command '**terraform destroy –auto-approve=true**' in terminal.



To create S3 bucket, again we have to create a new folder inside terraform folder and named it as S3. Again, create a file with name 'main.tf'.



S3 Script

```
provider "aws" {
   region = "us-west-2"
}
resource "aws_s3_bucket" "testbkt" {
  bucket = "mytftestbkt-kelly999"

  tags = {
    Name       = "My bucket"
  }
}
```

After saving the file, we have to run the command '**terraform init**' in terminal.

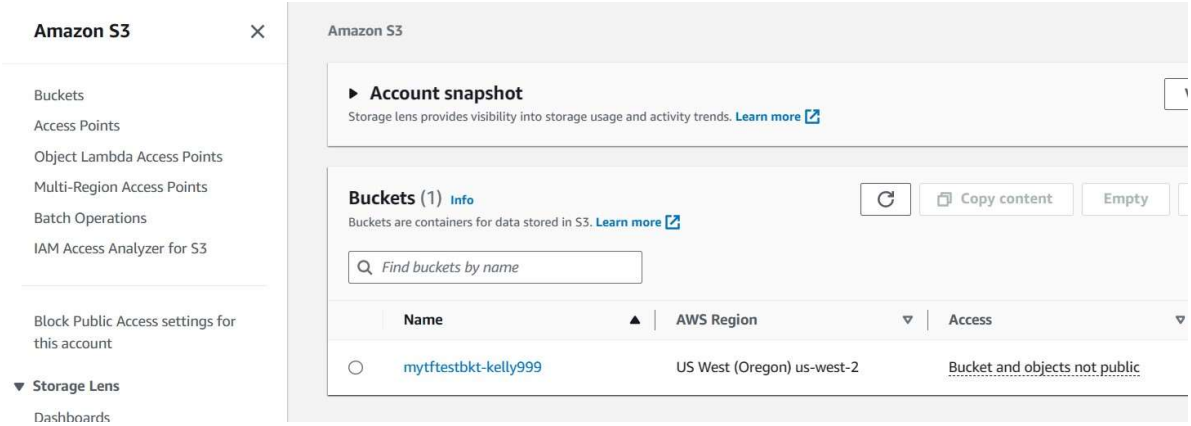
After that, successful initialization run the command '**terraform plan**' in terminal.

After that, run the command '**terraform apply –auto-approve=true**' in terminal.

After that, run the command '**terraform destroy –auto-approve=true**' in terminal.

```
      }
    + tags_all                    = {
        + "Name" = "My bucket"
      }
    + website_domain              = (known after apply)
    + website_endpoint            = (known after apply)
  }

Plan: 1 to add, 0 to change, 0 to destroy.
aws_s3_bucket.testbkt: Creating...
aws_s3_bucket.testbkt: Still creating... [10s elapsed]
aws_s3_bucket.testbkt: Still creating... [20s elapsed]
aws_s3_bucket.testbkt: Creation complete after 23s [id=mytftestbkt-kelly999]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
PS C:\Users\manne\OneDrive\Desktop\terraform\s3>
```

After creating S3 bucket, view it in the AWS console.



8) Using terraform script instance creation, connecting and installation of software's?

A) After writing script in main.tf file, save it and execute the following commands in terminal. For Ec2 instance creation and connecting to it we require key-pair. So, initially we have to create a key-pair. As follows

First create a directory to store the key-pair and after that enter into that directory and execute the command '**ssh-keygen -t rsa -f demo'** (key-pair name is demo)

Main.tf (script)

```
provider "aws" {
 region = "us-west-2"  # Replace with your preferred AWS region
}
resource "aws_key_pair" "test" {
 key_name     = "demo"
 public_key =  file("C:/Users/manne/OneDrive/Desktop/terraform/demo.pub")
}
resource "aws_instance" "web" {
 ami          = "ami-0c65adc9a5c1b5d7c"  # Replace with your desired AMI ID
 instance_type = "t2.micro"             # Replace with your desired instance type
 key_name     = aws_key_pair.test.key_name         # Replace with your key pair name
 vpc_security_group_ids = ["sg-026d6806f077f66a5"]   # your security group id to be replaced.
 tags = {
  Name = "Demo_Instance"
 }
 connection {
  type       = "ssh"
  user       = "ubuntu"
  private_key = file("C:/Users/manne/OneDrive/Desktop/terraform/demo")  # Replace with your private key path
  host       = self.public_ip
  timeout    = "1m"
  agent      = false
 }
 provisioner "remote-exec" {
  inline = [
   "sudo apt-get update",
   "sudo apt-get install tomcat9 tomcat9-admin -y",  #Installing tomcat server.
  ]
 }
}
```

We have to run the command '**terraform init**' in terminal.

```
PS C:\Users\manne\OneDrive\Desktop\terraform\instance connect with key> terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.10.0

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.
```

After that, successful initialization run the command '**terraform plan**' in terminal.

```
PS C:\Users\manne\OneDrive\Desktop\terraform\instance connect with key> terraform plan

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.web will be created
  + resource "aws_instance" "web" {
      + ami                                  = "ami-0c65adc9a5c1b5d7c"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + key_name                             = "demo"
      + key_name_prefix                      = (known after apply)
      + key_pair_id                          = (known after apply)
      + key_type                             = (known after apply)
      + public_key                           = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQC40y5A380J0I3XgbxZj9eSIHxHg1oF2HurGkRu2zGWbi+4z0vXMylCe1viEjQMGyCbk7DK4Qsu9vNR3k01r
lskffmHoACEg4ymET9UFopVK+c96aJidILKRMSok+YjPVnK1qwlrevCj6MjPt+B7T3rp1cW/6nibNw0JzjX6VgNAsnnQ9D53SBHHqn4XMddNYqp1KKK90W34ZXXlPWaGcNTEm0Lii60lWFXlgy+e
06hRicVzL9hKkklYc0uwLq0otDl7qMEdT/lVWa9w4MfcKgbZacHjN6Jc6roqwjTFFRFlLqb244m66+Yij5oarsYwYpHdZzscedi+b/8tS4OQiXxKiIluAfUiWeYDBWPxhzBypGzphaBQbl+YLVZE
qYND2F4nnGCUkM316FKefmMhTxcVk+N50+CyQLzdl1h0B8Xw+J2eptIaWokWW6H93z4RLz+eiCws30lf/202HtFmzXsPAZosmlneJP/K4h6odILNiziEDHh2q5oaYCSj+WM7IU= manne@Anil"
      + tags_all                             = (known after apply)
    }

Plan: 2 to add, 0 to change, 0 to destroy.
```

After that, for creation run the command '**terraform apply –auto-approve=true**' in terminal.

```
PS C:\Users\manne\OneDrive\Desktop\terraform\instance connect with key> terraform apply --auto-approve=true

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.web will be created
  + resource "aws_instance" "web" {
      + ami                                  = "ami-0c65adc9a5c1b5d7c"
      + 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)
aws_instance.web (remote-exec): Created symlink /etc/systemd/system/multi-user.target.wants/tomcat9.service → /lib/systemd/system/tomcat9.service.
aws_instance.web (remote-exec): Processing triggers for libc-bin (2.31-0ubuntu9.9) ...
aws_instance.web (remote-exec): Processing triggers for rsyslog (8.2001.0-1ubuntu1.3) ...
aws_instance.web (remote-exec): Processing triggers for man-db (2.9.1-1) ...
aws_instance.web (remote-exec): Processing triggers for ca-certificates (20211016ubuntu0.20.04.1) ...
aws_instance.web (remote-exec): Updating certificates in /etc/ssl/certs...
aws_instance.web (remote-exec): 0 added, 0 removed; done.
aws_instance.web (remote-exec): Running hooks in /etc/ca-certificates/update.d...

aws_instance.web: Still creating... [1m20s elapsed]
aws_instance.web (remote-exec): done.
aws_instance.web (remote-exec): done.
aws_instance.web: Creation complete after 1m28s [id=i-084149dbb98921116]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

After that, for termination run the command '**terraform destroy –auto-approve=true**'

```
PS C:\Users\manne\OneDrive\Desktop\terraform\tomcat install> terraform destroy --auto-approve=true
aws_key_pair.test: Refreshing state... [id=demo]
aws_instance.web: Refreshing state... [id=i-084149dbb98921116]

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.web will be destroyed
  - resource "aws_instance" "web" {
      - ami                                  = "ami-0c65adc9a5c1b5d7c" -> null
      - arn                                  = "arn:aws:ec2:us-west-2:488606592963:instance/i-084149dbb98921116" -> null
      - associate_public_ip_address          = true -> null
```

9) What is Terraform modules and Example script?

A) Terraform module is an encapsulated and reusable collection of Terraform configurations that represent a specific piece of infrastructure or a set of related resources. Modules help you organize your Terraform code and promote best practices like modularity, reusability, and maintainability.

Main.tf script

```
provider "aws" {
 region = "us-west-2"
}
module "vpc" {
 source = "terraform-aws-modules/vpc/aws"
 name = "my-vpc"
 cidr = "10.0.0.0/16"
 azs           = ["us-west-2a", "us-west-2b", "us-west-2c"]
 private_subnets = ["10.0.1.0/24", "10.0.2.0/24"]
 public_subnets  = ["10.0.101.0/24", "10.0.102.0/24"]
 enable_nat_gateway = false
 enable_vpn_gateway = false
 tags = {
  Terraform = "true"
  Environment = "dev"
 }
}
module "ec2_instance" {
 source  = "terraform-aws-modules/ec2-instance/aws"
 name = "single-instance"
 instance_type        = "t2.micro"
 key_name             = "demo_key"
 monitoring           = false
 vpc_security_group_ids = ["sg-07ba58c4c9914f12a"]
 subnet_id            = "subnet-0c72c059eb405c8b3"
 tags = {
  Terraform   = "true"
  Environment = "dev"
 }
}
```

After saving the file, we have to run the command '**terraform init**' in terminal.

After that, successful initialization run the command '**terraform plan**' in terminal.

After that, run the command '**terraform apply –auto-approve=true**' in terminal.

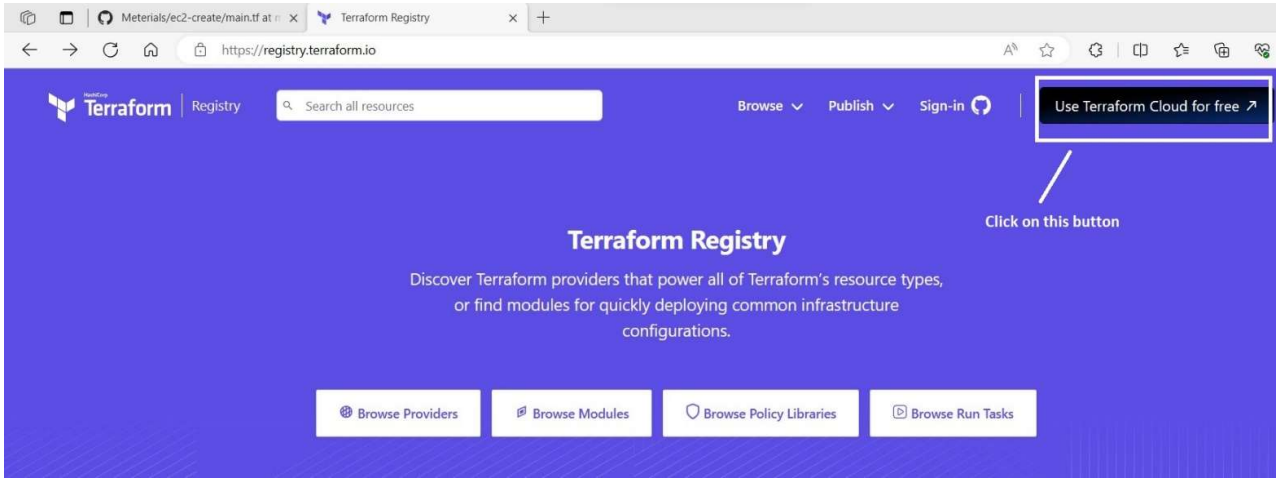
After that, run the command '**terraform destroy –auto-approve=true**' in terminal.

10) Terraform workspaces and cloud configuration?

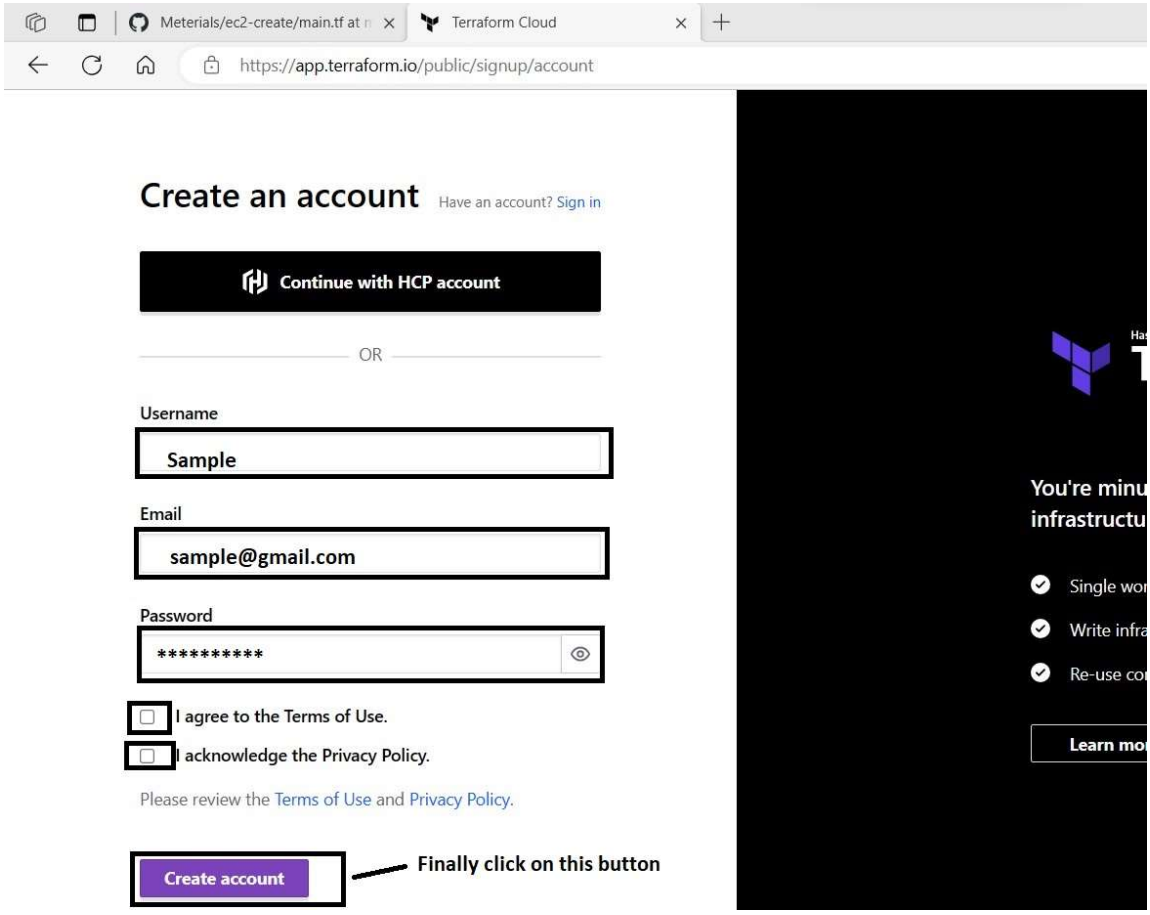
A) To create terraform registry account and configuring work spaces, follow the steps as mentioned below.

To register Link : https://registry.terraform.io

After opening the link, the following page will appear.

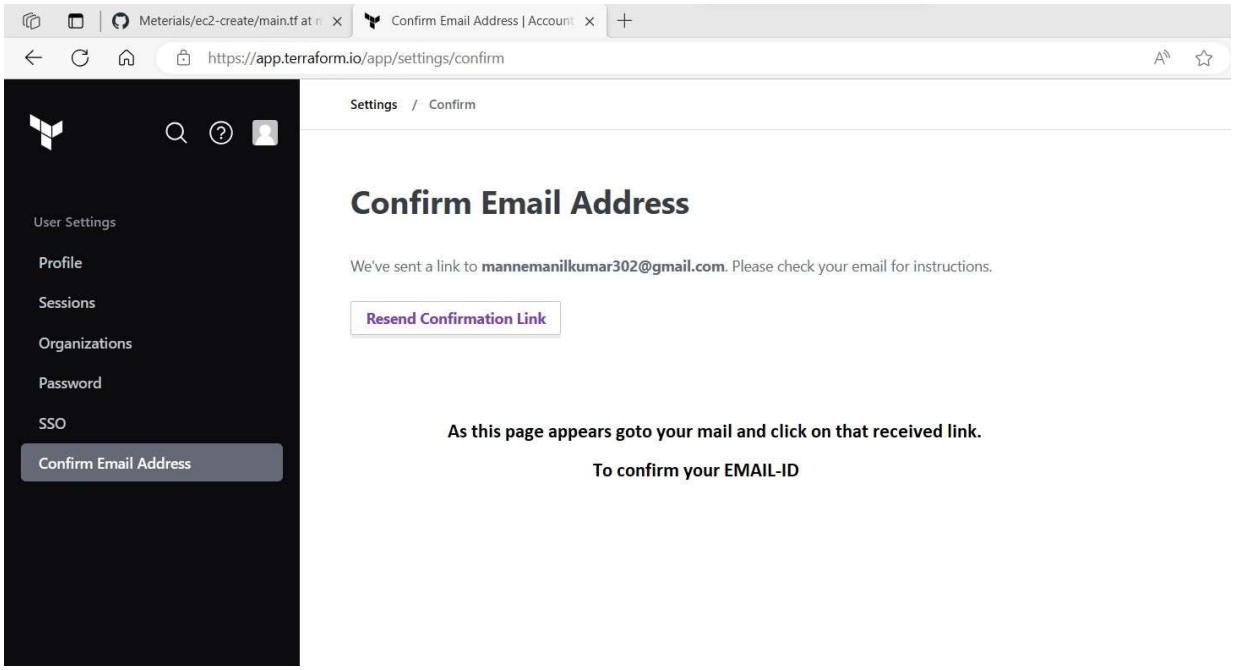


After clicking on that button, the following page will be displayed.

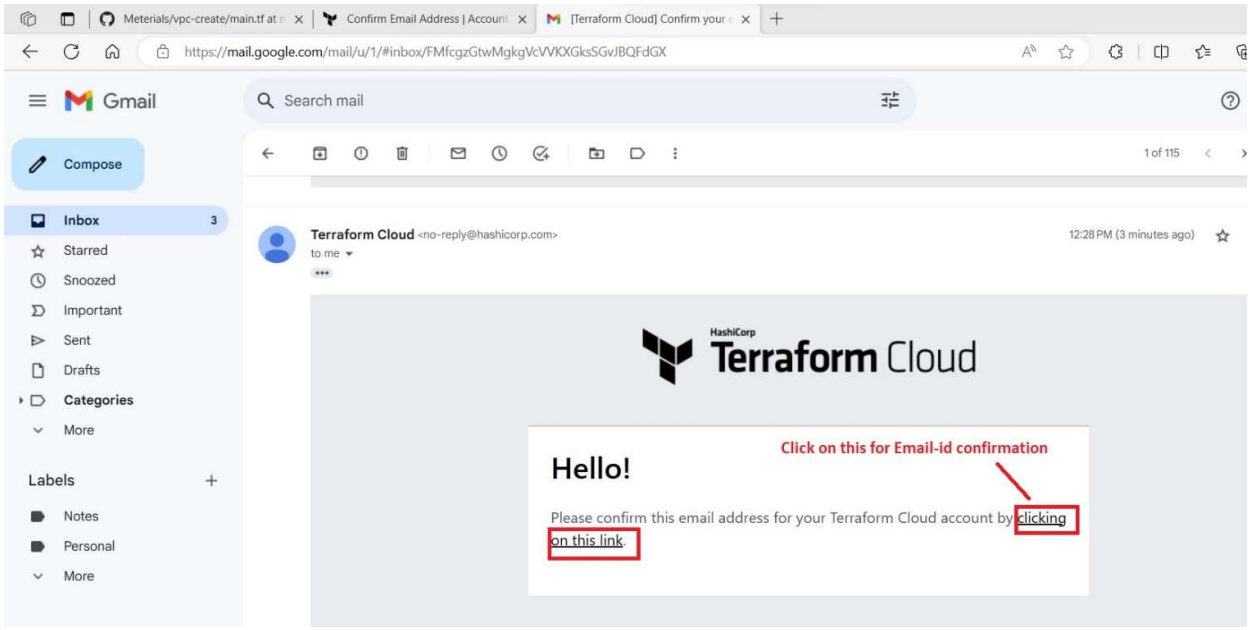


After clicking on button ‘create account’, the following page will appear. Again, we have to enter our password. Which was set by us at the time of account creation.

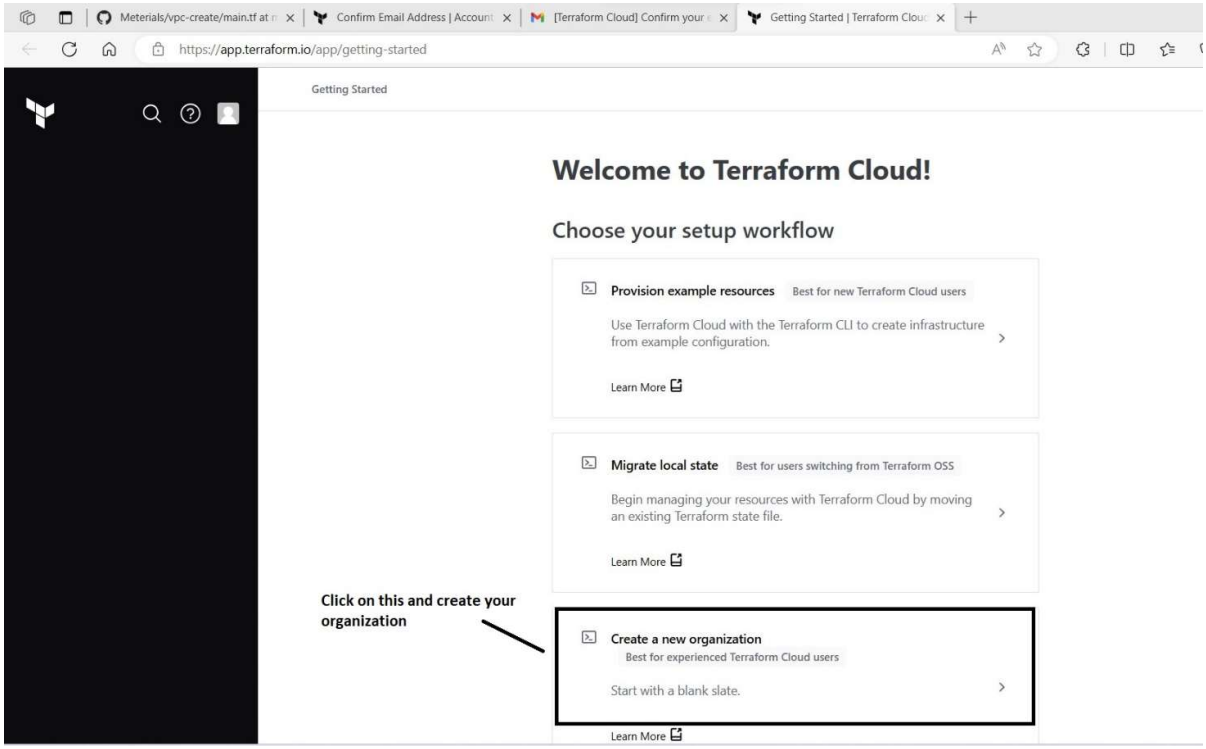
After clicking on the button ‘Authenticate’, the following page will appear.



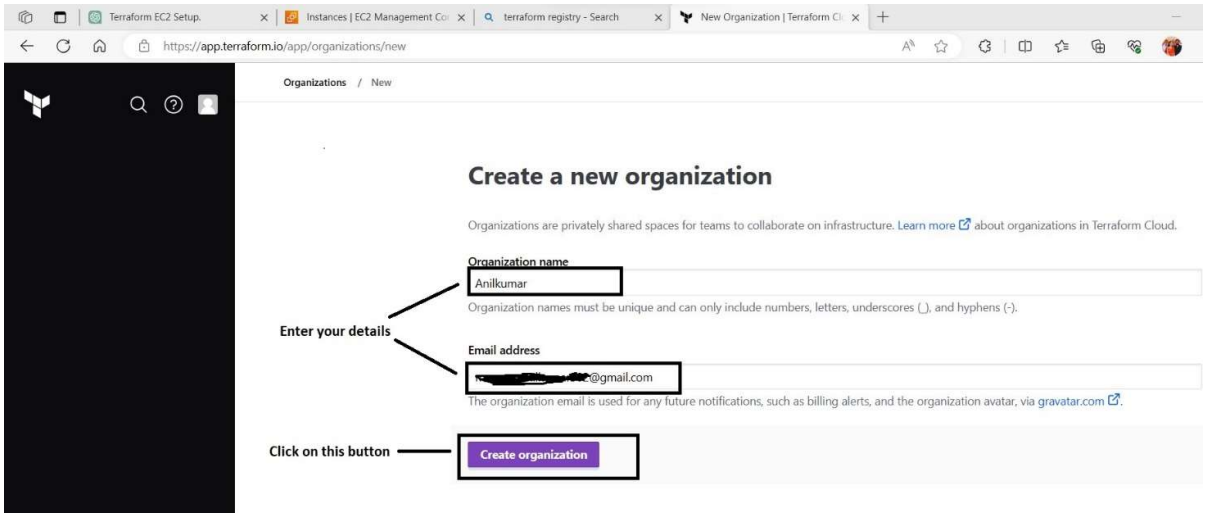
As this page appears, you have to login into your mail account through system as shown.



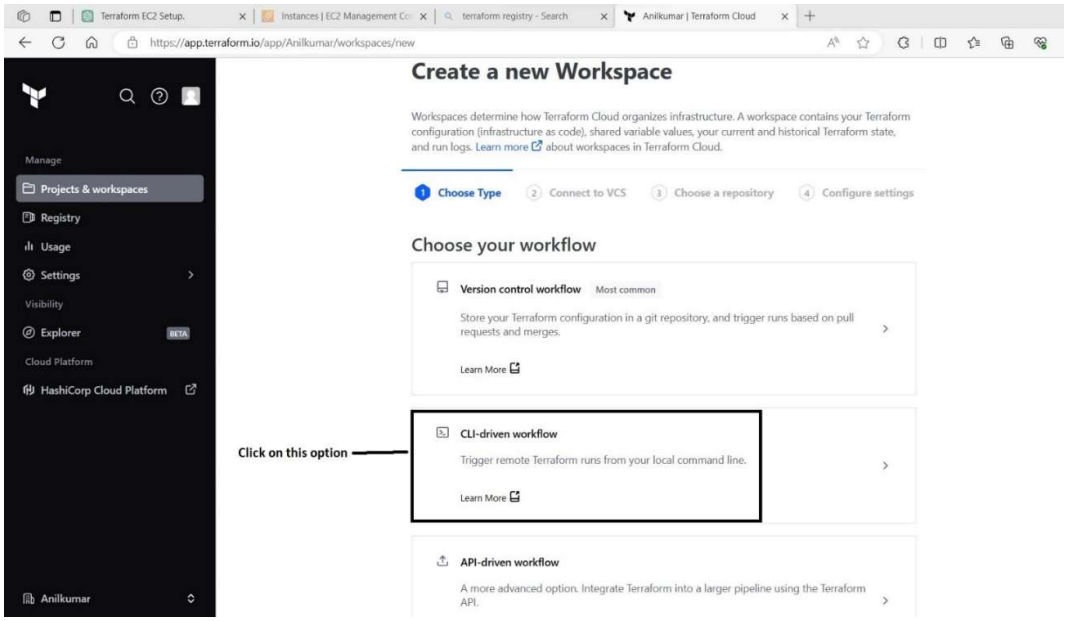
After clicking on the confirmation link, the following page will appear.



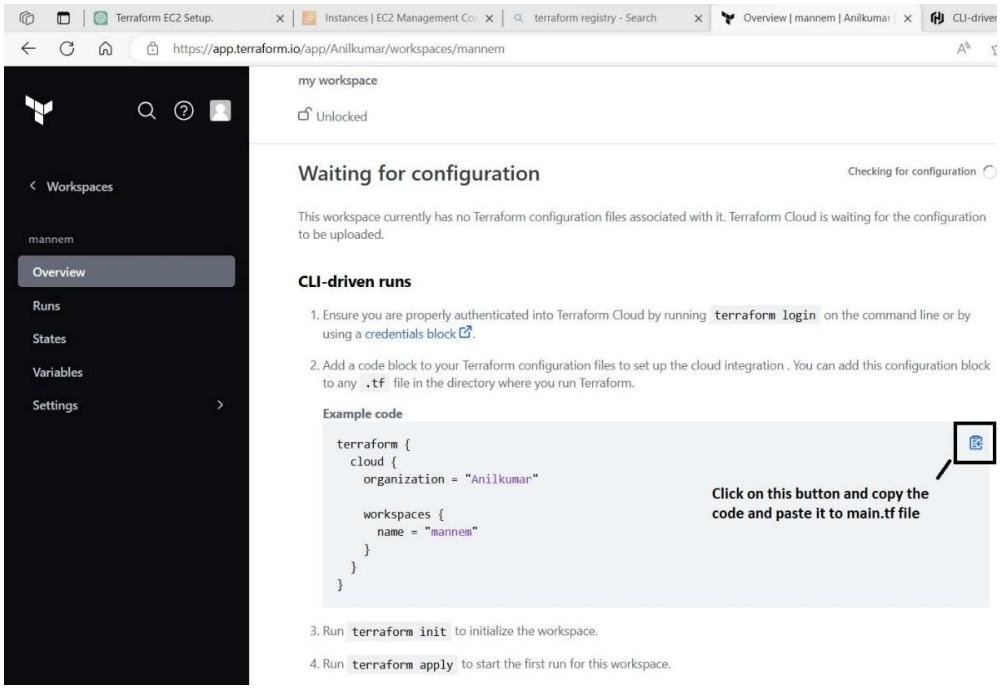
After clicking on the option ‘Create a new organization’ the following page will appear.



After clicking on that button ‘Create organization’ the following page will be appeared.



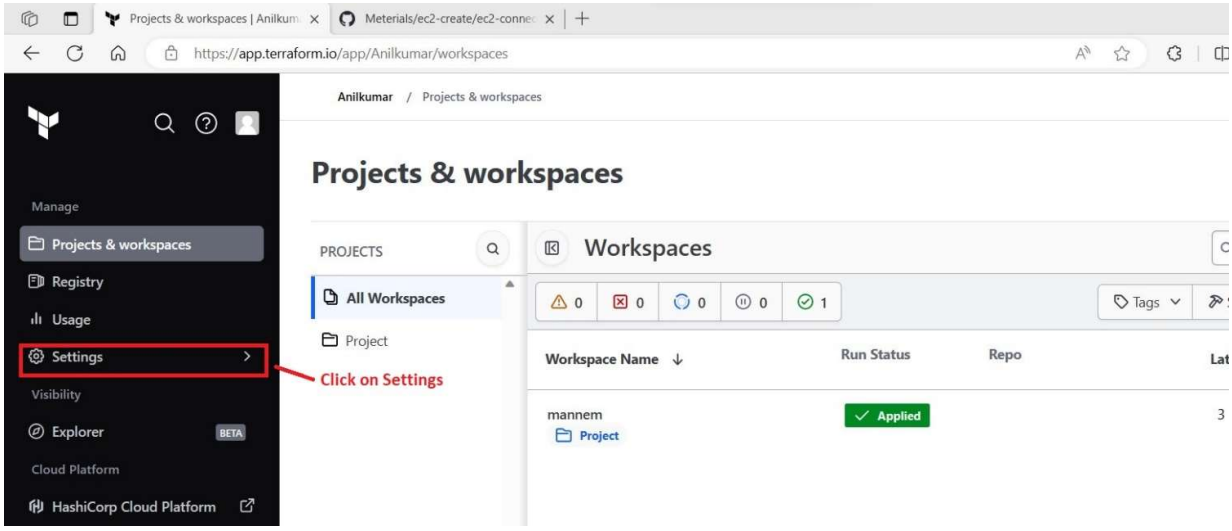
After selecting that option ‘CLI-driven workflow’, the following page will be displayed.



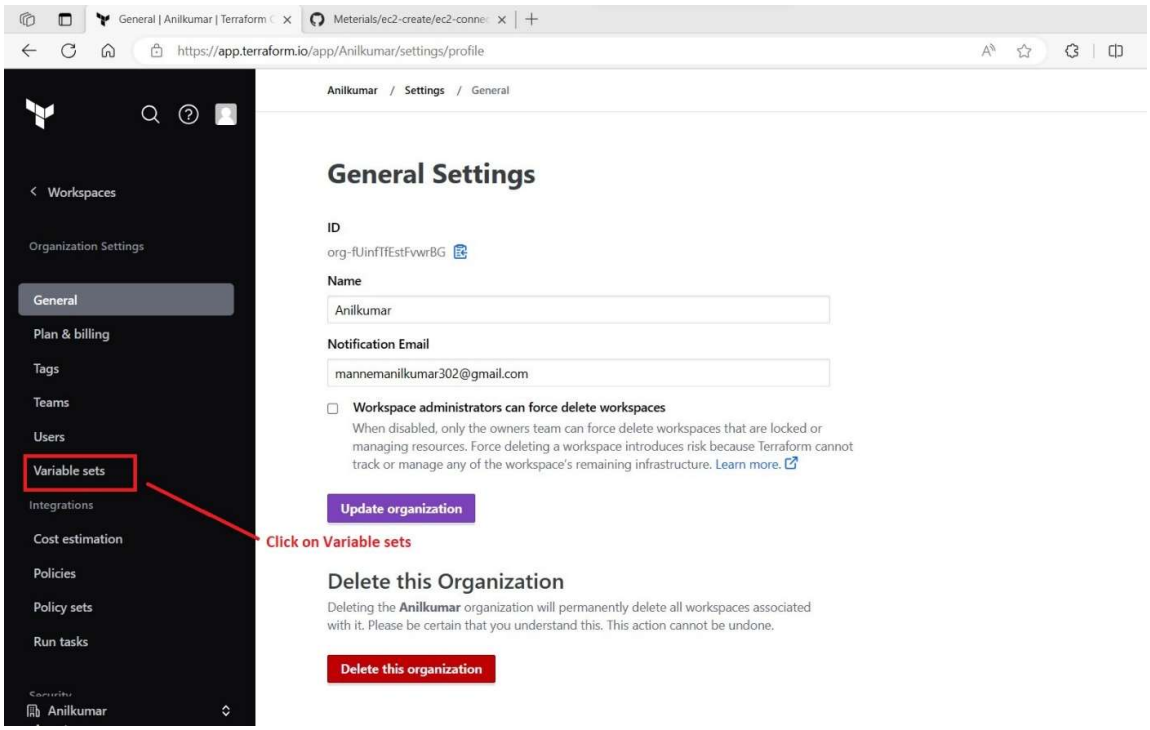
Main.tf (script)

```
provider "aws" {
   region = "us-west-2"
   }
terraform {
 cloud {
   organization = "Anilkumar"    # Code copied from terraform registry.
   workspaces {
     name = "mannem"
   }
 }
}
resource "aws_instance" "web" {
 ami   = "ami-08541bb85074a743a"
 instance_type = "t2.micro"
 tags = {
   Name = "My_First_Instance"
 }
}
```

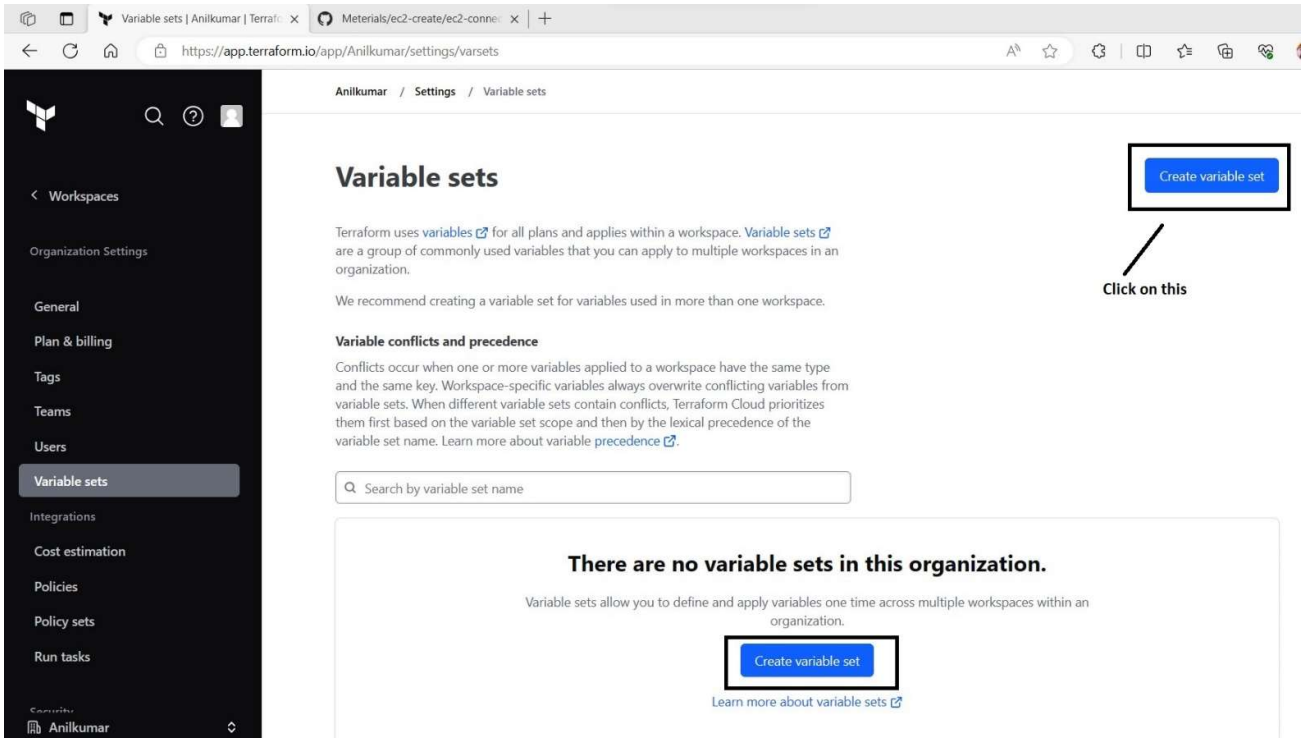
After copying that code in main.tf file, we have to create the variable sets in terraform registry as follows.



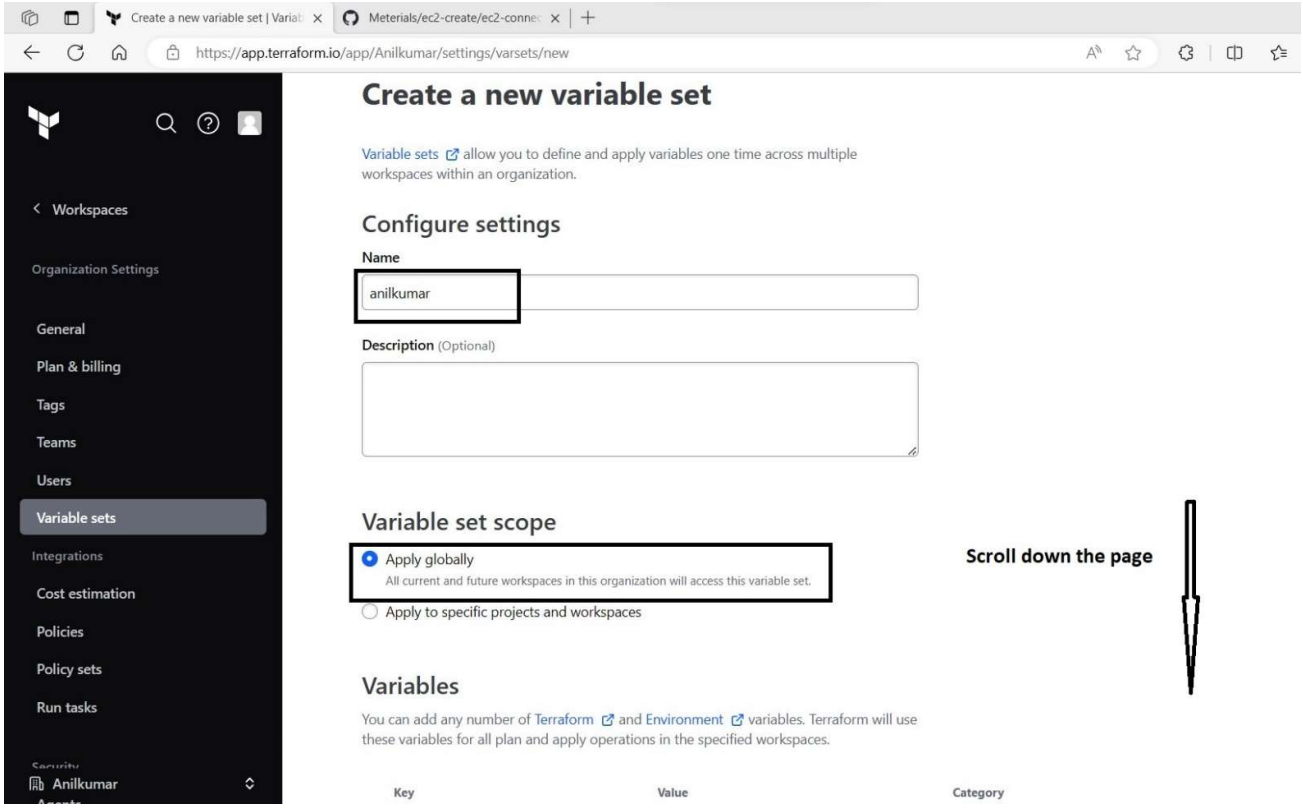
After clicking on settings, the following page will appear.



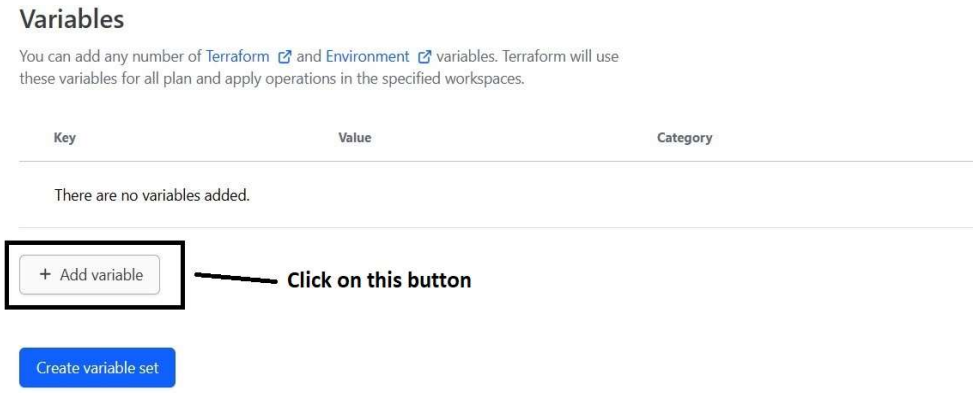
After clicking on variable sets, the following page will appear.



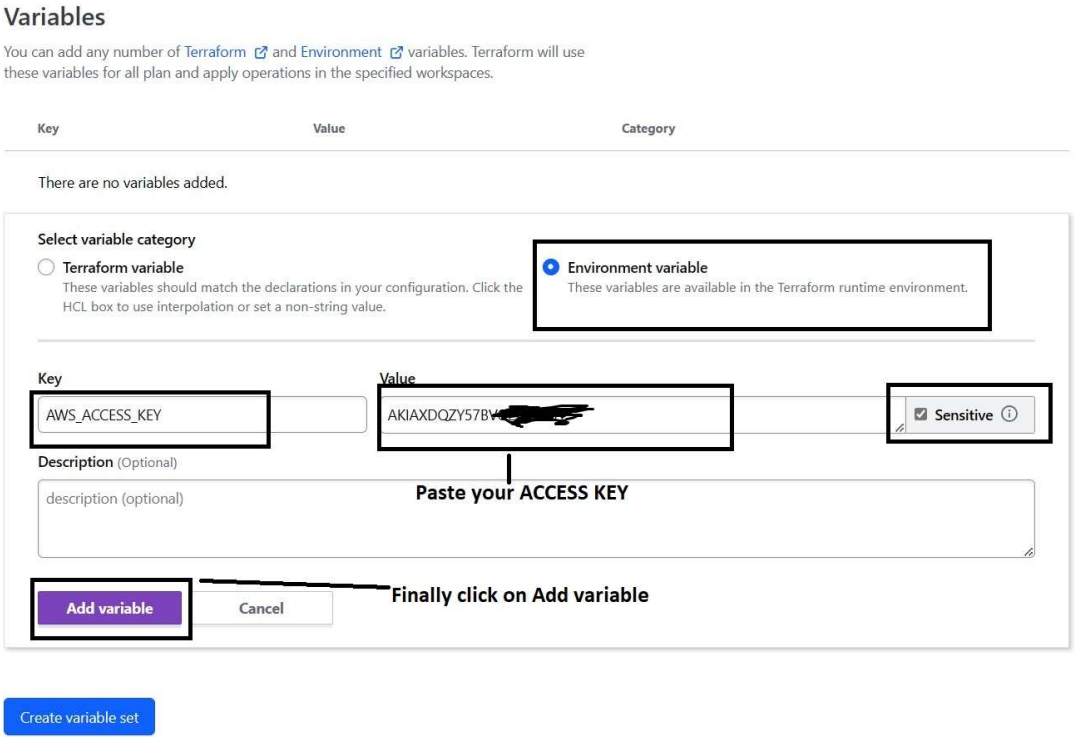
After clicking on create variable set, the following page will appear.



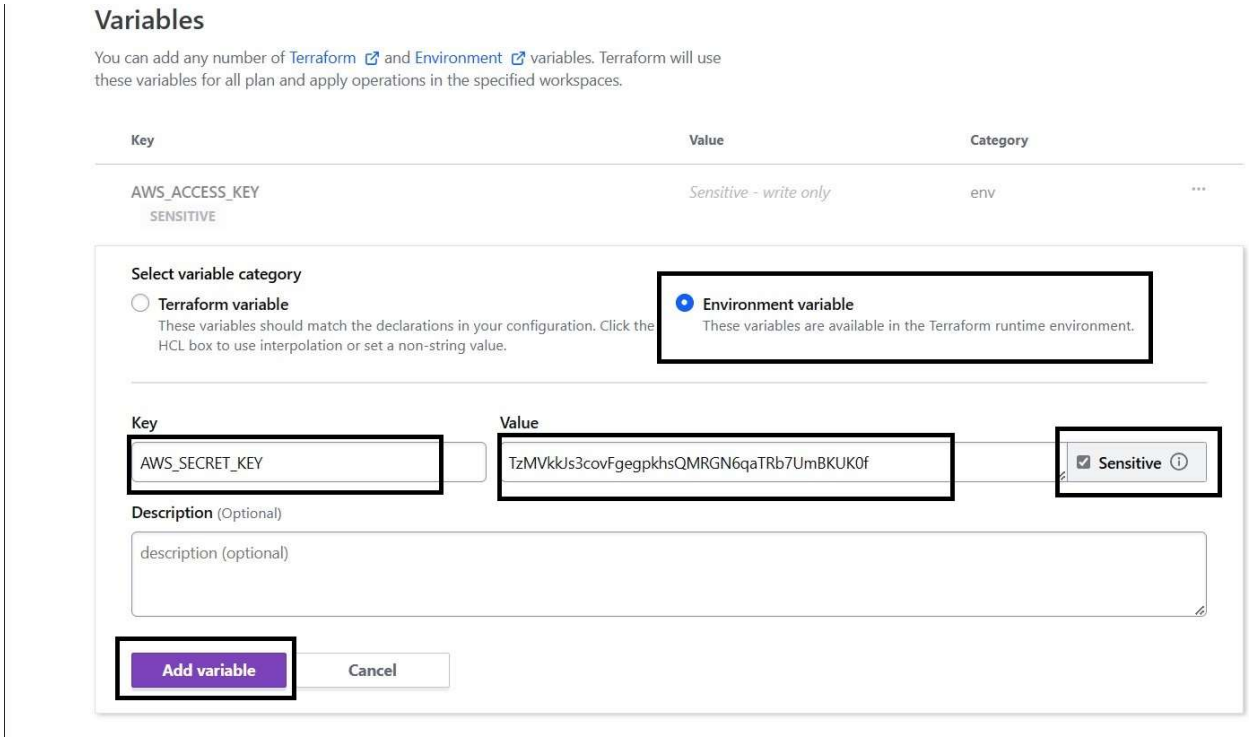
After scrolling down the page, the following options will be visible.



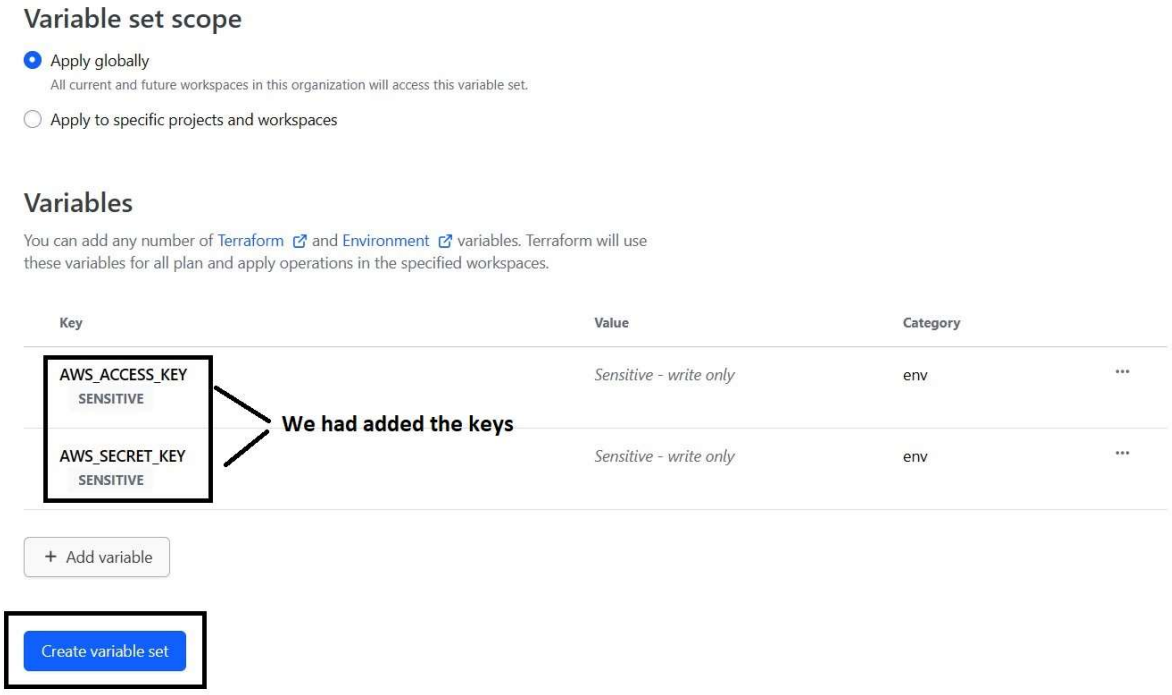
After click on that button ‘add variable’, the following page will appear.



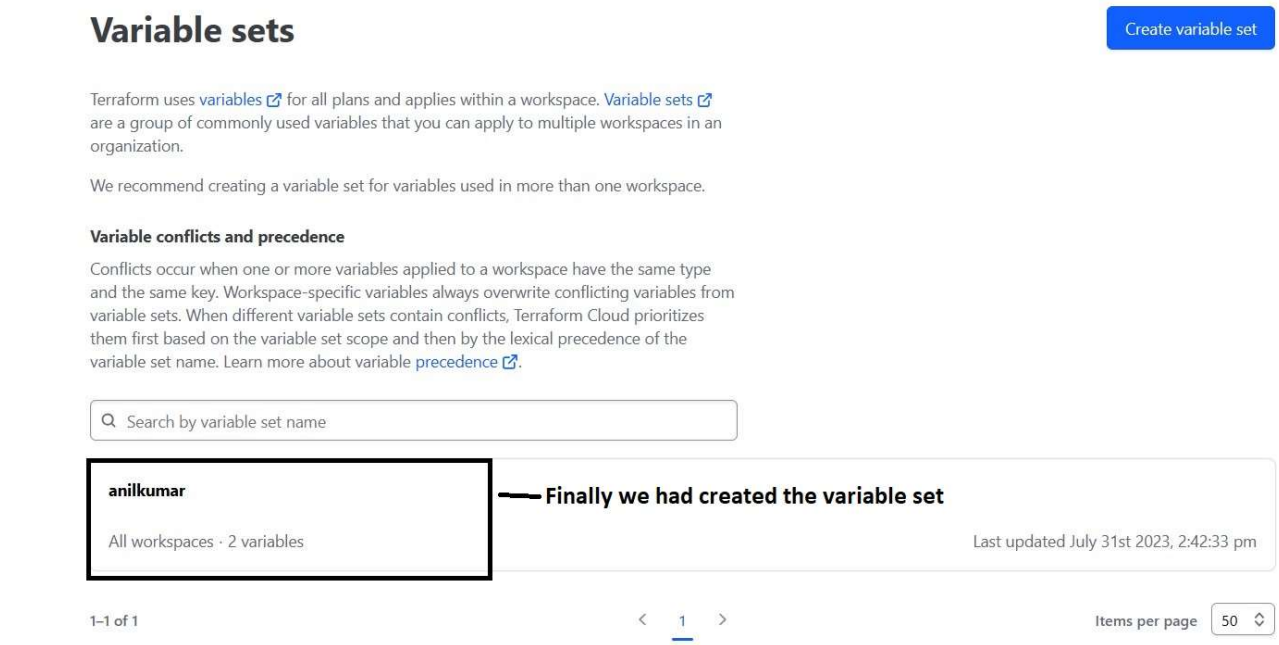
After clicking on ‘Add variable’, the following page will appear.



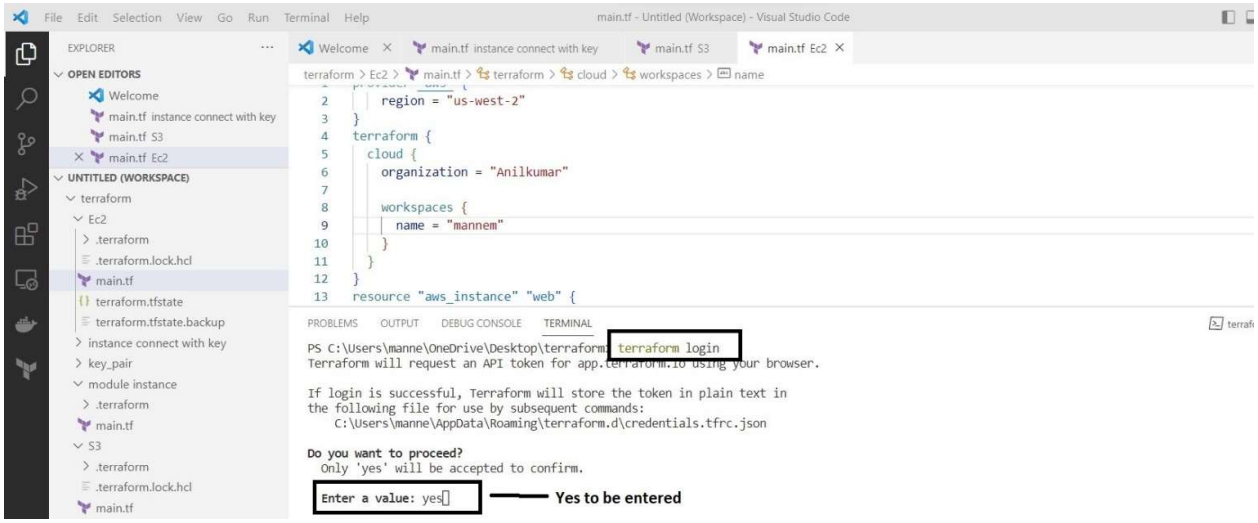
Again, we have click on ‘Add variable’, the following page will appear.



After clicking on ‘Create variable set’ button, the following page will be displayed.

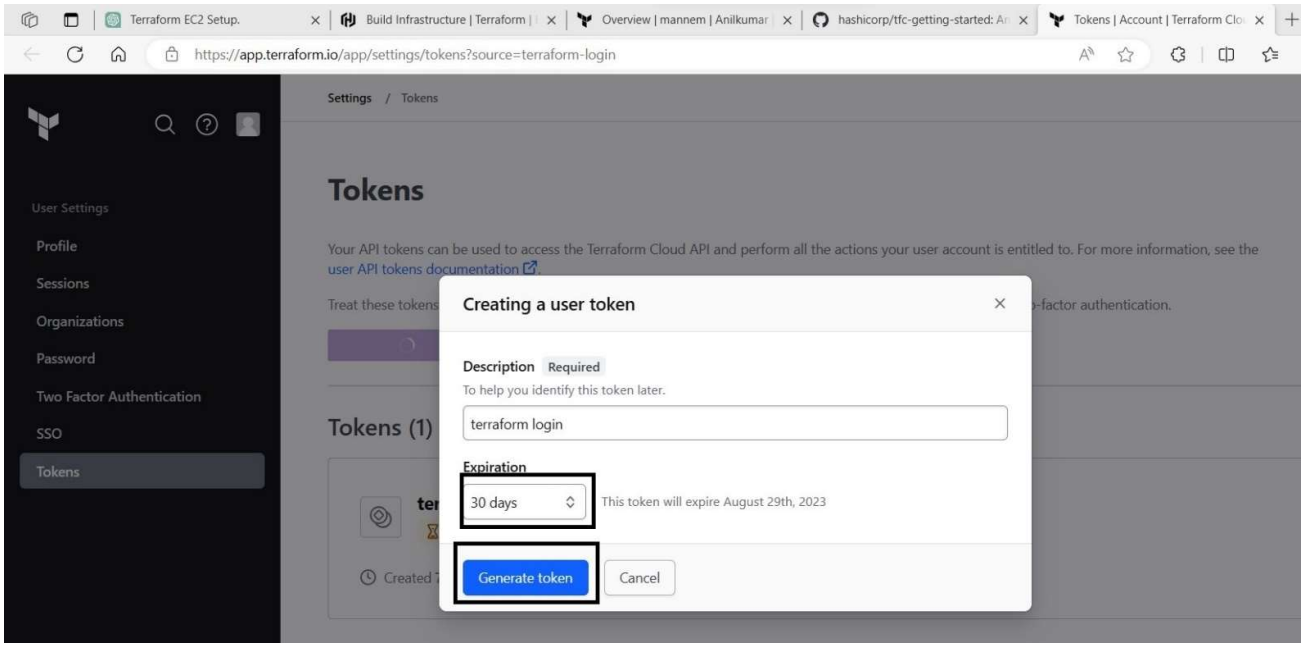


Then after, First run the command ‘**terraform login**’ in VS code terminal.

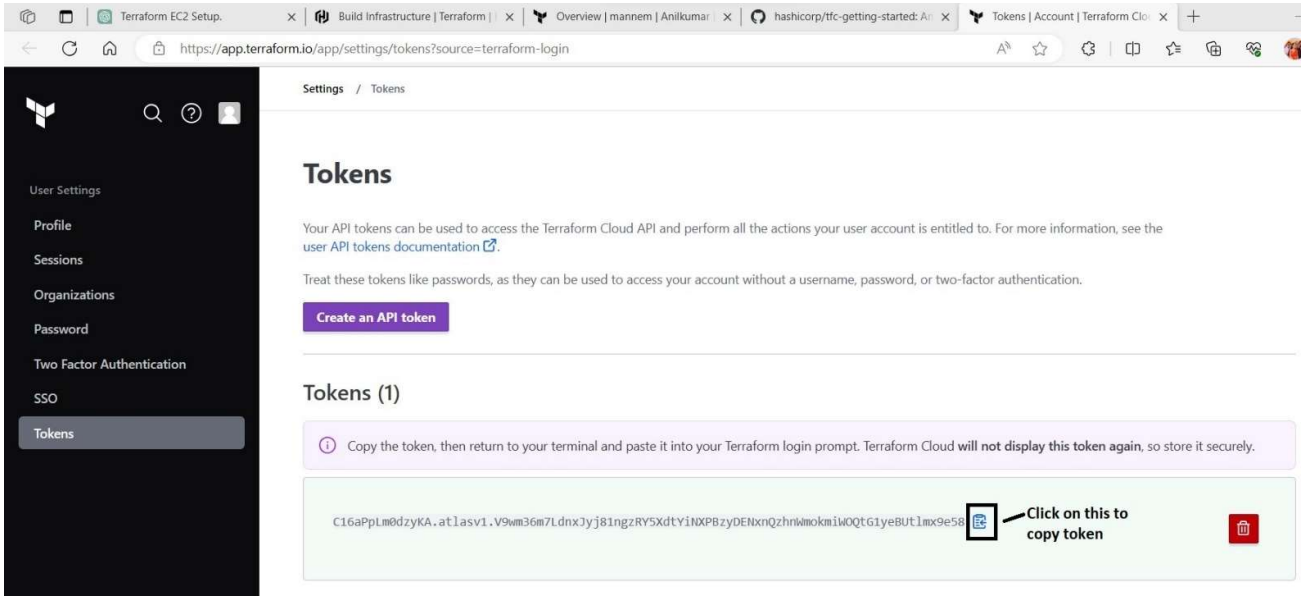


After entering yes, click enter. The following screen will appear and in parallel automatically web browser will be opened. Copy that generated token and paste it.

After entering yes, for token generation the following will appear.



After clicking the button ‘Generate token’, the following page will appear.



After pasting the token, click enter. The following screen will appear.

After successful login, run the command '**terraform init'** in the terminal.

```
PS C:\Users\manne\OneDrive\Desktop\terraform\Ec2> terraform init

Initializing Terraform Cloud...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.10.0

Terraform Cloud has been successfully initialized!

You may now begin working with Terraform Cloud. Try running "terraform plan" to
see any changes that are required for your infrastructure.

If you ever set or change modules or Terraform Settings, run "terraform init"
again to reinitialize your working directory.
```

After successful initialization, run the command '**terraform plan'** in the terminal.

```
PS C:\Users\manne\OneDrive\Desktop\terraform\Ec2> terraform plan
Running plan in Terraform Cloud. Output will stream here. Pressing Ctrl-C
will stop streaming the logs, but will not stop the plan running remotely.

Preparing the remote plan...

To view this run in a browser, visit:
https://app.terraform.io/app/Anilkumar/mannem/runs/run-R6Zna4Dedppo6Q3t

Waiting for the plan to start...

Terraform v1.5.4
on linux_amd64
Initializing plugins and modules...

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.web will be created
  + resource "aws_instance" "web" {
      + ami                                  = "ami-08541bb85074a743a"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)

        }
      + tenancy                              = (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      + user_data_replace_on_change          = false
      + vpc_security_group_ids               = (known after apply)
    }

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

After that, you have to run the command '**terraform apply –auto-approve=true**'.

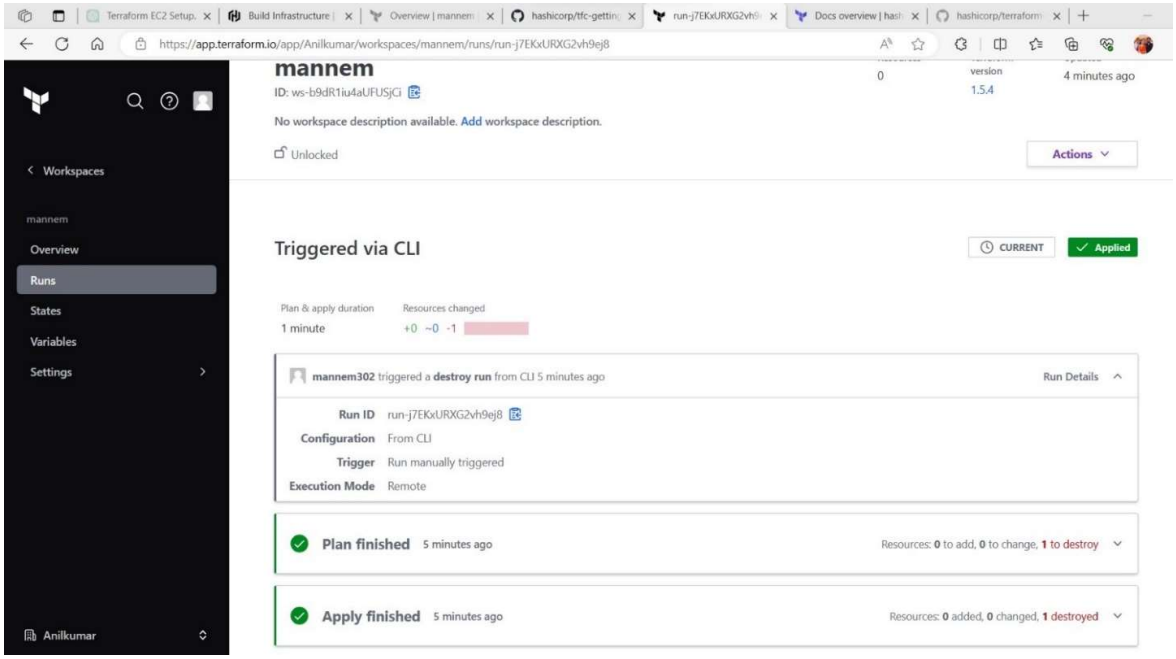
```
PS C:\Users\manne\OneDrive\Desktop\terraform\Ec2> terraform apply --auto-approve=true
Running apply in Terraform Cloud. Output will stream here. Pressing Ctrl-C
will cancel the remote apply if it's still pending. If the apply started it
will stop streaming the logs, but will not stop the apply running remotely.

Preparing the remote apply...

To view this run in a browser, visit:
https://app.terraform.io/app/Anilkumar/mannem/runs/run-vAkzoGaPXMKdZNUL

Waiting for the manually locked workspace to be unlocked...

Terraform v1.5.4
on linux_amd64
Initializing plugins and modules...

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


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

aws_instance.web: Creating...
aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Still creating... [20s elapsed]
aws_instance.web: Still creating... [30s elapsed]
aws_instance.web: Creation complete after 32s [id=i-07f06d10cb947255f]

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

Instance created and tfstate file was uploaded to terraform registry.

To terminate the instance, run the command '**terraform destroy –auto-approve=true**'.

```
aws_instance.web: Destroying... [id=i-0eed04333b1ba3c88]
aws_instance.web: Still destroying... [10s elapsed]
aws_instance.web: Still destroying... [20s elapsed]
aws_instance.web: Still destroying... [30s elapsed]
aws_instance.web: Still destroying... [40s elapsed]
aws_instance.web: Destruction complete after 40s

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

For confirmation purpose, we can view in terraform registry. As shown below.



11) Terraform backend configuration, to store terraform.tfstate file?

A) To store the terraform.tfstate file in S3 bucket, use the following script.

Main.tf (Script)

```
provider "aws" {
  region = "us-west-2"
}
terraform {
 backend "s3" {
  bucket       = "anilkumar9999"  # In S3 already, this bucket was available.
  key          = "C:/Users/manne/OneDrive/Desktop/terraform/Ec2/terraform.tfstate"
  region       = "us-west-2"
 }
}
resource "aws_instance" "web" {
 ami   = "ami-08541bb85074a743a"
 instance_type = "t2.micro"
 tags = {
  Name = "My_First_Instance"
 }
}
```

After writing the main.tf script, save the file and execute the below commands.

First run the command, '**terraform init'** in the terminal.

```
PS C:\Users\manne\OneDrive\Desktop\terraform\Ec2 s3 state file> terraform init

Initializing the backend...

Successfully configured the backend "s3"! Terraform will automatically
use this backend unless the backend configuration changes.

Initializing provider plugins...
```

After successful initialization, run the command, '**terraform plan'** in the terminal.

```
PS C:\Users\manne\OneDrive\Desktop\terraform\Ec2 s3 state file> terraform plan

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.web will be created
  + resource "aws_instance" "web" {
      + ami                                  = "ami-08541bb85074a743a"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)

        }
      + tenancy                              = (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      + user_data_replace_on_change          = false
      + vpc_security_group_ids               = (known after apply)
    }

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

After that run the command, '**terraform apply –auto-approve=true**' in terminal.

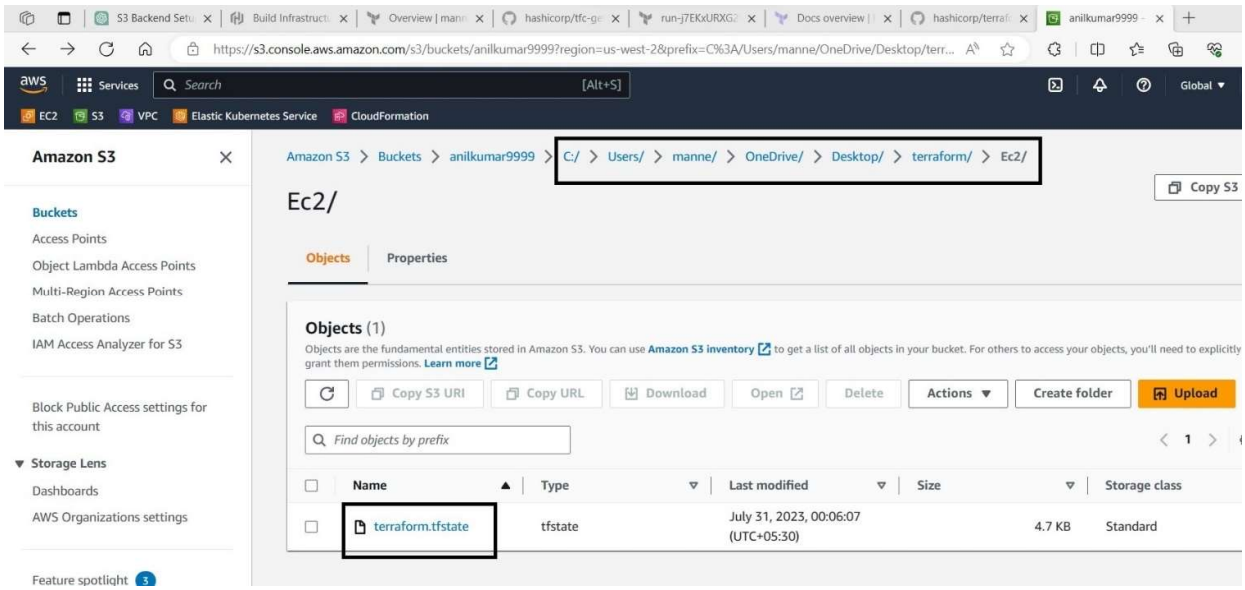
```
PS C:\Users\manne\OneDrive\Desktop\terraform\Ec2 s3 state file> terraform apply --auto-approve=true

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.web will be created
  + resource "aws_instance" "web" {
      + ami                                  = "ami-08541bb85074a743a"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)

      + tags_all                             = {
          + "Name" = "My_First_Instance"
        }
      + tenancy                              = (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      + user_data_replace_on_change          = false
      + vpc_security_group_ids               = (known after apply)
    }

aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Still creating... [20s elapsed]
aws_instance.web: Creation complete after 26s [id=i-080f4204420f50c54]

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

After that run the command, '**terraform destroy –auto-approve=true**' in terminal.

```
PS C:\Users\manne\OneDrive\Desktop\terraform\Ec2 s3 state file> terraform destroy --auto-approve=true
aws_instance.web: Refreshing state... [id=i-080f4204420f50c54]

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.web will be destroyed
  - resource "aws_instance" "web" {
      - ami                                  = "ami-08541bb85074a743a" -> null
      - arn                                  = "arn:aws:ec2:us-west-2:488606592963:instance/i-080f4204420f50c54" -> null
```

```
          - throughput           = 125 -> null
          - volume_id            = "vol-05e55067f9269d1f9" -> null
          - volume_size          = 8 -> null
          - volume_type          = "gp3" -> null
        }
    }

Plan: 0 to add, 0 to change, 1 to destroy.
aws_instance.web: Destroying... [id=i-080f4204420f50c54]
aws_instance.web: Still destroying... [id=i-080f4204420f50c54, 10s elapsed]
aws_instance.web: Still destroying... [id=i-080f4204420f50c54, 20s elapsed]
aws_instance.web: Still destroying... [id=i-080f4204420f50c54, 30s elapsed]
aws_instance.web: Destruction complete after 32s

Destroy complete! Resources: 1 destroyed.
```

For confirmation, view the S3 bucket in AWS console.



12) In terraform, what is terraform.tfstate file and its importance?

A) In Terraform, the state file (often referred to as terraform.tfstate) is a crucial component of the infrastructure management process. It is an automatically generated file that keeps track of the resources created and managed by Terraform. The state file records the current state of your infrastructure as defined in your Terraform configuration files. The terraform.tfstate file is a JSON-formatted file that contains detailed information about the resources Terraform manages, such as resource IDs, metadata, attributes, dependencies, and more. It acts as the source of truth for Terraform to understand the current state of your infrastructure. This file can be saved in terraform registry and as well as in S3 bucket for common state maintaining purpose in an organization.

13) Terraform variables creation (in different types)?

A) To declare variables in terraform, we need to create main.tf, variables.tf, values.tfvars and outputs.tf

Values.tfvars (script)

```
aws_region = "us-west-2"
cidr_block = ["192.168.0.0/16","192.168.0.0/24","192.168.1.0/24"]
vpc_name = "demovpc"
subnet_name = ["sn1","sn2"]
```

Outputs.tf (script)

```
output "vpc_name" {
value = var.vpc_name
}
output "subnets" {
 value = var.subnet_name
}
```

Main.tf(Script)

```
provider "aws" {
 region = var.aws_region
}
terraform {
 cloud {
  organization = "kellydevopsdemo"
     workspaces {
   name = "test"
  }
 }
}
resource "aws_vpc" "main" {
   cidr_block = var.cidr_block[0]
   tags = {
    Name = var.vpc_name
   }
}
resource "aws_subnet" "subnet1" {
 vpc_id = aws_vpc.main.id
 cidr_block = var.cidr_block[1]
 tags = {
  Name = var.subnet_name[0]
 }
}
resource "aws_subnet" "subnet2" {
  vpc_id = aws_vpc.main.id
  cidr_block = var.cidr_block[2]
  tags = {
   Name = var.subnet_name[1]
  }
}
```

Variables.tf(script)

```
variable "aws_region" {
 type = string
 #default = "us-west-2"
}
variable "vpc_name" {
 type = string
 #default = "testvpc"
}

variable "cidr_block" {
 type = list
 #default =
["192.168.0.0/16","192.168.0.0/24","192.168.1.0/24"]
}

variable "subnet_name" {
   type = list
   #default = ["sn1","sn2"]

}
```

After writing the scripts and saving the files, we have to execute the following commands '**terraform init**' in terminal. (screen shows in next page will be displayed)

```
PS C:\Users\manne\OneDrive\Desktop\terraform\variables execution> terraform init

Initializing Terraform Cloud...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.10.0

Terraform Cloud has been successfully initialized!

You may now begin working with Terraform Cloud. Try running "terraform plan" to
see any changes that are required for your infrastructure.

If you ever set or change modules or Terraform Settings, run "terraform init"
again to reinitialize your working directory.
```

After that run the command, '**terraform plan**' in terminal.

```
PS C:\Users\manne\OneDrive\Desktop\terraform\variables execution> terraform plan
Running plan in Terraform Cloud. Output will stream here. Pressing Ctrl-C
will stop streaming the logs, but will not stop the plan running remotely.

Preparing the remote plan...

To view this run in a browser, visit:
https://app.terraform.io/app/Anilkumar/mannem/runs/run-NNBSwBf61Hx9sCJk

Waiting for the plan to start...

Terraform v1.5.4
on linux_amd64
Initializing plugins and modules...

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_subnet.subnet1 will be created
  + resource "aws_subnet" "subnet1" {
      + arn                                            = (known after apply)
```

After that run the command, '**terraform plan –auto-approve=true**' in terminal.

```
PS C:\Users\manne\OneDrive\Desktop\terraform\Ec2 s3 state file> terraform apply --auto-approve=true

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.web will be created
  + resource "aws_instance" "web" {
      + ami                                  = "ami-08541bb85074a743a"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + tags_all                             = {
          + "Name" = "My_First_Instance"
        }
      + tenancy                              = (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      + user_data_replace_on_change          = false
      + vpc_security_group_ids               = (known after apply)
    }

aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Still creating... [20s elapsed]
aws_instance.web: Creation complete after 26s [id=i-080f4204420f50c54]

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

After that run the command, '**terraform destroy –auto-approve=true**' in terminal.

```
PS C:\Users\manne\OneDrive\Desktop\terraform\variables execution> terraform destroy --auto-approve=true
Running apply in Terraform Cloud. Output will stream here. Pressing Ctrl-C
will cancel the remote apply if it's still pending. If the apply started it
will stop streaming the logs, but will not stop the apply running remotely.

Preparing the remote apply...

To view this run in a browser, visit:
https://app.terraform.io/app/Anilkumar/mannem/runs/run-r46NCCfPCPzYtuoD

Waiting for the plan to start...

Terraform v1.5.4
on linux_amd64
Initializing plugins and modules...
aws_vpc.main: Refreshing state... [id=vpc-00e9e458b5577b124]
aws_subnet.subnet1: Refreshing state... [id=subnet-0587e5bc96e2afee7]
aws_subnet.subnet2: Refreshing state... [id=subnet-06be1fbead7f1e96f]

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

```
Changes to Outputs:
  - subnets  = [
      - "sn1",
      - "sn2",
    ] -> null
  - vpc_name = "testvpc" -> null

aws_subnet.subnet2: Destroying... [id=subnet-06be1fbead7f1e96f]
aws_subnet.subnet1: Destroying... [id=subnet-0587e5bc96e2afee7]
aws_subnet.subnet1: Destruction complete after 1s
aws_subnet.subnet2: Destruction complete after 1s
aws_vpc.main: Destroying... [id=vpc-00e9e458b5577b124]
aws_vpc.main: Destruction complete after 1s

Apply complete! Resources: 0 added, 0 changed, 3 destroyed.
```

Variables, we can declare directly or we can pass it as file or we can pass it as a command line argument.

Variable passing through file command is **‘terraform apply --auto-approve=true -var-file="values.tfvars" ‘**
Note:- In variables.tf we have to comment the default values.

Variables passing through command line is ‘terraform apply -var= ‘cidr_block=[“IP”,”IP”] -var= ‘subnet_name=[“sn1”,”sn2”]’ -var= “aws_region=us-west-2”