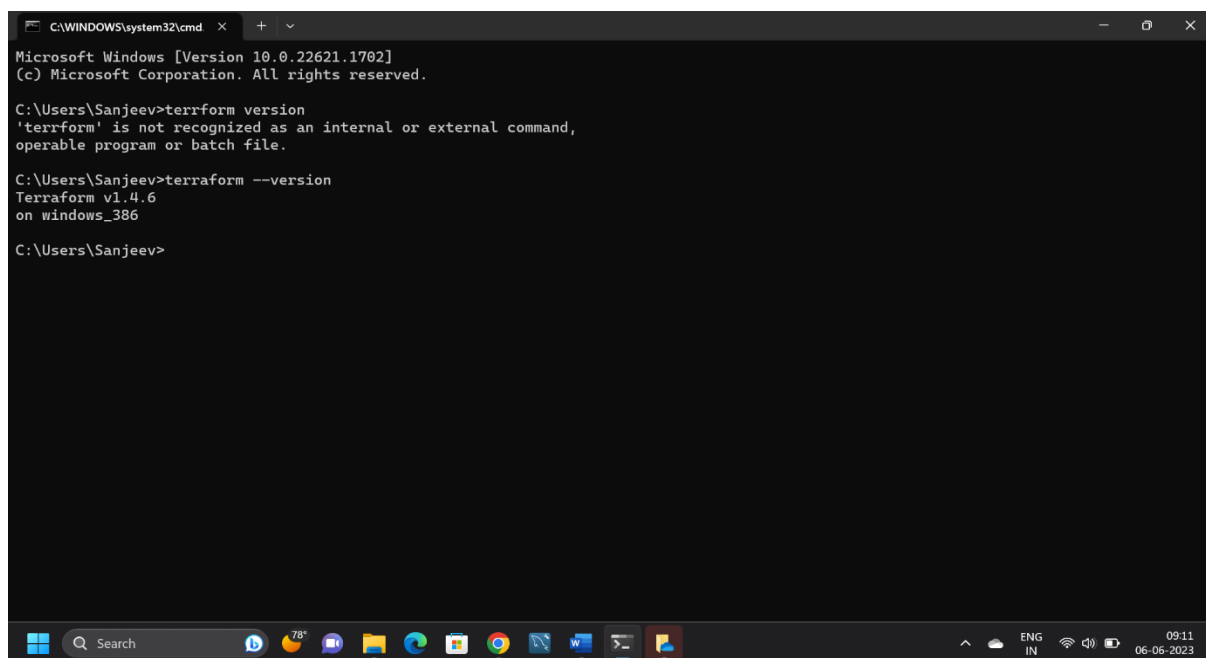
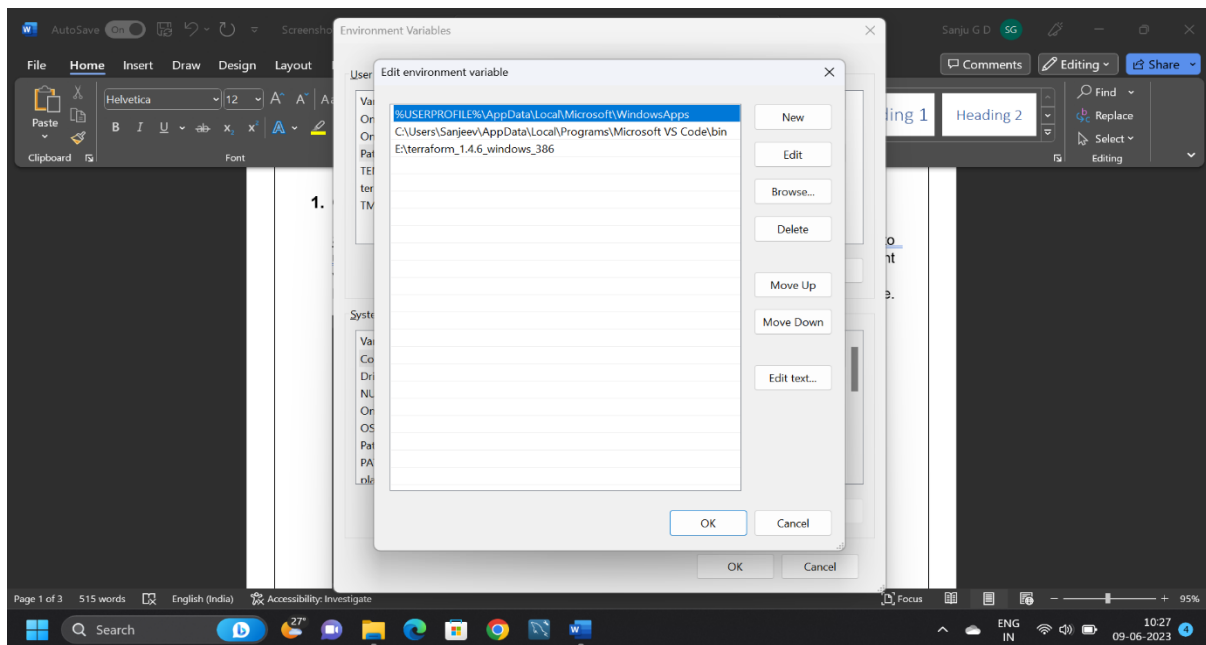


## 1. Creating an EC2 instance using Terraform.

Step-1: First we have to download the terraform file in our local machine. We have to unzip the file and move to the drive. Copy the path of the file and open Edit environment variables > environment variables > path > and paste the path and tap ok.

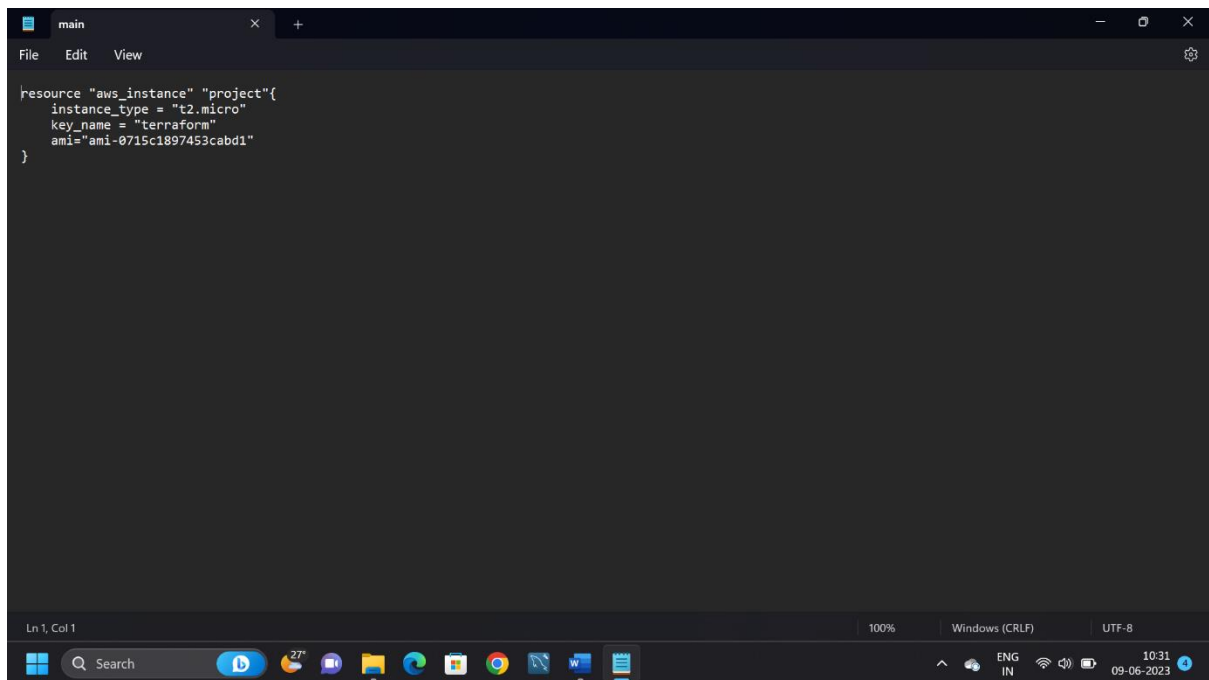
Now the terraform is installed in our local machine. Now we want to create a EC2 instance.



Step-2: Create a folder named terraform-project and cd into the folder. Now create a folder named as script and now we have to create two terraform files. These files have extension “.tf”. we can name those two files as “main.tf” and “provider.tf”. Now we have to add providers and resource commands to those two files.

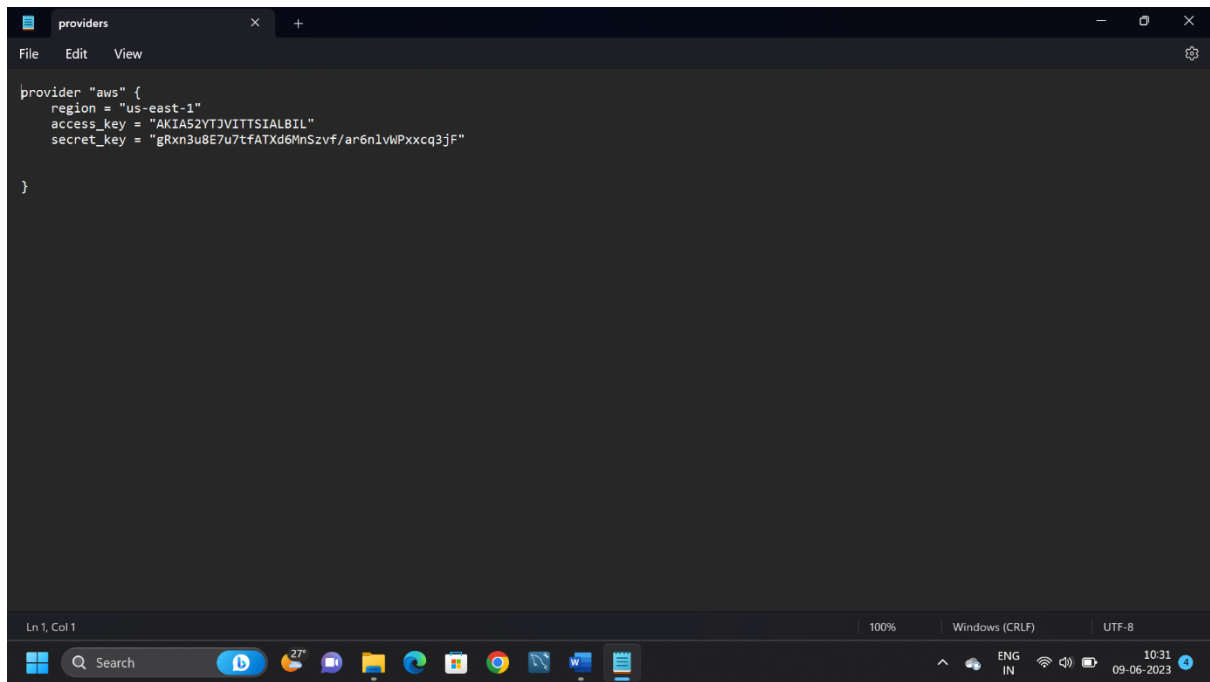
Step-3: Now open the main.tf by giving the command ‘notepad main.tf’ and open the notepad and define the resource.

```
resource "aws_instance" "web" {  
    instance_type = "t2.micro"  
    key_name = "terraform"  
    ami="ami-053b0d53c279acc90"  
}
```



Now open the provider.tf by giving the command ‘notepad provider.tf’ and open the notepad and define the resource.

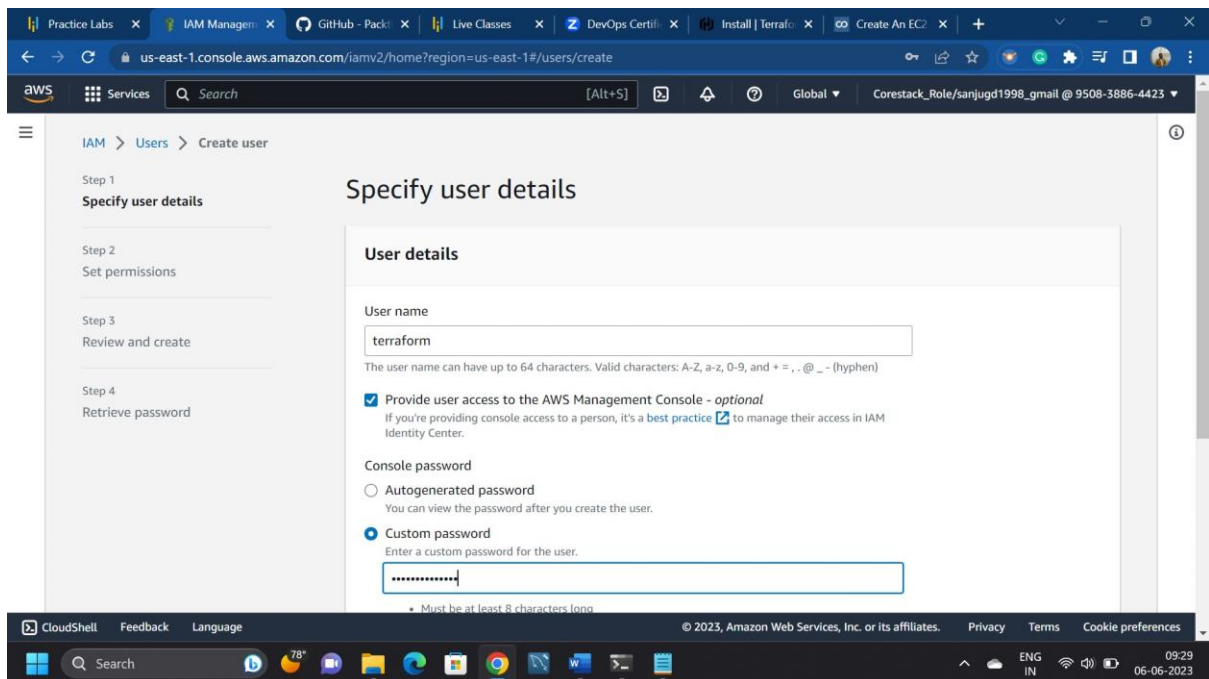
```
provider "aws" {  
    region = "us-east-1"  
    access_key = "AKIA52YTJVITXICODEU5"  
    secret_key = "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"  
}
```

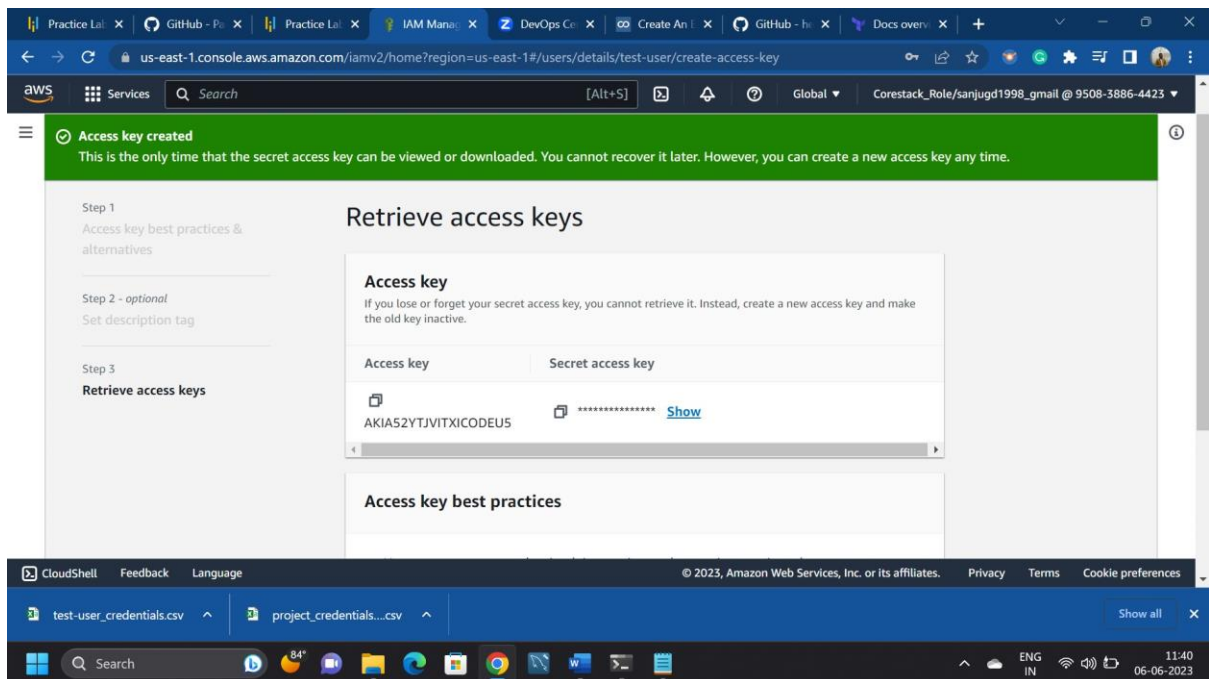
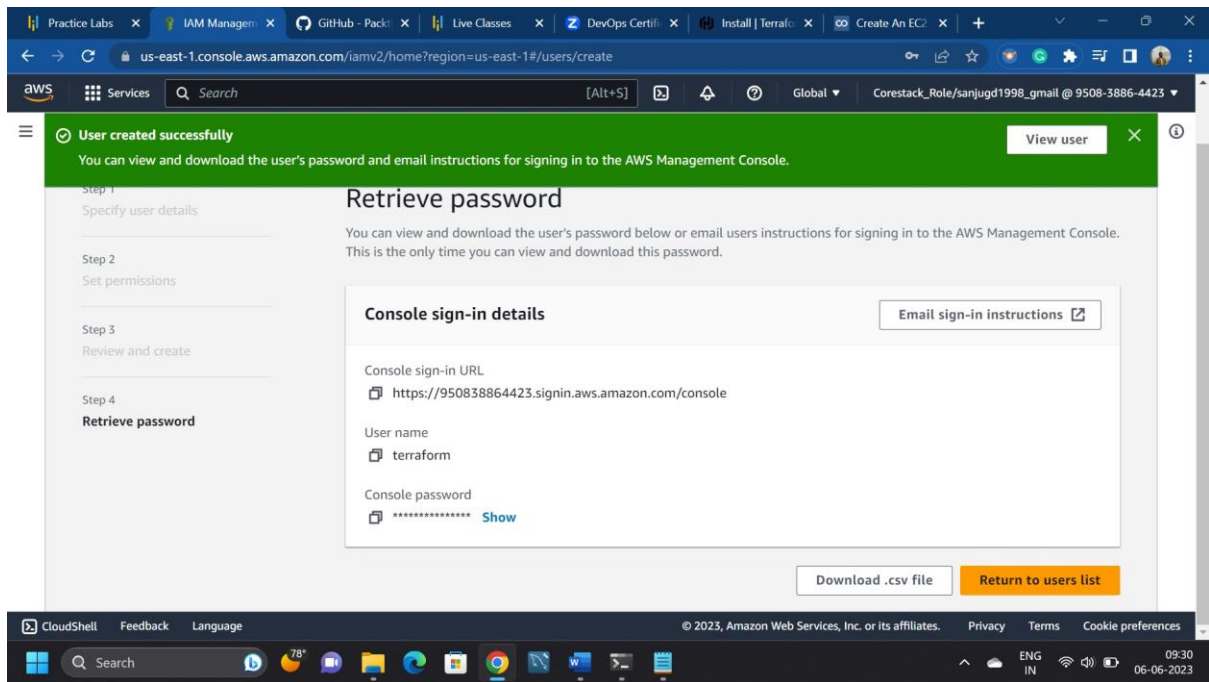


```
provider "aws" {  
  region = "us-east-1"  
  access_key = "AKIA52YTJVITTSIALBIL"  
  secret_key = "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"  
}
```

The screenshot shows a code editor window titled 'providers' with a menu bar (File, Edit, View) and a settings icon. The code is a Terraform provider configuration for AWS. The status bar at the bottom indicates 'Ln 1, Col 1', '100%', 'Windows (CRLF)', and 'UTF-8'. The Windows taskbar is visible at the very bottom.

We can generate the access\_key and secret\_key by creating a user at IAM in aws.





Now after configuring the user we have to run the commands

## Terraform init

```
C:\WINDOWS\system32\cmd. x + v

The following dependency selections recorded in the lock file are inconsistent with the current configuration:
- provider registry.terraform.io/hashicorp/aws: required by this configuration but no version is selected

To make the initial dependency selections that will initialize the dependency lock file, run:
terraform init

C:\Users\Sanjeev\terraform-project\script>terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.1.0...
- Installed hashicorp/aws v5.1.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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\Sanjeev\terraform-project\script>
```

## Terraform plan

```
C:\WINDOWS\system32\cmd. x + v

C:\Users\Sanjeev\terraform-project\script>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.project will be created
+ resource "aws_instance" "project" {
  + ami                      = "ami-053b0d53c279acc90"
  + 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_state           = (known after apply)
  + instance_type            = "t2-micro"
  + ipv6_address_count       = (known after apply)
  + ipv6_addresses           = (known after apply)
  + key_name                 = "terraform1"
  + monitoring               = (known after apply)
}
```

*Terraform apply.*

```
C:\WINDOWS\system32\cmd. x + v

+ outpost_arn                = (known after apply)
+ password_data              = (known after apply)
+ placement_group            = (known after apply)
+ placement_partition_number = (known after apply)
+ primary_network_interface_id = (known after apply)
+ private_dns                 = (known after apply)
+ private_ip                 = (known after apply)
+ public_dns                 = (known after apply)
+ public_ip                  = (known after apply)
+ secondary_private_ips      = (known after apply)
+ security_groups             = (known after apply)
+ source_dest_check          = true
+ subnet_id                  = (known after apply)
+ tags_all                   = (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.

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.web: Creating...
aws_instance.web: Still creating... [10s elapsed]
```

Now our instance is created in our AWS account.

```
C:\WINDOWS\system32\cmd. x + v

+ outpost_arn                = (known after apply)
+ password_data              = (known after apply)
+ placement_group            = (known after apply)
+ placement_partition_number = (known after apply)
+ primary_network_interface_id = (known after apply)
+ private_dns                 = (known after apply)
+ private_ip                 = (known after apply)
+ public_dns                 = (known after apply)
+ public_ip                  = (known after apply)
+ secondary_private_ips      = (known after apply)
+ security_groups             = (known after apply)
+ source_dest_check          = true
+ subnet_id                  = (known after apply)
+ tags_all                   = (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)
}

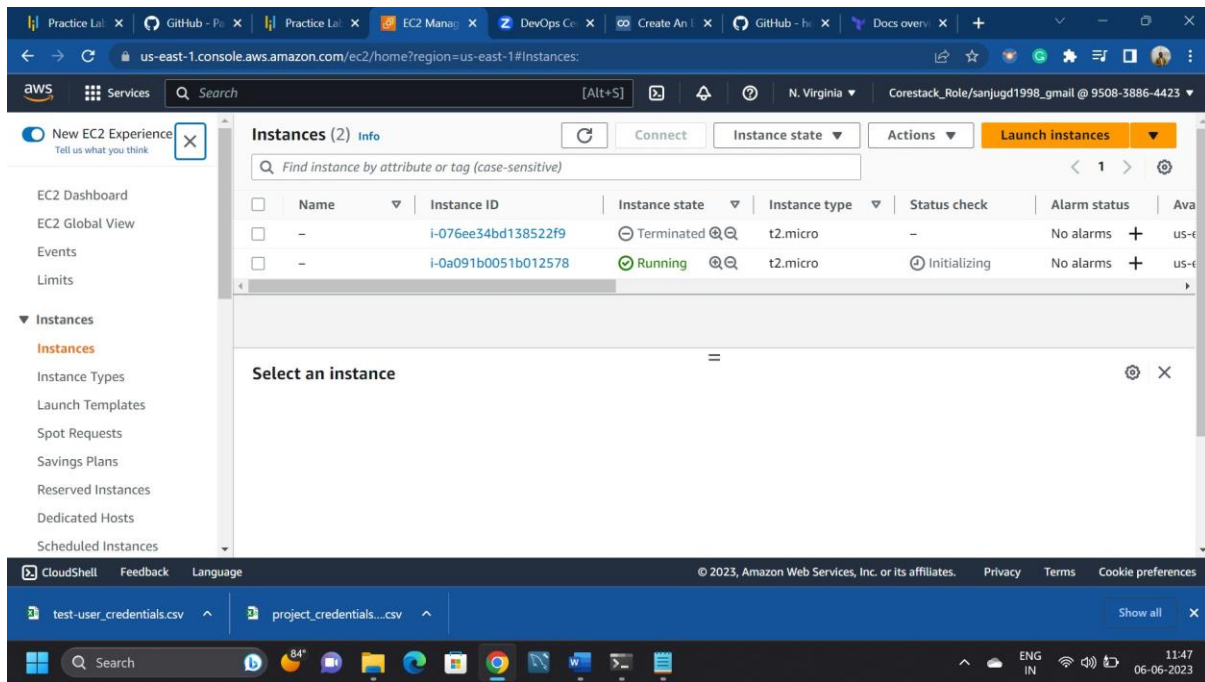
Plan: 1 to add, 0 to change, 0 to destroy.

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.web: Creating...
aws_instance.web: Still creating... [10s elapsed]
```

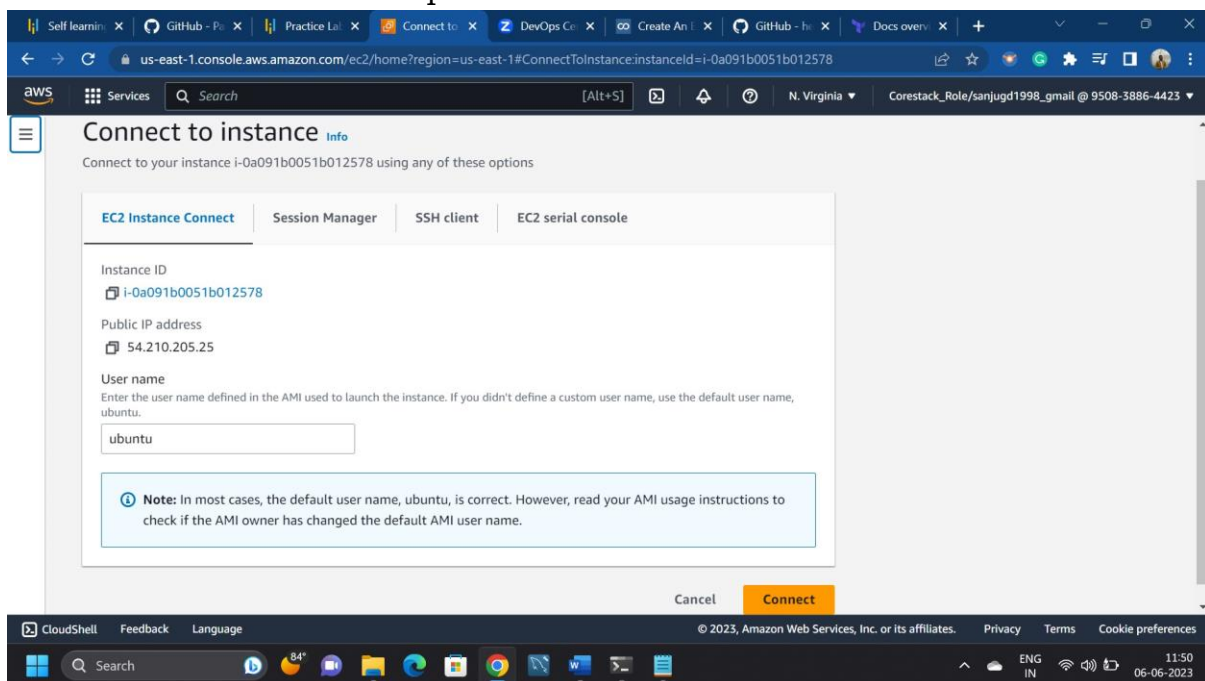


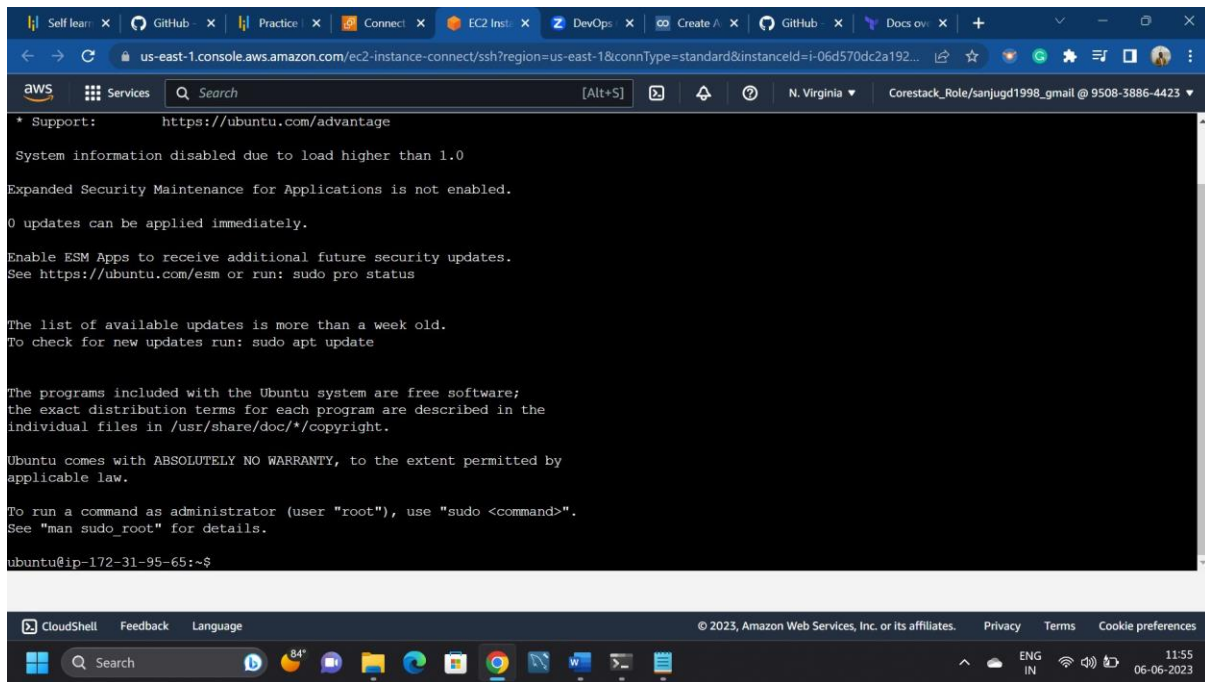


SS

## 2. Connecting to the EC2 instance.

Now we can see that our EC2 instance is running successfully, we will find the option in AWS to connect to our instance from there we can connect to our instance. A terminal will open and now we are connected to our instance.





The screenshot shows a terminal window within the AWS CloudShell interface. The terminal displays the following text:

```
* Support: https://ubuntu.com/advantage

System information disabled due to load higher than 1.0

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

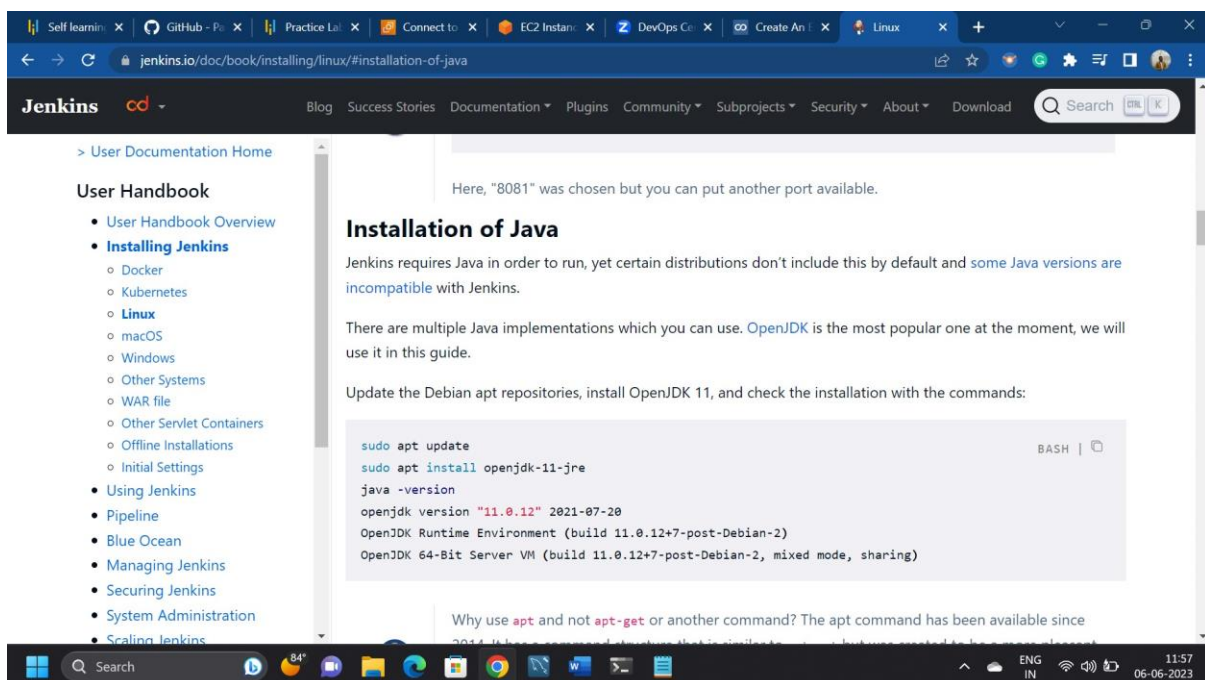
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-95-65:~$
```

### 3. Installing Java in our EC2 instance.

We need to run few commands in our terminal to install Java in our local machine. To install Java the commands that we should run are,

```
# sudo apt update
# sudo apt install openjdk-11-jre
```



The screenshot shows the Jenkins documentation page for Linux installation. The page title is "Installation of Java". The content includes the following text:

Here, "8081" was chosen but you can put another port available.

### Installation of Java

Jenkins requires Java in order to run, yet certain distributions don't include this by default and some Java versions are incompatible with Jenkins.

There are multiple Java implementations which you can use. OpenJDK is the most popular one at the moment, we will use it in this guide.

Update the Debian apt repositories, install OpenJDK 11, and check the installation with the commands:

```
sudo apt update
sudo apt install openjdk-11-jre
java -version
```

openjdk version "11.0.12" 2021-07-20  
OpenJDK Runtime Environment (build 11.0.12+7-post-Debian-2)  
OpenJDK 64-Bit Server VM (build 11.0.12+7-post-Debian-2, mixed mode, sharing)

To check if Java is installed or not we need to run this command.

```
# java -version
```

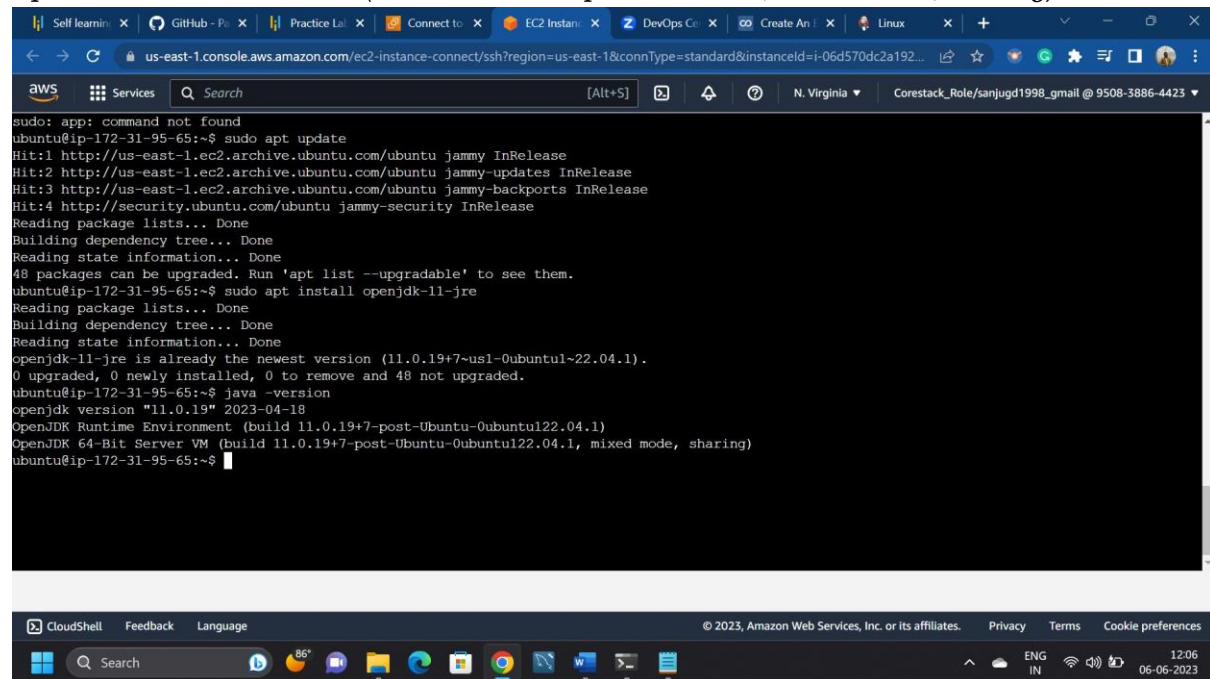


If Java is installed we will find the result as this,

*Openjdk version "11.0.12" 2021-07-20*

*OpenJDK Runtime Environment (build 11.0.12+7-post-Debian-2)*

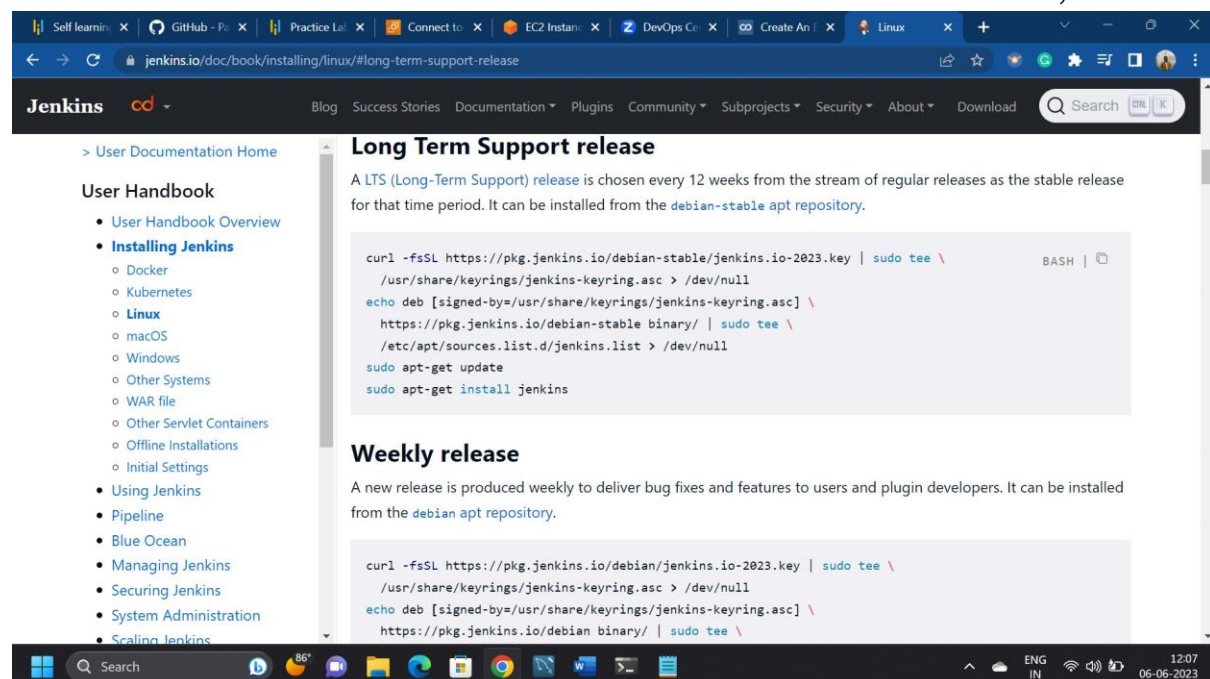
*OpenJDK 64-Bit Server VM (build 11.0.12+7-post-Debian-2, mixed mode, sharing)*



```
ubuntu@ip-172-31-95-65:~$ sudo apt update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu jammy-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
48 packages can be upgraded. Run 'apt list --upgradable' to see them.
ubuntu@ip-172-31-95-65:~$ sudo apt install openjdk-11-jre
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
openjdk-11-jre is already the newest version (11.0.19+7-us1-0ubuntu1~22.04.1).
0 upgraded, 0 newly installed, 0 to remove and 48 not upgraded.
ubuntu@ip-172-31-95-65:~$ java -version
openjdk version "11.0.19" 2023-04-18
OpenJDK Runtime Environment (build 11.0.19+7-post-Ubuntu-0ubuntu122.04.1)
OpenJDK 64-Bit Server VM (build 11.0.19+7-post-Ubuntu-0ubuntu122.04.1, mixed mode, sharing)
ubuntu@ip-172-31-95-65:~$
```

## 4. Installing Jenkins in our EC2 instance.

We need to run few commands in our terminal to install Jenkins in our local machine. To install Jenkins the commands that we should run are,



```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
```

**Weekly release**

```
curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo tee \
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian binary/ | sudo tee \
```

```
# curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo
tee \
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
```

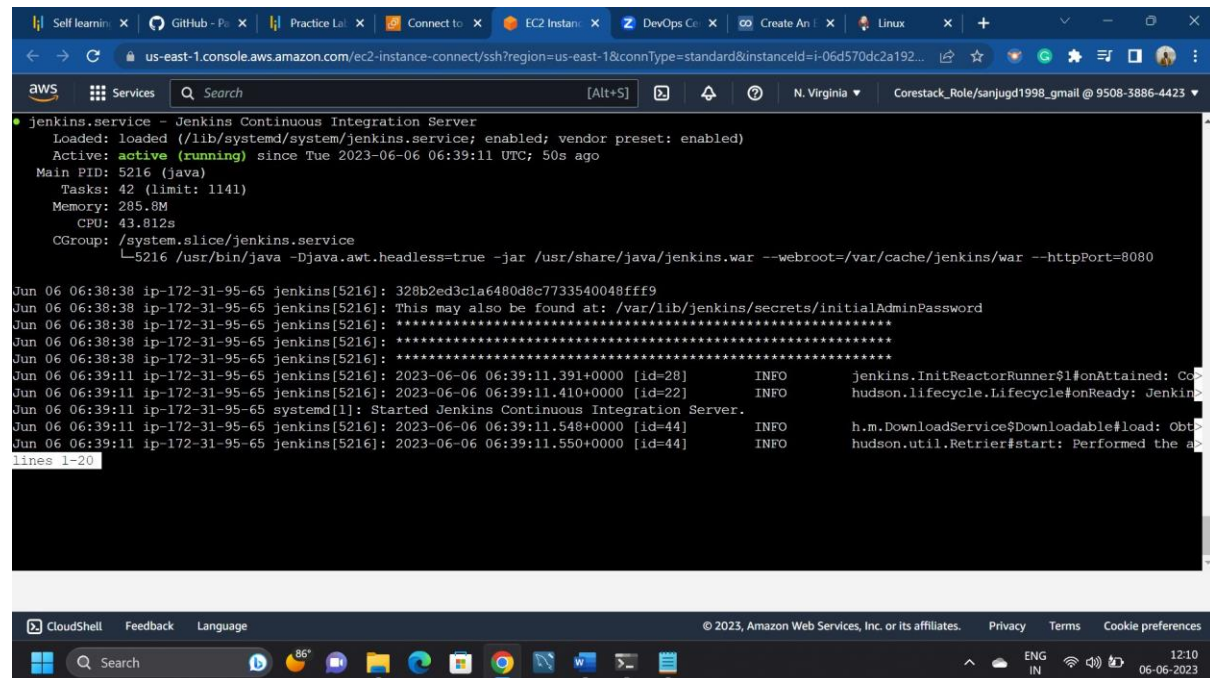
```
# echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null
```

```
# sudo apt-get update
```

```
# sudo apt-get install Jenkins
```

To check if Java is installed or not we need to run this command.

```
# service Jenkins status
```



```
jenkins.service - Jenkins Continuous Integration Server
Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
Active: active (running) since Tue 2023-06-06 06:39:11 UTC; 50s ago
Main PID: 5216 (java)
Tasks: 42 (limit: 1141)
Memory: 285.0M
CPU: 43.812s
CGroup: /system.slice/jenkins.service
└─5216 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Jun 06 06:38:38 ip-172-31-95-65 jenkins[5216]: 328b2ed3c1a6480d8c7733540048fff9
Jun 06 06:38:38 ip-172-31-95-65 jenkins[5216]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Jun 06 06:38:38 ip-172-31-95-65 jenkins[5216]: *****
Jun 06 06:38:38 ip-172-31-95-65 jenkins[5216]: *****
Jun 06 06:39:11 ip-172-31-95-65 jenkins[5216]: 2023-06-06 06:39:11.391+0000 [id=28] INFO jenkins.InitReactorRunner$1#onAttained: Co
Jun 06 06:39:11 ip-172-31-95-65 jenkins[5216]: 2023-06-06 06:39:11.410+0000 [id=22] INFO hudson.lifecycle.Lifecycle#onReady: Jenkin
Jun 06 06:39:11 ip-172-31-95-65 systemd[1]: Started Jenkins Continuous Integration Server.
Jun 06 06:39:11 ip-172-31-95-65 jenkins[5216]: 2023-06-06 06:39:11.548+0000 [id=44] INFO h.m.DownloadService$Downloadable#load: Obt
Jun 06 06:39:11 ip-172-31-95-65 jenkins[5216]: 2023-06-06 06:39:11.550+0000 [id=44] INFO hudson.util.Retrier#start: Performed the a
```

## 5. Installing Python in our EC2 instance.

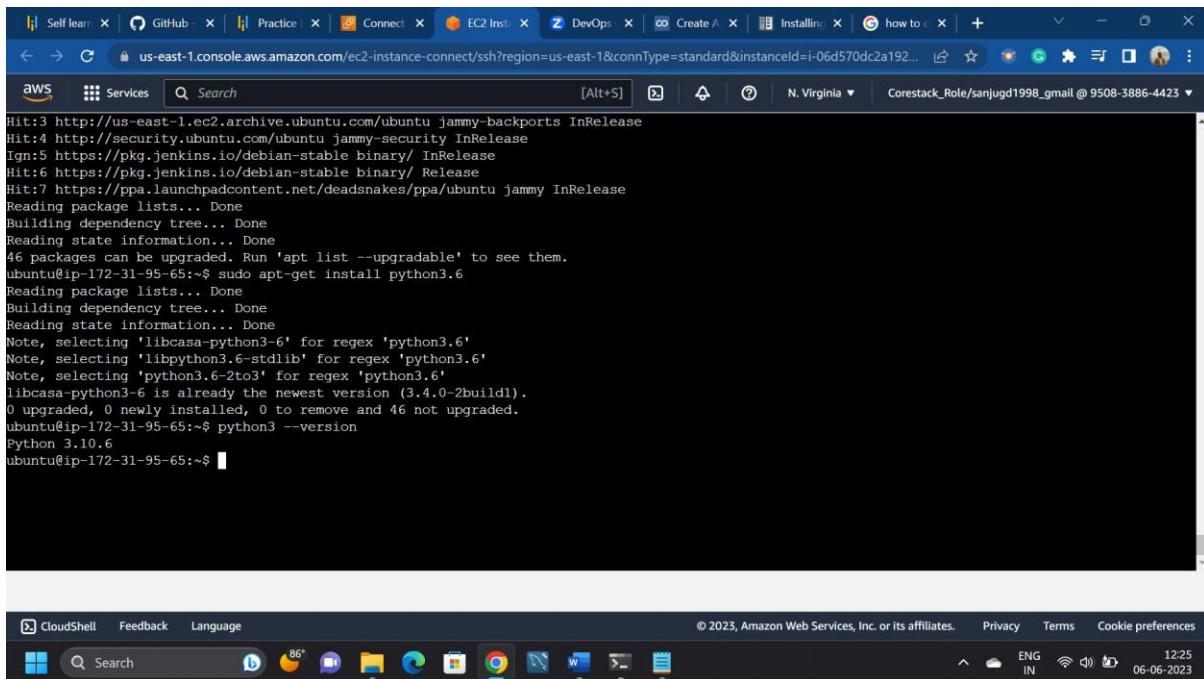
We need to run few commands in our terminal to install Python in our local machine. To install python the commands that we should run are,

```
# sudo apt-get install python3.6
```

To check the python is installed or not

```
# python3 --version.
```

These are the steps and procedure that I have taken to complete the requirements.



The screenshot shows an AWS CloudShell terminal window with the following content:

```
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu jammy-security InRelease
Ign:5 https://pkg.jenkins.io/debian-stable binary/ InRelease
Hit:6 https://pkg.jenkins.io/debian-stable binary/ Release
Hit:7 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
46 packages can be upgraded. Run 'apt list --upgradable' to see them.
ubuntu@ip-172-31-95-65:~$ sudo apt-get install python3.6
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Note, selecting 'libcasa-python3-6' for regex 'python3.6'
Note, selecting 'libpython3.6-stdlib' for regex 'python3.6'
Note, selecting 'python3.6-2to3' for regex 'python3.6'
libcasa-python3-6 is already the newest version (3.4.0-2build1).
0 upgraded, 0 newly installed, 0 to remove and 46 not upgraded.
ubuntu@ip-172-31-95-65:~$ python3 --version
Python 3.10.6
ubuntu@ip-172-31-95-65:~$
```

The terminal window is part of the AWS CloudShell interface, showing the AWS logo, Services menu, and search bar. The bottom of the window displays the CloudShell status bar with feedback, language, and copyright information.