EXPERIMENT NO. 01

Aim: To understand the benefits of Cloud Infrastructure and Setup AWS Cloud9 IDE, Launch AWS Cloud9 IDE and Perform Collaboration Demonstration.

Steps:

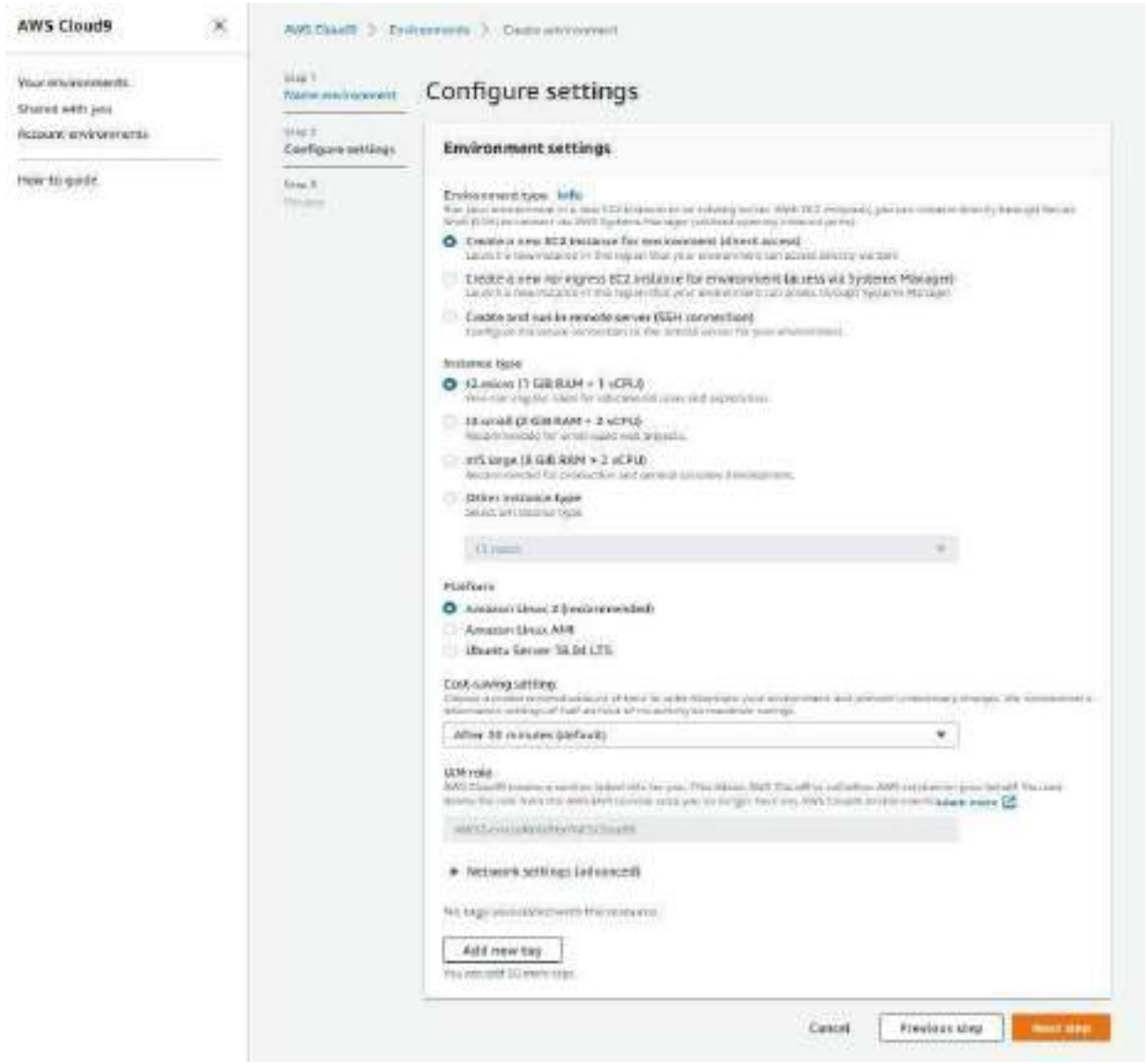
1. Login with your AWS account.

2. Navigate to Cloud 9 service from Developer tools section as below:

3. Click on Create Environment :

4. Provide name for the Enviornment (WebAppIDE) and click on next.

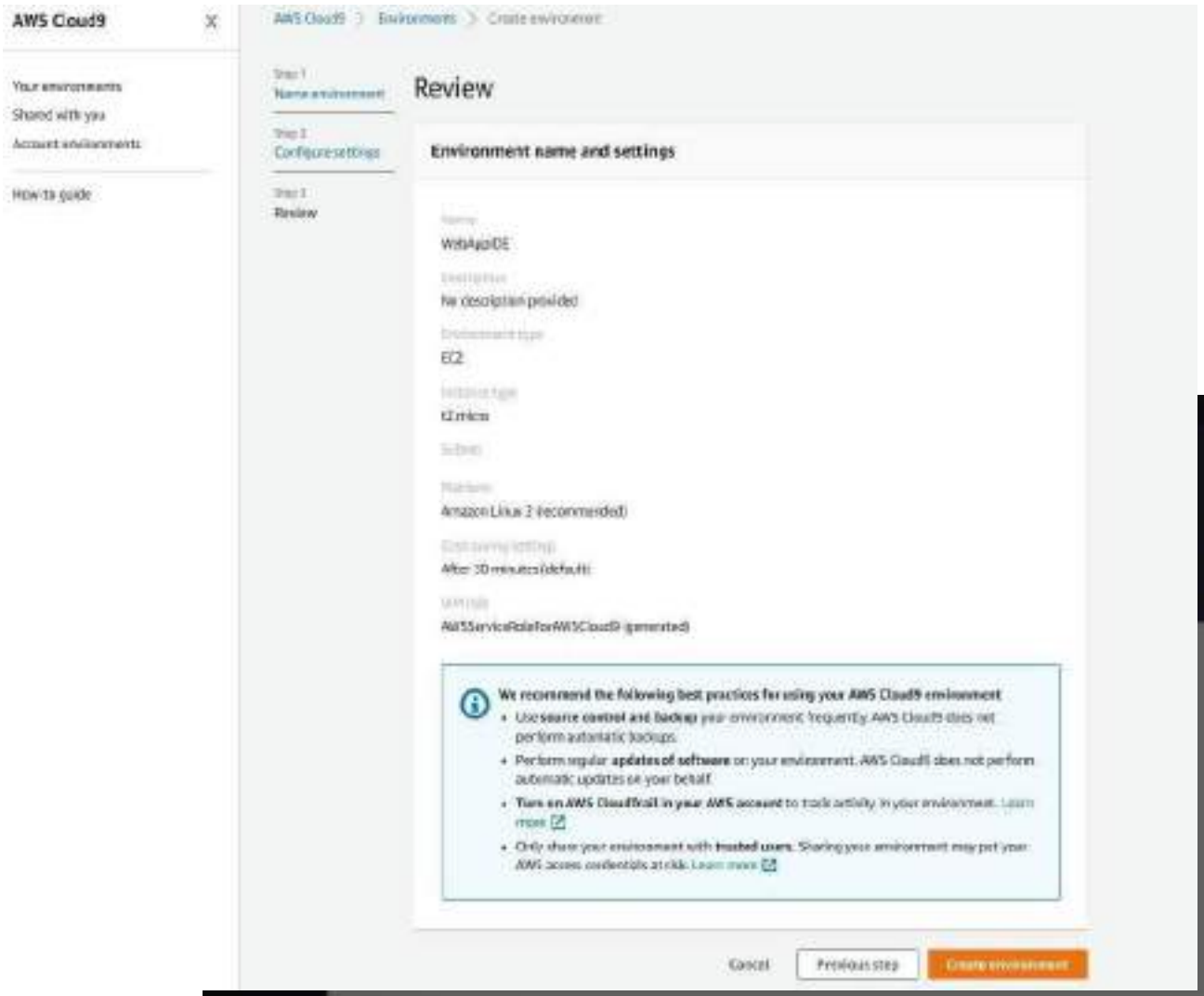
5. Keep all the Default seetings as shown in below:

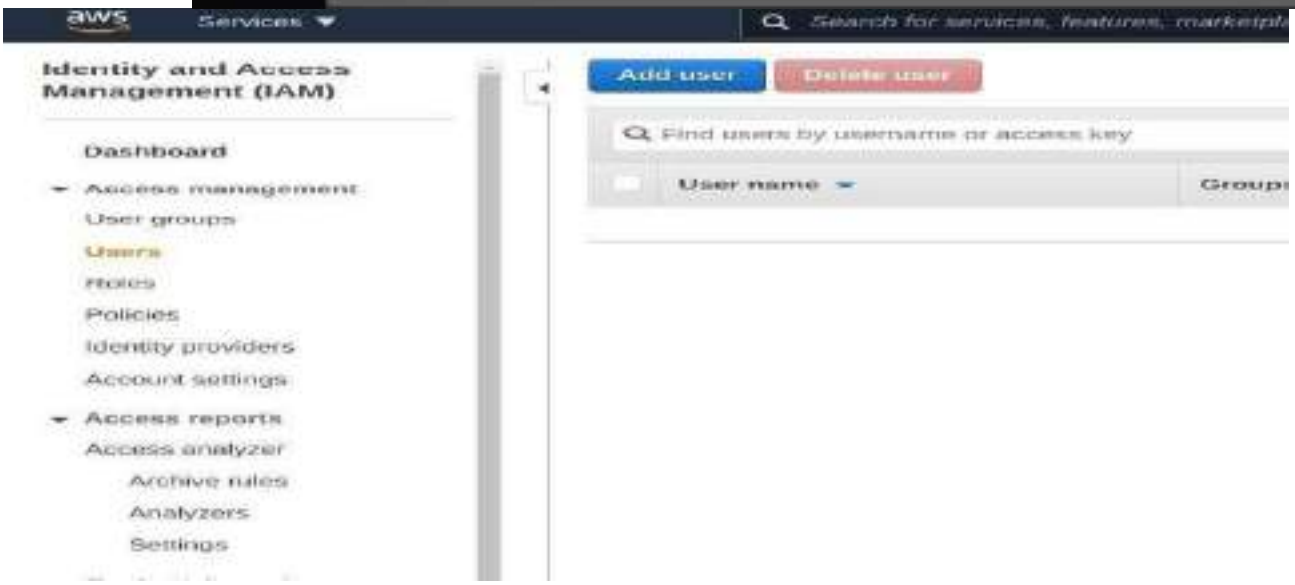


6. Review the Enviornment name and Settings and click on Create Enviornment:

It will take few minutes to create aws instance for your Cloud 9 Enviornment.

7. Till that time open IAM Identity and Access Management in order to Add user In other tab.

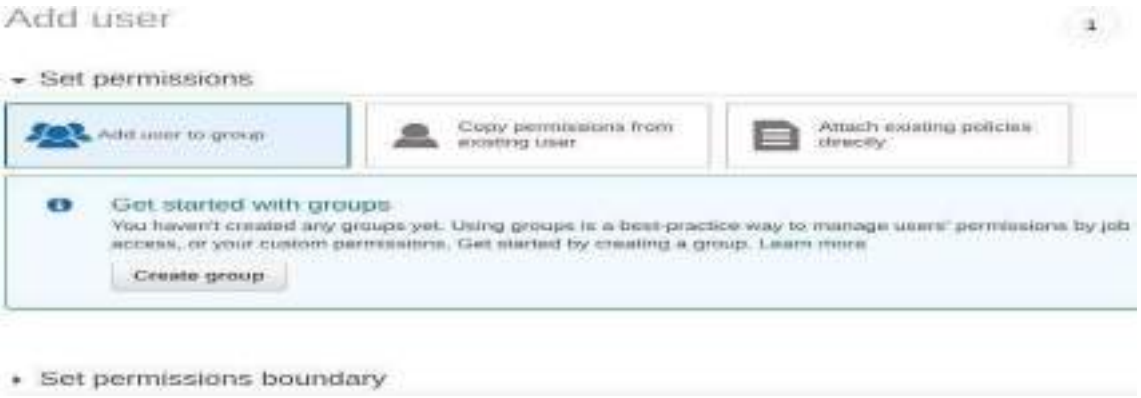




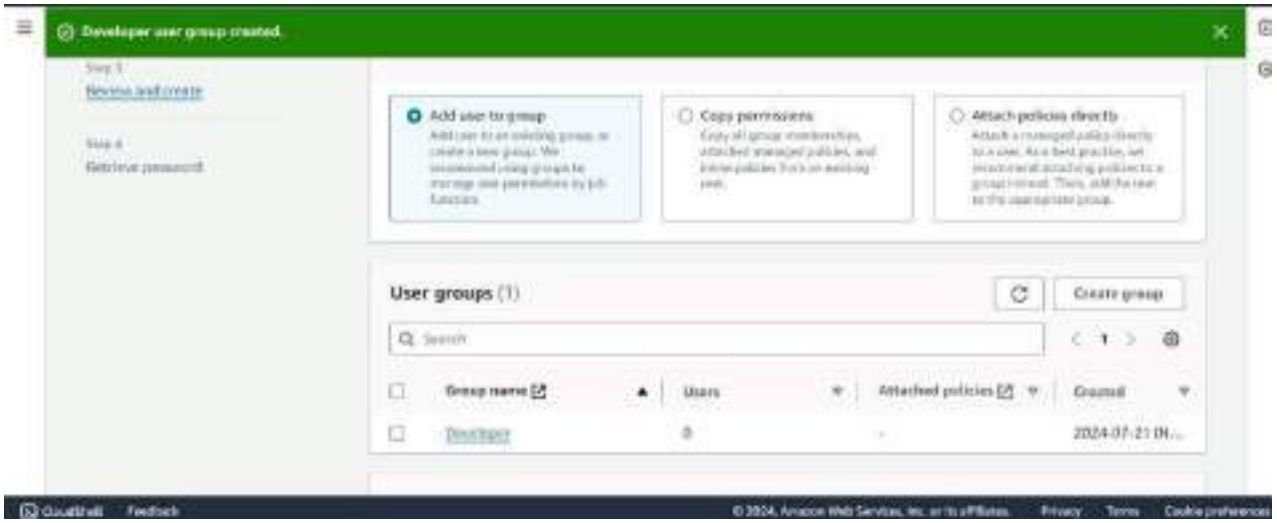
8. Add user provide manual password if you want and click on Next permission tab.

9.

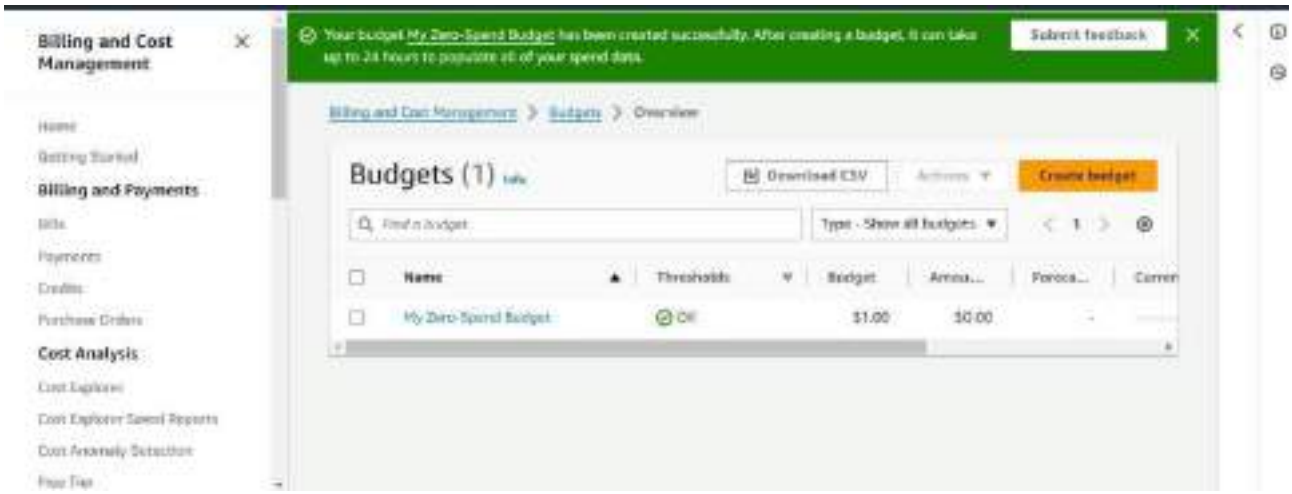
Click on Create group



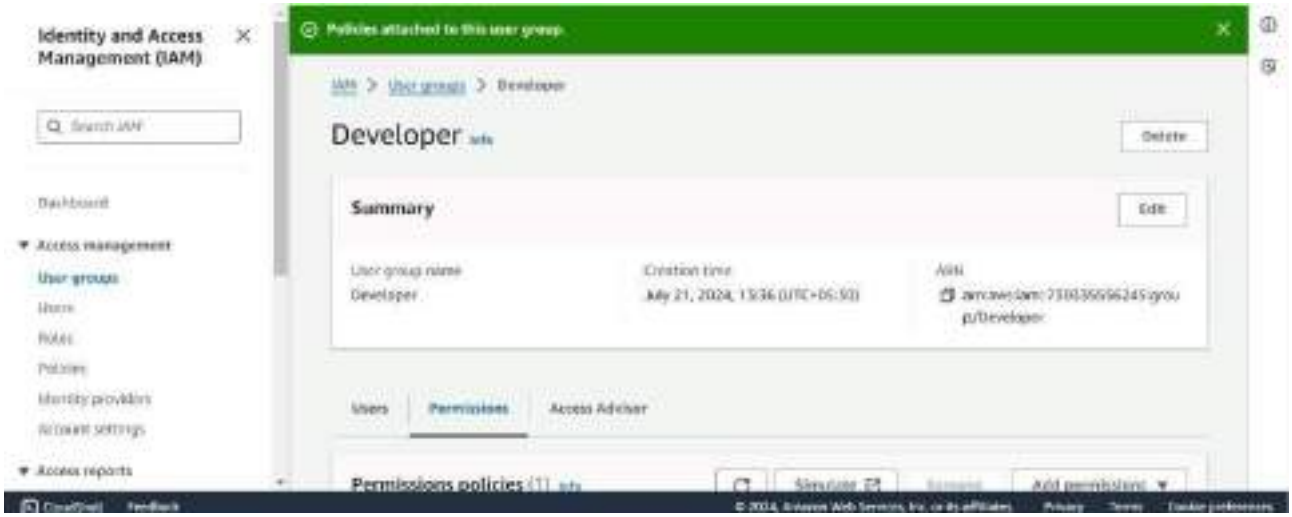
10. Provide group name and click on create group.



11. After that group is created click on next if u want to provide tag else click on Review for user settings and click on create user as shown in fig.



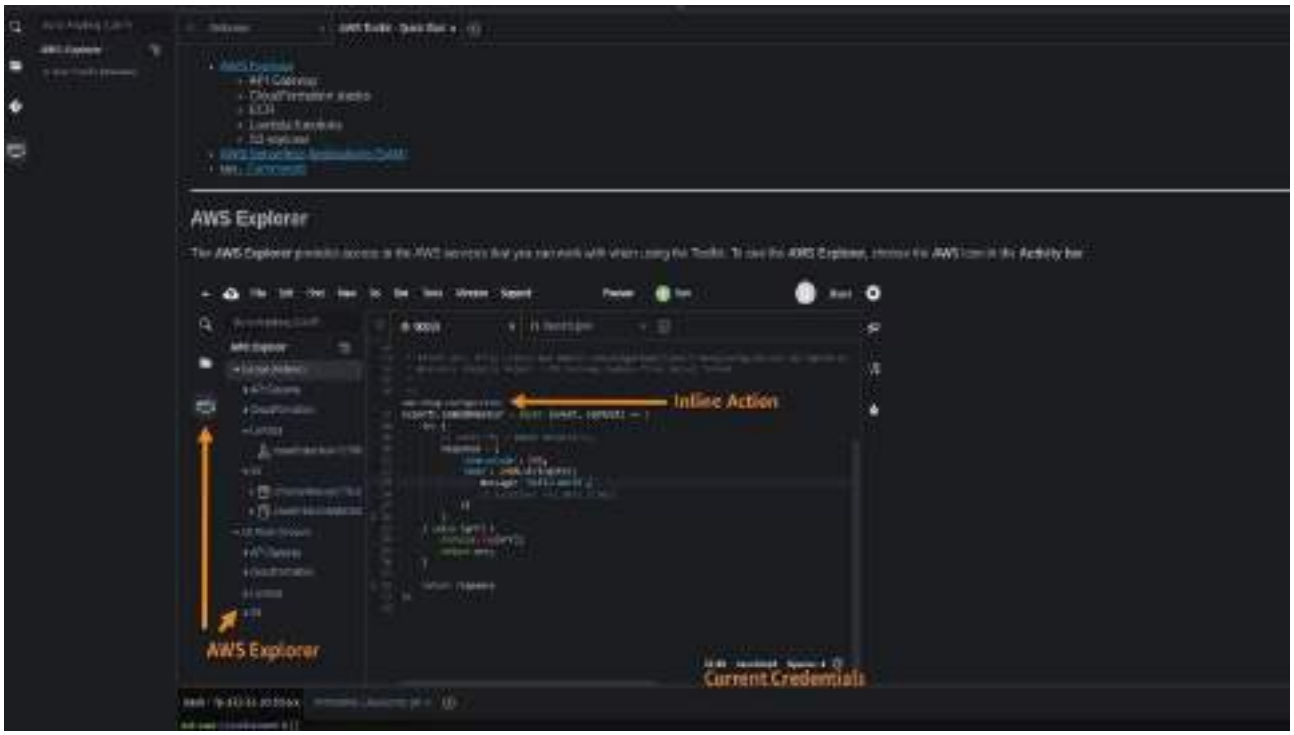
12. Now close that window and Navigate to user Groups from left pane in IAM.



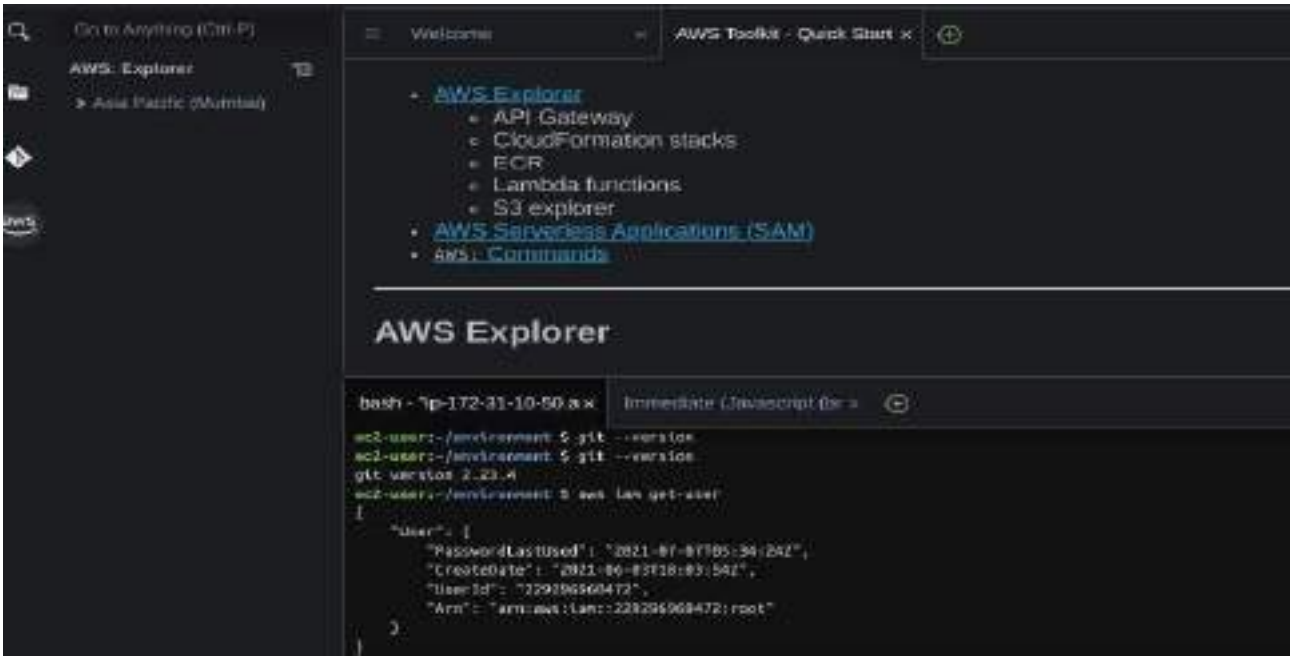
13. click on your group name which you have created and nevigate to permission tab as shown:

14. Now click on Add permission and select Attach Policy after that search for Cloud9 related policy and select Awscloud9EnviornmentMember policy and add it.

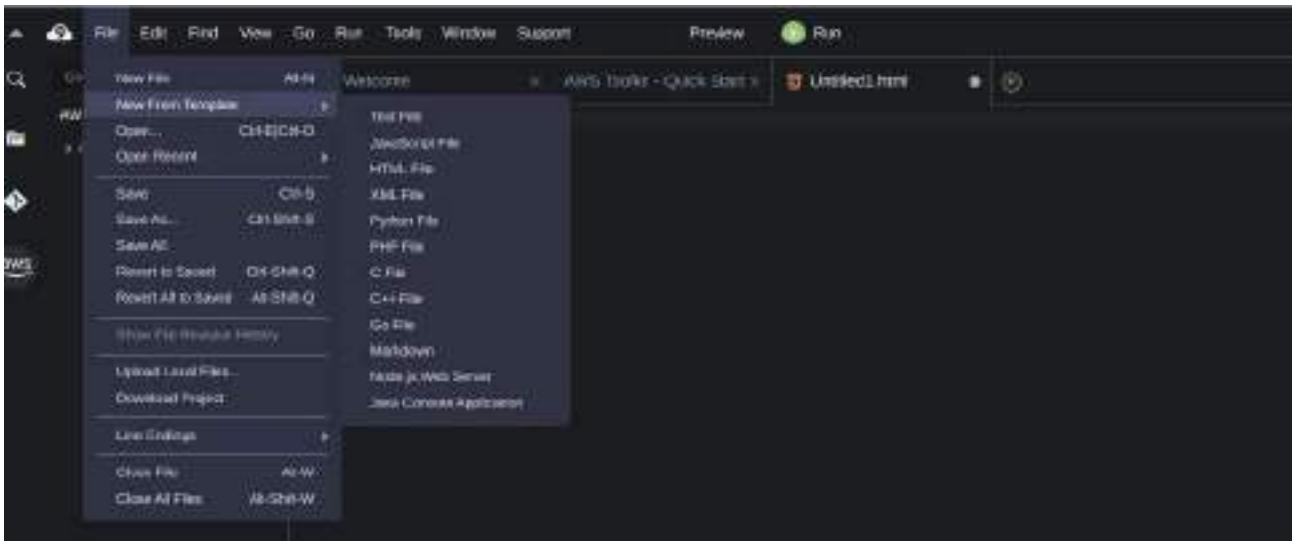
15. now we move towards our cloud9 IDE Enviornment tab it shows as shown :



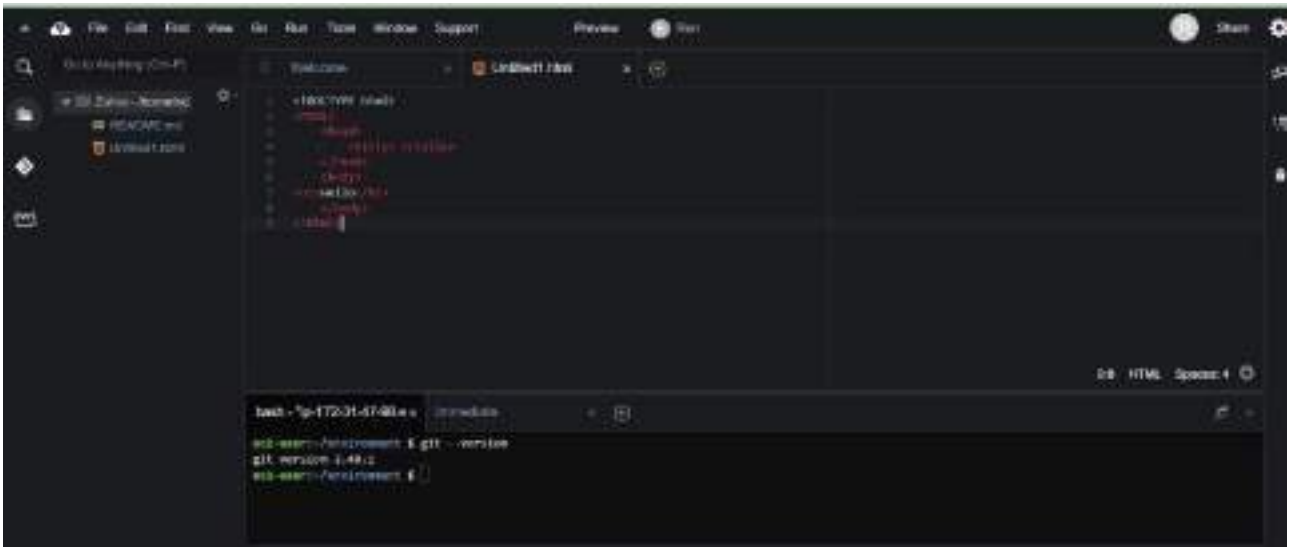
16. If you check at bottom side Cloud9 IDE also giving you and aws CLI for command operations: as we here checked git version, iam user details and so on...



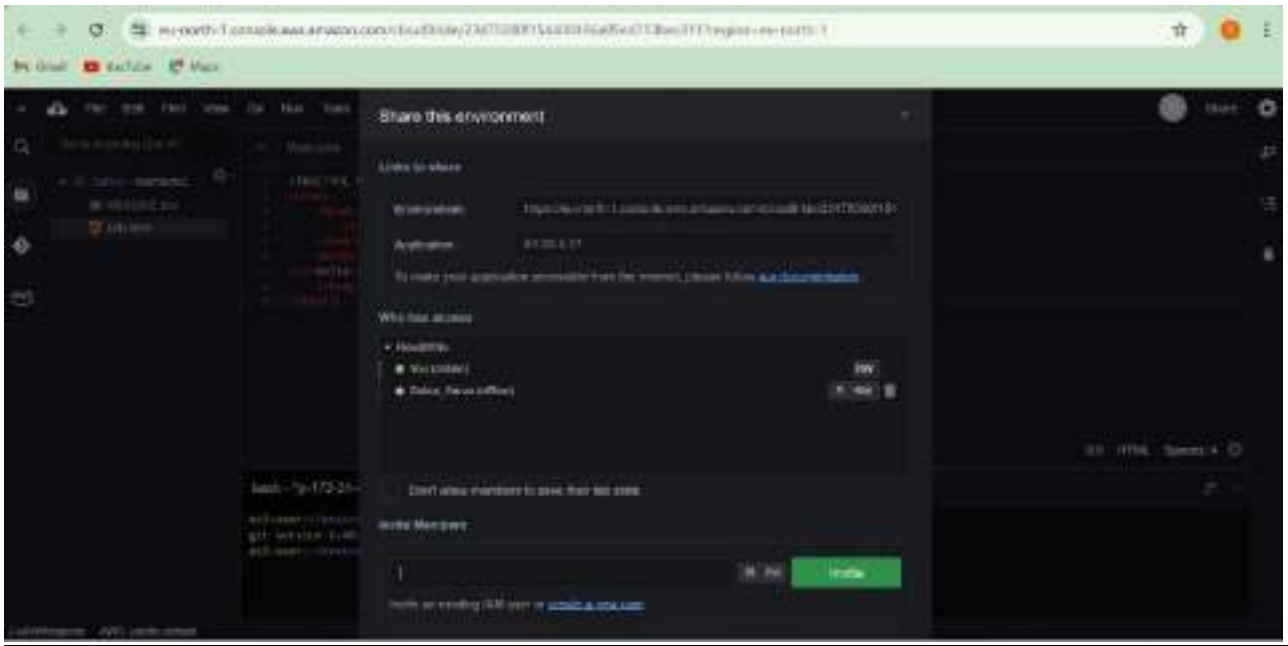
17. Now we will setup collaborative enviornment Click on File you can create new file or choose from template, here m opting html file to collaborate.

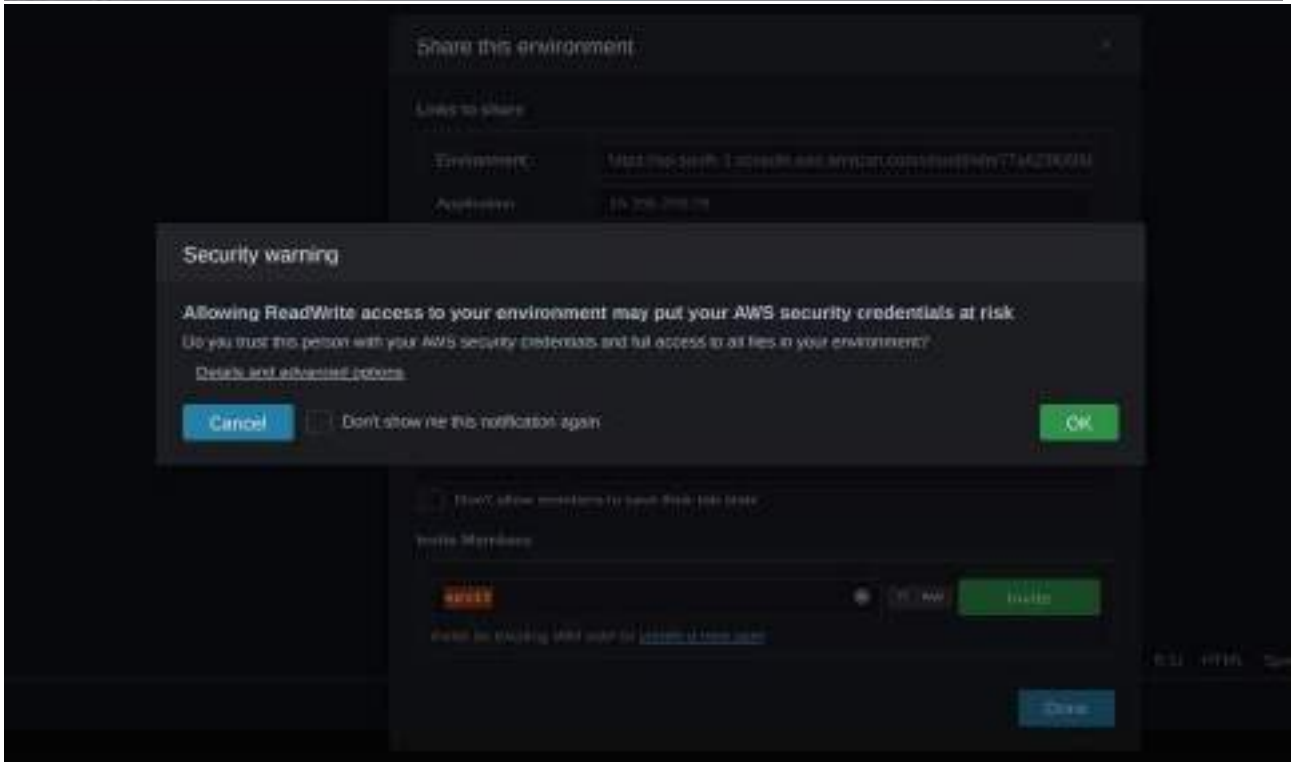


18. Edit html file and save it



19. now in order to share this file to collaborate with other members of your team click on Share option on Roght Pane and username which you created in IAM before into Invite members and enable persmission as RW (Read and Write) and click on Done. Click OK for Security warning.

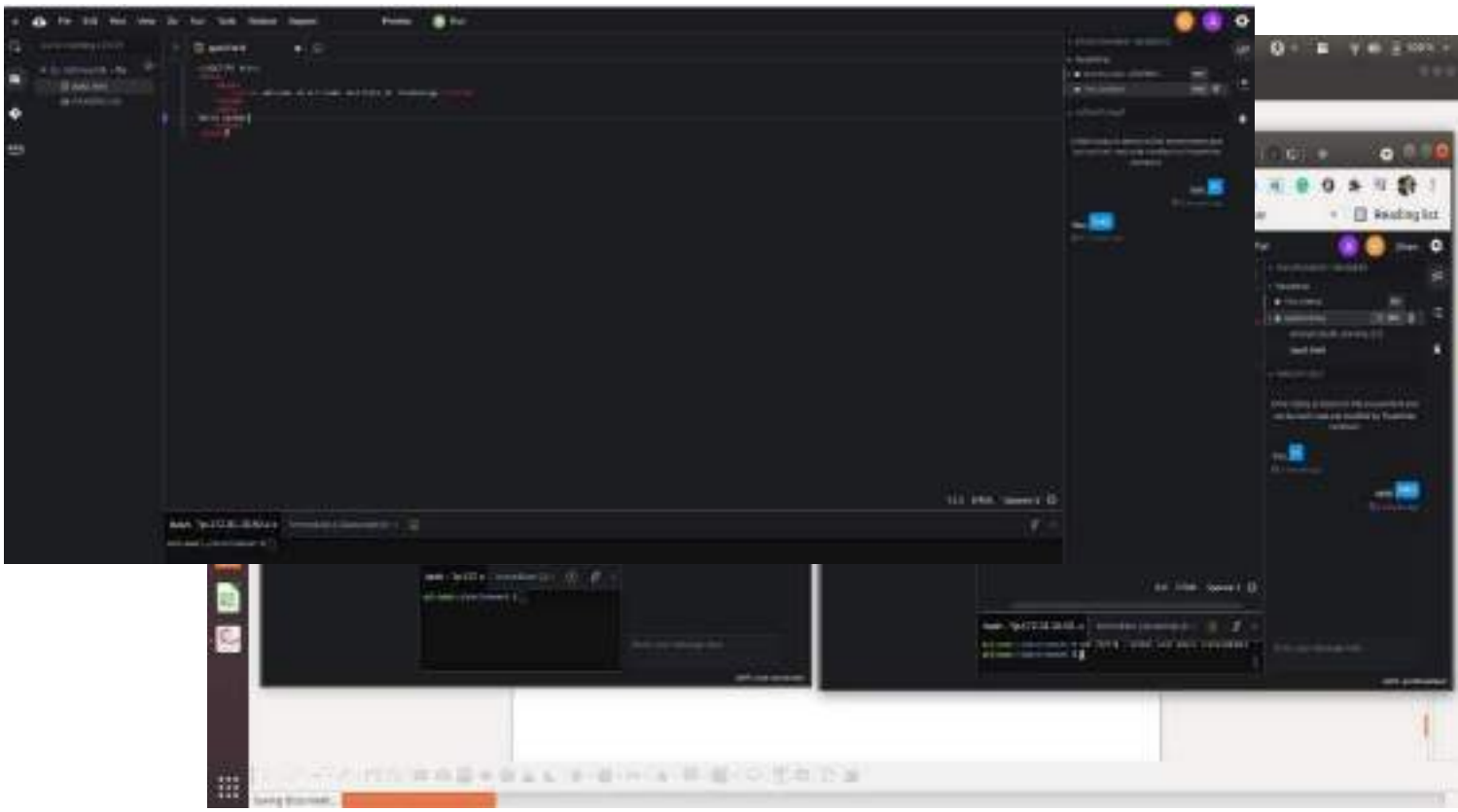




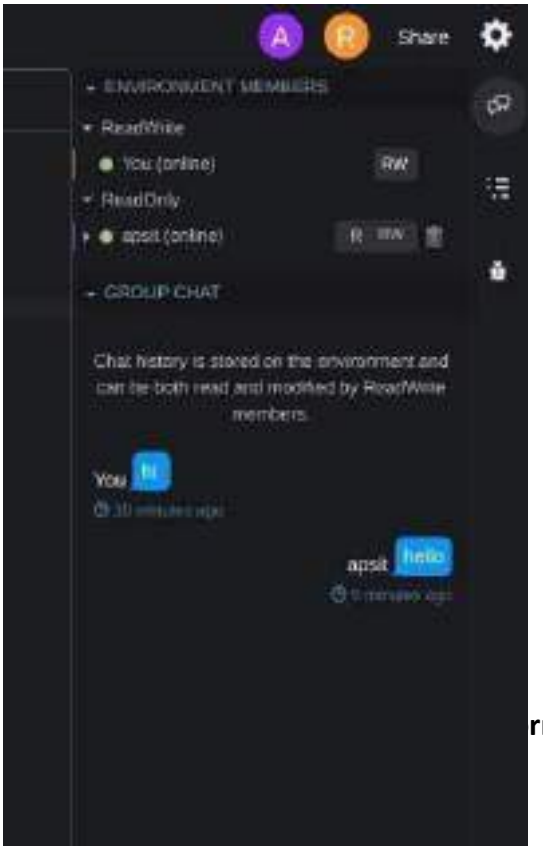
20. Now Open your Browsers Incognito Window and login with IAM user which you configured before.

21. After Successful login with IAM user open Cloud9 service from dashboard services and click on shared with you enviornment to collaborate.

22. Click on Open IDE you will same interface as your other member have to collaborate in real time, also you all within team can do group chats as shown below:



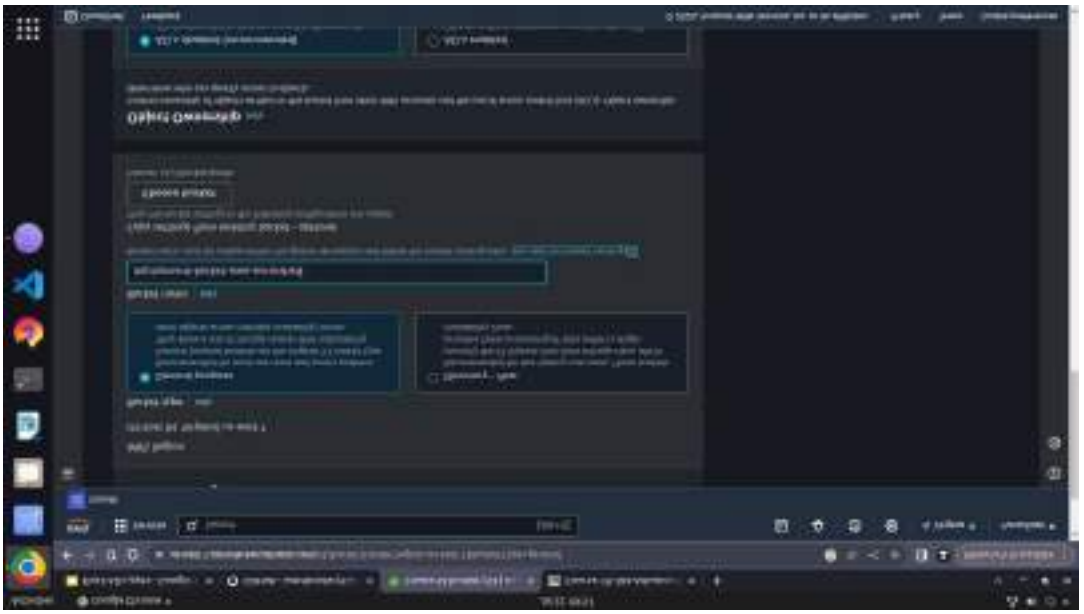
24. you can also exploresettings where you can update permissions of yourtemmates as from RW to R only or you can removeuser too.

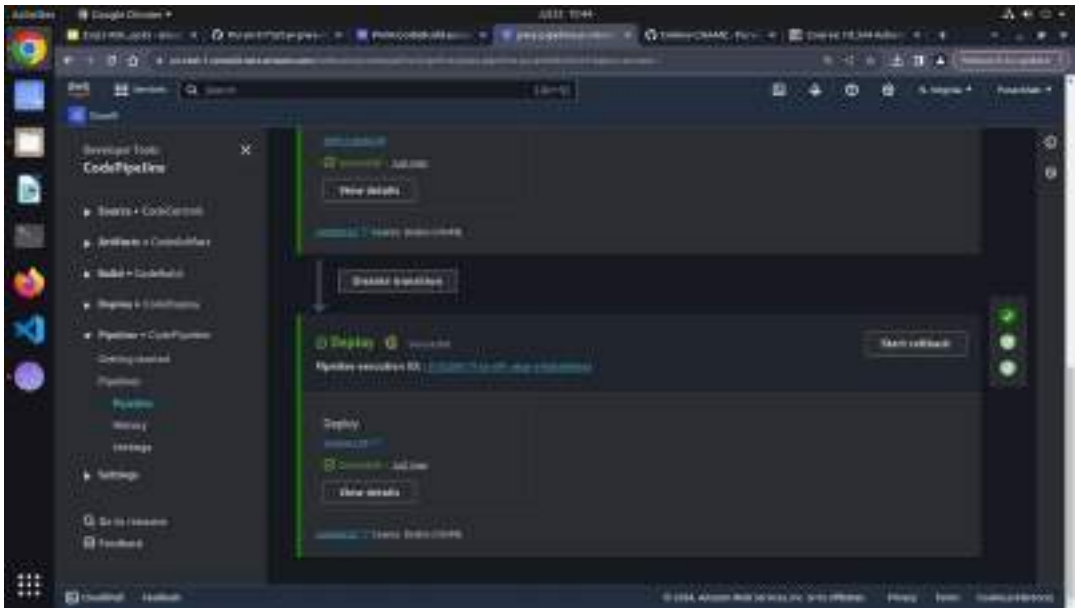


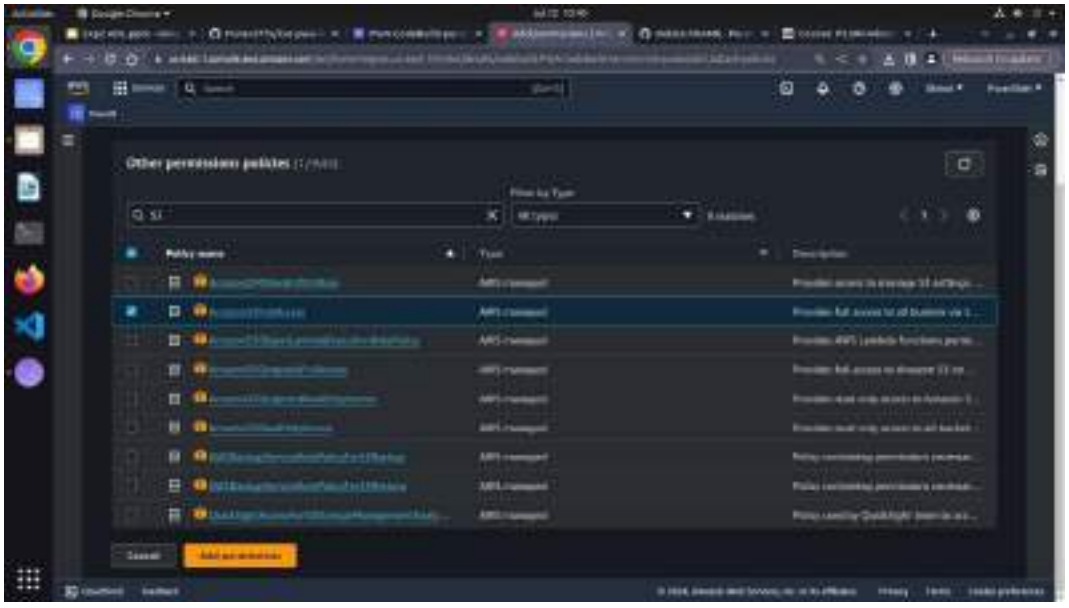
EXPERIMENT NO. 02

Aim: To Build Your Application using AWS CodeBuild and Deploy on S3 / SEBS using AWS CodePipeline, deploy Sample Application on EC2 instance using AWS CodeDeploy.

Step1: Create a deployment environment




CodePipeline
View details
Disable transition
Deploy
Stop rollback
View details

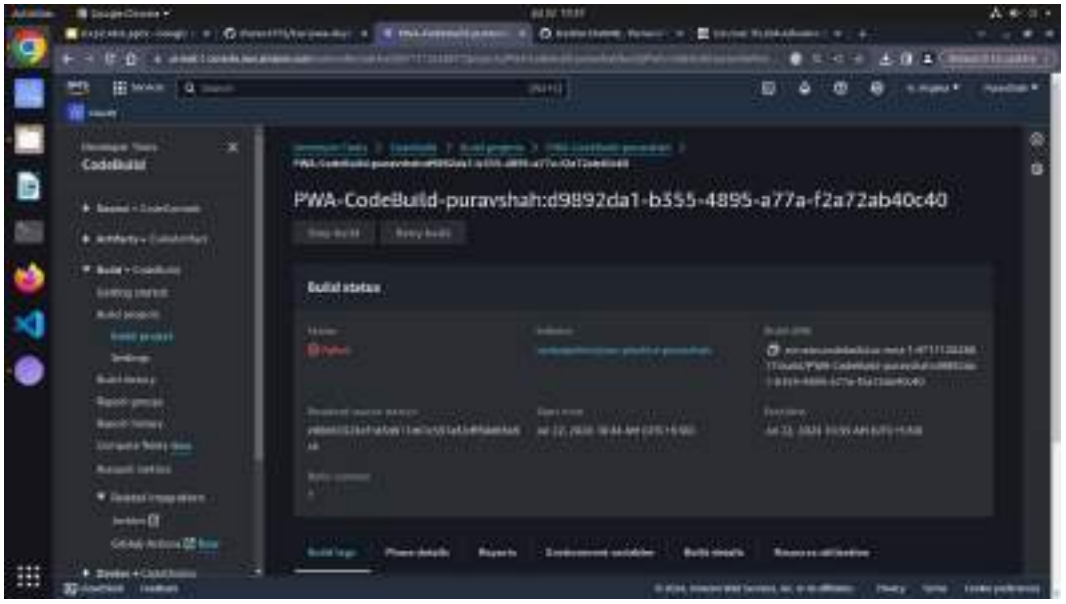

Other permissions policies
Policy name
Type
Description

Step2: Get a copy of the sample code

In this step, you will retrieve a copy of the sample app's code and choose a source to host the code.

The pipeline takes code from the source and then performs actions on it.

You can use one of three options as your source: a GitHub repository, an Amazon S3 bucket, or an AWS CodeCommit repository. Select your preference and follow the steps below:



a. If you plan to use Amazon S3 as your source, you will retrieve the sample code from the AWS GitHub repository, save it to your computer, and upload it to an Amazon S3 bucket.

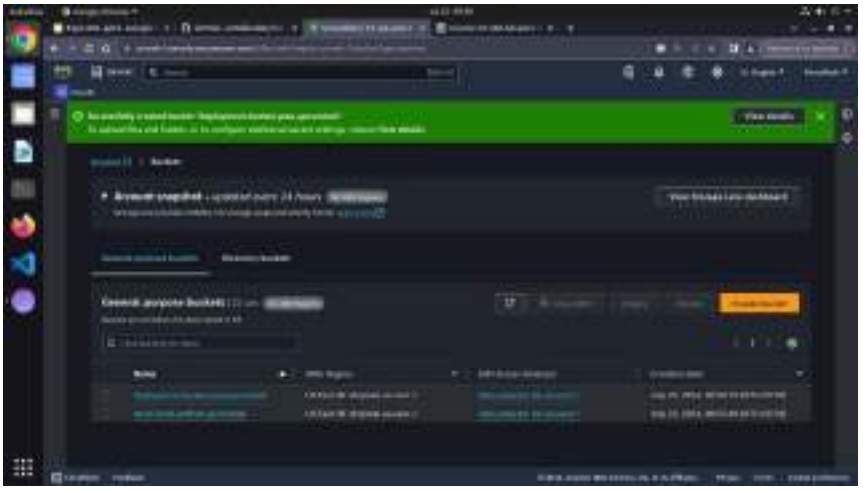
- Visit our GitHub repository containing the sample code at https://github.com/imoisharma/aws-codepipeline-s3-codedeploy-linux-2.0
- Click the dist folder.

b. Save the source files to your computer:

- Click the file named aws-codepipeline-s3-aws-codedeploy_linux.zip
- Click View Raw.
- Save the sample file to your local computer.

c. open the Amazon S3 console and create your Amazon S3 bucket:

- Click Create Bucket



- Bucket Name: type a unique name for your bucket, such as awscodepipeline-demobucket-variables. All bucket names in Amazon S3 must be unique, so use one of your own, not one with the name shown in the example.
- Region: In the drop-down, select the region where you will create your pipeline, such as ap-South-1
- Click Create.

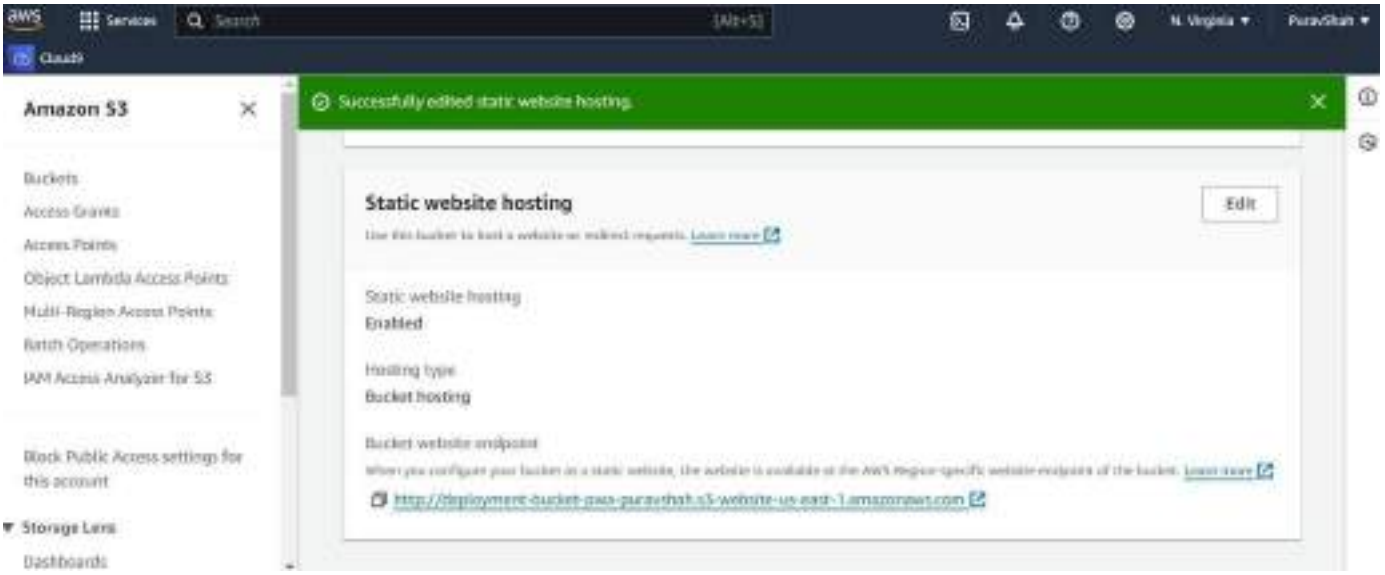
d. The console displays the newly created bucket, which is empty.

- Click Properties.
- Expand Versioning and select Enable Versioning. When versioning is enabled, Amazon S3 saves every version of every object in the bucket.

e. You will now upload the sample code to the Amazon S3 bucket:

- Click Upload.
- Follow the on-screen directions to upload the .zip file containing the sample code you downloaded from GitHub.


PWA-CodeBuild-puravshah
Configuration
Build history


Amazon S3
Successfully edited static website hosting.
Static website hosting
Edit
Static website hosting
Enabled
Hosting type
Bucket hosting
Bucket website endpoint
http://deployment-bucket-pwa-puravshah.s3-website-us-east-1.amazonaws.com
Buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards

Step3: Create your Pipeline

In this step, you will create and configure a simple pipeline with two actions: source and deploy. You will provide CodePipeline with the locations of your source repository and deployment environment.

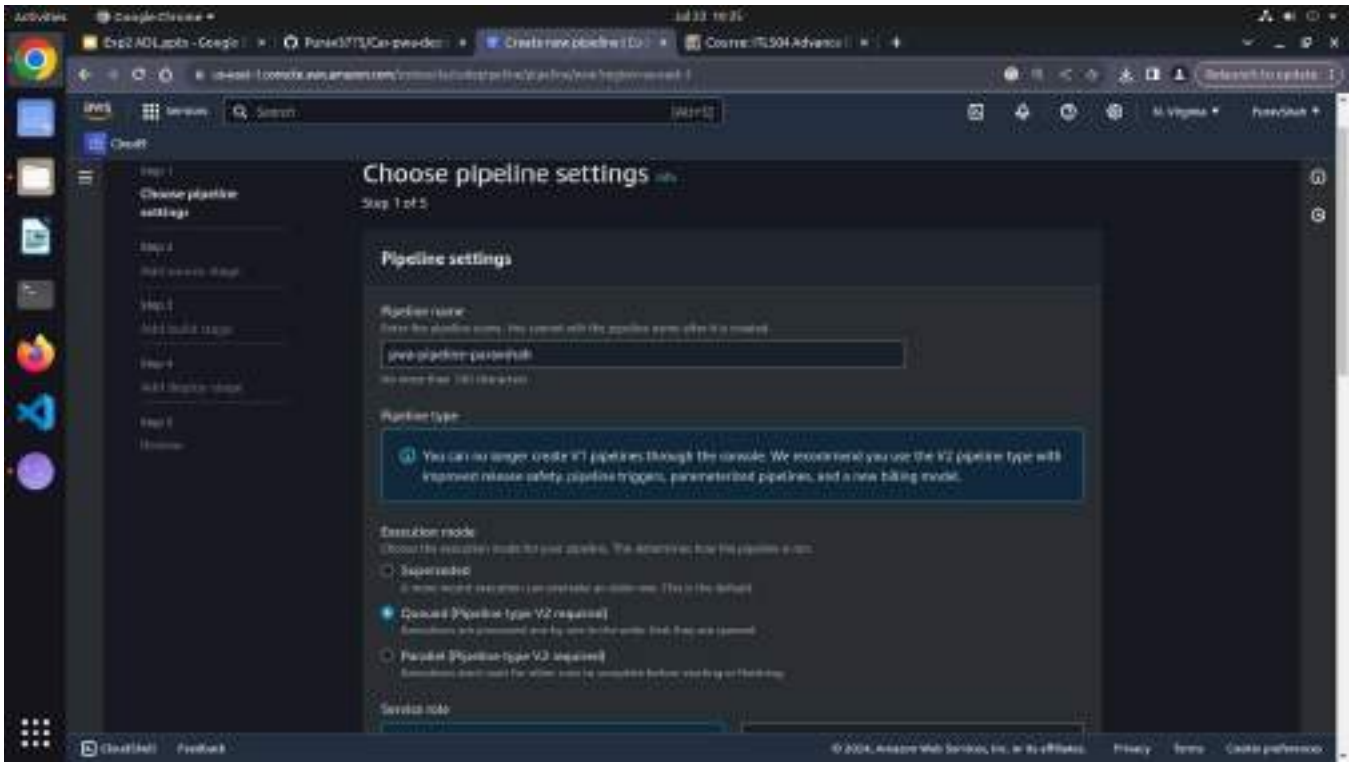
A true continuous deployment pipeline requires a build stage, where code is compiled and unit tested. CodePipeline lets you plug your preferred build provider into your pipeline. However, in this we will skip the build stage.

Goto Pipeline again and create it

In above you can give zip file name in S3 object Key and choose bucket name which you created

In Step 4: Deploy Stage:

- Deployment provider: Click AWS Elastic Beanstalk.
- Application name: MYEBS.
- Environment name: Click Myebs-env.
- Click Next step.



After your pipeline is created, the pipeline status page appears and the pipeline automatically starts to run. You can view progress as well as success and failure messages as the pipeline perform each action.

To verify your pipeline ran successfully, monitor the progress of the pipeline as it moves through each stage. The status of each stage will change from No executions yet to In Progress, and then to either Succeeded or Failed. The pipeline should complete the first run within a few minutes.

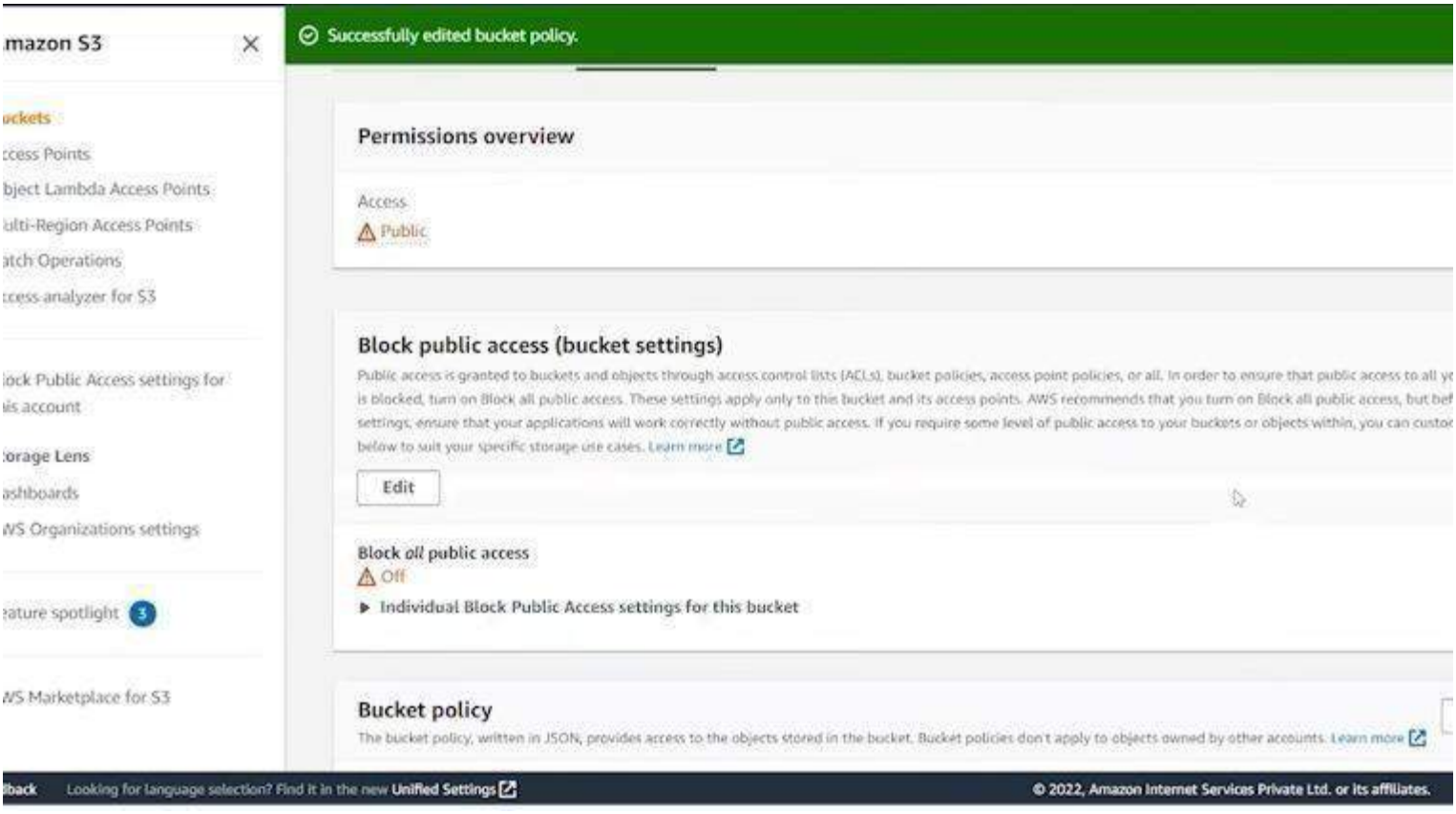
Now go to your EBS environment and click on the URL to view the sample website you deployed.

You have successfully created an automated software release pipeline using AWS CodePipeline!

Using CodePipeline, you created a pipeline that uses GitHub, Amazon S3, or AWS CodeCommit as the source location for application code and then deploys the code to an Amazon EC2 instance managed by AWS Elastic Beanstalk.

Step 5: Commit a change and then update your app Step

6: Clean up your resources



To avoid future charges, you will delete all the resources you launched throughout this tutorial, which includes the pipeline, the Elastic Beanstalk application, and the source you set up to host the code.

a. First, you will delete your pipeline:

b. Second, delete your Elastic Beanstalk application:

Conclusion: Write your own findings.

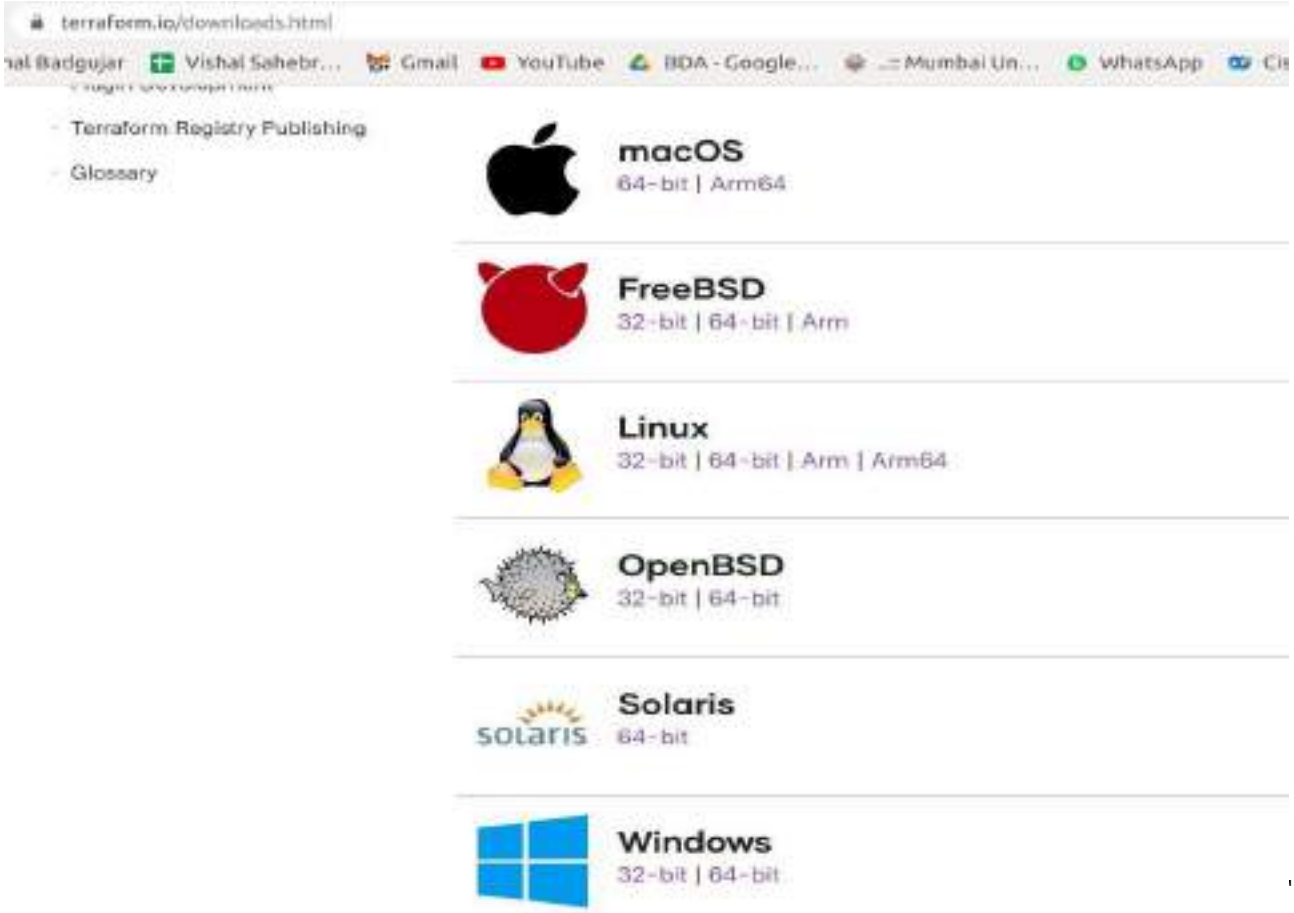
EXPERIMENT NO. 05

Aim: To understand terraform lifecycle, core concepts/terminologies and install it on a Linux Machine.

Terraform Installation Steps on Ubunu18.04

Step: 1 Terraform uses HashiCorp Configuration Language (HCL) to manage environments of Operators and Infrastructure teams. To download go to site https://www.terraform.io/downloads.html

Select the appropriate package for your operating system and architecture.



Step:2 unzip the archive by using below command

```
vishal@master:~$ unzip terraform_1.0.3_linux_amd64.zip
```

The archive will extract a single binary called **terraform**.

Step 3: Change the directory to unzipped folder

```
vishal@master:~$ cd terraform_1.0.3_linux_amd64/
```

and Move the terraform binary to a directory included in your system's PATH in my case */usr*/local/bin/

```
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~/Desktop/terraform_1.9.3_linux_amd64$ sudo mv terraform /usr/local/bin/
[sudo] password for apsit:
Sorry, try again.
[sudo] password for apsit:
```

Step 4: To check whether Terraform is installed, run:

```
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~/Desktop/terraform_1.9.3_linux_amd64$ terraform -v
Terraform v1.9.3
on linux_amd64
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~/Desktop/terraform_1.9.3_linux_amd64$
```

EXPERIMENT NO. 06

Aim: To Build, change, and destroy AWS infrastructure Using Terraform.

Pre-requistes:

1. Install the AWS CLI version 2 on Linux

Follow these steps from the command line to install the AWS CLI on Linux.

Install curl on linux

```
vishal@apsit:~$ sudo apt-get install curl
```

vishal@apsit:~$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

```
vishal@apsit:~$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 41.8M  100 41.8M    0     0  2529k      0  0:00:16  0:00:16 --:--:-- 2555k
```

vishal@apsit:~$ sudo apt install unzip

```
vishal@apsit:~$ sudo apt install unzip
```

vishal@apsit:~$ sudo unzip awscliv2.zip

```
vishal@apsit:~$ sudo unzip awscliv2.zip
```

vishal@apsit:~$ sudo ./aws/install

```
vishal@apsit:~$ sudo ./aws/install
You can now run: /usr/local/bin/aws --version
```

vishal@apsit:~$ aws --version

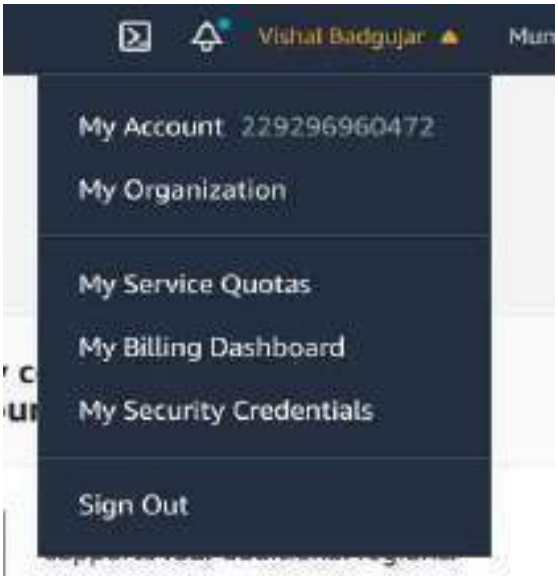
it should display the below outout.

aws-cli/2.1.29 Python/3.8.8 Linux/5.4.0-1038-aws exe/x86_64.ubuntu.18 prompt/off

```
vishal@apsit:~$ aws --version
aws-cli/2.2.25 Python/3.8.8 Linux/5.4.0-80-generic exe/x86_64.ubuntu.18 prompt/off
```

2. Create a new access key if you don't have one. Make sure you download the keys in your local machine.

Login to AWS console, click on username and go to My security credentials.



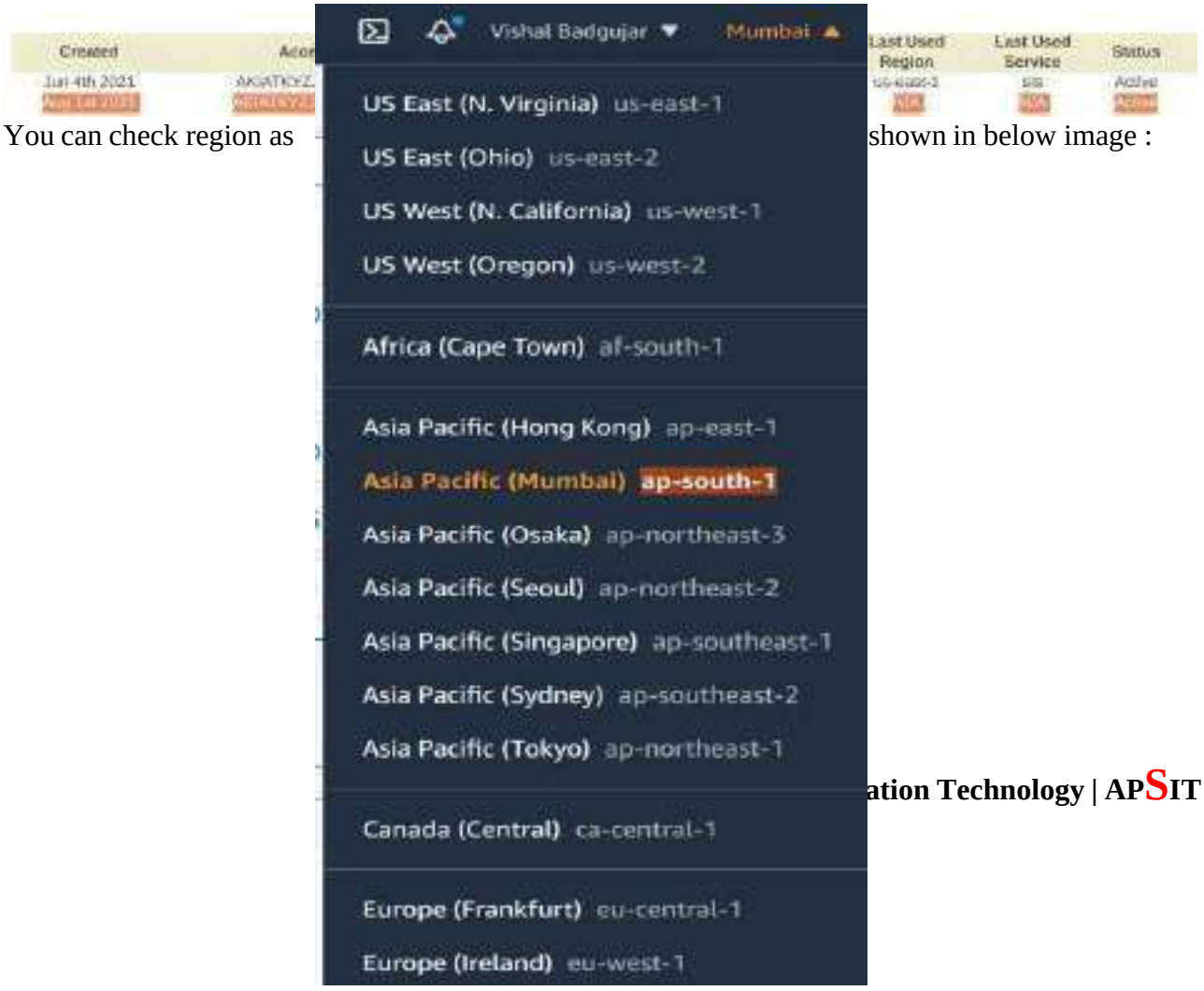
Continue on security credentials, click on access keys

Perform below commands in Linux where you have installed Terraform

First setup your access keys, secret keys and region code locally.

vishal@apsit:~$aws configure

You can check region as shown in below image :



```
vishal@apsit:~$ aws configure
AWS Access Key ID [None]: AKIATKYZJ6PMFLTCGGPV
AWS Secret Access Key [None]: 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
Default region name [None]: ap-south-1
Default output format [None]:
vishal@apsit:~$
```

Create one Directory for Terraform project in which all files of terraform we can save

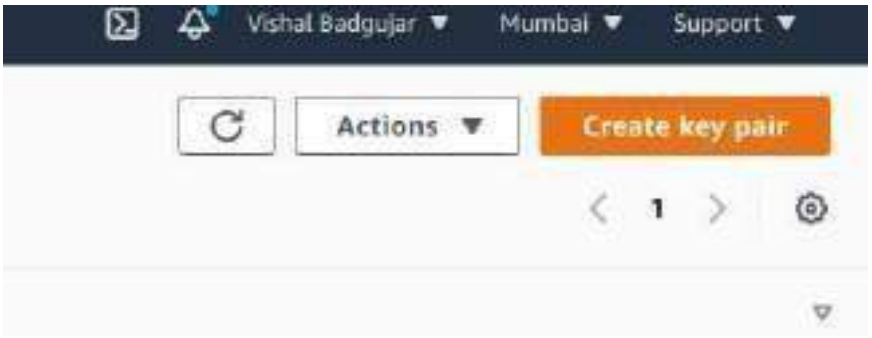
vishal@apsit:~$ cd ~
vishal@apsit:~$ mkdir project-terraform
vishal@apsit:~$ cd project-terraform

```
vishal@apsit:~$ mkdir project-terraform
vishal@apsit:~$ cd project-terraform/
vishal@apsit:~/project-terraform$
```

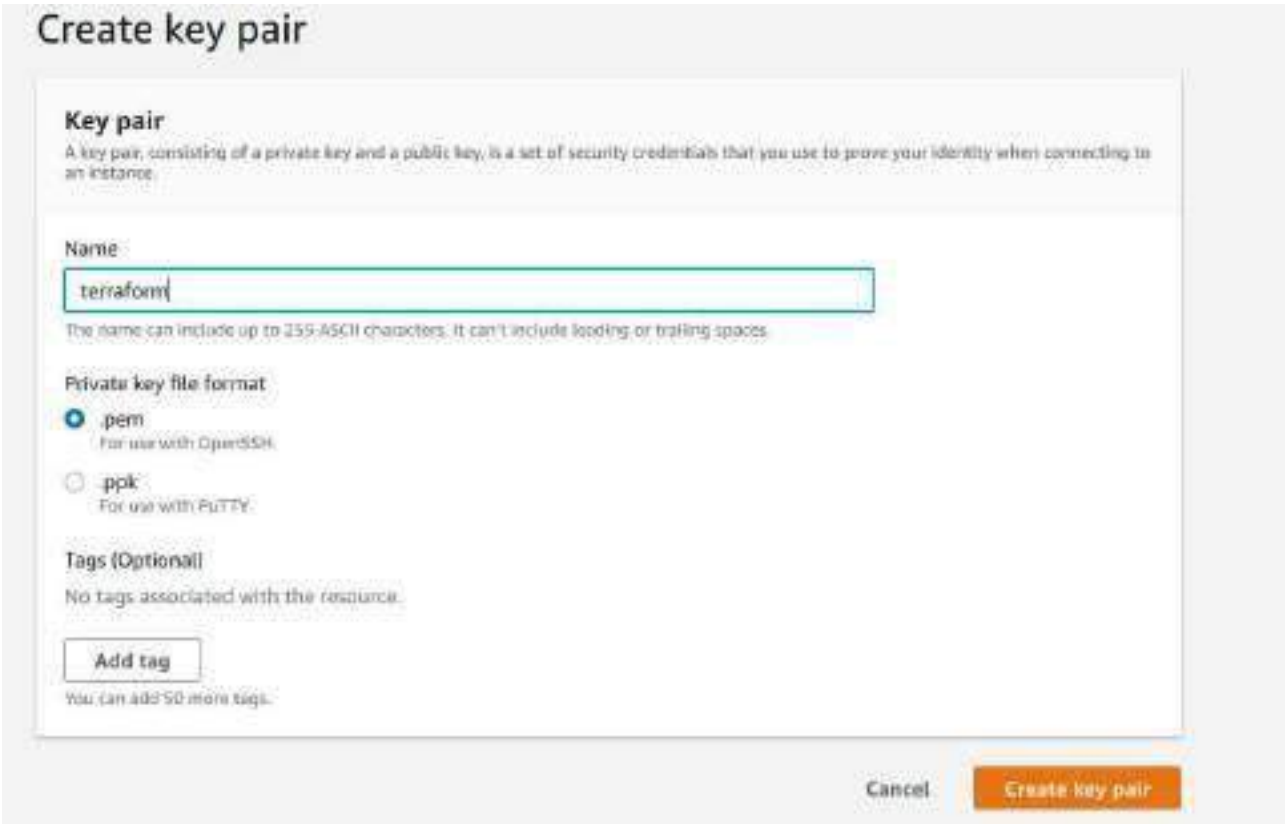
Create Terraform Files

vishal@apsit:~$ sudo nano variables.tf

In order to provide key name in variables first create key pair as shown:

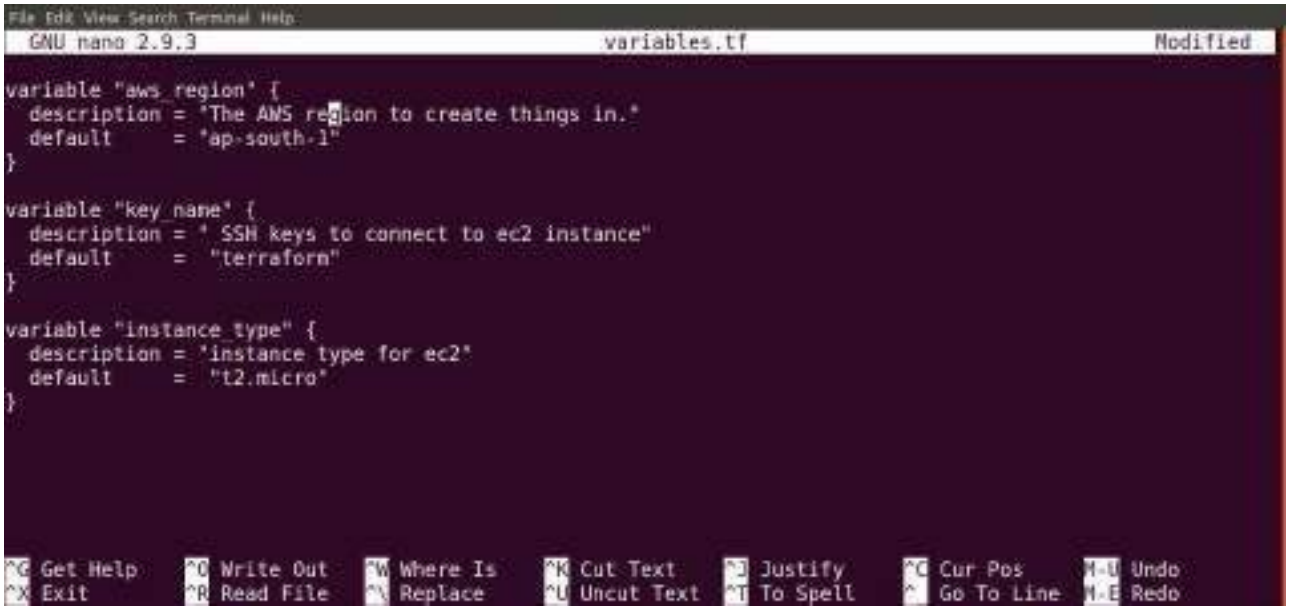


Give name to key pair file as **terraform**

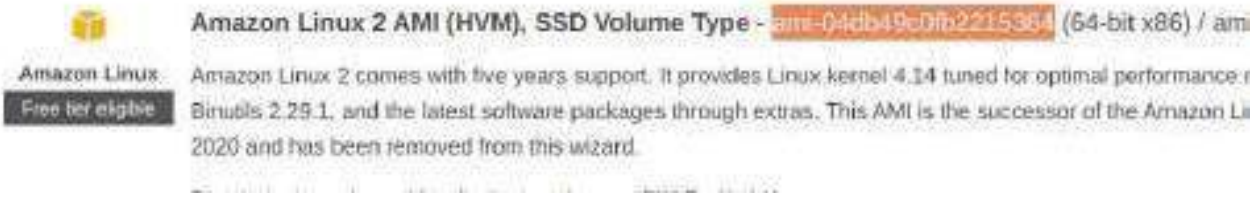


Key pair is generated

Use your Region and Key name in variable.tf as shown and provide instance type which you want to create.



After creating variable terraform file note down the AMI ID of instance which u want to create which we will use to configure our instance in main.tf file.



Now create main.tf file:

```
vishal@apsit:~/project-terraform$ sudo nano main.tf
```

```
provider "aws" {
  region = var.aws_region
}

#Create security group with firewall rules
resource "aws_security_group" "security_jenkins_port" {
  name        = "security_jenkins_port"
  description = "security group for jenkins"

  ingress {
    from_port  = 8080
    to_port    = 8080
    protocol   = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

 ingress {
    from_port = 22
```

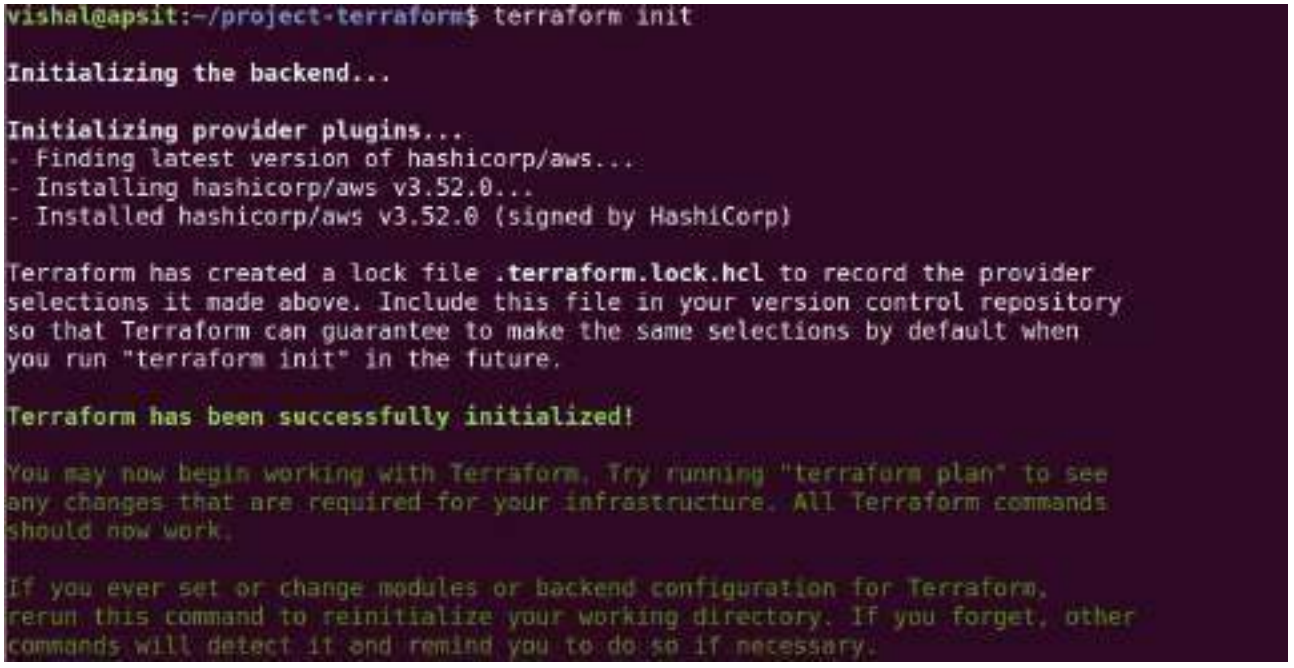
```
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
# outbound from jenkis server
  egress {
    from_port  = 0
    to_port     = 65535
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags= {
    Name = "security_jenkins_port"
  }
}

resource "aws_instance" "myFirstInstance" {
  ami           = "ami-0b9064170e32bde34"
  key_name = var.key_name
  instance_type = var.instance_type
  security_groups= [ "security_jenkins_port"]
  tags= {
    Name = "jenkins_instance"
  }
}
```

```
# Create Elastic IP address
resource "aws_eip" "myFirstInstance" {
  vpc      = true
  instance = aws_instance.myFirstInstance.id
tags= {
    Name = "jenkins_elstic_ip"
  }
}
```

Put AMI-ID in above highlighted space and Now execute the below command:

```
vishal@apsit:~/project-terraform$ terraform init
```

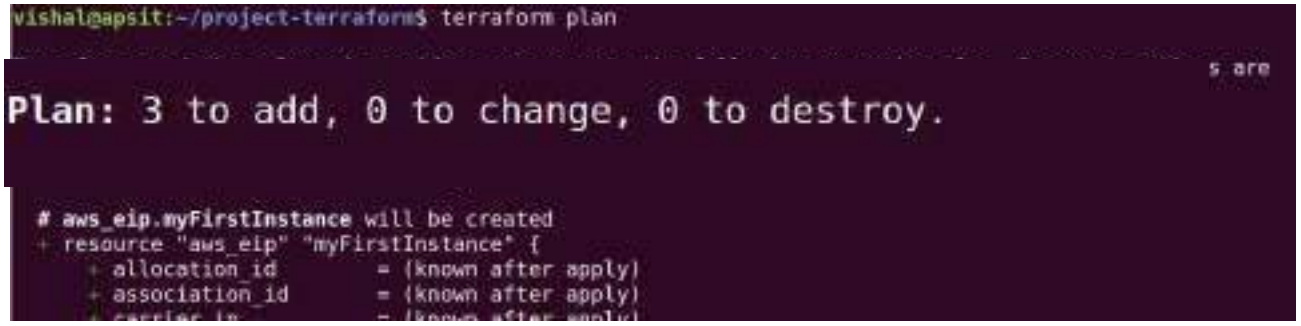
you should see like below screenshot.



Execute the below command

```
vishal@apsit:~/project-terraform$ terraform plan
```

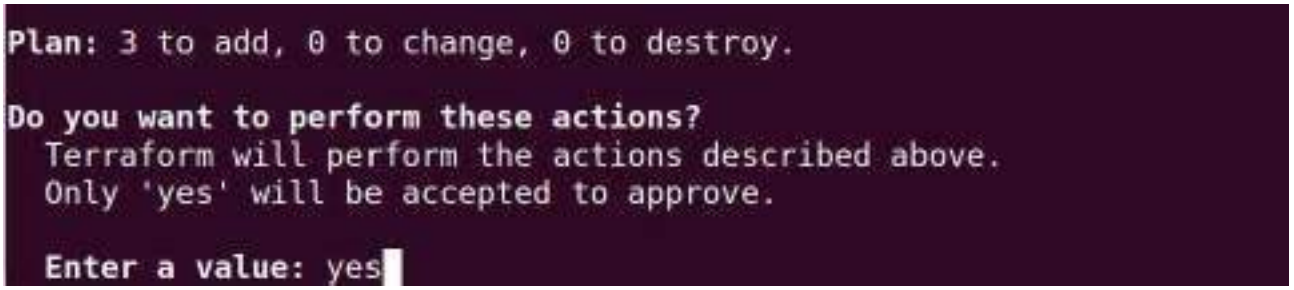
the above command will show how many resources will be added.
Plan: 3 to add, 0 to change, 0 to destroy.



Execute the below command

```
vishal@apsit:~/project-terraform$ terraform apply
```

Provide the value as Yes for applying terraform



Plan: 3 to add, 0 to change, 0 to destroy.

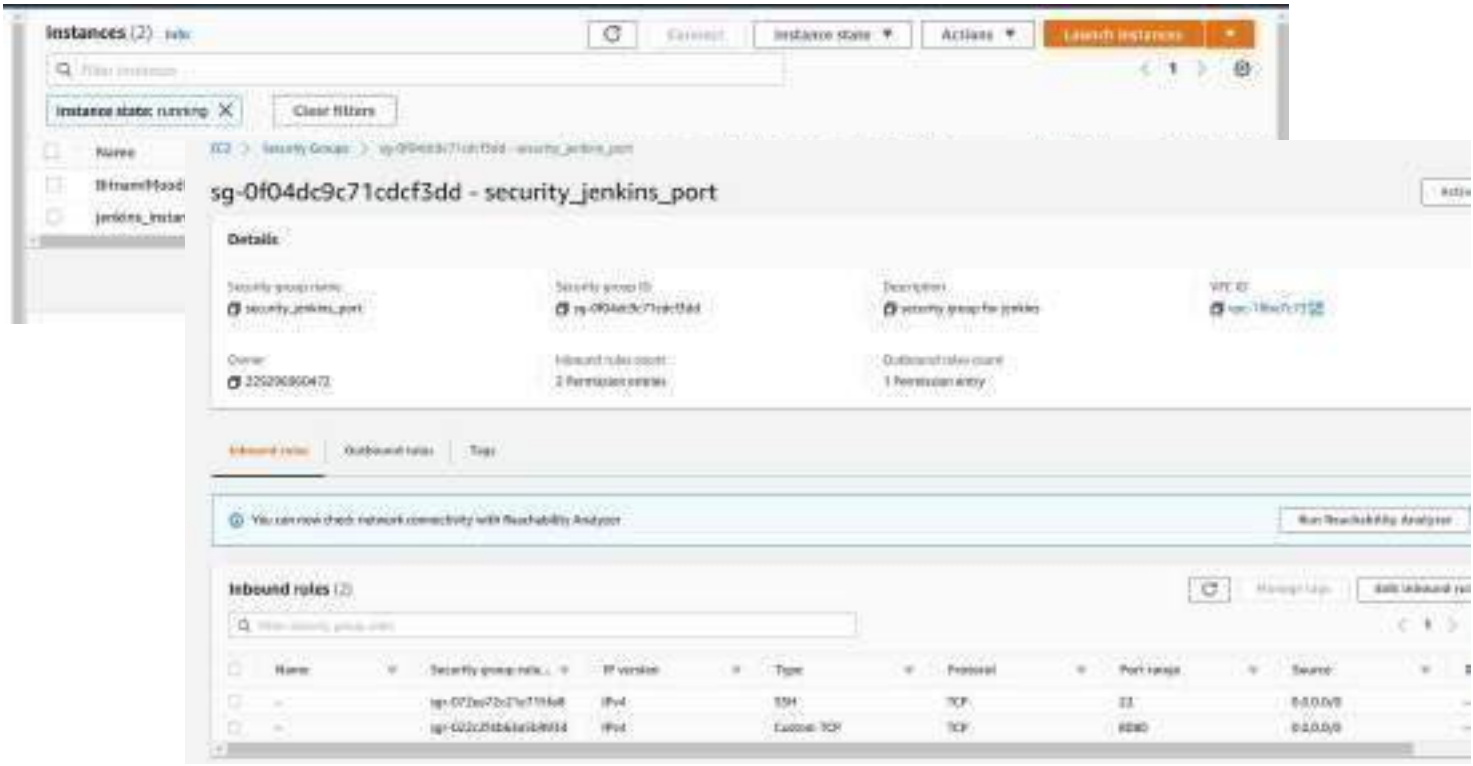
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

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

```
Plan: 2 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.myFirstInstance: Creating...
aws_instance.myFirstInstance: Still creating... [10s elapsed]
aws_instance.myFirstInstance: Still creating... [20s elapsed]
aws_instance.myFirstInstance: Still creating... [30s elapsed]
aws_instance.myFirstInstance: Creation complete after 32s [id=i-0a4a0fb7e55252d0f]
aws_eip.myFirstInstance: Creating...
aws_eip.myFirstInstance: Creation complete after 1s [id=eipalloc-0fd8f60524b10fc93]

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

Now login to EC2 console, to see the new instances up and running, you can see Jenkins_instance is up and running which we deploy from terraform.



You can also check the security group resource details which you created from terraform :

Terraform destroy

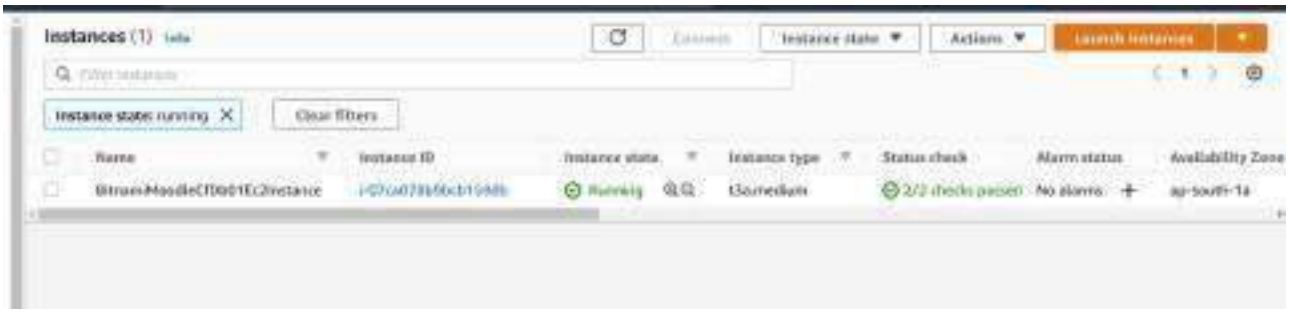
you can also destroy or delete your instance by using terraform destroy command :

```
vishal@apsit:~/project-terraform$ terraform destroy
```

```
Plan: 0 to add, 0 to change, 3 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes
```

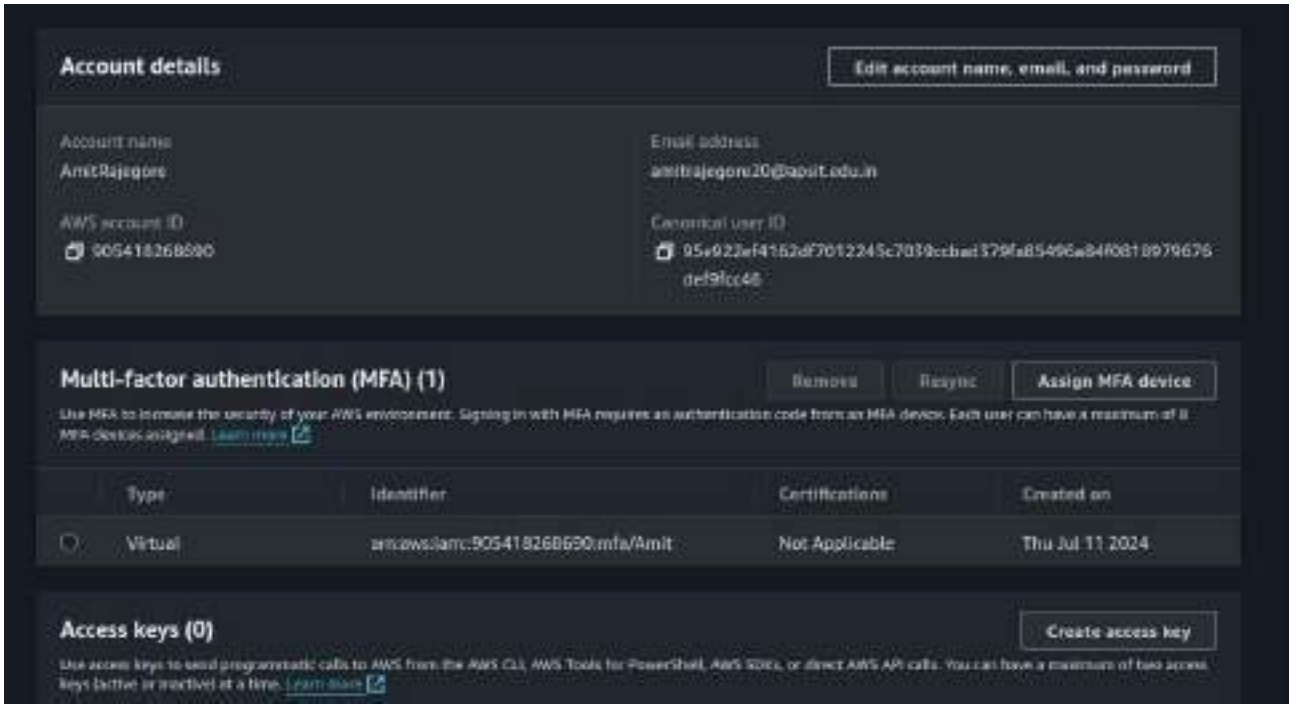
```
  Enter a value: yes

aws_eip.myFirstInstance: Destroying... [id=eipalloc-0fd8f60524b10fc93]
aws_security_group.security_jenkins_port: Destroying... [id=sg-0f04dc9c71cdcf3dd]
aws_eip.myFirstInstance: Destruction complete after 2s
aws_instance.myFirstInstance: Destroying... [id=i-0a4a0fb7e55252d0f]
aws_security_group.security_jenkins_port: Still destroying... [id=sg-0f04dc9c71cdcf3dd, 10s elapsed]
aws_instance.myFirstInstance: Still destroying... [id=i-0a4a0fb7e55252d0f, 10s elapsed]
aws_security_group.security_jenkins_port: Still destroying... [id=sg-0f04dc9c71cdcf3dd, 20s elapsed]
aws_instance.myFirstInstance: Still destroying... [id=i-0a4a0fb7e55252d0f, 20s elapsed]
aws_security_group.security_jenkins_port: Still destroying... [id=sg-0f04dc9c71cdcf3dd, 30s elapsed]
aws_instance.myFirstInstance: Still destroying... [id=i-0a4a0fb7e55252d0f, 30s elapsed]
aws_security_group.security_jenkins_port: Destruction complete after 38s
aws_instance.myFirstInstance: Still destroying... [id=i-0a4a0fb7e55252d0f, 40s elapsed]
aws_instance.myFirstInstance: Destruction complete after 40s

Destroy complete! Resources: 3 destroyed.
```

Now you can see instance which you created by using terraform is deleted successfully from aws console also you can check it will removed successfully:



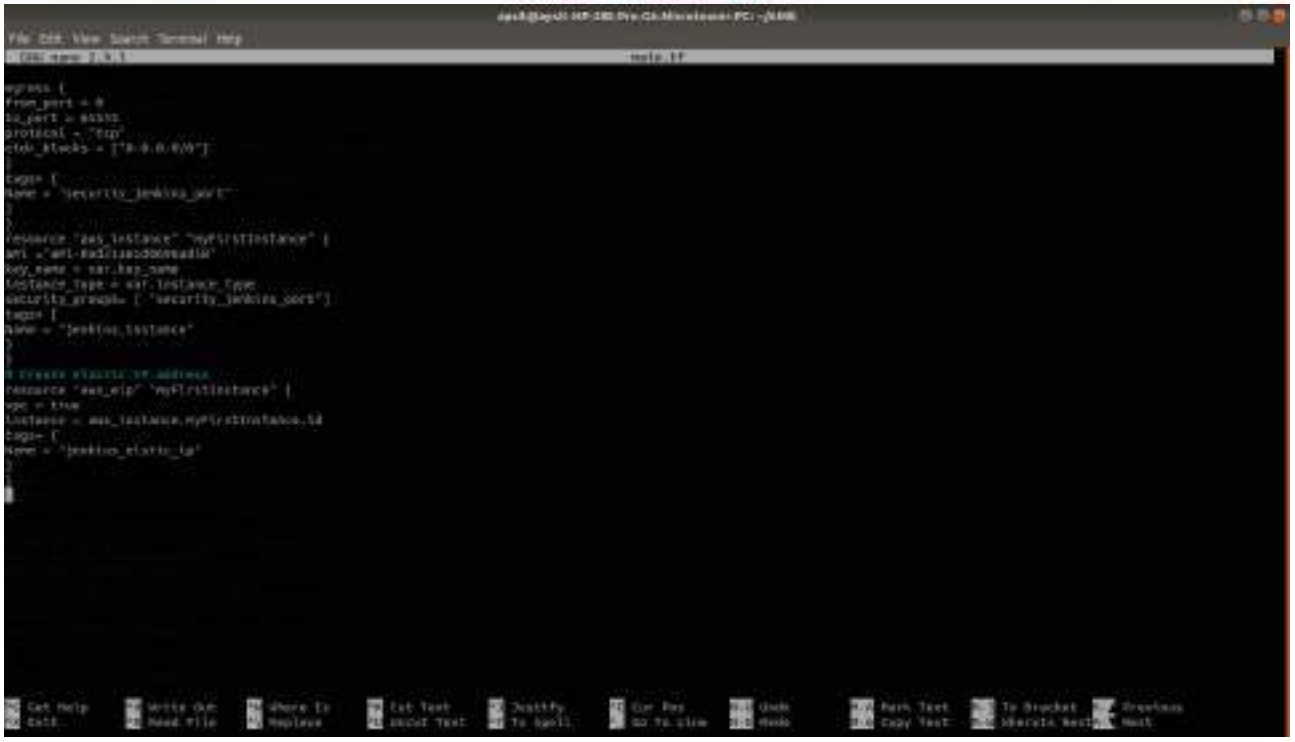
All the Resources including Security groups, EC2 instances using terraform will be deleted. In this way we can automate infrastructure set up using terrform in aws cloud.

OUTPUT SCREENSHOTS :




```
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~$ aws --version
aws-cli/2.17.20 Python/3.11.9 Linux/5.4.0-150-generic exe/x86_64.ubuntu.18
```

```
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~$ aws configure
AWS Access Key ID [****************4PM7]: AKIA5FTZCMAJ02RJ4PM7
AWS Secret Access Key [****************8lfl]: rjwwlW4vRWjkQx8chYGgc7tUhljTbyry9GUV8lfl
Default region name [Global]: ap-south-1
Default output format [None]:
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~$ cd -
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~$ cd Amit
bash: cd: Amit: No such file or directory
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~$ mkdir AMit
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~$ cd AMit
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~/AMit$ pwd
/home/apsit/AMit
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~/AMit$ sudo nano variables.tf
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~/AMit$ sudo nano main.tf
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~/AMit$ sudo nano variables.tf
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~/AMit$ sudo nano main.tf
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~/AMit$
```



```
apslt@apslt-HP-280-Pro-G6-Microtower-PC:~/AMLt$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.60.0...
- Installed hashicorp/aws v5.60.0 (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.
apslt@apslt-HP-280-Pro-G6-Microtower-PC:~/AMLt$
```

```
Warning: Argument is deprecated

  with aws_eip.myFirstInstance,
  on main.tf line 42, in resource "aws_eip" "myFirstInstance":
  42:   vpc = true

use domain attribute instead

(and one more similar warning elsewhere)


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_security_group.security_jenkins_port: Creating...
aws_instance.myFirstInstance: Creating...
aws_security_group.security_jenkins_port: Creation complete after 2s [id=sg-0fc1761930fa1c626]
aws_instance.myFirstInstance: Still creating... [10s elapsed]
aws_instance.myFirstInstance: Still creating... [20s elapsed]
aws_instance.myFirstInstance: Still creating... [30s elapsed]
aws_instance.myFirstInstance: Creation complete after 31s [id=i-087984fb40869846c]
aws_eip.myFirstInstance: Creating...
aws_eip.myFirstInstance: Creation complete after 2s [id=eipalloc-0b60dd06adsadd9e]

Warning: Argument is deprecated

  with aws_eip.myFirstInstance,
  on main.tf line 42, in resource "aws_eip" "myFirstInstance":
  42:   vpc = true

use domain attribute instead


Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
apsit@apsit-HP-Z8G-Pro-G8-Microtower-PC:~/Anil$
```

```
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_eip.myFirstInstance: Destroying... [id=eipalloc-0b6080056a01add96]
aws_security_group.security_jenkins_port: Destroying... [id=sg-0fc1761930fa1c626]
aws_eip.myFirstInstance: Destruction complete after 1s
aws_instance.myFirstInstance: Destroying... [id=i-087904fb40869846c]
aws_security_group.security_jenkins_port: Still destroying... [id=sg-0fc1761930fa1c626, 10s elapsed]
aws_instance.myFirstInstance: Still destroying... [id=i-087904fb40869846c, 10s elapsed]
aws_security_group.security_jenkins_port: Still destroying... [id=sg-0fc1761930fa1c626, 20s elapsed]
aws_instance.myFirstInstance: Still destroying... [id=i-087904fb40869846c, 20s elapsed]
aws_security_group.security_jenkins_port: Still destroying... [id=sg-0fc1761930fa1c626, 30s elapsed]
aws_instance.myFirstInstance: Still destroying... [id=i-087904fb40869846c, 30s elapsed]
aws_security_group.security_jenkins_port: Destruction complete after 37s
aws_instance.myFirstInstance: Still destroying... [id=i-087904fb40869846c, 40s elapsed]
aws_instance.myFirstInstance: Destruction complete after 40s

Destroy complete! Resources: 3 destroyed.
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~/AWS$
```

Conclusion: The knowledge of applying IaC (Infrastructure as Code) was quite amusing to learn. It is very well explained in a simple proficient way . It broadens our knowledge domain of Devops and its practical application.