AGENDA

Monday, June 19, 2023 11:50 AM

MAIN AGENDA

1. Automating the build of cloud infrastructure
2. Automating the process of deploying the application and in the created infra resources
3. Deploying in the different environments like (dev, testing, prod)
4. No manual intervention involved

COMPONENTS :

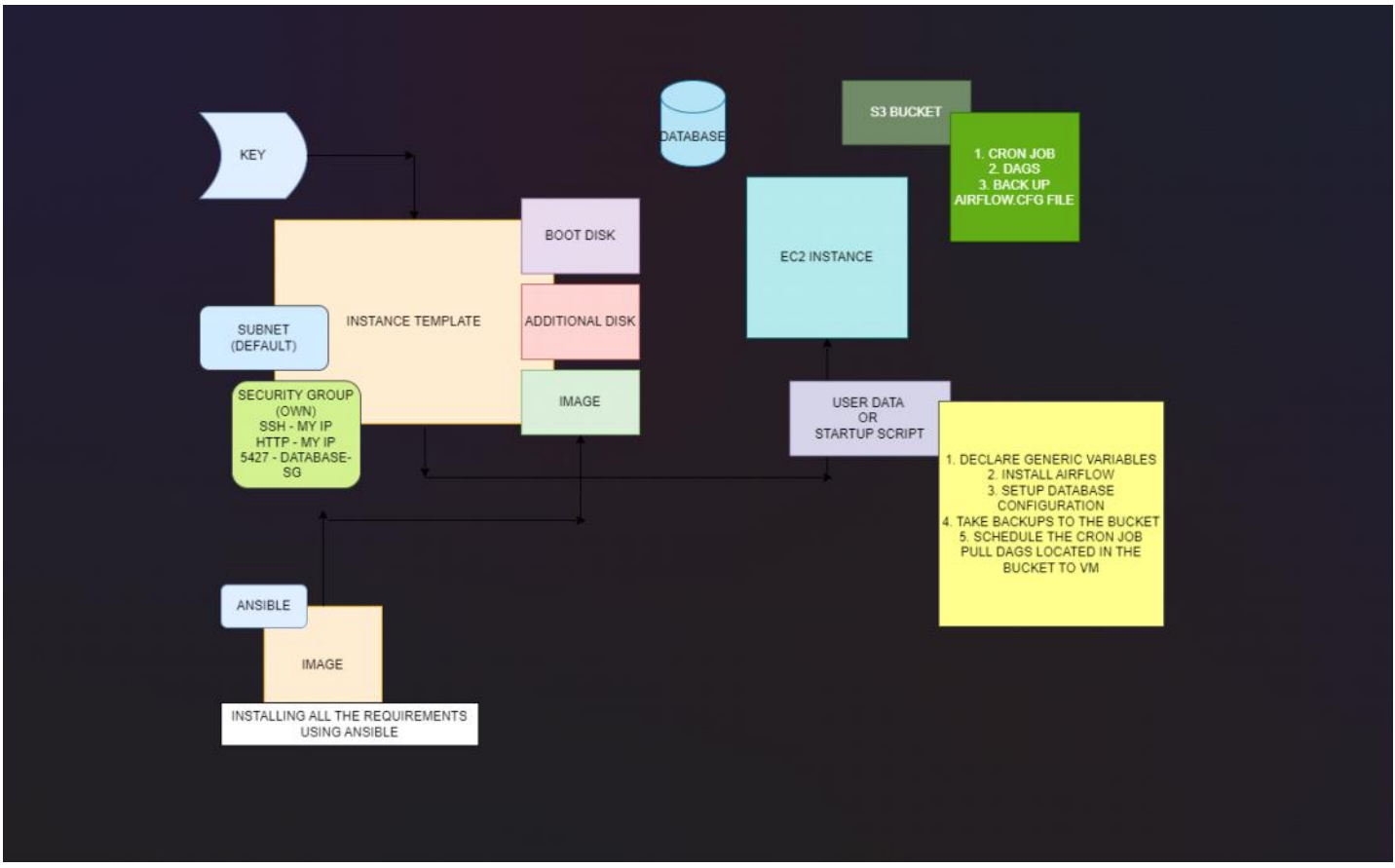
1. AWS CLOUD CONSOLE
   1.1 EC2 INSTANCE
   1.2 INSTANCE TEMPLATE
   1.3 UBUNTU
   1.4 REMOTE DATABASE SERVICES
   1.5 S3 BUCKET
   1.6 LOAD BALANCER
   1.7 SECURITY GROUP
   1.8 AUTO SCALING
   1.9 HEALTH CHECKS

2. JENKINS
3. TERRAFORM
4. GIT
5. ANSIBLE
6. SHELL SCRIPTING

ADVANTAGES OF THIS :

ARCHITECTURE

Tuesday, December 19, 2023 10:20 PM

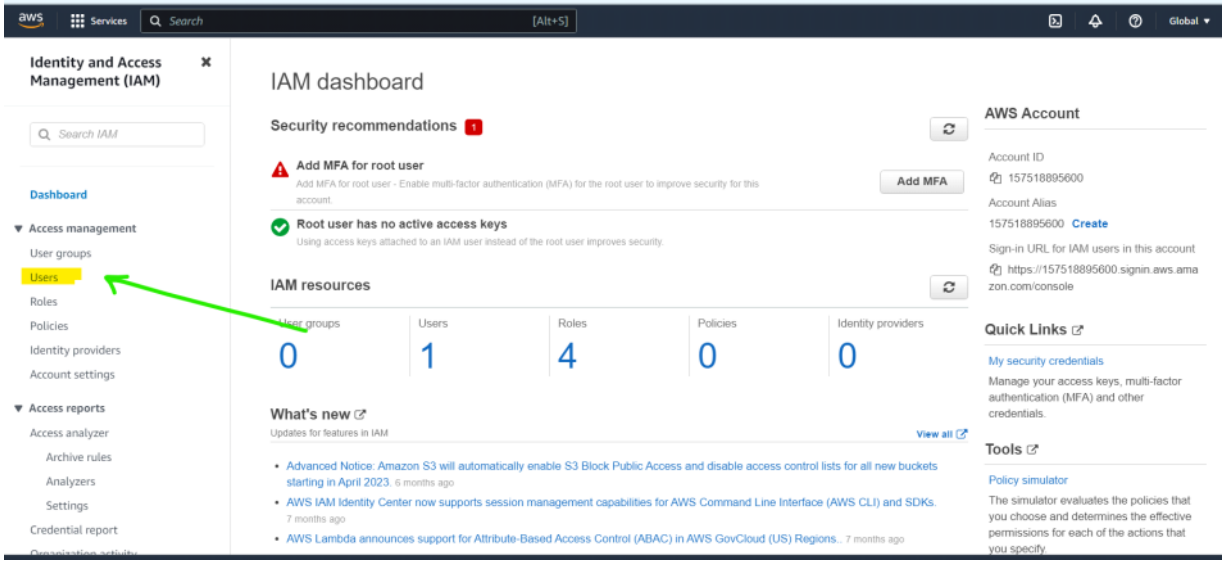


AWS CLI AUTHENTICATION

Monday, June 19, 2023 2:23 PM

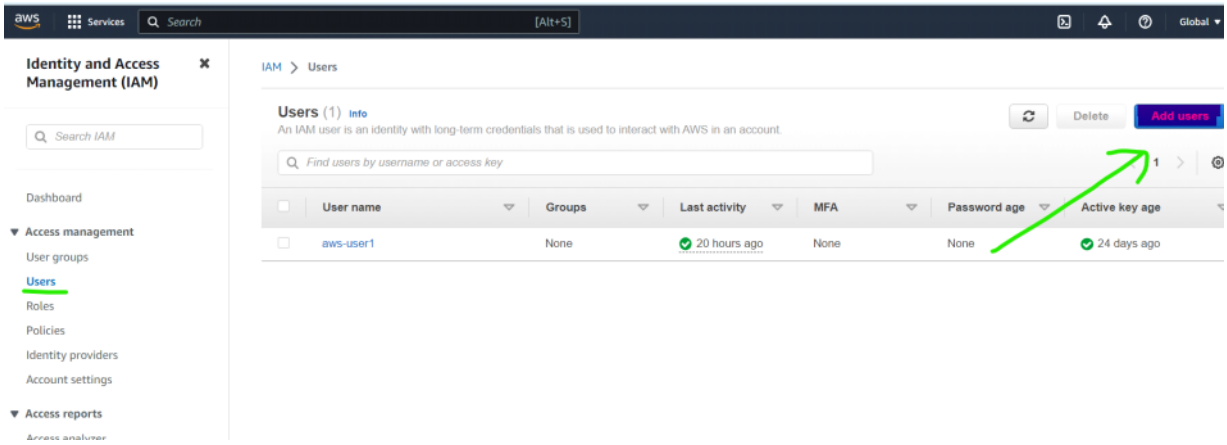
→ This is the Documentation steps on how to authenticate AWS CLI in our local machine.
→ First, Open AWS Console and login to it as a root user
→ Now, we have to create a IAM user, with the required permissions to it. So, whatever permissions are provided, it will allow the specified roles to that IAM user
→ Now, Search for IAM > users

→



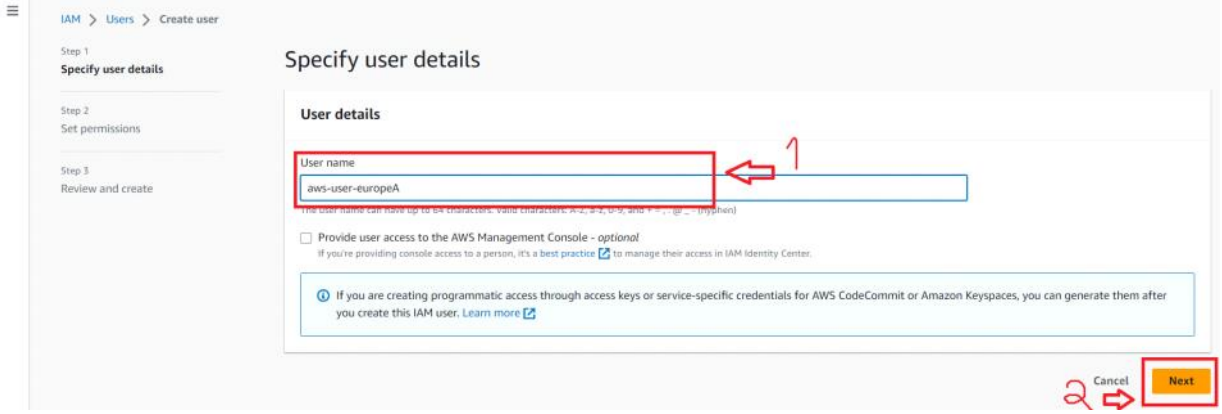
→ Click on Add Users

→

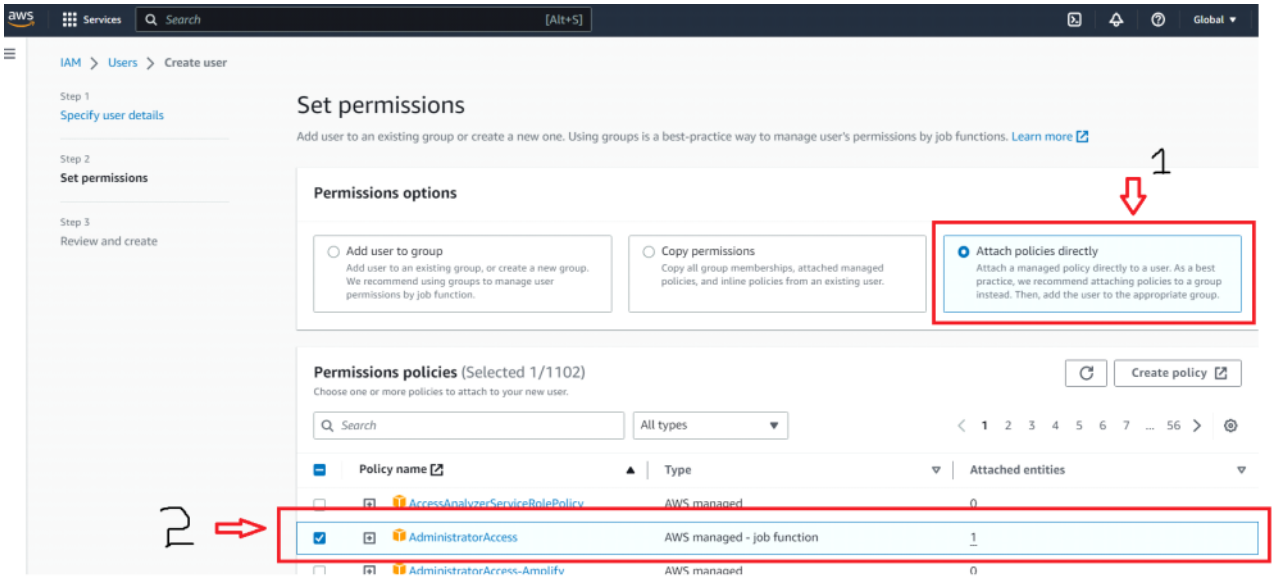


→ Provide the user name and click on NEXT

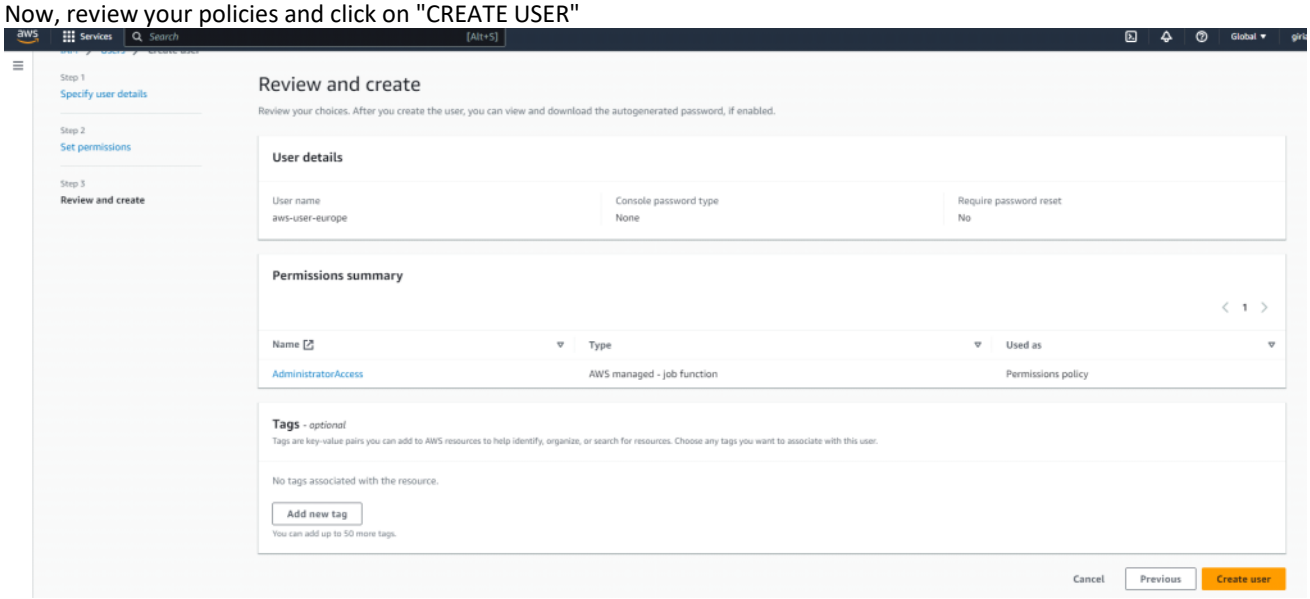
→



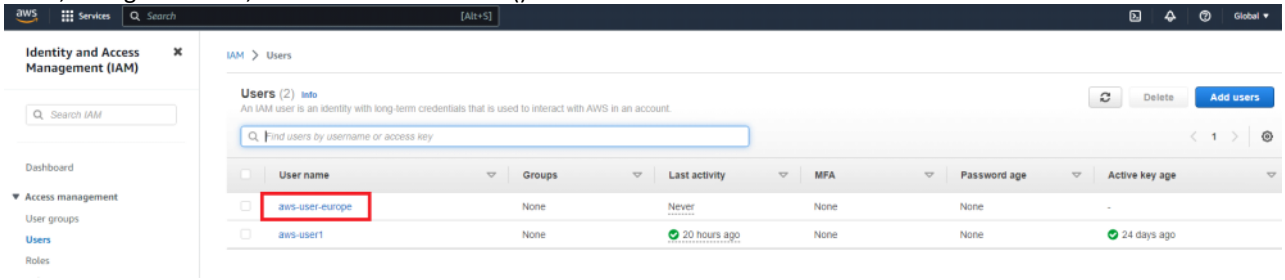
→ Now, select the "Attach policies directly" and check mark the "AdministratorAccess"
→ NOTE: You can select the Policy name as per your requirement, If you want to provide instance access only, then select the option "AmazonEC2ContainerRegistryFullAccess"
→ Now click on NEXT

→ 

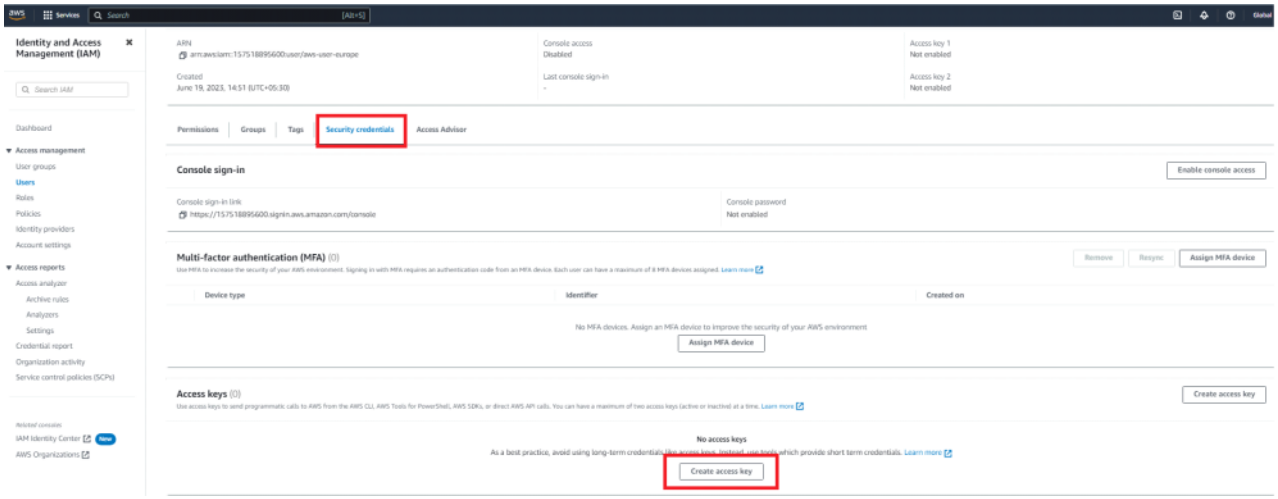
→ Now, review your policies and click on "CREATE USER"

→ 

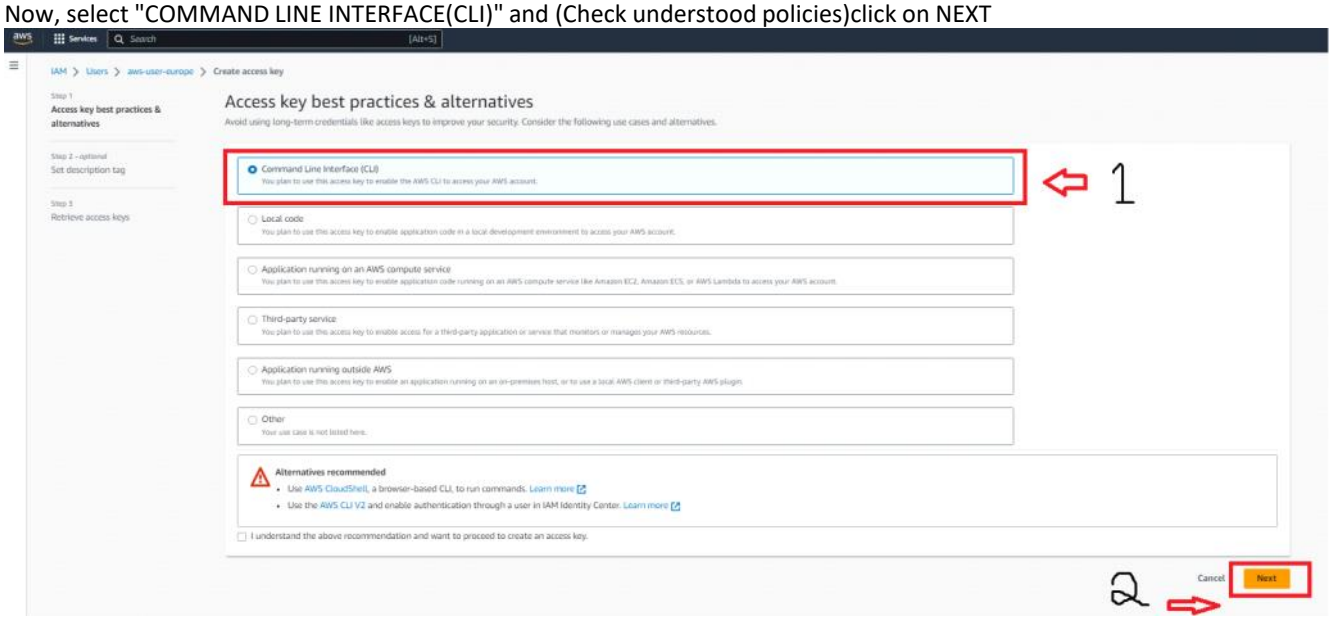
→ Once, user got created, click on that user name ()

→ 

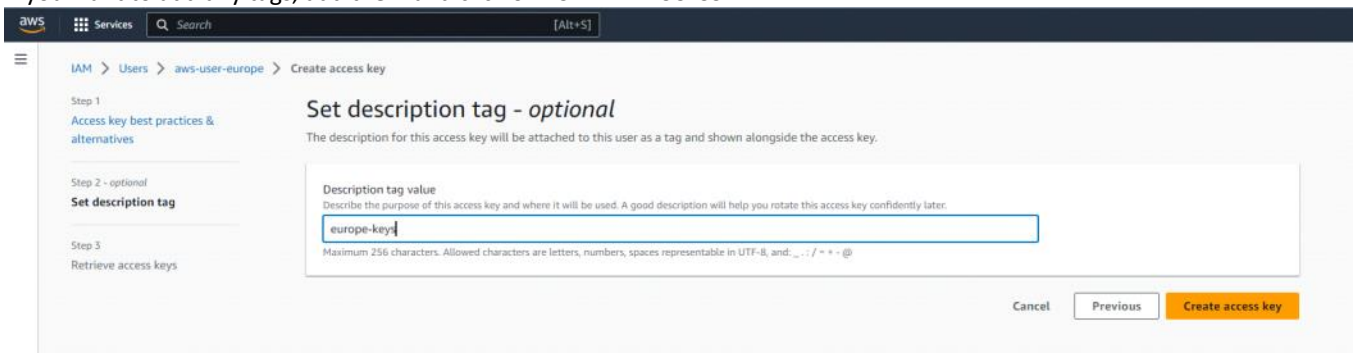
→ Go to SECURITY CREDENTIALS tab > Access Keys > Click on "CREATE ACCESS KEYS"

→ 

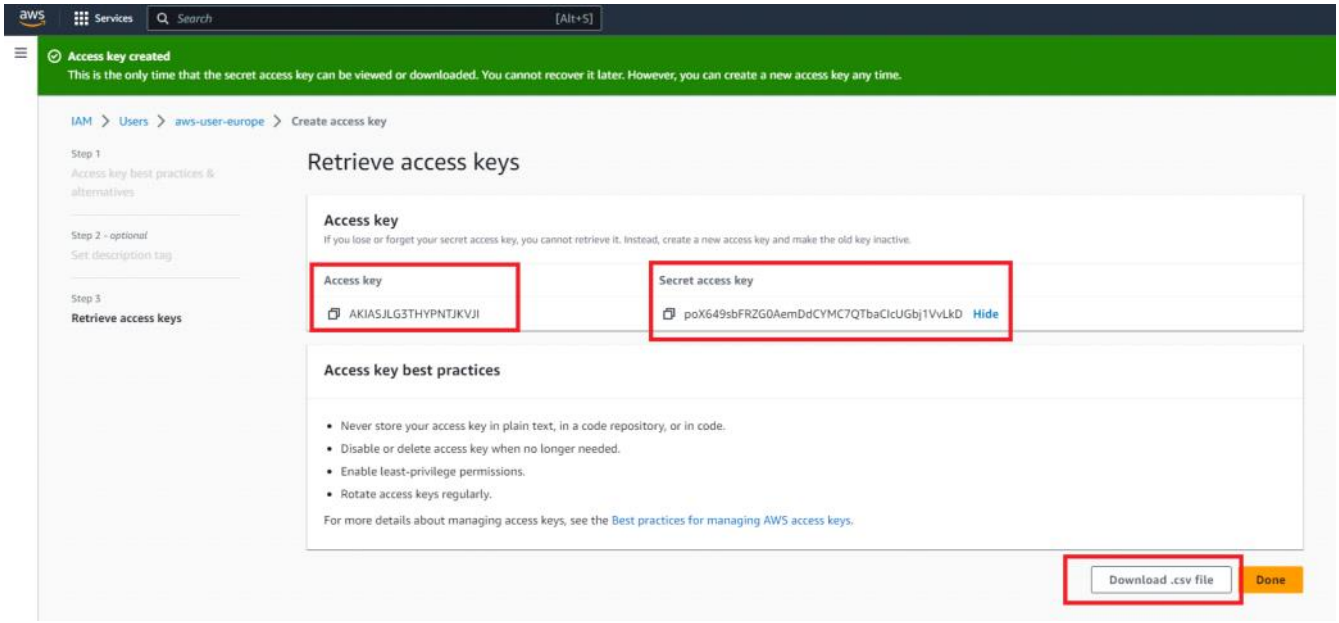
→ Now, select "COMMAND LINE INTERFACE(CLI)" and (Check understood policies)click on NEXT

→ 

→ If you want to add any tags, add them and click on "CREATE ACCESS KEY"

→ 

→ Now, it will provide you the ACCESS KEY and SECRET KEY(COPY THEM)
→ Try to download the data in .CSV file by click the "DOWNLOAD .CSV" option. It will save in your local machine.

→ 

→ Click on DONE
→ By this, you got an user credentials. So, you can authenticate to aws cloud by using these.

→ The next step is to Authenticate the AWS Cloud in our local machine
→ Open GIT BASH in your computer
→ Run the below command
→ ***$ aws configure***

```
AWS Access Key ID [****************2GHR]: AKIASJLG****************
AWS Secret Access Key [****************x9DE]: LNKGr1fq0L/****************************
Default region name [us-east-1]: us-east-1
Default output format [json]: json
```

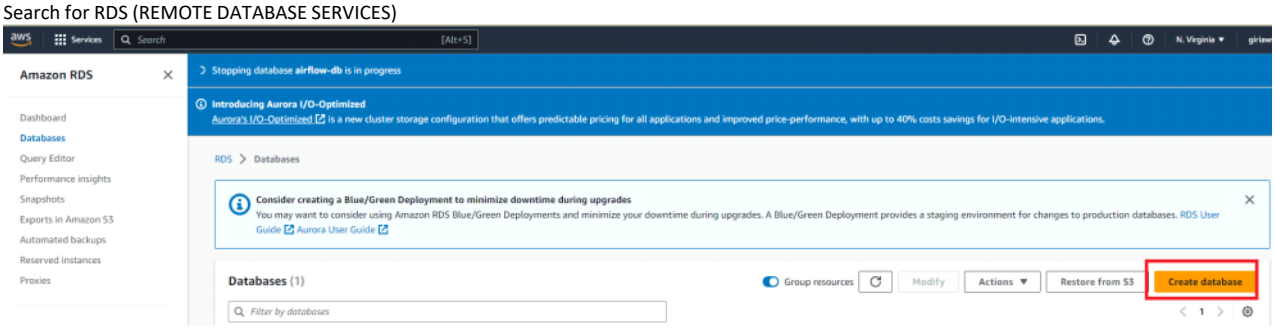
→ Enter your "ACCESS KEY", "SECRET KEY" , "REGION" and "OUTPUT FORMAT as : JSON"
→ Now, your authentication in your local machine is completed. So, you can access the aws cloud without any password

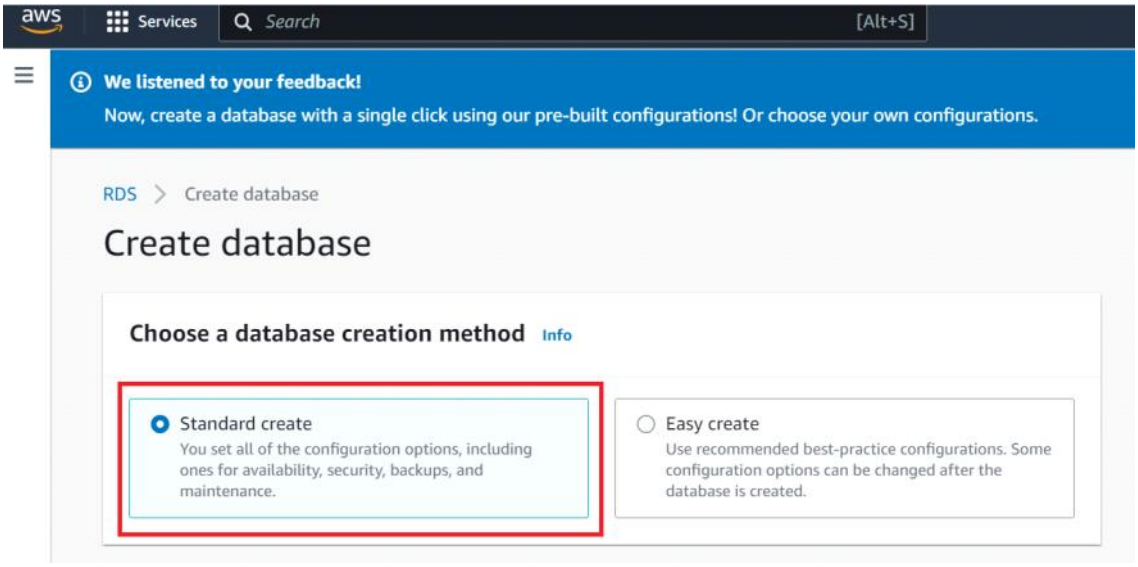
→ Run the below command to check you configuration
→ ***$ aws sts get-caller-identity***

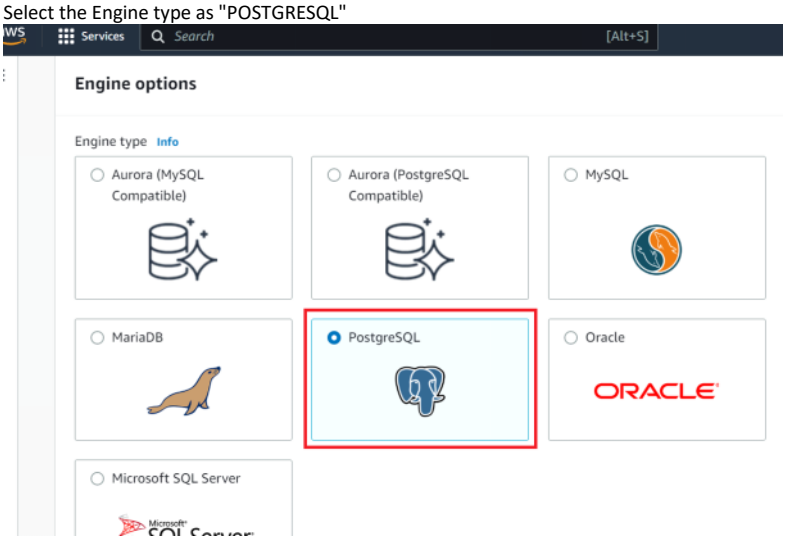
```
{
    "UserId": "AIDASJLGXYZYXZABCAB",
    "Account": "123445245600",
    "Arn": "arn:aws:iam::123445245600:user/aws-user-europe"
}
```

DATABASE SETUP

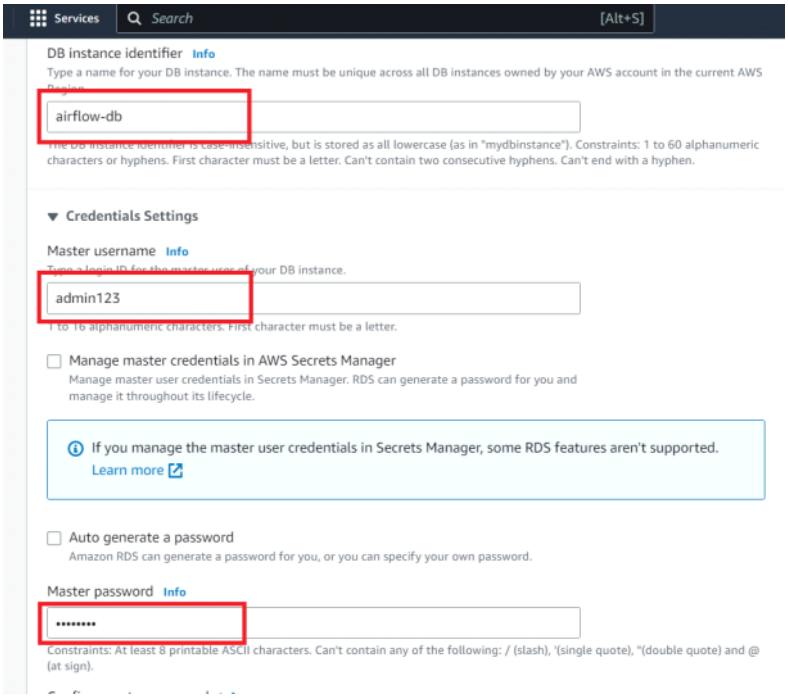
Thursday, June 22, 2023 12:08 PM

→ Open AWS CONSOLE
→ Search for RDS (REMOTE DATABASE SERVICES)
→ 

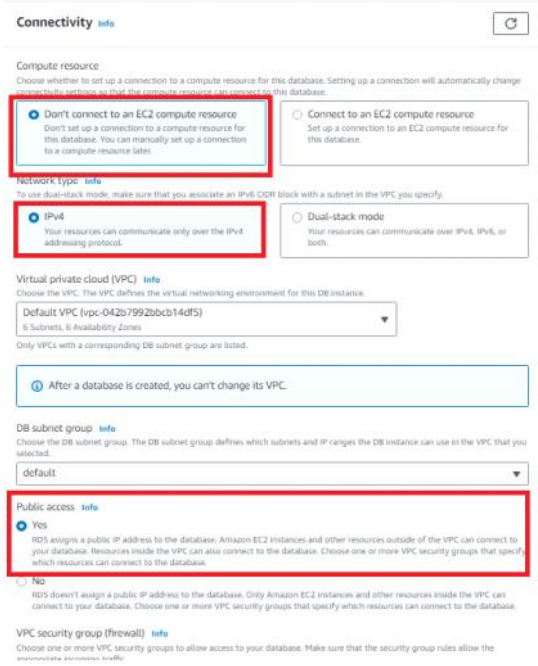
→ Click on "CREATE DATABASE"
→ Now, Create a database with the below parameters
→ Select "STANDARD CREATE"
→ 

→ Select the Engine type as "POSTGRESQL"
→ 

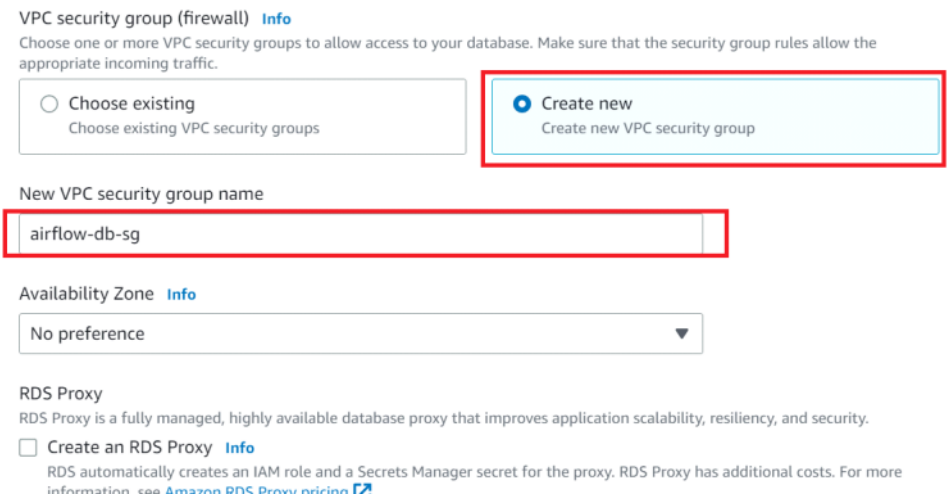
→ Here, you need to provide "DATABASE INSTANCE NAME", "MASTER USERNAME" and "MASTER PASSWORD"
→ I provided the details as below
- DATABASE INSTANE IDENTIFER : airflow-db
- MASTER USERNAME : admin123
- MASTER PASSWORD : admin@123

→ 

→ Choose preferred storage, instance configuration, and dbclass as very minimum type (db.t3.micro)
→ Choose PUBLIC ACCESS : YES (or else you can't connect to your database)

→ 

→ Here, I'm creating my own SECURITY GROUP and providing my own rules to the database
→ So, select "CREATE NEW" and provide a name to your new SG : "airflow-db-sg"

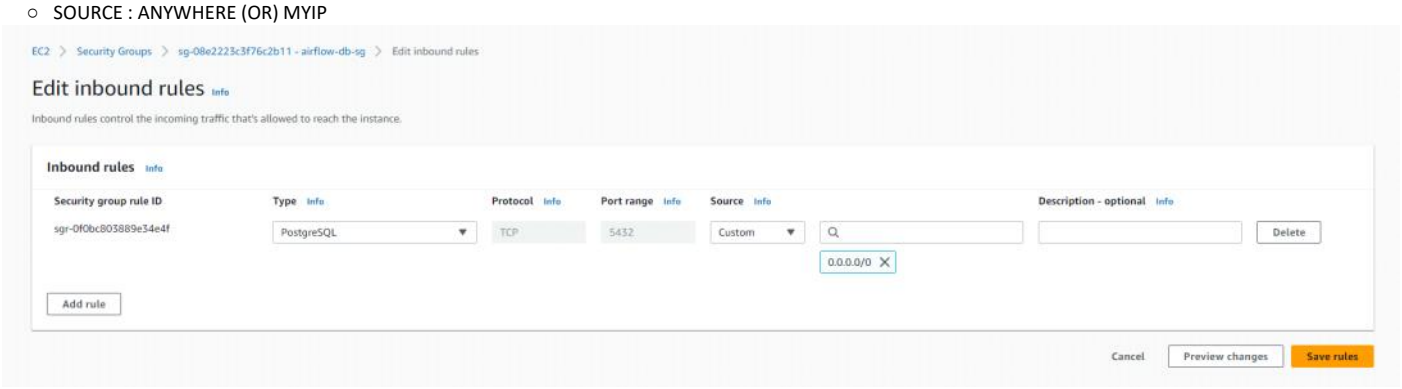


→ Now, set up the rule like below :

→ You can also set the rules within your IP as well

- TYPE : POSTGRESQL
- PROTOCOL : TCP
- PORT RANGE : 5432
- SOURCE : ANYWHERE (OR) MYIP

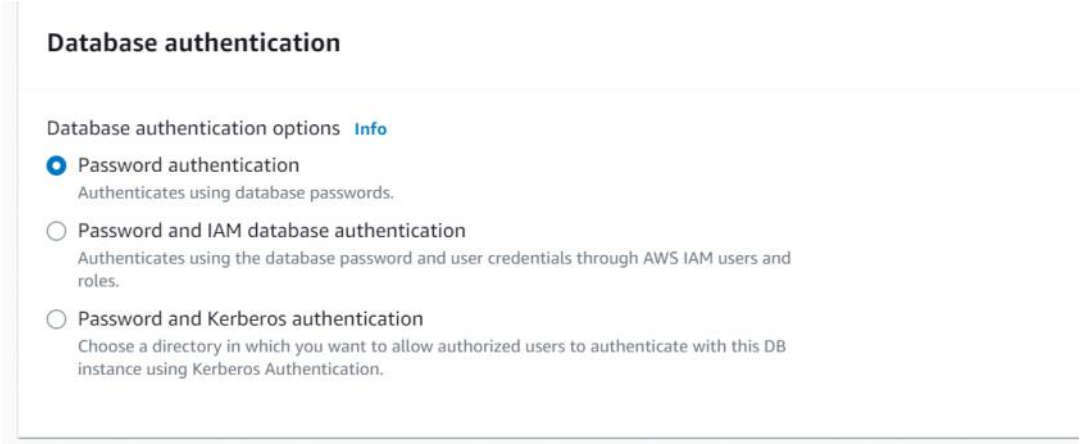
→



→ Click on "SAVE RULES"

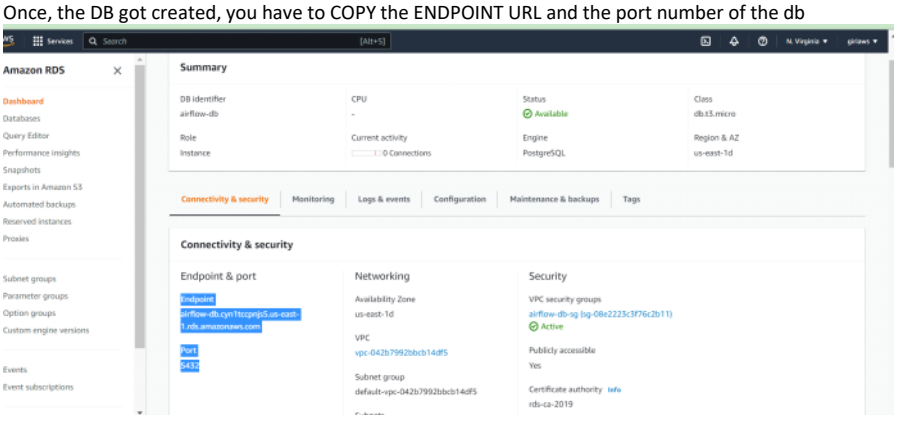
→ Select the DATABASE AUTHENTICATION as : "PASSWORD AUTHENTICATION"

→



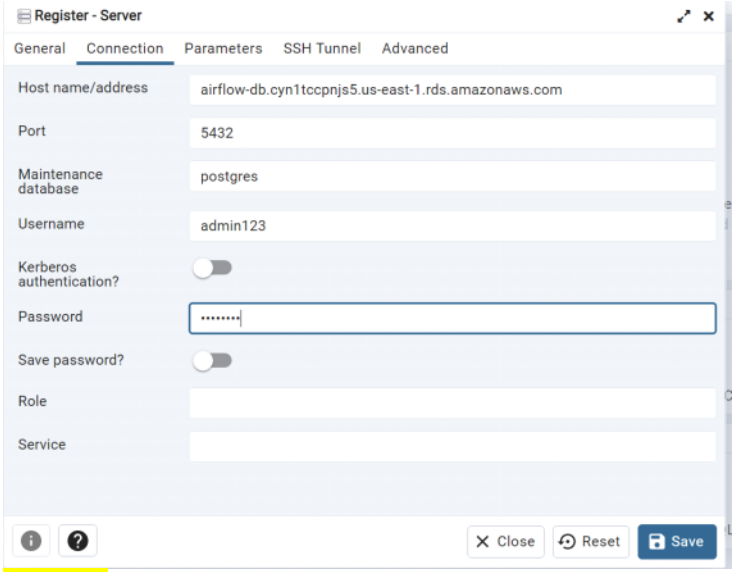
→ Keep the other options as default and click on "CREATE DATABASE"

→ Once, the DB got created, you have to COPY the ENDPOINT URL and the port number of the db



→ Now, let's try to connect with the PGADMIN
→ (INSTALL PGADMIN in your computer)
→ Create a password for PGADMIN (you have to enter the password everytime, when you entering into the server)
→ MY PGADMIN PASSWORD : kishandev
→
→ In PGADMIN, click on "ADD NEW SERVER" :

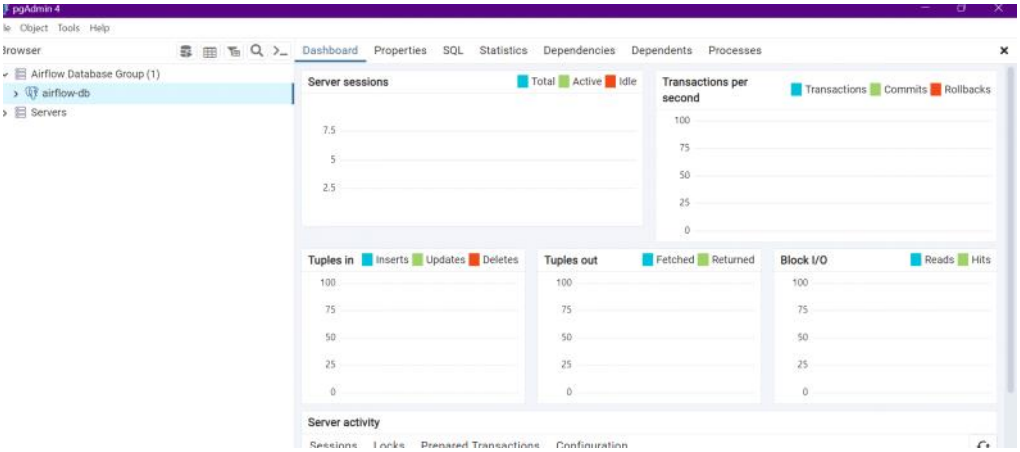
→
→ AF
→ Give "HOSTNAME", "USERNAME" and "PASSWORD" which we provided in the AWS database



→
HOST NAME : airflow-db.cyn1tccpnjs5.us-east-1.rds.amazonaws.com
USERNAME : admin123
Password : admin123

→ Click on save
→ It will connect to your database :

→



→ These are the steps to create and connect to DATABASE

STANDARD SETUP OF JENKINS

Monday, June 19, 2023 12:04 PM

JENKINS :

Jenkins is a Java-based open-source automation platform with plugins designed for continuous integration. It is used to continually create and test software projects, making it easier for developers and Devops Engineers to integrate changes to the project and for consumers to get a new build. It also enables you to release your software continuously by interacting with various testing and deployment methods

From <https://www.spiceworks.com/tech/devops/articles/what-is-jenkins/>

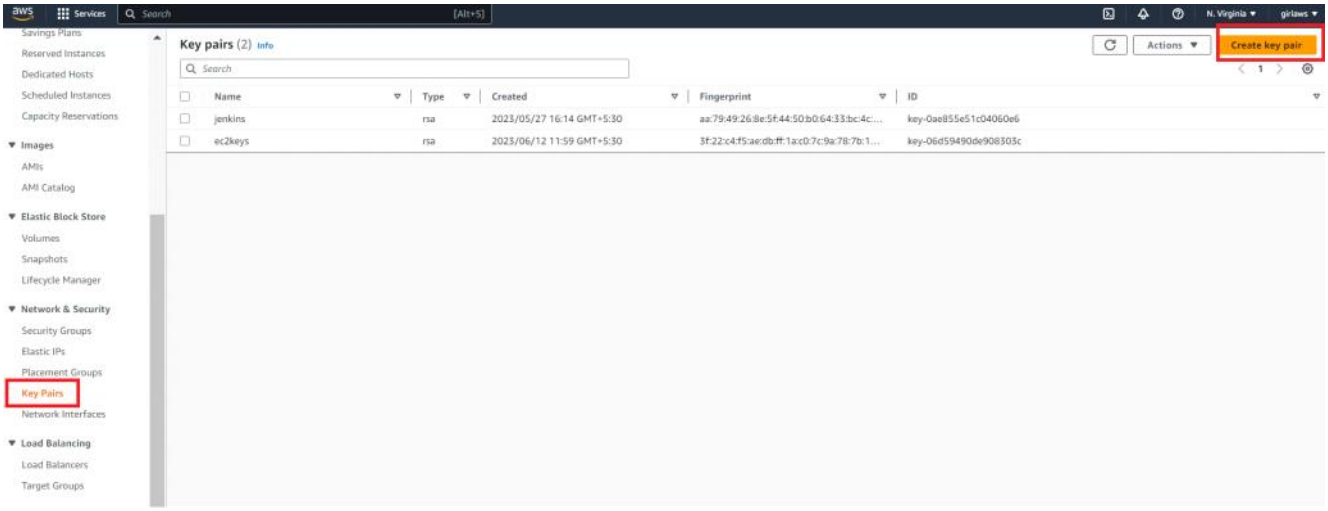
1. This is a Continuous Integration/ Continuous Delivery (CI/CD) tool.
2. Deploying code into production
3. Enabling task automation
4. Reducing the time it takes to review a code
5. Driving continuous integration

HOW TO SETUP A JENKINS SERVER IN EC2 INSTANCE

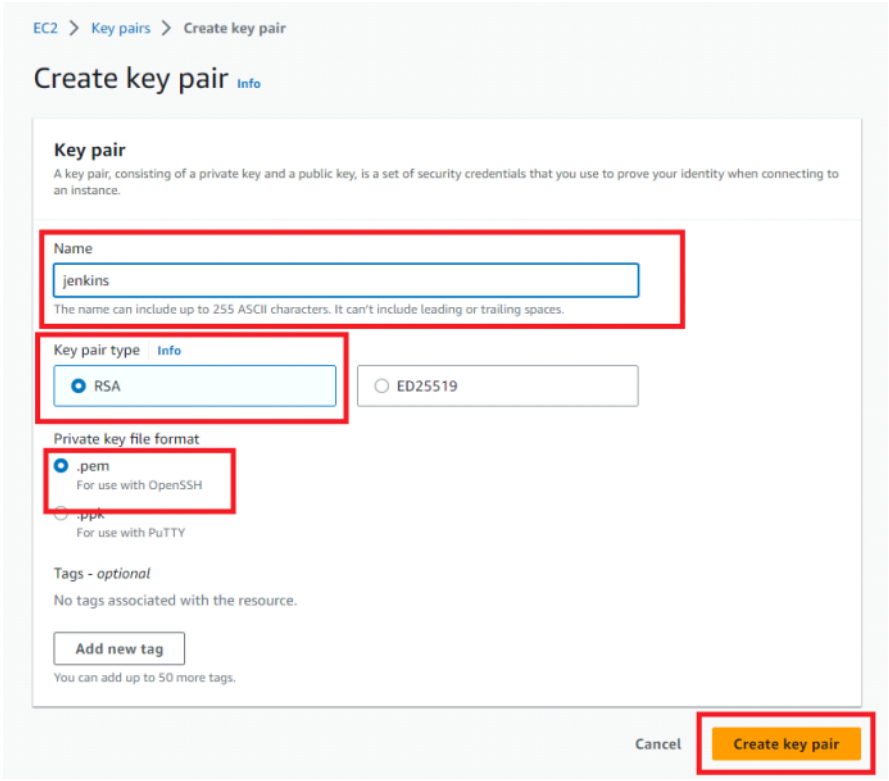
→ REQUIREMENTS :
- EC2 INSTANCE
- SECURITY GROUP
- KEY-PAIR
- JAVA 11
- JENKINS INSTALLATION SCRIPT

→ **CREATING THE KEY-PAIR (MANUALLY):**

→ GO to NETWORK AND SECURITY > KEY PAIRS > Click on "CREATE KEY PAIR"

→ 

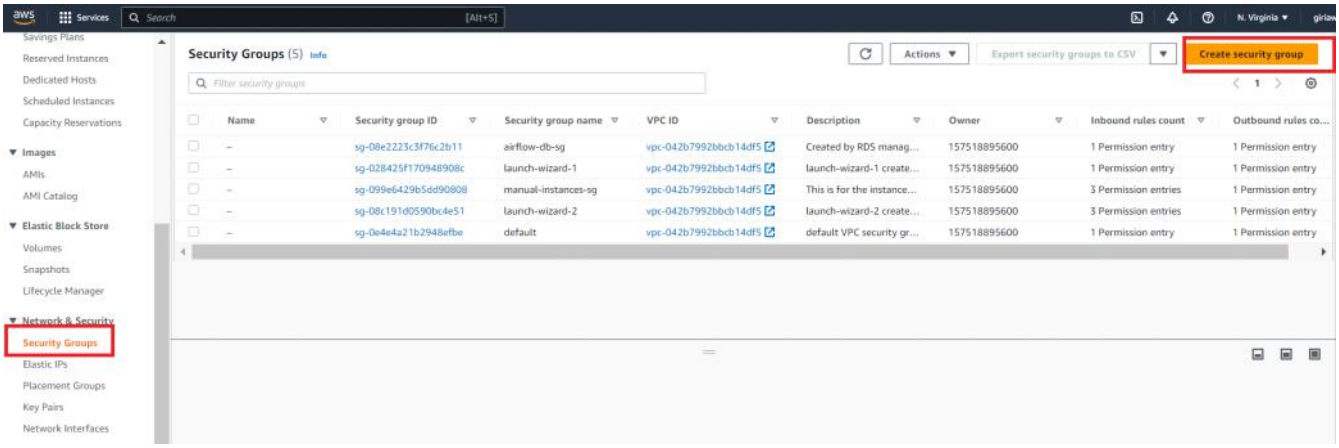
→ PROVIDE THE FOLLOWING DETAILS :
- NAME : jenkins
- KEY PAIR TYPE : RSA
- PRIVATE KEY FORMAT : .PEM

→ 

→ Click on CREATE KEY PAIR. It will create a key-pair and it will download a .pem file to your local machine. Based on the .pem file you can able to SSH into the VM

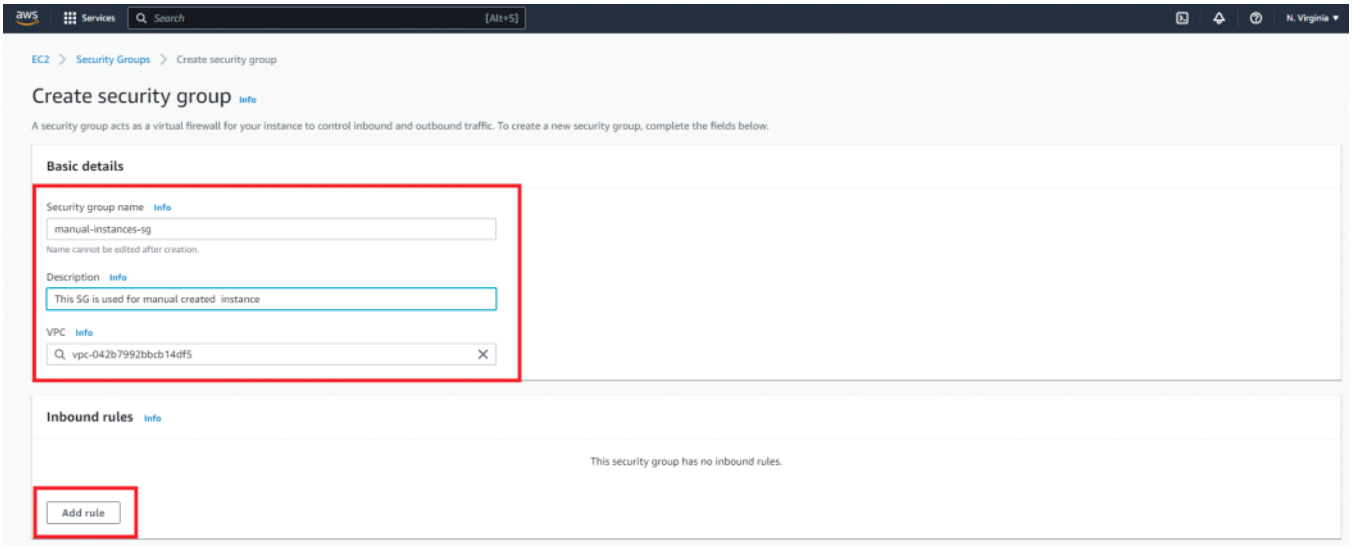
→ **CREATING THE SECURITY GROUP (MANUALLY):**

→ Go to NETWORK AND SECURITY > CREATE SECURITY GROUP

→ 

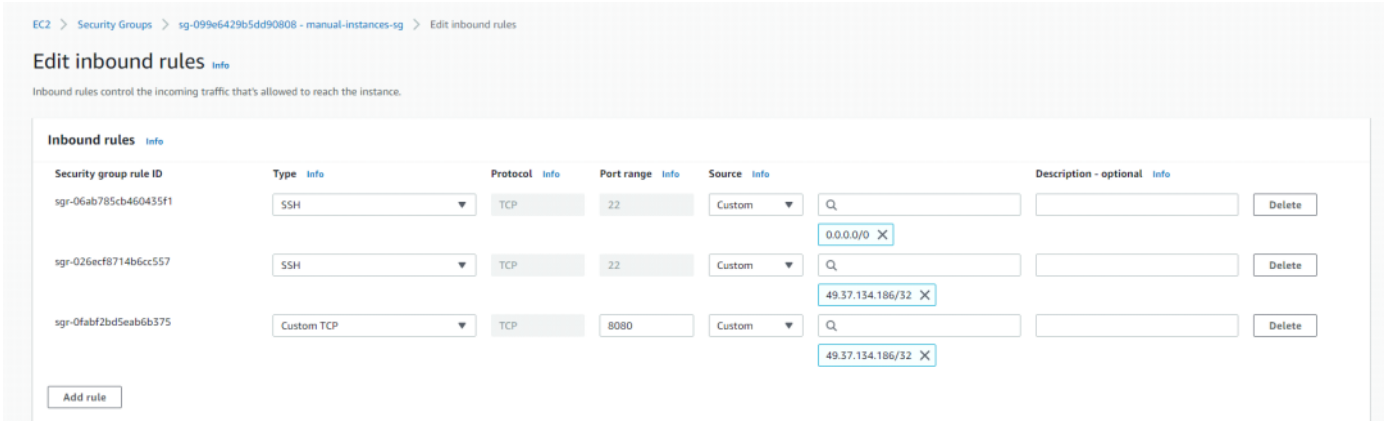
→ PROVIDE THE DETAILS :

- SECURITY GROUP NAME : manual-instances-sg
- DESCRIPTION : This SG is used for manual created instance
- VPC : <KEEP DEFAULT VPC ONLY>

→ 

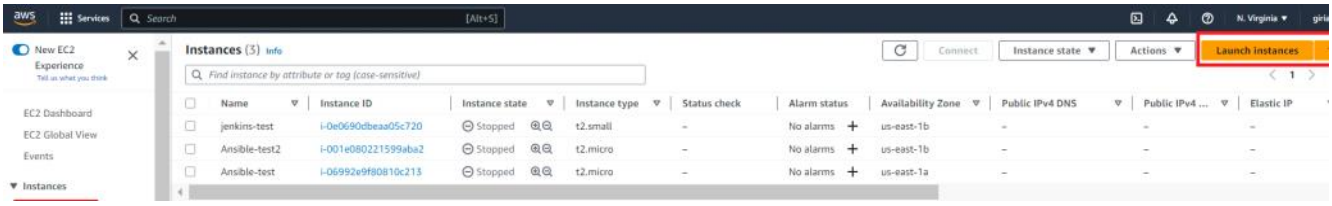
→ ADD THE BELOW RULES :

- SSH-PORTRANGE:22-ANYWHERE
- SSH-PORTRANGE:22-MYIP
- CUSTOMTCP(OR)-PORTRANGE:8080-MYIP

→ 

→ Click on "CREATE SECURITY GROUP". It will create the Security Group (FIREWALL)

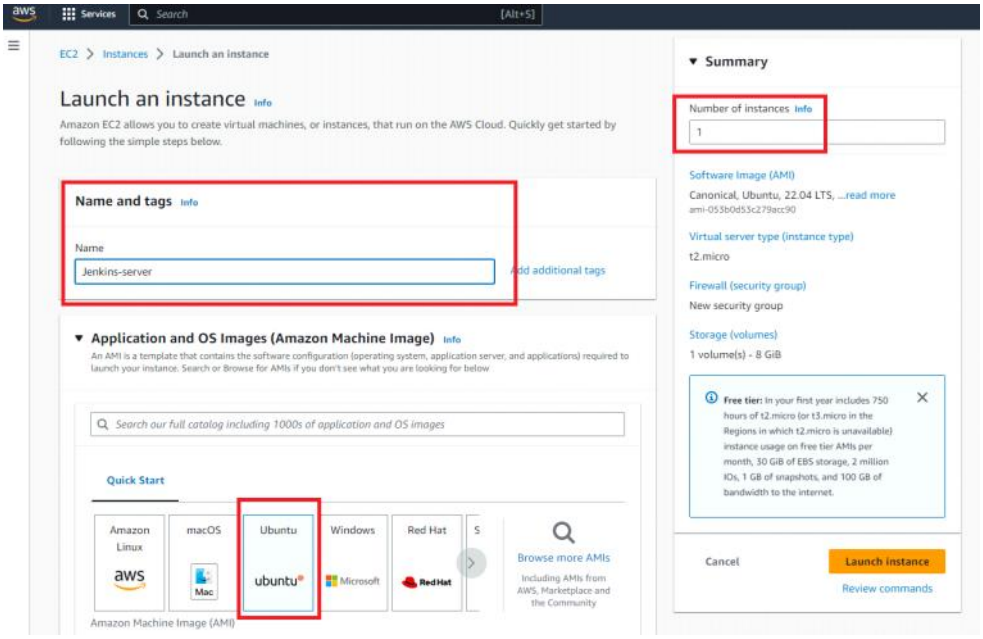
→ **CREATING AN EC2 INSTANCE (MANUALLY) :**
→ First, Login to AWS CONSOLE
→ Go to EC2 SECTION

→ 

→ In Instances -> CLICK on create in LAUNCH INSTANCE
→ Provide the below parameters :

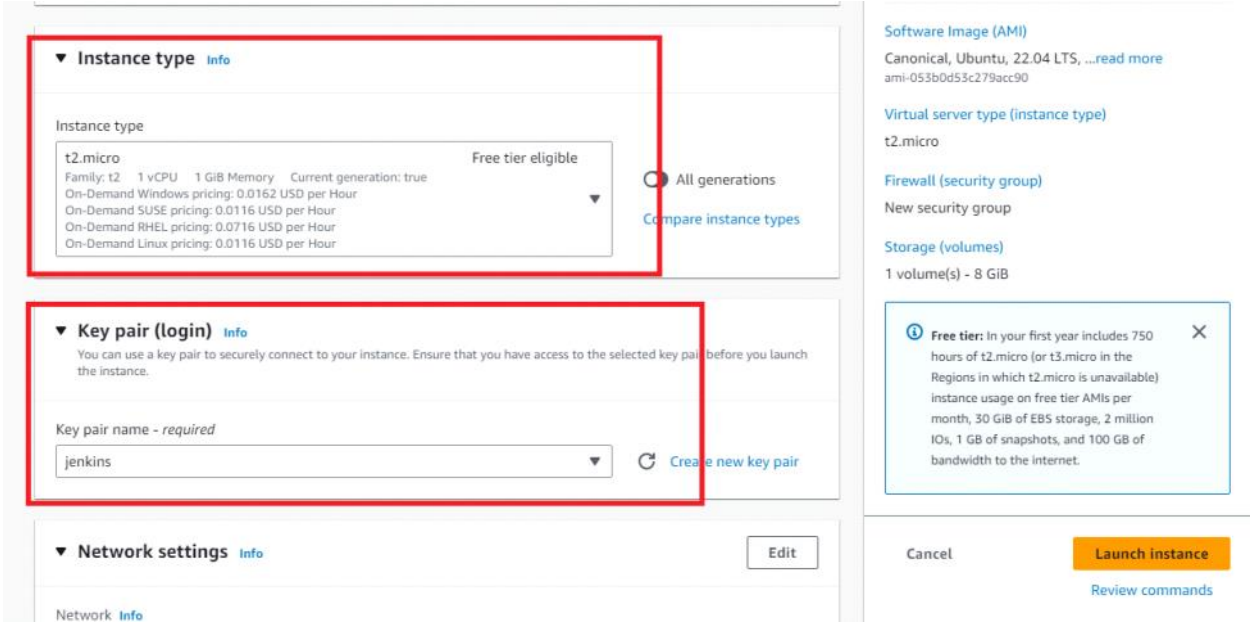
- NAME : Jenkins-server
- Image : UBUNTU
- Number of instances : 1

→



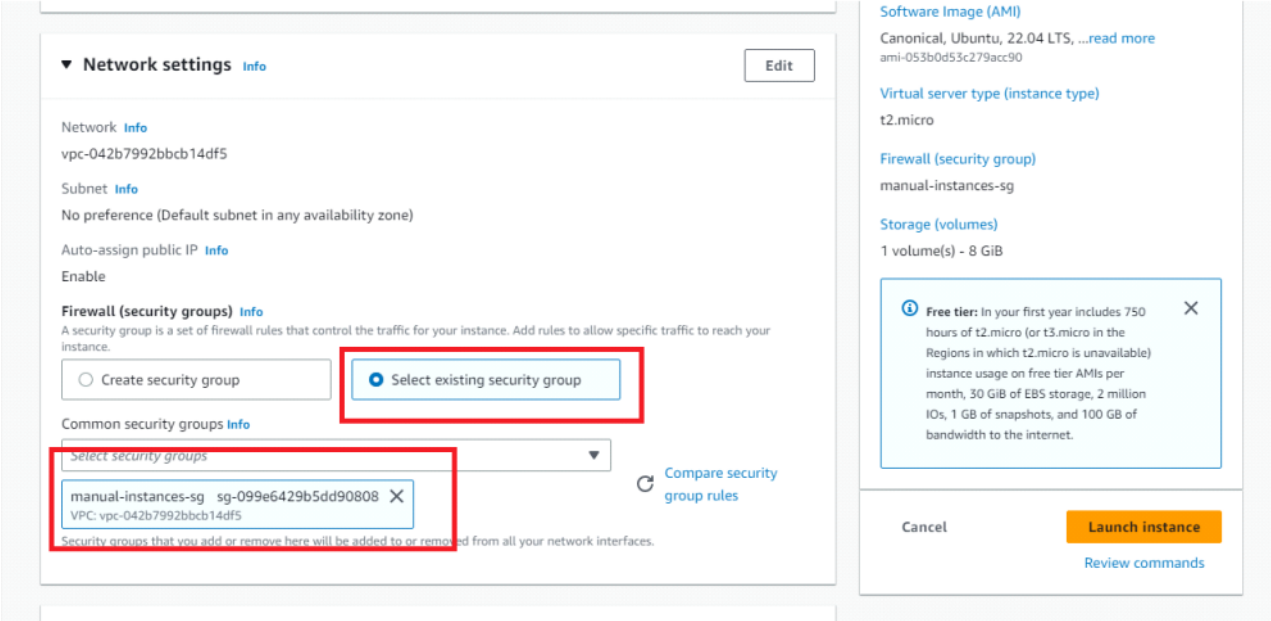
- INSTANCE TYPE : t2.micro
- NOTE: JENKINS HAS MINUMUM REQUIREMENT OF HAVING INSTANCE TYPE : t2.small
- KEYPAIR : jenkins

→



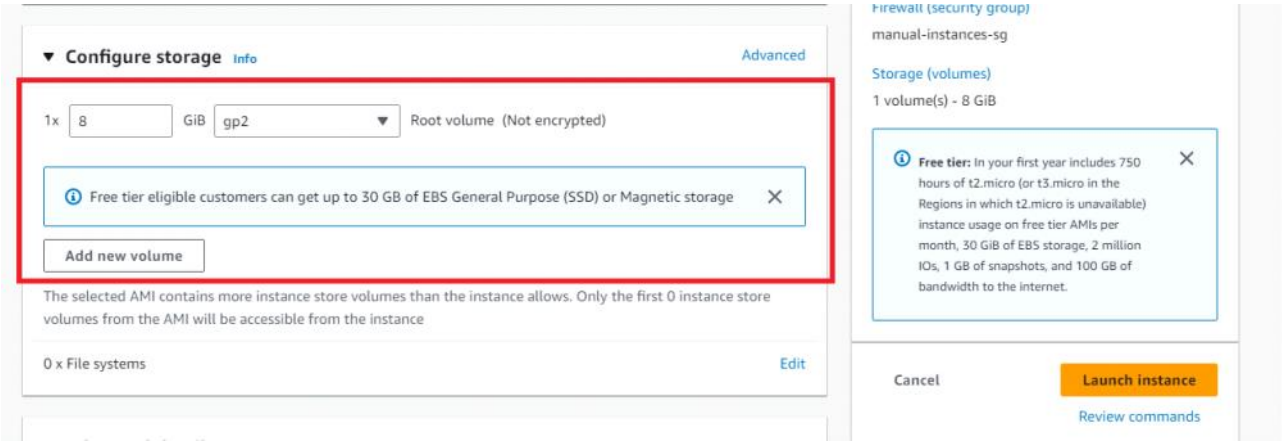
- SECURITY GROUP : (FIREWALL) - SELECT EXISITNG SECURITY GROUP
- CHOOSE : "manual-instances-sg" (we created earlier)

→



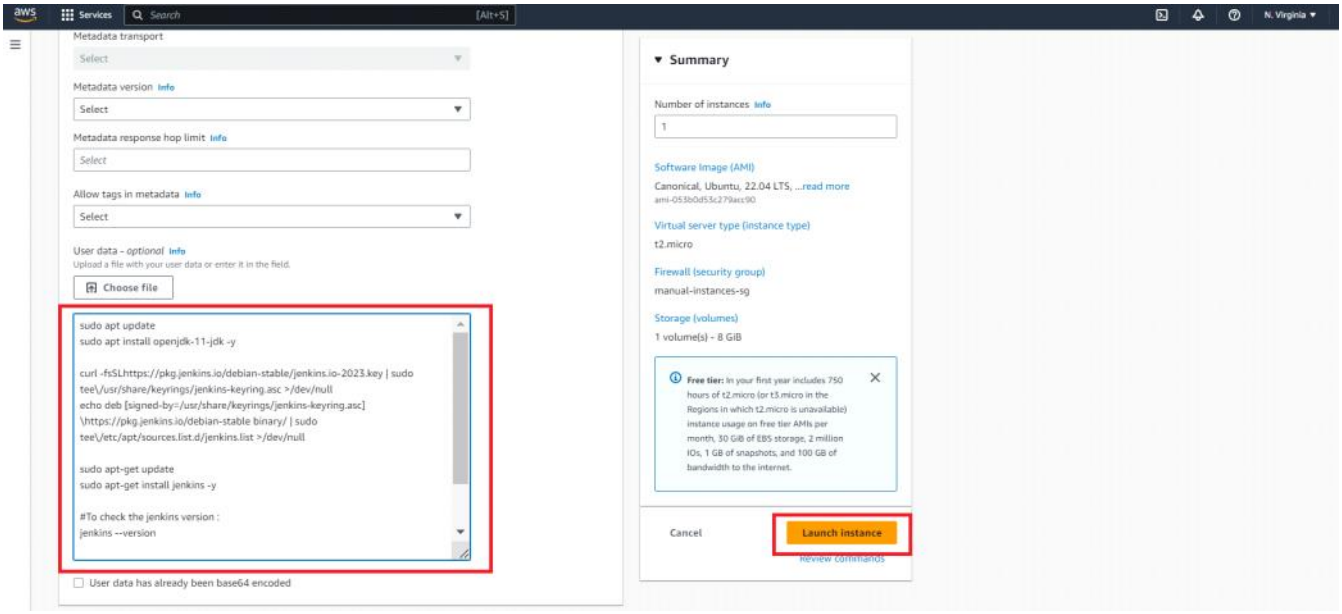
- The STORAGE : 8GB

→



→ CONFIGURING THE JENKINS SETUP INSIDE THE EC2 VM (MANUALLY)
→ There are 2 ways to configure the script :
- THROUGH USER DATA
- VIA SSH INTO VM AND SETTING UP THE JENKINS INSTALLATION VIA COMMANDS

→ **METHOD : 1 - THROUGH USER DATA**
→ In USER DATA :
→ ADD THE INSTALLATION SCRIPT and click on "LAUNCH INSTANCE"

→ 

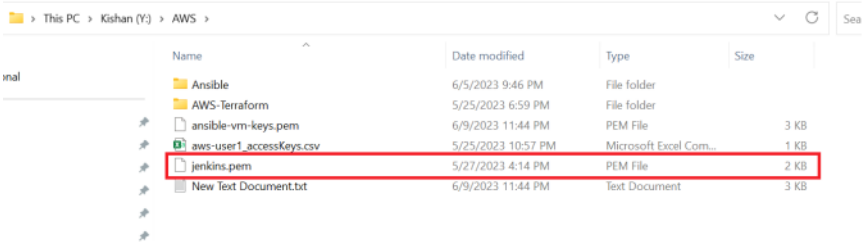
→ Once, the instance got created, the script in the user data will automatically run the instance

→ **METHOD: 2 - VIA SSH INTO VM AND SETTING UP THE JENKINS INSTALLATION VIA COMMANDS**

→ Login to the VM via SSH

→ Copy the VM "PUBLIC IP" : 3.80.89.180

→ Go to the path where your jenkins.pem file located

→ 

→ Open GIT BASH in this path and run the below command

→ *$ ssh -i jenkins.pem ubuntu@3.80.89.180*

→ 

→ Click on ENTER, it will as for fingerprint: <TYPE : YES>

→ Now, you'll able to login to the VM

→ **NOTE: IF YOU UNABLE TO LOGIN TO THE VM, THEN CHECK THE SECURITY GROUP RULES. MOSTLY YOUR LOCAL IP CHANGES EVERYTIME. SO, SET IT AGAIN.**

→ Now, Run the commands one after one with (#ROOT USER - *$sudo su*)and it will install the jenkins in your VM

→ **JENKINS SCRIPT :**

```
#THIS COMMAND IS FOR REGULAR UPDATES FOR UBUNTU
sudo apt update

#JENKINS REQUIRED THE JAVA-11 VERSION. THIS COMMAND WILL INSTALL JAVA-11
```

```
(JDK-11)
sudo apt install openjdk-11-jdk -y


#THIS WILL SET UP THE JENKINS CONFIGURATIONS INSIDE THE VM
curl -fsSLhttps://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


#AGAIN RUNNING THIS COMMAND FOR REGULAR UPDATES
sudo apt-get update

#THIS WILL INSTALL JENKINS IN YOUR VM
sudo apt-get install jenkins -y


#To check the jenkins version :
jenkins --version

#THESE COMMANDS WILL ENABLE THE JENKINS SERVER
systemctl start jenkins
systemctl enable jenkins
systemctl status jenkins
```

→ JENKINS will run in the port: 8080
Now, copy the Public IP and search http://<public-ip>:8080
http://44.204.172.179:8080/
http://44.204.172.179:8080/

→ Start setting up the jenkins with standard default plugins only

→ Create the user and password for it

- 

➢ REFRENCE LINKS :
https://www.jenkins.io/doc/book/installing/linux/
https://www.jenkins.io/doc/book/installing/linux/#debianubuntu
How to Setup & Install Jenkins in AWS EC2 Ubuntu Instance | DevOps

JENKINS INTEGRATING WITH GIT

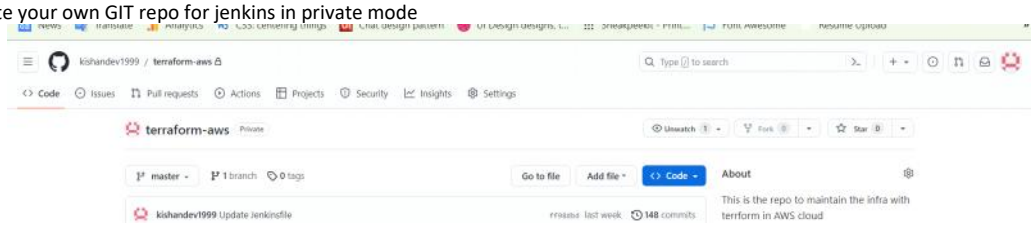
Sunday, June 25, 2023 5:23 PM

AUTHENTICATING GIT TO JENKINS WITH SSH KEYS :

→ REQUIREMENTS :

- GIT
- GIT REPO (PRIVATE MODE)
- SSH KEYS
- PERSONAL ACCESS TOKEN (FROM GIT)
- JENKINS SERVER

→ Create your own GIT repo for jenkins in private mode

- 

→ GENERATING SSH KEYS :

→ Open git bash in your local computer machine (not in VM) and generate the ssh keys by using below command

→ *$ssh-keygen*

→ If the keys are already exists, then it will ask to overwrite the file or else, it will create the new key pairs

- ```
asus@KishanDev MINGW64 /y/AWS
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/y/Devops Udemy/ssh-keys-folder/.ssh/id_rsa
):
/y/Devops Udemy/ssh-keys-folder/.ssh/id_rsa already exists.
```

→ Go to the path and check your keys

- ```
asus@KishanDev MINGW64 ~/.ssh
$ ls
id_rsa  id_rsa.pub  known_hosts  known_hosts.old
```

→ You'll find the keys : "id_rsa", "id_rsa.pub", "known_hosts"

→ Id_rsa.pub : This is the public key. Very short key

- ```
$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCiugC9ld3DJRt6Fyo+9gs4+dp+7Ex8pZSz7JdiOS
rLrxBbSUlWFgitRvzADQRwtURZtHnKLDbdBt4ewd6ITnkZQVwPVMGEj6FcSAO86AThWxnSCKOPMkmp
I4vmjGtJOfOK9tNL6DFdpbeoE2D5ViDcNhgAv7O5ha2rrEogNIq+PahYVgOdi5CnILk1MIwFPiYnbd
GqzcSp41yRU4IpJegix7goySh7GqIkv9XGqTXz94nKUkXoTcUnrh4djdkso7bc9E1YtzAZvgjRNArB
cMu2P3b8MEBVrangYszUxGp8C2Z5jSD3nANdwKt3oTM+kzqjE3f4WOGo8CII6he+AEqWZASuZMpkVd
3Gl62ThIHRChmUC6CLPSM466PEPAwlbEQdopYXfFeG+OFw1NgJRqRofT2EHz/PbxRNTpYR8jIJ5oR7
XuwnGiH86to+khpgB/YgIwbYugaNWBDxSB9g9dgkFBiQKOSzW+OVgcpieA7yAkDaIsucCuk= asus@
shanDev
```

→ Id_rsa : This is private key. Very long key
```

- 

→ Known-hosts will store the hosting ip's and login data

POINTS TO REMEMBER :

1. Why does the public and private key change for a user in the same machine?
2. In continuation of that question, why can't I use my old public and private key to access a repository when nothing else has changed?
3. Will the keys change if I create them again using ssh-keygen -t rsa?

ANSWERS :

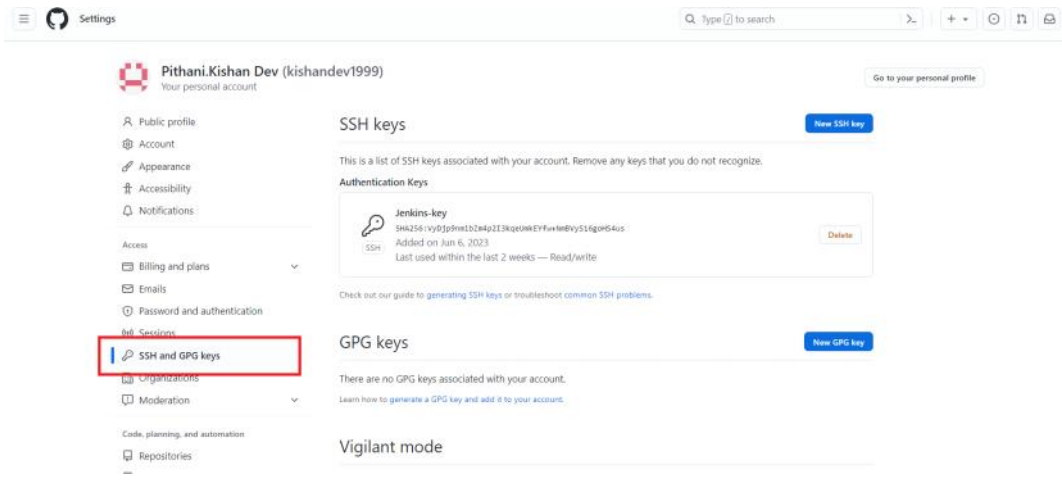
1. A key pair is not tied to you as a user or your machine. Imagine having the keys to your house stolen. The keys aren't bound to you, anyone can use them to unlock your doors. In the exact same way, if I get my hands on your private key, I can use it to make changes to your Github repositories.
2. If you wish to access your repositories using both your new keys and your old ones, you need to make sure the *public* keys for *both* key pairs are uploaded to Github. If Github only has your new key, it cannot authenticate you if you use your old key pair.
3. Yes, the keys will change. That's sort of the point. Imagine buying a new lock for your door. If your old keys are the same as your new ones, it won't do you any good, esp. if you were changing the lock because someone stole your old key. If you were to have one of your ssh keys compromised, you could simply generate new keys & upload the corresponding public keys to Github after deleting the old ones. You can also have multiple key pairs. Generate as many as you wish and use them for different purposes. This is similar to having multiple keys for multiple locks in your house.
   Also NEVER EVER give away your private keys. The private key is what is used to authenticate you as a person. If I get my hands on your private key, I can fake being you. Github can't tell the difference. No one but you needs to know what your private key is. If you have uploaded your private key somewhere, I recommend deleting it, generating a new pair & uploading the new public key just to be safe.

From <https://unix.stackexchange.com/questions/121311/does-public-and-private-key-remain-the-same-for-a-user-or-system>

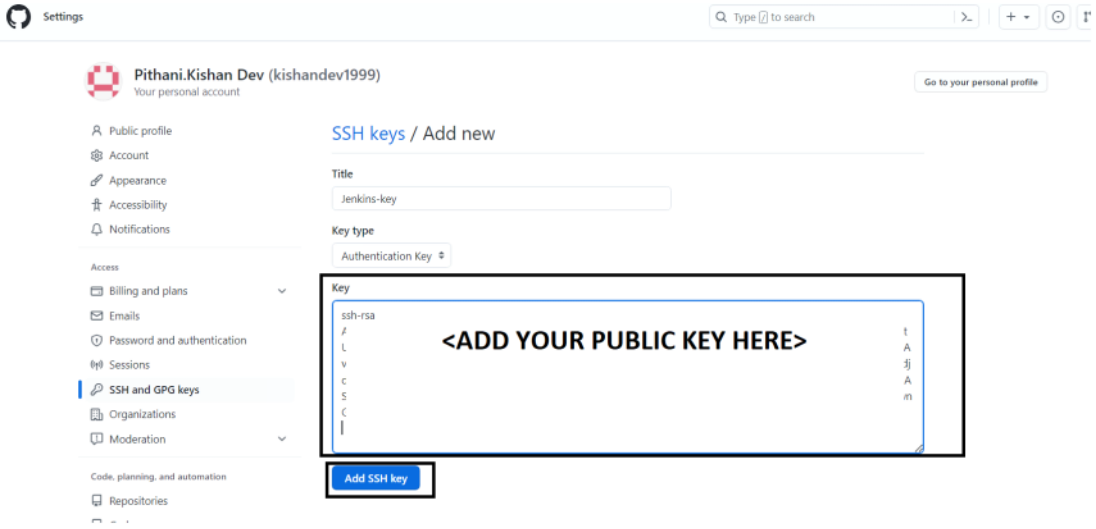
→ ADDING THEE LOCAL MACHINE KEYS TO GIT REPO :

→ Now go to git hub > settings > SSH AND GPG KEYS >

→ Click on the NEWSSHKEY and add your "PUBLIC KEY" over there :

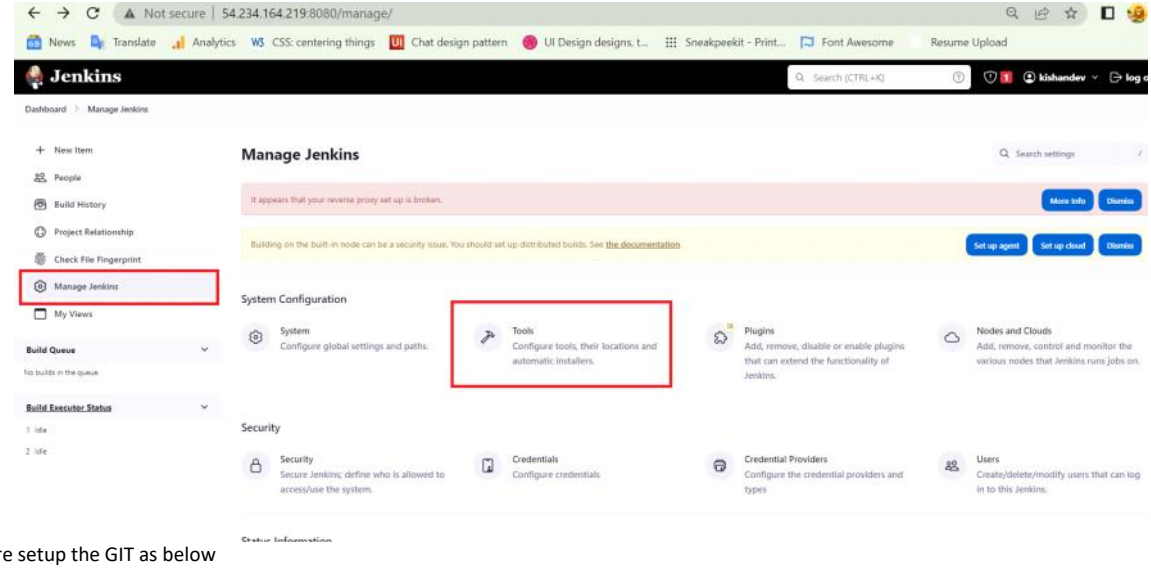


→ Give the TITLE
→ KEY TYPE : <AUTHENTICATION KEY>
→ KEY : <ADD YOUR PUBLIC KEY> id_rsa.pub (short key)
→ Click on ADD SSH KEY

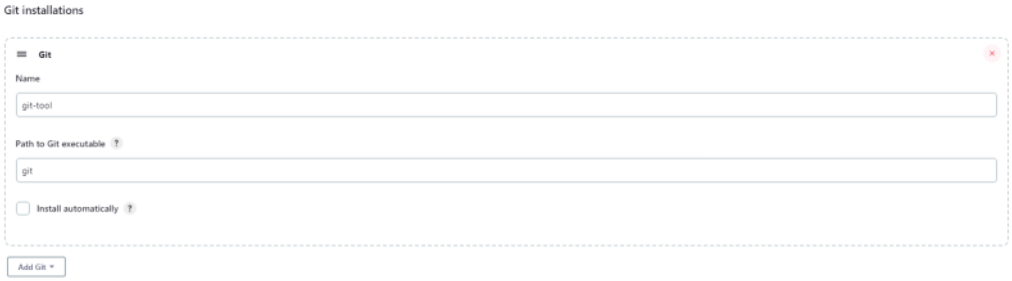


→ GIT INSTALLATION IN JENKINS SERVER VIA JENKINS CONSOLE :

→ Login to jenkins with created username and password
→ Firstly, we need to seed install the GIT in Jenkins
→ Go to DASHBOARD > MANAGE JENKINS > TOOLS

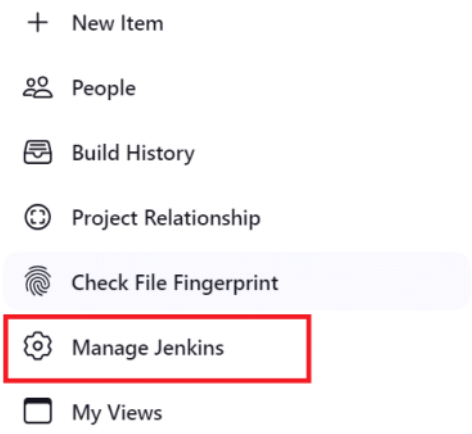


→ There setup the GIT as below
→ This will install GIT automatically into your server

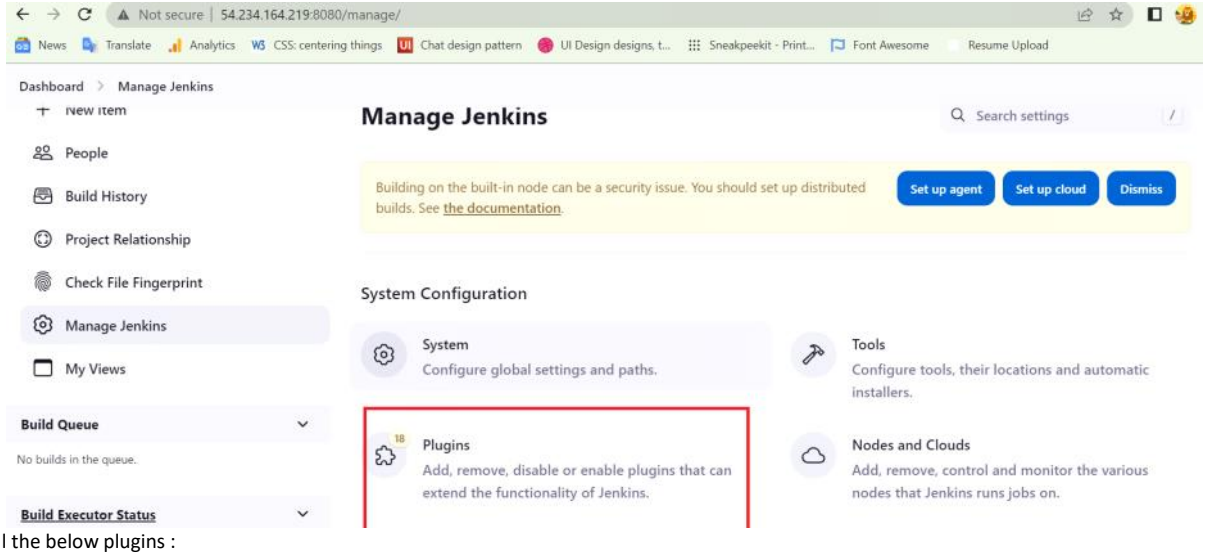
- 

→ PLUGINS :

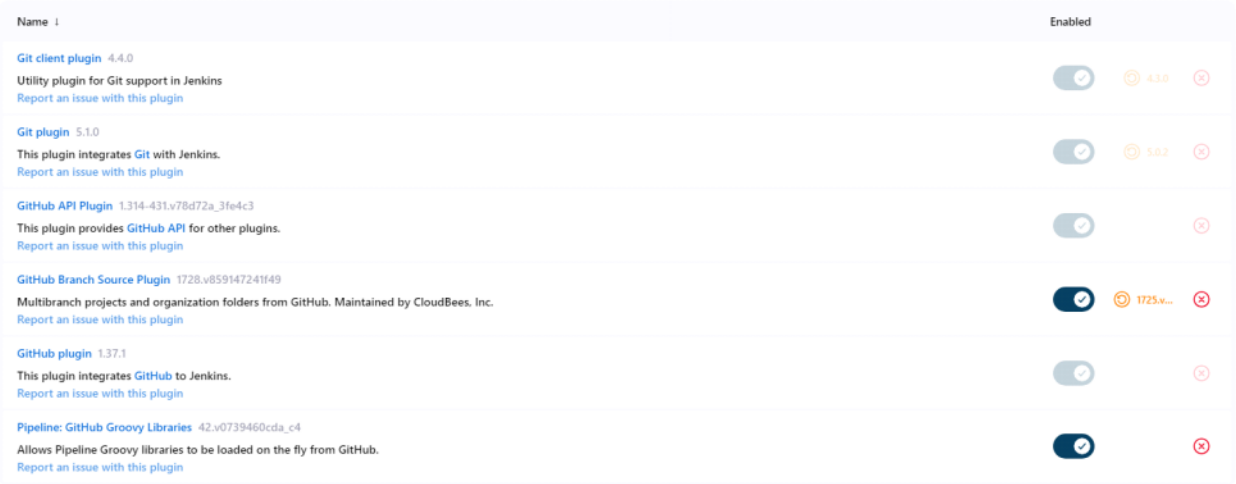
→ Go to Jenkins console
→ We need to install the required plugins in GIT for JENKINS
→ In Jenkins DASHBOARD > Go to manage JENKINS

- 

→ Go to PLUGINS

- 

→ Install the below plugins :

- 

Git client pluginVersion4.4.0 Utility plugin for Git support in Jenkins Report an issue with this plugin	4.3.0
Git pluginVersion5.1.0 This plugin integrates Git with Jenkins. Report an issue with this plugin	5.0.2
GitHub API PluginVersion1.314-431.v78d72a_3fe4c3 This plugin provides GitHub API for other plugins. Report an issue with this plugin	
GitHub Branch Source PluginVersion1728.v859147241f49 Multibranch projects and organization folders from GitHub. Maintained by CloudBees, Inc. Report an issue with this plugin	1725.vd391eef681a_e
GitHub pluginVersion1.37.1 This plugin integrates GitHub to Jenkins. Report an issue with this plugin	
Pipeline: GitHub Groovy LibrariesVersion42.v0739460cda_c4 Allows Pipeline Groovy libraries to be loaded on the fly from GitHub. Report an issue with this plugin	

→ CREDENTIALS SETUP :

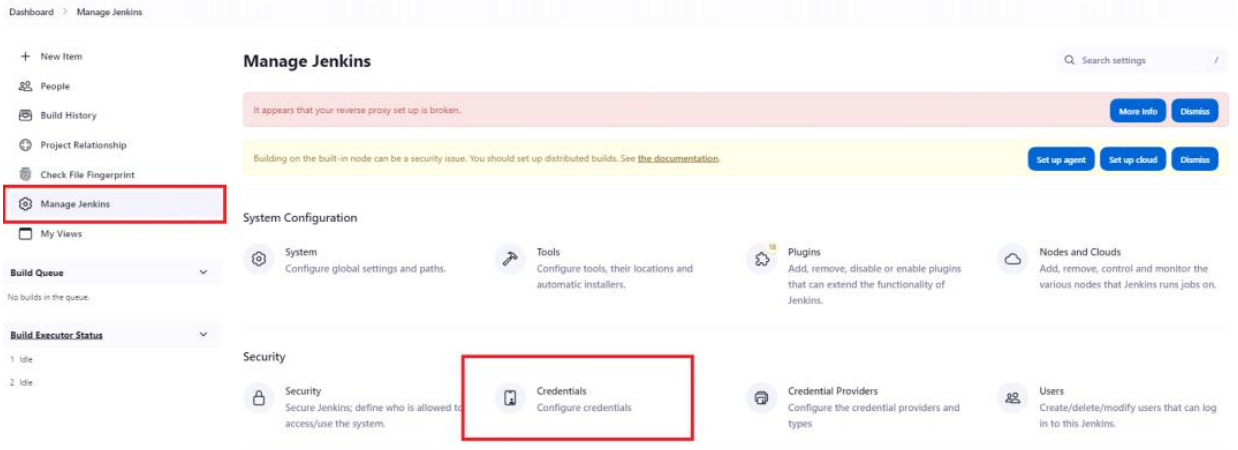
→ Now, we need to setup the CREDENTIALS for this GIT

- For GIT, There are 2 ways to setup the credentials
  1. VIA SSH KEYS (SSH)
  2. VIA PERSONAL ACCESS TOKEN (HTTP)

→ We will create the CREDENTIALS for both

1. VIA SSH KEYS (SSH)

→ Go to DASHBOARD > MANAGE JENKINS > CREDENTIALS



→

→ Click on ADD CREDENTIALS

- 

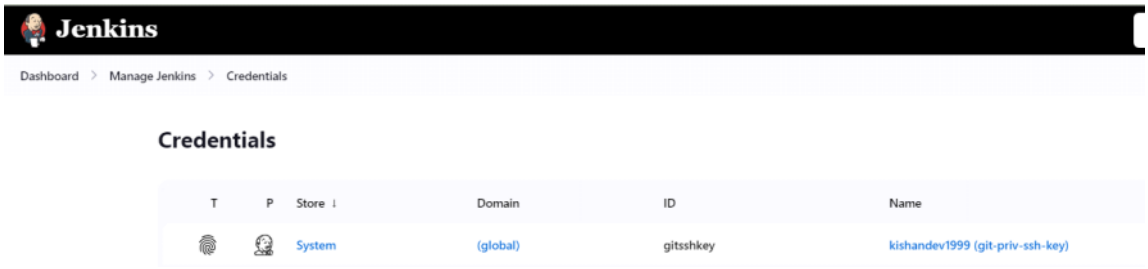
→ Add the CREDENTIALS as below

- SCOPE : GLOBAL

- ID : gitsshkey (provide a ID by your own)
- Description : git-key-priv
- Username : kishandev1999 (YOUR GIT USER NAME)

→ In PRIVATE KEY > Add your private key of your local machine :

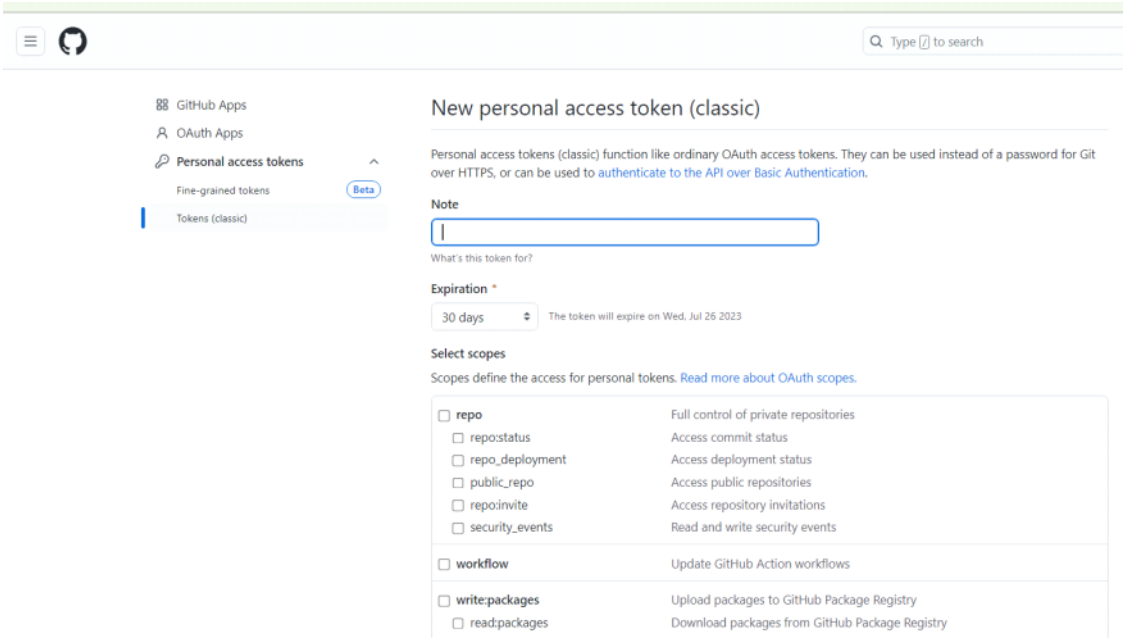
→ Leave the passphrase as empty
→ Click on "CREATE"
→ The credentials will be created as below screenshot



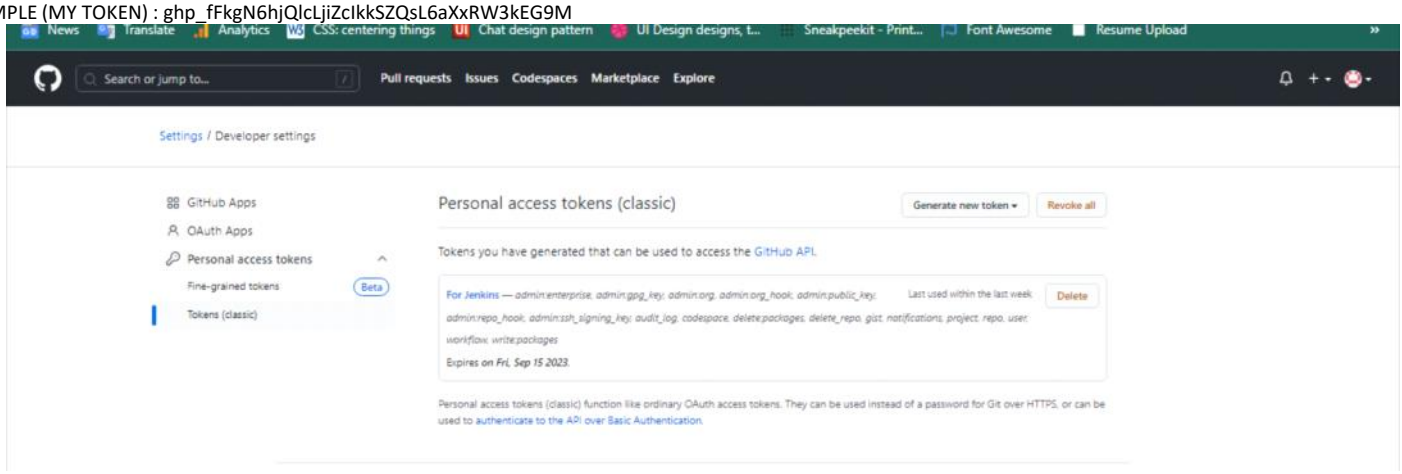
→ Through this, we will setup the credentials with SSH KEYS

2. VIA PERSONAL ACCESS TOKEN (HTTP)

→ Go to GIT > SETTINGS > DEVELOPER SETTINGS > PERSONAL ACCESS TOKEN (TOKENS- CLASSIC)
→ Provide a NAME for your token and select the expire period for this token (30 days or 90 days)
→ Select the required permission for this token to provide the access in git (I selected all)

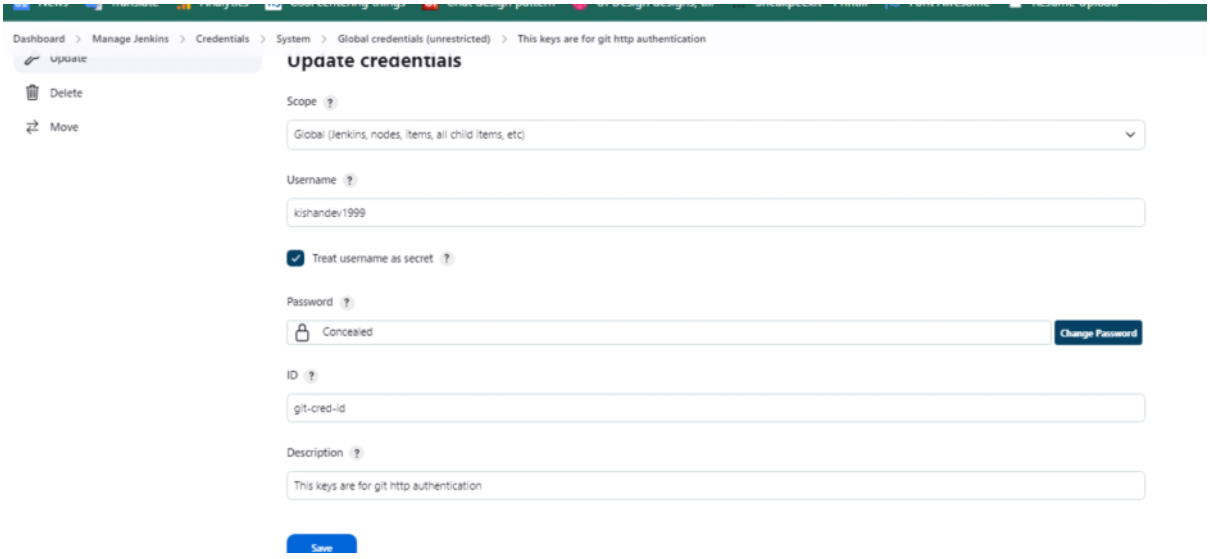


→ Click on "GENERATE TOKEN"
→ It will generate a token, Copy the token Immediately
→ EXAMPLE (MY TOKEN) : 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

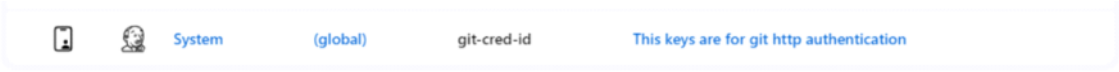


→ Go to DASHBOARD > MANAGE JENKINS > CREDENTIALS
→ Click on ADD CREDENTIALS
→ Add the CREDENTIALS as below :

- SCOPE : GLOBAL
- USERNAME : kishandev1999 (YOUR GIT USER NAME)
- Check (treat username as secret)
- PASSWORD : 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 (YOUR PERSONAL ACCESS TOKEN GENERATED FROM GIT)
- ID : git-cred-id (GIVE YOUR OWN ID)
- DESCRIPTION : THIS CREDENTIALS ARE FOR GIT HTTP AUTHENTICATIONS
- CLICK ON "SAVE"

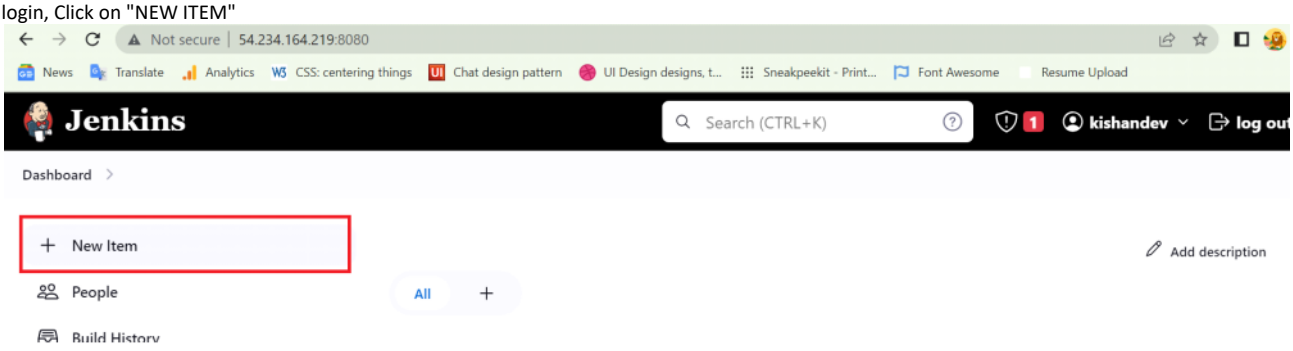


→ Your Credentials will create like this

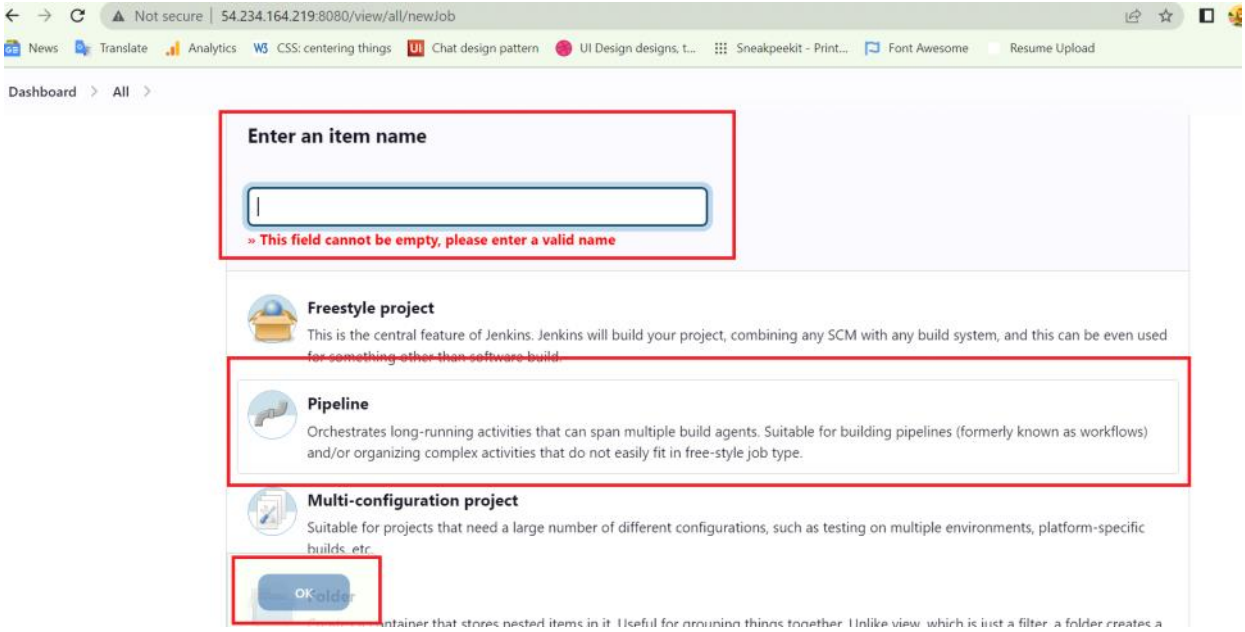
- 

→ CREATING THE JENKINS PIPELINE WITH GIT AS SCM :

→ After login, Click on "NEW ITEM"

- 

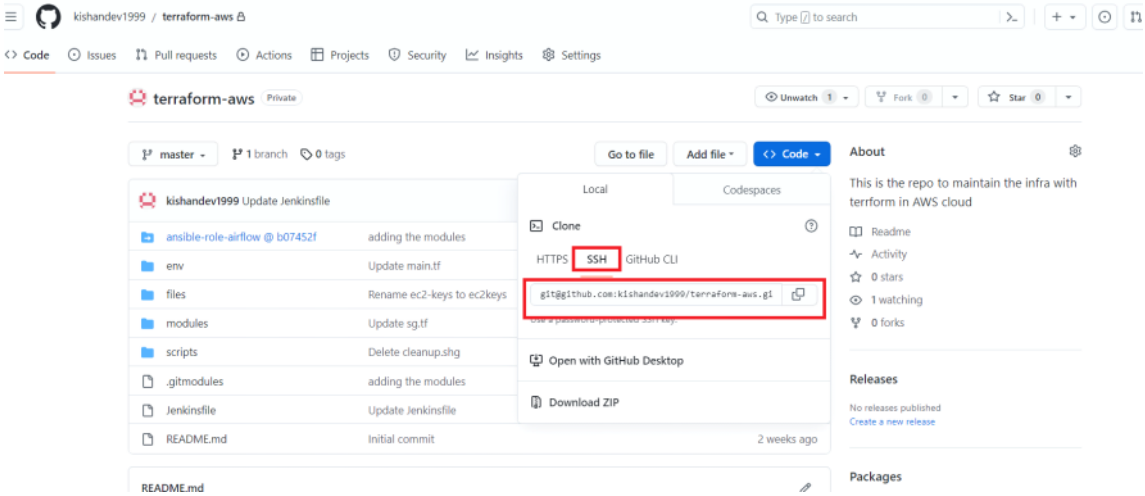
→ Now,

- 

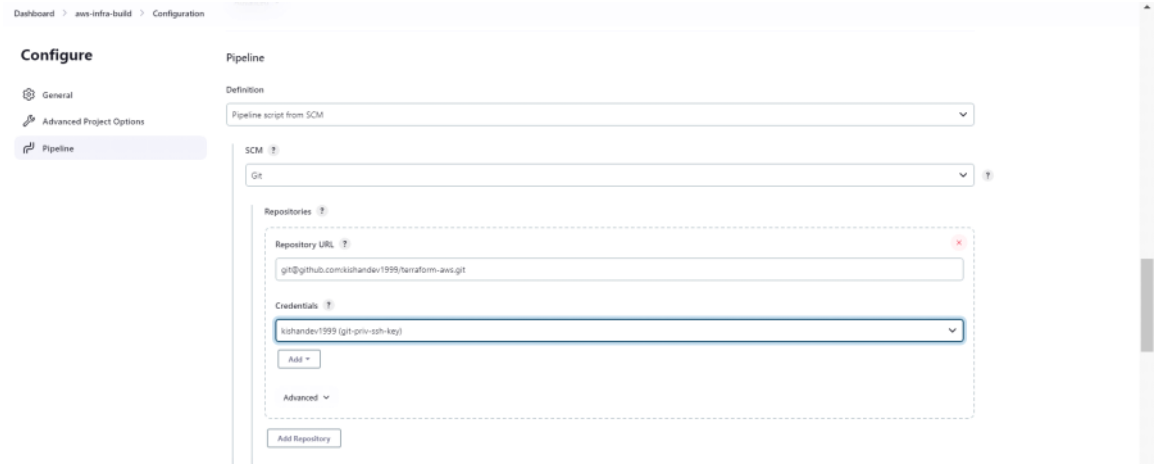
→ Now go to your pipeline > configure
→ Now, there are 2 ways to setting up the git SCM
→ As we created the 2 credentials will see the both ways of setting up the Jenkins with git

1. VIA SSH KEYS :

→ If you are choosing the SSH KEYS as authenticating credentials, then use this link from your GIT repo

- 

→ In the Jenkins > Go to your pipeline > configure
→ Provide the REPOSITORY URL and select the credentials which we created with private keys

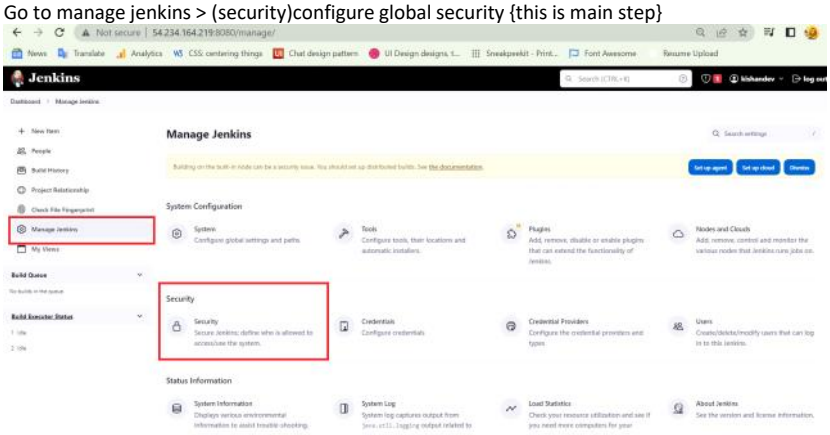
- 

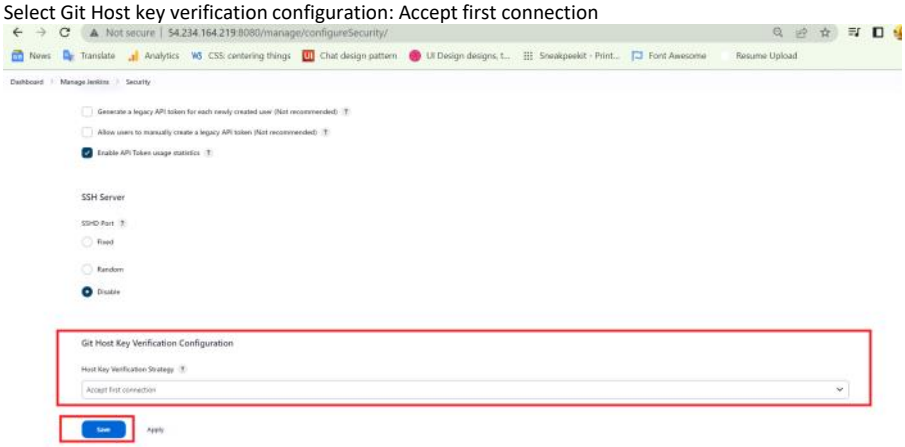
→ NOTE : YOU WILL GET AN ERROR LIKE THIS BELOW

- 

→ This is due to may be the credentials setup you did wrongly. Cross check once again

→ Or This would be the main reason : You need to set up the accept the first connection in jenkins

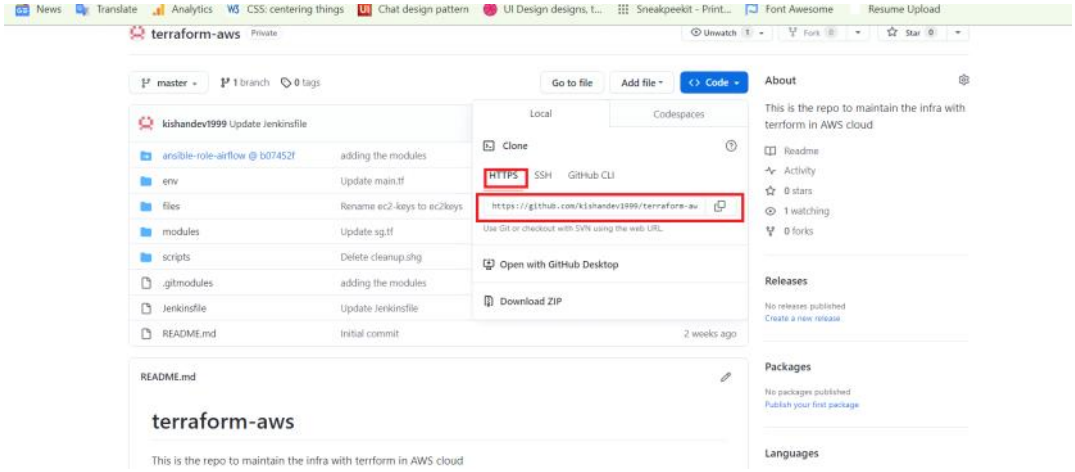
1. Go to manage jenkins > (security)configure global security {this is main step}
2. 

3. Select Git Host key verification configuration: Accept first connection
4. 

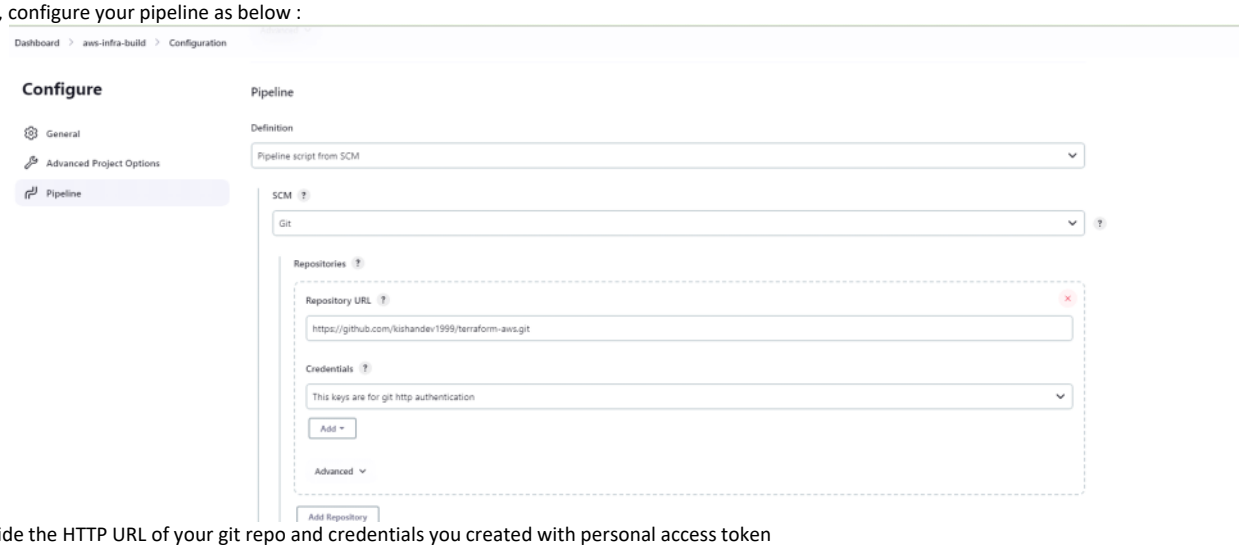
→ Now, that error will disappear automatically.

2. VIA PERSONAL ACCESS TOKEN :

→ Now, copy the HTTP URL of your git repo :



→ Now, configure your pipeline as below :



→ Provide the HTTP URL of your git repo and credentials you created with personal access token
→ Click on save
→ Now, your GIT is authenticated to Jenkins pipeline

→ Here, I explained 2 ways of authenticating the Git with jenkins. You can prefer your own way to authenticate. For this project, I preferred 2nd method of authenticating (VIA PERSONAL ACCESS TOKEN). Because, It helped me to fetch the submodules data which we are using in the main module. This will be cover up in submodules section

➢ REFERENCES :
https://www.guru99.com/jenkins-github-integration.html

Authenticating git to the jenkins
SSH integration between Jenkins & GitHub | Jenkins GitHub Integration using SSH Keys |Jenkins GitHub

Jenkins #5 | Integrate GitHub private Repository with Jenkins job using SSH Keys

How to Add Git Credentials in Jenkins

JENKINS INTEGRATING WITH AWS

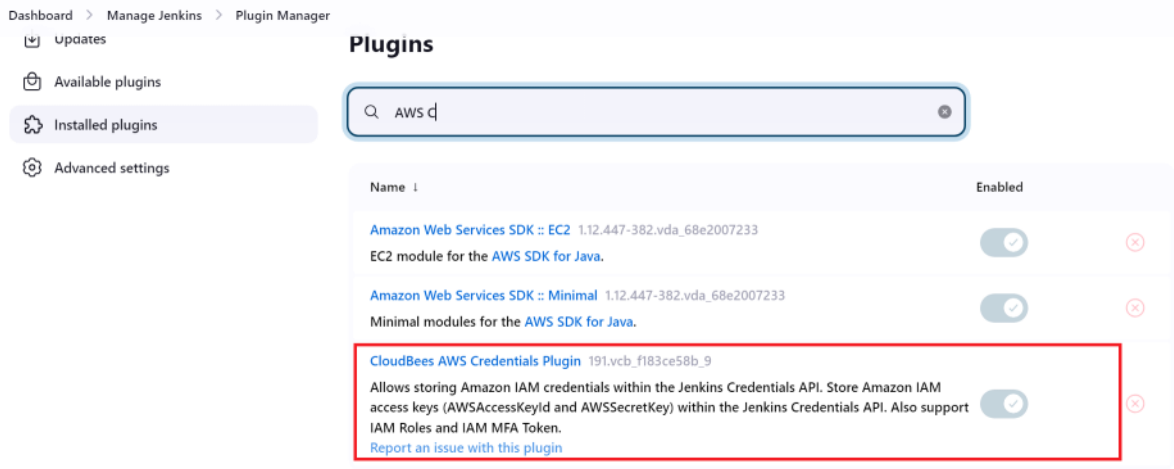
Monday, June 26, 2023 11:30 PM

INTEGRATING AWS WITH THE JENKINS :

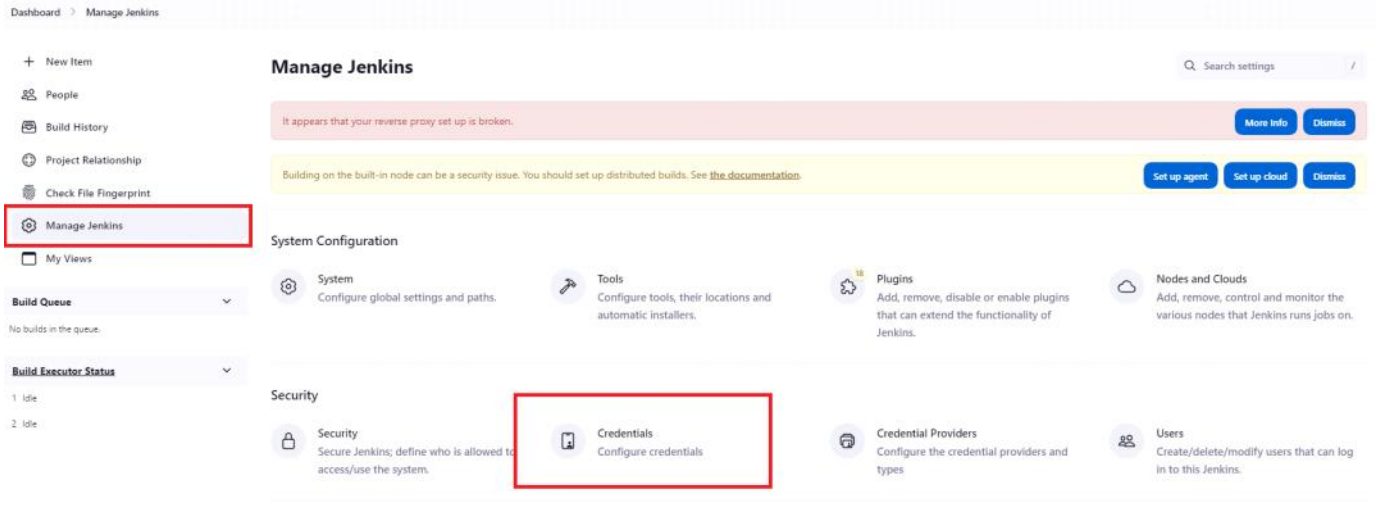
PLUGINS REQUIRED :

1. CLOUD BEE AWS CREDENTIALS
2. TERRAFORM

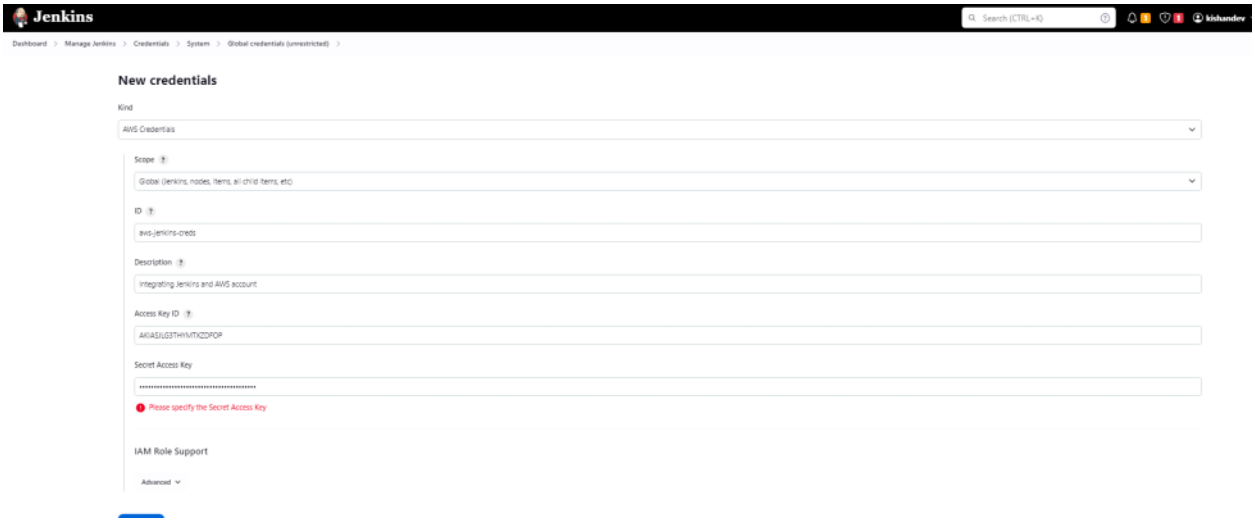
→ FIRST CREATE AN IAM ROLE AND GIVE THE ADMINSTRATIVE ACCESS TO IT
→ Check the "AWS CLI AUTHENTICATION" document on how to create the role and setup
→ Get the ACCESS KEY and SECRET KEY for it
→ Now go to jenkins -> settings -> Plugin manager -> Install "Cloudbees AWS credentials"



→ Now, we will setup the AWS credentials in jenkins
→ Go to MANAGE JENKINS > "credentials"



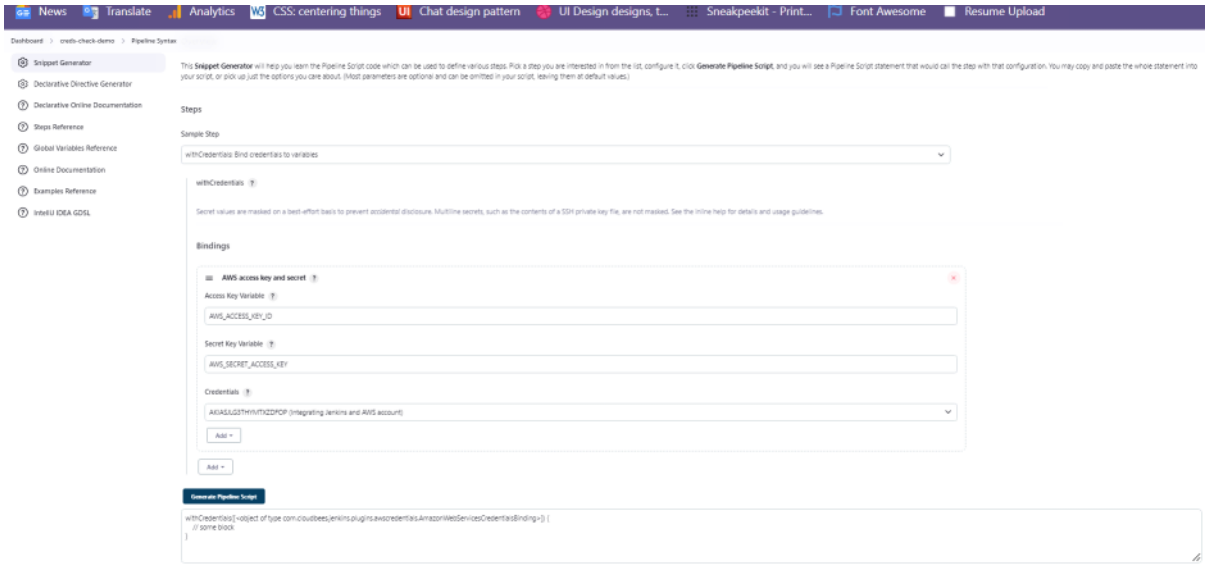
→ Setup the credentials as below
  - KIND : AWS CREDENTIALS
  - SCOPE : GLOBAL
  - ID : AWS-JENKINS-CREDS
  - DESCRIPTION : INTEGRATING JENKINS AWS ACCOUNT
  - ACCESS KEY ID : <YOUR ACCESS KEY>
  - SECRET KEY ID : <YOUR SECRET KEY>

→ 

→ Click on SAVE

→ Now, we will create a new pipeline

- So, click on new and go to the bottom, you'll fine pipleline syntax, click on it, you'll see this page.
- Fill with credentials and select your AWS credentials
- When you click on generate syntax. It will generate the syntax

→ 

→ You'll get the syntax like this :

```
withCredentials([<object of type com.cloudbees.jenkins.plugins.awscredentials.AmazonWebServicesCredentialsBinding>]) {
   // some block
}
In the <object of type, -- you need to add appropriate syntax value
```

→ The above syntax is not completely correct syntax

→ The proper syntax is mentioned below

```
pipeline{
   agent any
   stages {
   stage("AWS DEMO") {
      steps {
         withCredentials([[
            $class: 'AmazonWebServicesCredentialsBinding',
            credentialsId: 'aws-jenkins-creds',
            ACCESS_KEY: 'AWS_ACCESS_KEY_ID',
            SECRET_KEY: 'AWS_SECRET_ACCESS_KEY'
            ]]) {

         sh "aws s3 ls"
```

```
        sh "aws ec2 describe-instances --region=us-east-1"
          }
    }
    }
}
}
```

→ Before, running the pipeline. We need to install the AWS CLI in the jenkins VM
→ So, SSH into the VM and install the AWS CLI
→ *$apt install awscli*
→ Check the below screenshots for reference :

```
Last login: Mon May 29 09:11:04 2023 from 49.37.132.20
ubuntu@ip-172-31-82-1:~$ sudo su
root@ip-172-31-82-1:/home/ubuntu# aws s3 ls
Command 'aws' not found, but can be installed with:
snap install aws-cli  # version 1.15.58, or
apt  install awscli   # version 1.22.34-1
See 'snap info aws-cli' for additional versions.
root@ip-172-31-82-1:/home/ubuntu# apt  install awscli
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
```

→

```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@ip-172-31-82-1:/home/ubuntu# aws --version
aws-cli/1.22.34 Python/3.10.6 Linux/5.19.0-1025-aws botocore/1.23.34
root@ip-172-31-82-1:/home/ubuntu# aws s3 ls
Unable to locate credentials. You can configure credentials by running "aws co
igure".
```

→ Now trigger the pipeline, it will works

→ NOTE :
You need "t2.small" range instance
→ This is the minimum instance type to run the Jenkins server

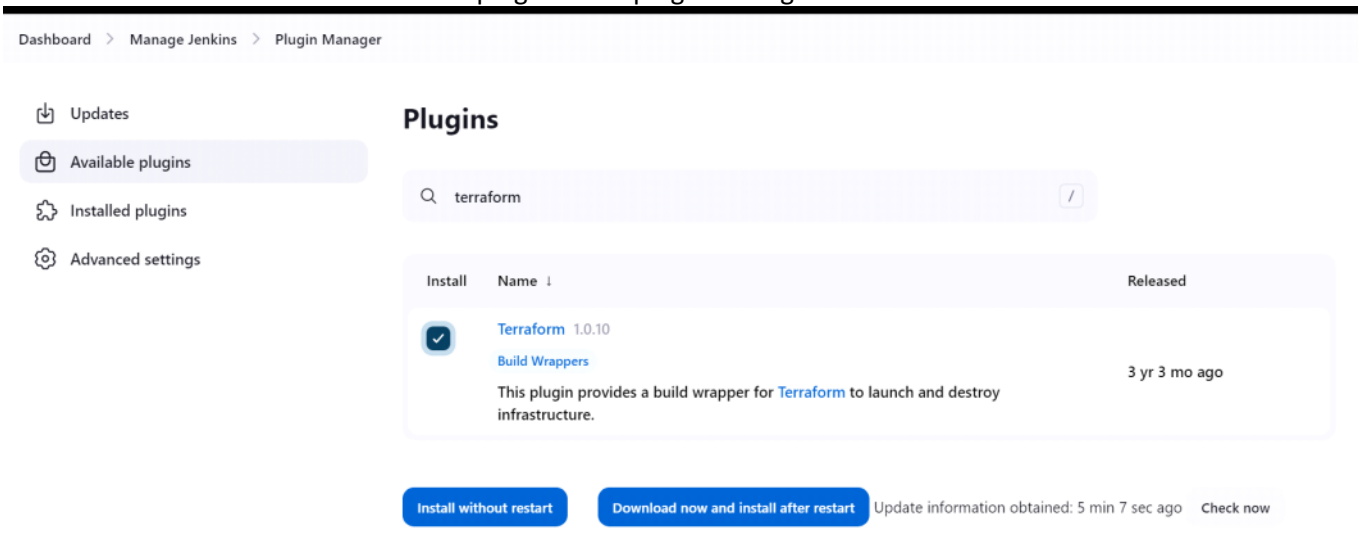
➢ HELPFUL LINKS :
https://andresaaap.medium.com/how-to-install-docker-aws-cli-eksctl-kubectl-for-jenkins-in-linux-ubuntu-18-04-3e3c4ceeb71
Demo - How to integrate AWS & Jenkins | AWS Credentials Plugin | For DevOps Training - 9886611117

TERRAFORM SETUP IN JENKINS

Monday, June 26, 2023 11:28 PM

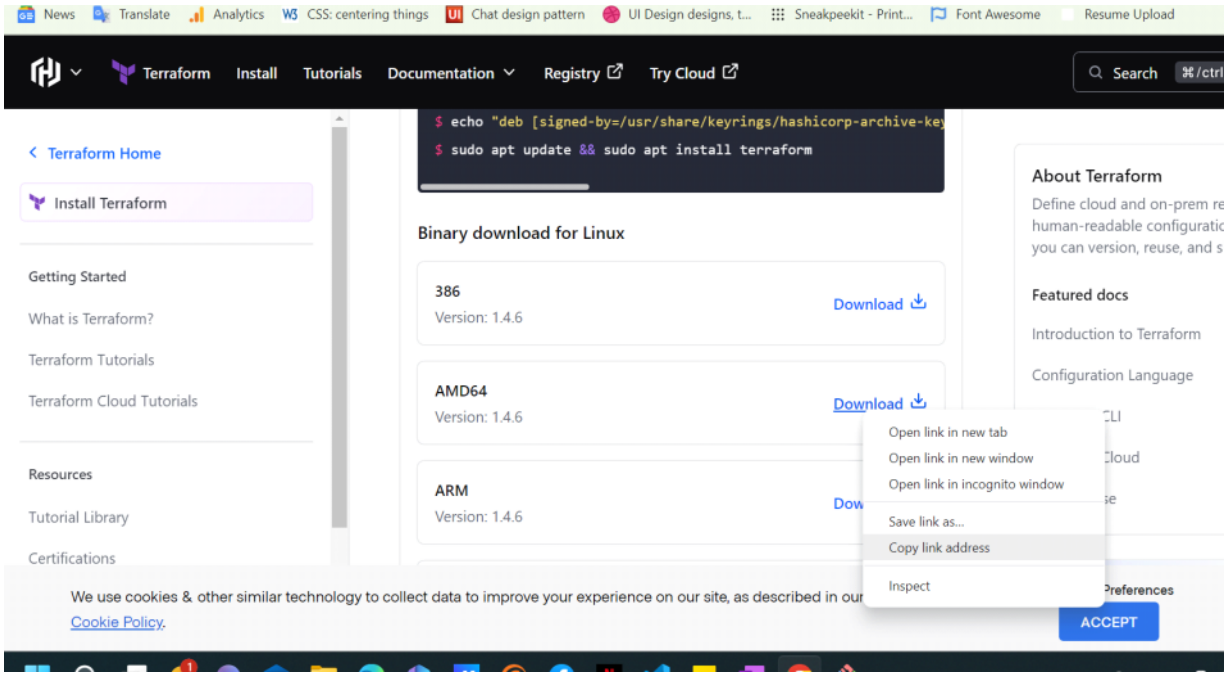
→ Firstly we need to set up the terraform inside the jenkins :
→ You need to install this terraform plugin from "plugin manager":

→



→ Click on install without restart.
→ Once, it got installed, then you need to set up the terraform installation using the "global tool configuration"
Go to manage jenkins > "global tool configuration" > Terraform

- Here you can choose the terraform to install automatically
- But, we'll try to install the terraform inside the VM and then we'll provide the path in the jenkins
- Go to this page : https://developer.hashicorp.com/terraform/downloads?product_intent=terraform



→ Select the linux and copy the link of 64 bit
→ Run the below commands in your jenkins machine after doing ssh

$ ssh -i jenkins.pem ubuntu@3.91.226.196
$ wget https://releases.hashicorp.com/terraform/1.4.6/terraform_1.4.6_linux_amd64.zip
$which terraform
$ apt install unzip

$unzip terraform_1.4.6_linux_amd64.zip
$mv terraform /usr/bin
$which terraform
$terraform version

LOGS :

```
asus@KishanDev MINGW64 /y/AWS
$ ssh -i jenkins.pem ubuntu@3.91.226.196
The authenticity of host '3.91.226.196 (3.91.226.196)' can't be established.
ED25519 key fingerprint is SHA256:vaxdHTDOCt0t7s5CYH0SHkbLQh2JB5h7bN8sik4dCbQ.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:108: 44.204.172.179
    ~/.ssh/known_hosts:111: 54.172.187.122
    ~/.ssh/known_hosts:112: 54.235.229.128
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '3.91.226.196' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.19.0-1025-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon May 29 09:11:02 UTC 2023

  System load:  0.0               Processes:             98
  Usage of /:   37.1% of 7.57GB   Users logged in:       0
  Memory usage: 73%               IPv4 address for eth0: 172.31.82.1
  Swap usage:   0%

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.

   https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

8 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

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


Last login: Sun May 28 12:30:37 2023 from 49.37.132.20
ubuntu@ip-172-31-82-1:~$ sudo su
root@ip-172-31-82-1:/home/ubuntu# wget https://releases.hashicorp.com/terraform/1.4.6/terraform_1.4.6
_linux_amd64.zip
--2023-05-29 09:13:16--  https://releases.hashicorp.com/terraform/1.4.6/terraform_1.4.6_linux_amd64.zip
Resolving releases.hashicorp.com (releases.hashicorp.com)... 108.138.85.30, 108.138.85.31, 108.138.85.53, ...
Connecting to releases.hashicorp.com (releases.hashicorp.com)|108.138.85.30|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 20779821 (20M) [application/zip]
Saving to: 'terraform_1.4.6_linux_amd64.zip'
```

```
terraform_1.4.6_lin 100%[===================>]  19.82M  --.-KB/s    in 0.1s

2023-05-29 09:13:16 (146 MB/s) - ‘terraform_1.4.6_linux_amd64.zip’ saved [20779821/20779821]

root@ip-172-31-82-1:/home/ubuntu# which terraform
root@ip-172-31-82-1:/home/ubuntu# ls
jenkins-setup.sh  terraform_1.4.6_linux_amd64.zip
root@ip-172-31-82-1:/home/ubuntu# unzip terraform_1.4.6_linux_amd64.zip
Command 'unzip' not found, but can be installed with:
apt install unzip
root@ip-172-31-82-1:/home/ubuntu# apt install unzip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Suggested packages:
  zip
The following NEW packages will be installed:
  unzip
0 upgraded, 1 newly installed, 0 to remove and 8 not upgraded.
Need to get 174 kB of archives.
After this operation, 385 kB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 unzip amd64
6.0-26ubuntu3.1 [174 kB]
Fetched 174 kB in 0s (6580 kB/s)
Selecting previously unselected package unzip.
(Reading database ... 66753 files and directories currently installed.)
Preparing to unpack .../unzip_6.0-26ubuntu3.1_amd64.deb ...
Unpacking unzip (6.0-26ubuntu3.1) ...
Setting up unzip (6.0-26ubuntu3.1) ...
Processing triggers for man-db (2.10.2-1) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

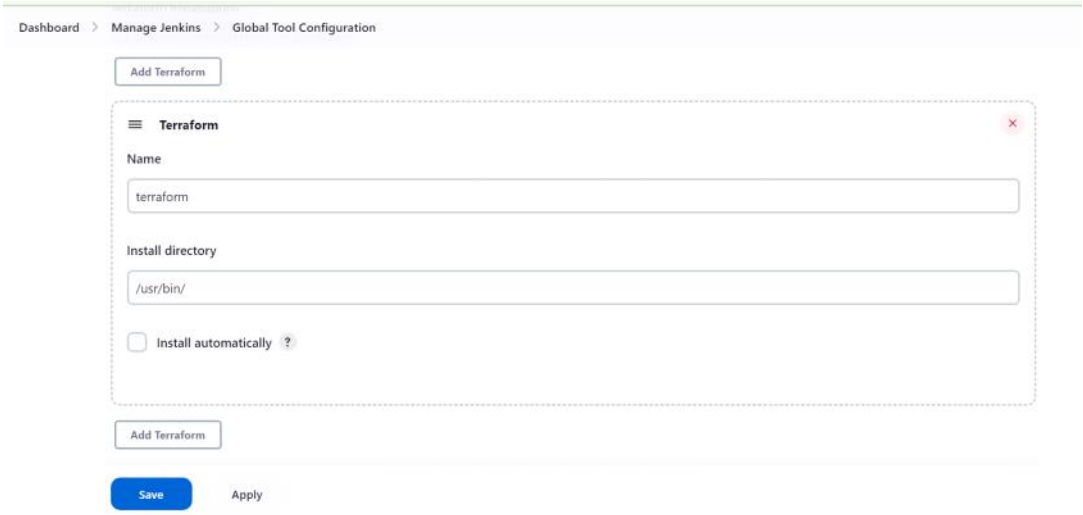
No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@ip-172-31-82-1:/home/ubuntu# ls
jenkins-setup.sh  terraform_1.4.6_linux_amd64.zip
root@ip-172-31-82-1:/home/ubuntu# unzip terraform_1.4.6_linux_amd64.zip
Archive:  terraform_1.4.6_linux_amd64.zip
  inflating: terraform
root@ip-172-31-82-1:/home/ubuntu# ls
jenkins-setup.sh  terraform  terraform_1.4.6_linux_amd64.zip
root@ip-172-31-82-1:/home/ubuntu# mv terraform /usr/bin
root@ip-172-31-82-1:/home/ubuntu# ls
jenkins-setup.sh  terraform_1.4.6_linux_amd64.zip
root@ip-172-31-82-1:/home/ubuntu# which terraform
/usr/bin/terraform
root@ip-172-31-82-1:/home/ubuntu# terraform version
Terraform v1.4.6
on linux_amd64
```

```
root@ip-172-31-82-1:/home/ubuntu#
```

→ Now, the terraform is in this path : /usr/bin/

→ Provide this path in your jenkins, your jenkins will take terraform from there



→ Click and save and apply

➢ REFRERENCES :

- Launch AWS EC2 with Jenkins | DevOps Tutorial

  https://registry.terraform.io/
  Automated Deployment from GitHub to EC2 instance using Jenkins CI/CD | Part 3

  Jenkins and AWS EC2 Connection Over SSH | Jenkins CI/CD EC2 Instance |

  Automate EC2 setup using Jenkins & Ansible | Automate Infrastructure setup using Ansible and Jenkins

  This link will have all the cloud related examples for terraform
  For AWS :
  https://registry.terraform.io/providers/hashicorp/aws/latest
  https://registry.terraform.io/providers/hashicorp/aws/latest/docs

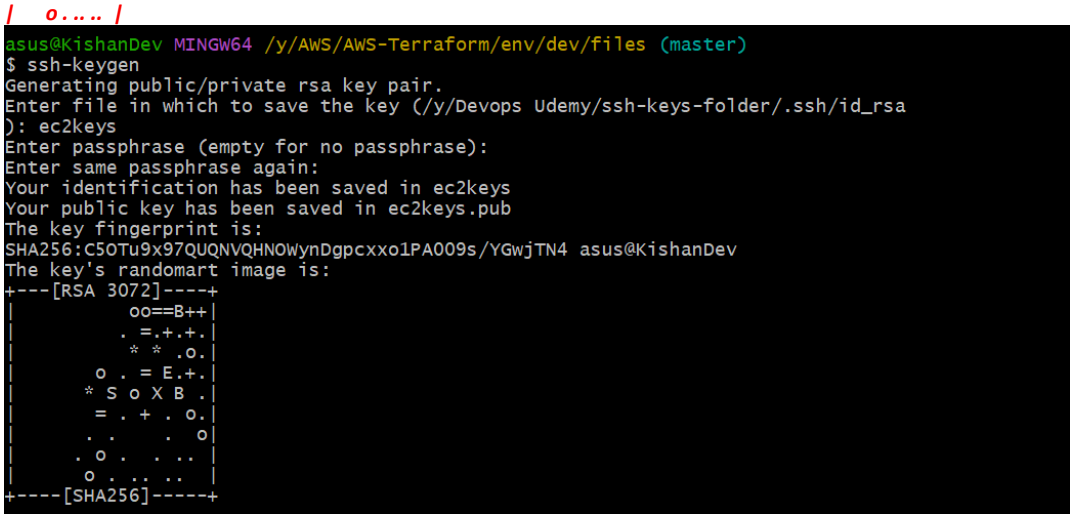
ADDING KEYS

Thursday, June 22, 2023 10:22 PM

→ First select a folder and generate the keys by using the below command:
ssh-keygen

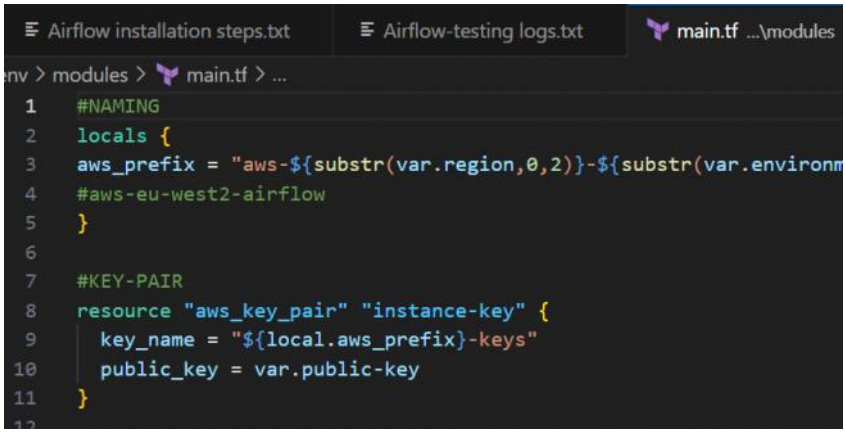
→ Once the keys are generated, provide the name as them as you like
(EX : ec2keys)

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/y/Devops Udemy/ssh-keys-folder/.ssh/id_rsa): ec2keys
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ec2keys
Your public key has been saved in ec2keys.pub
The key fingerprint is:
SHA256:C5OTu9x97QUQNVQHNOWynDgpcxxo1PA009s/YGwjTN4 asus@KishanDev
The key's randomart image is:
+---[RSA 3072]----+
|      oo==B++|
|     . =.+.+.|
|      * * .o.|
|    o . = E.+.|
|    * S o X B .|
|     = . + . o.|
|    .. .  . o|
|   . o . ... |
|    o . .. .  |
+----[SHA256]-----+
```



→ In terraform, create the key-pair using these generated keys (in modules section) :

```
#KEY-PAIR
resource "aws_key_pair" "instance-key" {
  key_name = "${local.aws_prefix}-keys"
  public_key = var.public-key
}
```

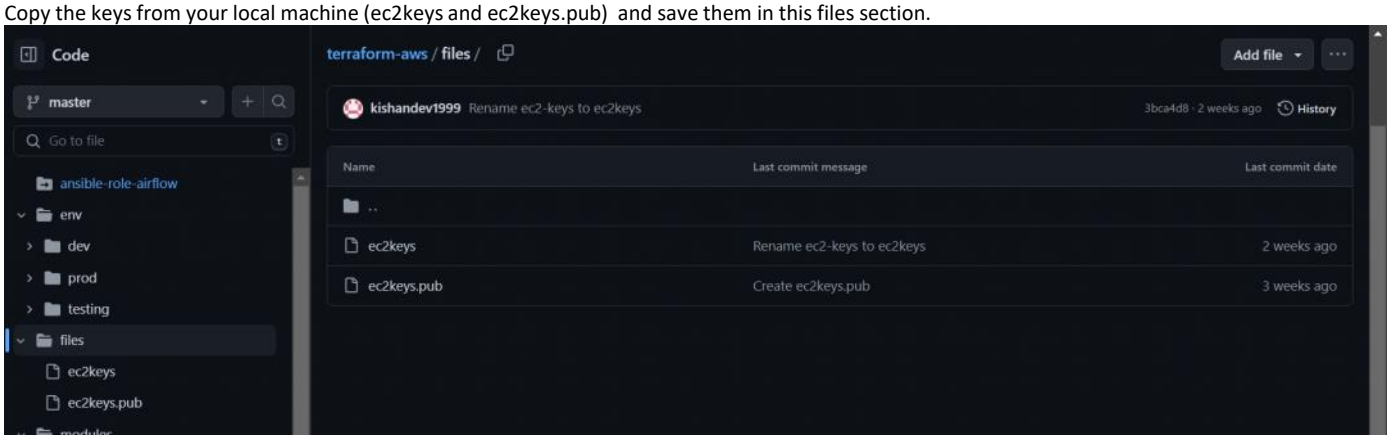


→ Attach these keys to the "instance-template"

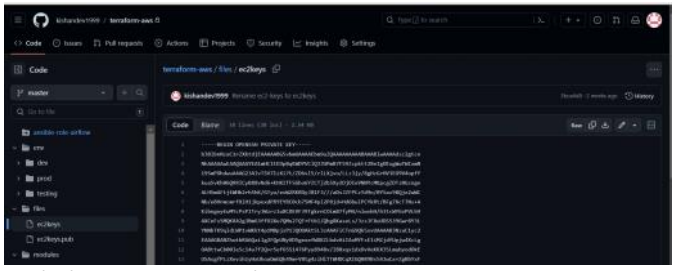
→ key_name = aws_key_pair.instance-key.key_name

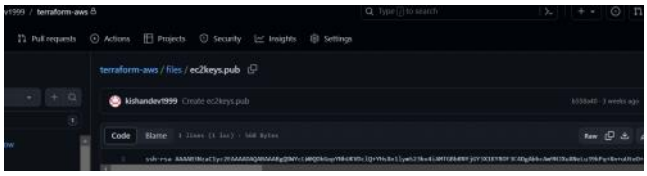
→ Copy the keys from your local machine (ec2keys and ec2keys.pub) and save them in this files section.

→



→





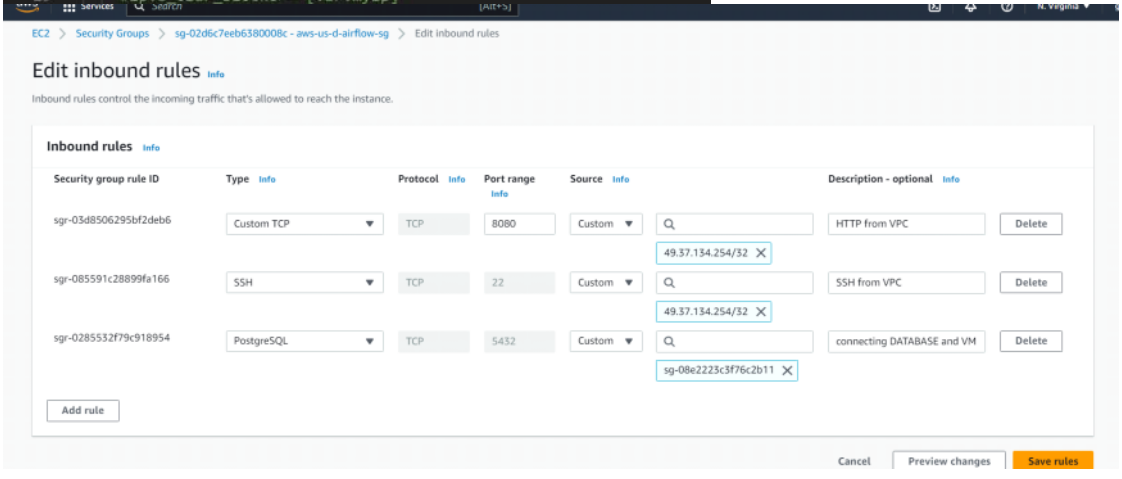
→ Apply the security group rules

→ SSH-PORT:22-MY IP

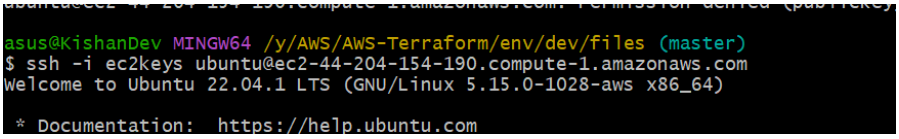
→

```
env > modules > sg.tf > resource "aws_security_group" "instance-sg"
1   resource "aws_security_group" "instance-sg" {
2     name        = "${local.aws_prefix}-sg"
3     description = "Allow MYIP inbound traffic"
4     vpc_id      = "vpc-042b7992bbcb14df5"   #ATTACH THE EXISTING VPC I
5
6   #INBOUND RULE
7     ingress {
8       description      = "SSH from VPC"
9       from_port        = 22
10      to_port          = 22
11      protocol         = "tcp"
12      cidr_blocks      = [var.myip]
13      #ipv6_cidr_blocks = [var.myip]
14    }
15    ingress {
16      description      = "HTTP from VPC"
17      from_port        = 8080
18      to_port          = 8080
19      protocol         = "tcp"
20      cidr_blocks      = [var.myip]
21      #ipv6_cidr_blocks = [var.myip]
22    }
23    ingress {
24      description      = "connecting DATABASE and VM"
25      from_port        = 5432
26      to_port          = 5432
27      protocol         = "tcp"
28      security_groups  = ["sg-08e2223c3f76c2b11"]
29      #ipv6_cidr_blocks = [var.myip]
```

→



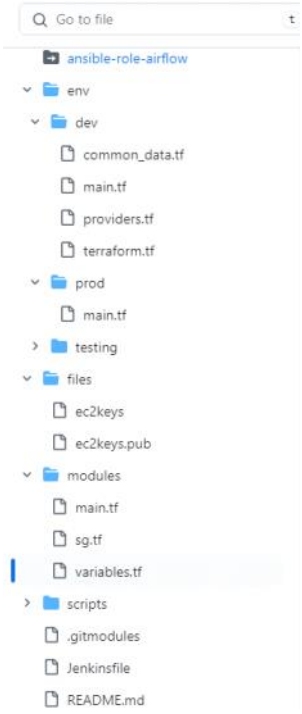
Run this command to connect or SSH into the VM:
$ ssh -i ec2keys ubuntu@ec2-44-204-154-190.compute-1.amazonaws.com
Or
$ ssh -i ec2keys ubuntu@44.204.154.190
$ssh -I <key-name> username@public-ip



TERRAFORM

Monday, June 26, 2023 11:33 PM

→ In GIT -> create a private repo. Here I created a repo with name "terraform-aws"
→ Created file architecture as below :

→ 

→ MODULES

→ **(main.tf)**
→ In terraform, Modules are the centralized code where we can recall their values and repeatedly and created no.of resources as per out requirement
→ In this Repo, I created a modules section with main.tf, sg.tf, variables.tf
→ Now, the main code will be scripted in main.tf
→ The security policies (firewall rules) will be in the sg.tf
→ Variables are defined in variables.tf

- **LOCALS**

```
#FOR NAMING THE AWS COMPONENTS WE ARE USING THESE LOCALS
#LOCALS
locals {
aws_prefix = "aws-${substr(var.region,0,2)}-${substr(var.environment,0,1)}-${var.application}"
#aws-eu-west2-airflow
}
```

→ This locals are like reusable variables. If you want to define the repeated values in your code again and again, then just simply define in locals and use the local values in your code.
→ I created this local for the standard naming convention

- `"aws-` //This will simply give name aws
- `${substr(var.region,0,2)}-` //This will take the value for region and cut the first two values as substring EX. europe - it will take (eu)
- `${substr(var.environment,0,1)}-` //This will take the first letter of the environment variable
- `${var.application}"` //this will attach the application name mentioned in the variables
- `The final output will be like :` #aws-eu-west2-airflow

- **KEY-PAIRS :**

```
#KEY-PAIR
#THIS WILL CREATE KEY PAIR FOR THE INSTANCES
resource "aws_key_pair" "instance-key" {
  key_name = "${local.aws_prefix}-keys"
  public_key = var.public-key
}
```

→ This will create the key-pairs
→ The key name will be created with the locals defined above
→ If you want to call anything defined in the locals, you need to call them as ${local.<defined_variable_name>}
→ The key we are calling from the variables

- ADDITIONAL DISK

```
#ADDITIONAL DISK CREATION
#THIS WILL CREATE AN ADDITIONAL DISK
resource "aws_ebs_volume" "additional-disk" {
  availability_zone = var.zone
  size              = var.add-disk-size
  tags = {
    Name = "${local.aws_prefix}-ini-additional-disk"
    Application = var.application
    device_name = var.add-disk-device-name
  }
}
```

→ This will create an ADDITIONAL DISK
→ Availability zone will be defined in the variables
→ Size also we are defining in static
→ Tags - We are using the locals to declared the name accordingly, Application will be the application name defined in the vaiables
→ device_name [DEFINED IN TAGS SECTION ONLY]= (This is the main important time in declaring the device name here)
For this ADDITIONAL DISK, the devicename defined as "/dev/sdb"

```
# variable defined in variables.tf
variable "add-disk-device-name" {
  default = "/dev/sdf"
}
```

→ REFER THE TABLE FOR DEVICE NAMES

Type of Virtualization	Available String Blocks	Reserved for Root	Instance Store Volumes	Recommended for EBS Volumes
Paravirtual	**/dev/sd[a-z]** **/dev/sd[a-z][1-15]** **/dev/hd[a-z]** **/dev/hd[a-z][1-15]**	**/dev/sda1**	**/dev/sd[b-e]**	**/dev/sd[f-p]** **/dev/sd[f-p][1-6]**
HVM	**/dev/sd[a-z]** **/dev/xvd[b-c][a-z]**	Differs by AMI **/dev/sda1** or **/dev/xvda**	**/dev/sd[b-e]** **/dev/sd[b-y]** (hs1.8xlarge)	**/dev/sd[f-p]**

REFERENCE <https://www.softnas.com/docs/softnas/v3/html/ebs_volumes_and_device_mapping_print.html>
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/device_naming.html>

- **LAUNCH TEMPLATE :**

```
resource "aws_launch_template" "linux" {
  #count = var.linux ? 1 : 0
  name = "${local.aws_prefix}-linux-machine-templates"
  instance_type = var.instance-type                    #t2.micro
  key_name = aws_key_pair.instance-key.key_name
  image_id = var.image-id
  vpc_security_group_ids = [aws_security_group.instance-sg.id]

  #BOOT DISK
  block_device_mappings {
   device_name = var.boot-disk-device-name              #/dev/sda1
  ebs {
    volume_size = var.boot-disk-size                  #10
    delete_on_termination = true
    volume_type = var.volume-type                     #STANDARD
    }
  }
  monitoring {
    enabled = true
  }
  placement {
    availability_zone = var.zone
  }
```

```
  tag_specifications {
    resource_type = "instance"
    tags = {
    Name = "${local.aws_prefix}"
    script = "${var.application}-init"
    image   = var.image_name
    }
  }

  #DISK TAGS
  tag_specifications {
    resource_type = "volume"
    tags = {
    #count = lenght(var.disk-device-name-list)
    Name = "${local.aws_prefix}-ini-boot-disks"
    #script = "${var.application}-init"
    #image   = var.image_name
    }
  }
  #user_data = var.init-script
  #aws_launch_template.linux.tag_specifications.tags.scrit.value
}
```

→ In this template, We configure the below components
- Keys
- Instance Type
- Image
- Security group
- Boot Disk
- Disk Tags
- User data

→ KEYS : We are using the key-pairs which are already created above in KEY PAIRS section
→ Instance Type: It is taken from the variables (t2.micro, t2.small…)
→ Image : Here we are using UBUBTU image declared in the variable section
→ Security Group : This will map the security group which we create in sg.tf file with its id "aws_security_group.instance-sg.id"
→ Boot Disk : Instance definitely needs ROOT VOLUME (without this we can't create the instance)
- The device name will be mapped as "/dev/sda1"
- SIZE : 10GB
- Volume Type : STANDARD

→ TAGS SPECIFIATION : This will map the tag to only instance template for instance and volume separately
→ USER DATA: If you want to add any script manually, you can uncomment and use this variable. Now, here I'm provisioning through ansible and shell scripting. So, I commented this.

- **LAUNCHING INSTANCE FROM TEMPLATE:**

```
resource "aws_instance" "instance-linux" {
 #count = 1
 availability_zone = var.zone
 launch_template {
  name = aws_launch_template.linux.name
  #id  = aws_launch_template.linux.id
 }
 connection {
    type      = "ssh"
    host      = self.public_ip
    user      = "ubuntu"
    private_key = var.private-key
    timeout    = "4m"
 }
 provisioner "local-exec" {
  command = "echo ${self.private_ip} >> /var/lib/jenkins/workspace/aws-infra-build/private_ip.txt"
 }

 provisioner "local-exec" {
  command = "echo ${self.public_ip} >> /var/lib/jenkins/workspace/aws-infra-build/public_ip.txt"
}
 tags = {
  Name = "${local.aws_prefix}-instance"
  #Name = "${local.aws_prefix}-instance-${count.index}"
  #Script = var.source_script
 }
}
```

→ AVAILABILITY ZONE: provide through variables

→ LAUNCH TEMPLATE : calling the created template via name
→ CONNECTION : via SSH and providing the key
→ LOCAL-EXEC : Getting the public IP and private IP of the VM and pasting in the files
→ TAGS : For NAME of the VM

- **ATTACHING THE ADDITIONAL DISK AFTER VM GOT CREATED :**

→ Once, the VM gets created based on the template attached. After that we are attaching the created ADDITIONAL DISK to this VM

```
resource "aws_volume_attachment" "ebs-additional-disk" {
  device_name = "/dev/sdf"
  volume_id = aws_ebs_volume.additional-disk.id
  instance_id = aws_instance.instance-linux.id

}
```

→ DEVICE NAME : "/dev/sdf" <REFER THE TABLE>
{DEVICE NAME you have to define here. Because, the instance has to use this additional volume apart from the root volume}
→ VOULME_ID : attach the additional disk with ID reference
→ INSTACE_ID : attach the ID of instance to which this ADDITIONAL DISK you are planning to attach

→ **(Sg.tf)**
→ Creating the security group (Firewall Rules) :

```
resource "aws_security_group" "instance-sg" {
  name        = "${local.aws_prefix}-sg"
  description = "Allow MYIP inbound traffic"
  vpc_id      = "vpc-042b7992bbcb14df5"   #ATTACH THE EXISTING VPC ID

 #INBOUND RULE
  ingress {
    description      = "SSH from VPC"
    from_port        = 22
    to_port          = 22
    protocol         = "tcp"
    cidr_blocks      = [var.myip]
    #ipv6_cidr_blocks = [var.myip]
  }
 ingress {
    description      = "SSH from VPC"
    from_port        = 22
    to_port          = 22
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"]
    #ipv6_cidr_blocks = [var.myip]
  }
  ingress {
    description      = "HTTP from VPC"
    from_port        = 8080
    to_port          = 8080
    protocol         = "tcp"
    cidr_blocks      = [var.myip]
    #ipv6_cidr_blocks = [var.myip]
  }
  ingress {
    description      = "connecting DATABASE and VM"
    from_port        = 5432
    to_port          = 5432
    protocol         = "tcp"
    security_groups  = ["sg-08e2223c3f76c2b11"]
    #ipv6_cidr_blocks = [var.myip]
  }

 #OUTBOUND RULE
  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
```

```
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }

  tags = {
    Name = "airflow-security-group"
  }
}
```

→ VPC : Using the default VPC id
→ Apply only inbound rules
→ TCP - 22 - MYIP (IP DEFINED IN VARIABLES)
→ HTTP - 8080 - ["0.0.0.0/0"] (Means anywhere fromIPV4)
→ DATABASE - 5232 - "DATABASE SECURITY GROUP" > "CONNECTING INSTANCE TO DATABASE "
→ DON'T DO ANY CHANGES IN THE OUTBOUND RULES

→ **(Variables.tf)**

```
variable zone {
   default = ""
}

variable region {
   default = ""
}

variable AMI-ID {
   default = ""
}

variable project-name {
   default = ""
}

variable instance-type {
   default = ""
}

variable "environment" {
   default = ""
}

variable "project" {
   default = ""
}

variable volume-type {
   default = ""
}

variable linux {
   default = ""
}
variable key-name {
   default = ""
}

variable add-disk-size {
   default =""
}

variable boot-disk-size {
   default = ""
}

variable image-id {
   default = ""
}

variable subnet-id {
   default = ""
}

variable public-key {
   default = ""
```

```
}

variable application {
   default = ""
}

variable security-group {
   default = ""
}

variable myip {
   default = ""
}

variable private-key {
   default = ""
}

variable user {
   default = ""
}

variable out-pub-ip {
   default = ""
}

variable out-priv-ip {
   default = ""
}

variable source_script {
   default = ""
}

variable destination_loc {
   default = ""
}

variable init-script {
   default = ""
}

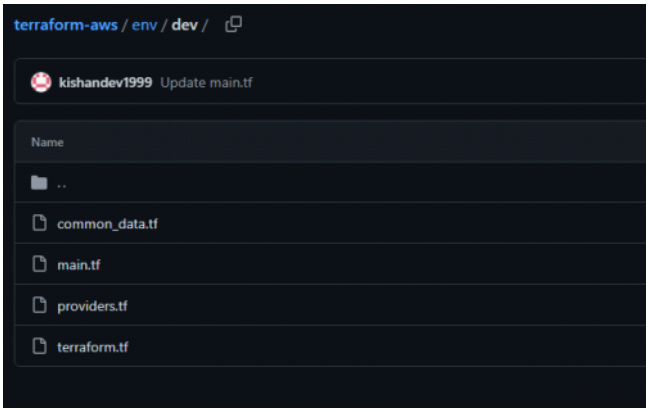
variable image_name {
   default = ""
}
variable "boot-disk-device-name" {
  default = "/dev/sda1"
}

variable "add-disk-device-name" {
  default = "/dev/sdb"
}
```

FROM THE FOLDERS : ENV -> DEV :

→ Files architecture

→



→ (Providers.tf)

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

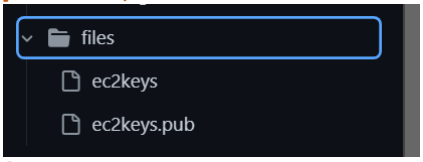
→ In this providers.tf, we will mention the cloud provider and the region

→ (Main.tf)

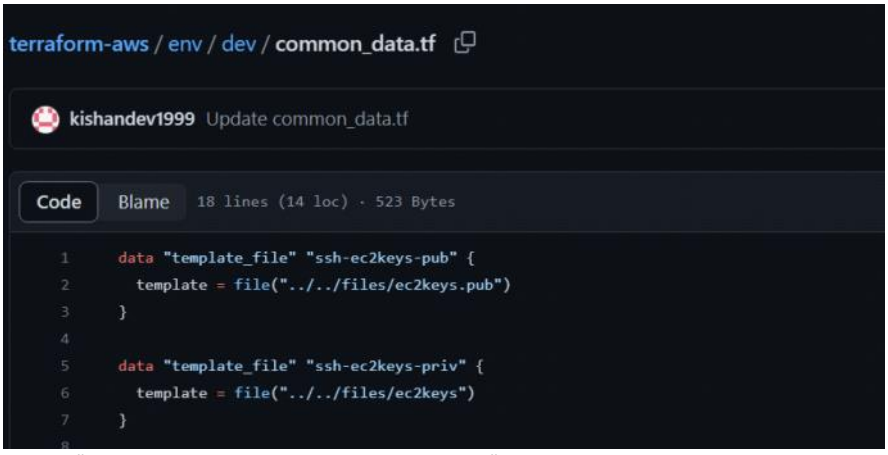
```
#Airflow#
module "airflow" {
  source        = "../../modules"
  region        = var.region
  zone          = var.zone
  application   = "airflow"
  instance-type  = "t2.micro"
  environment    = var.environment
  volume-type    = "standard"
  image-id       = var.image
  image_name     = "ubuntu"
  add-disk-size  = 10
  boot-disk-size = 10
  public-key     = data.template_file.ssh-ec2keys-pub.rendered
  private-key    = data.template_file.ssh-ec2keys-priv.rendered
  myip           = "49.37.134.186/32" #"192.168.29.40/32"
  security-group = var.security-group
  user           = var.user
  #source_script   =  "../scripts/airflow-init.sh"
  #destination_loc = "/tmp/airflow-init.sh"
  #init-script = base64encode(data.template_file.airflow-script.rendered)
  #"${base64encode(data.template_file.test.rendered)}"
}
```

→ In main.tf, we will call the modules with convenient name [module "airflow"]

- Source : "call the modules (folder)" [NOTE : ../../ This means the modules folder is in the 2 steps back from the current path. So, please give the ../ accordingly]
- Region : calling region from variables in terraform.tf files
- Zone : calling region from variables in terraform.tf files
  [NOTE: Declare the common values, EX. ZONE, REGION in variables]
- Environment : calling from variables (development, testing, production)
- Volume-type : standard
- Image : providing the image ID in terraform.tf [ami-0557a15b87f6559cf]
- Add-disk-size (additional disk size) : 10
- Boot-disk-size (boot disk size) : 10
- Public-key = *data.template_file.ssh-ec2keys-pub.rendered*
- Private-key = *data.template_file.ssh-ec2keys-priv.rendered*
  [THIS MEANS, THE PUBLIC KEY AND THE PRIVATE KEY ARE CALLING FROM THE FILES PRESENT IN THE BELOW SCREENSHOT]



- *[THESE KEYS ARE RENDERED IN THE commondata.tf]*



- My-ip = "enter your computer IP ADDRESS MANUALLY"
- Security group = (No need to define anything. It is already declared in launch template)
- User = defined in terraform.tf [ubuntu]

→ If you are declaring the startup script in the user data use this values

#source_script = "../scripts/airflow-init.sh"
#destination_loc = "/tmp/airflow-init.sh"
#init-script = base64encode(data.template_file.airflow-script.rendered)
#"${base64encode(data.template_file.test.rendered)}"

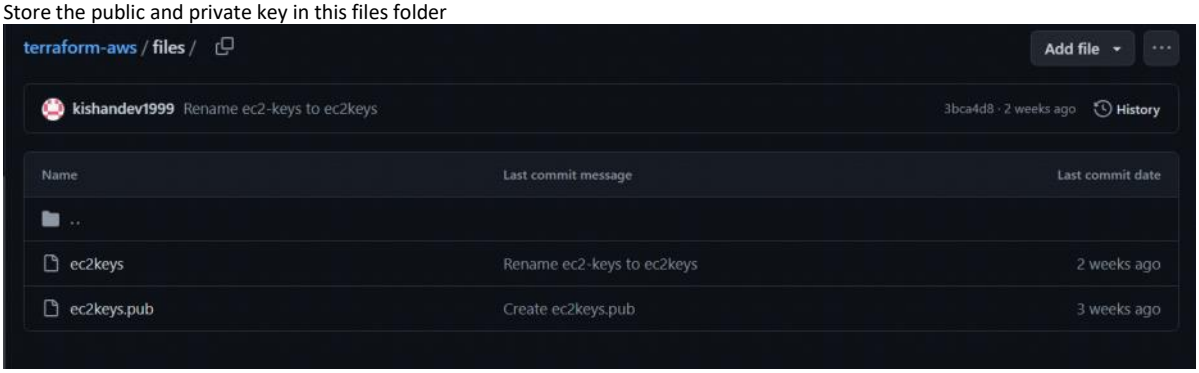
→ *This is the source script (present in /scripts/ folder in git repo and placing the destination location /tmp/airflow)*
→ *The initscript should be in base64decoder*

→ (common_data.tf)

```
data "template_file" "ssh-ec2keys-pub" {
 template = file("../../files/ec2keys.pub")
}

data "template_file" "ssh-ec2keys-priv" {
 template = file("../../files/ec2keys")
}
```

→ We are configuring our keys in the VM through this common_data.tf
→ Generate the keys in your local machine by : *$ssh-keygen*
→ Store the public and private key in this files folder



→ *template = file("../../files/ec2keys")*
→ *[note: the path ../../files/ec2keys, This ../../ means the files folder located two steps back from the present director]*
→ *This will map the public key which mentioned in the modules*

Modules/main.tf

```
#KEY-PAIR
#THIS WILL CREATE KEY PAIR FOR THE INSTANCES
resource "aws_key_pair" "instance-key" {
  key_name = "${local.aws_prefix}-keys"
  public_key = var.public-key
}
```

→ This public key will render the keys in the files folder and create the key-pair, the common_data.tf will helps to fetch the keys from the file using "data"

→ (terraform.tf)

```
variable "region" {
 default = "us-east-1"
}

variable "zone" {
```

```
 default = "us-east-1a"
}

variable "user" {
 default = "ubuntu"
}

variable "environment" {
 default = "development"
}

variable "subnet" {
 default = ""
}

variable "image" {
 default = "ami-0557a15b87f6559cf"
}
#ami-0557a15b87f6559cf -- UBUNTU

variable "ssh" {
 default = ""
}

variable "security-group" {
 default = ""
}

variable "source_script" {
 default = ""
}

variable "destination_loc" {
 default = ""
}

variable "public-key" {
 default = ""
}

variable "private-key" {
 default = ""
}

variable "myip" {
 default = ""
}
```

SCRIPTS :

→ *scripts/ansible-airflow-config.sh*

→ This is the script to run the Ansible inside the JENKINS

```
#!/bin/bash
pwd
##Getting the IP address :
present_path="/var/lib/jenkins/workspace/aws-infra-build"
cd $present_path
##cd /var/lib/jenkins/workspace# cd aws-infra-build
#pwd

private_ip=`cat private_ip.txt`
echo $private_ip

public_ip=`cat public_ip.txt`
echo $public_ip

chmod 700 /var/lib/jenkins/workspace/aws-infra-build/files/ec2keys
ansible_dir="/var/lib/jenkins/workspace/aws-infra-build/ansible-role-airflow/tasks/main.yaml"
keys_dir="/var/lib/jenkins/workspace/aws-infra-build/files/ec2keys"
sleep 45
```

```
ansible-playbook -i ${public_ip}, -u ubuntu --private-key=${keys_dir} ${ansible_dir}
echo "ansible-playbook -i ${public_ip}, -u ubuntu --private-key=${keys_dir} ${ansible_dir}"
```

→ First we are collecting the IP address of created VM, which is copied in the public.txt file
→ *public_ip=`cat public_ip.txt`*
→ Now, Providing the permissions for the private key file
→ *chmod 700 /var/lib/jenkins/workspace/aws-infra-build/files/ec2keys*
→ This is the ansible command to run the playbook in the VM
→ $ *ansible-playbook -i ${public_ip}, -u ubuntu --private-key=${keys_dir} ${ansible_dir}*
→ SYNTAX : ansible-playbook -i <${PUBLIC IP OF VM}> -u ububtu --private-key=${PRIVATE-KEY OF VM} ${ANSIBLE MAIN.YAML FILE}
→ i is inventory, -u is username of VM

→ *scripts/cleanup.sh*
→ This cleanup.sh will clear the files like pubilc_ip.txt and private_ip.txt, in order to repeat the occurrence of IP address

```
##Getting the IP address :
present_path_cu="/var/lib/jenkins/workspace/aws-infra-build"
cd $present_path_cu
pwd

priv_file=private_ip.txt
pub_file=public_ip.txt

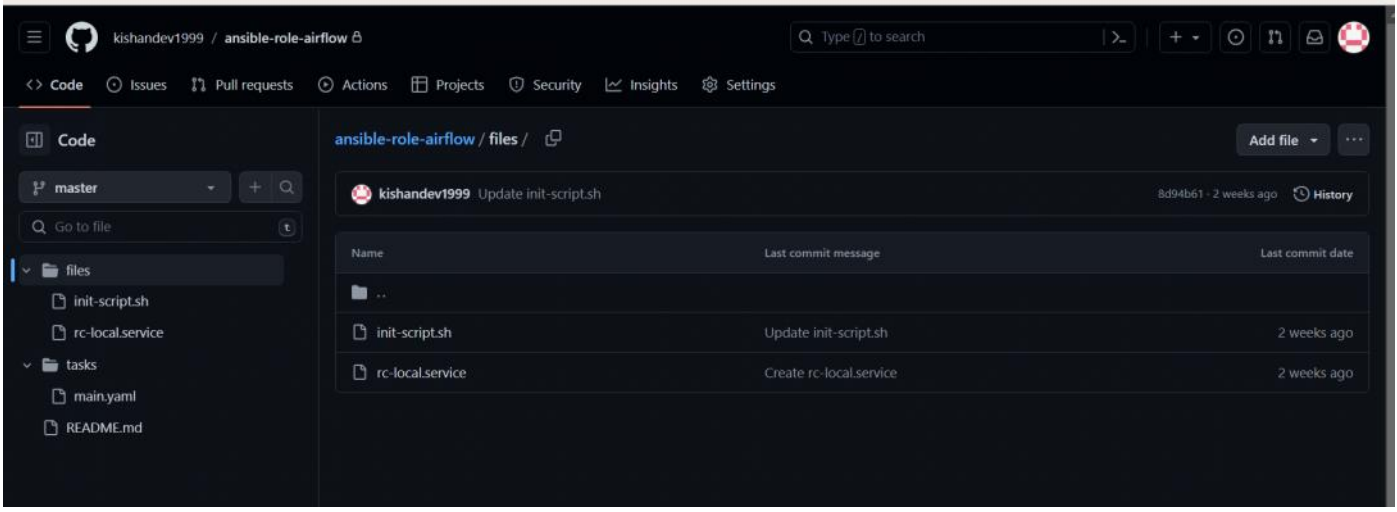
if [ -f "$priv_file" ]; then
rm private_ip.txt
else
echo "file not exists"
fi


if [ -f "$pub_file" ]; then
rm public_ip.txt
else
echo "file not exists"
fi
```

ANSIBLE

Monday, June 26, 2023 11:34 PM

→ For Ansible configuration, create a new repo in git:

→ 

→ There are two folders : files and tasks
→ In files, We have init-script(STARTUP SCRIPT) and rc-local.service
→ In tasks, you will fine main.yaml file
→ First We, will see main.yaml configuration

→ Main.yaml

→ Here

```
--
- name : Updating the packages
  hosts : all
  become: yes

  vars :
    - remote_install_path: /var/lib/jenkins/workspace/aws-infra-build/ansible-role-airflow/files/

  tasks:
    - name : update and upgrade apt packages
      apt :
        upgrade : yes
        update_cache : yes

    - name : install python
      apt :
        name : python3-pip
        state: latest

    - name : install sqlite3
      apt :
        name : sqlite3
        state : latest

    - name : install airflow
      pip :
        name : apache-airflow

    - name : install postgre
      apt :
        name : postgresql-contrib
        state : latest

    - name : set the initilisation of startup script
      block :
      - name : copy init-script file
        copy :
          src : "{{ remote_install_path }}/init-script.sh"
          dest : /tmp/
          owner : root
          group : root
          mode : '0775'

    - name : setting up the rc-local. service file
```

```
  block :
  - name : copy rc-local.service
    copy :
      src : "{{ remote_install_path }}/rc-local.service"
      dest : /etc/systemd/system/
      owner : root
      group : root
      mode : '0775'

  - name : Create a rc.local file
    copy :
      dest : /etc/rc.local
      content: |
        #!/bin/bash


  - name : Changing the permissions for rc.local
    file :
      path : /etc/rc.local
      state : file
      owner : root
      group  : root
      mode : '0755'


  - name : Enable a service
    service :
      name : rc-local
      enabled : yes

  - name : start the rc.local service
    service :
      name : rc-local
      state: started

  - name : Copy the conent of init scipt to rc.local
    copy :
      src : "{{ remote_install_path }}/init-script.sh"
      dest: /etc/rc.local
      owner : root
      group : root
      mode  : '0775'

  - name : Message
    debug :
      msg : "Now, This machine will reboot. SSH connection will be disconnected, ignore the below error"

  - name : rebooting the machine
    command : reboot
```

```
=============================================================================================
- name : Updating the packages
  hosts : all
  become: yes

  vars :
    - remote_install_path: /var/lib/jenkins/workspace/aws-infra-build/ansible-role-airflow/files/
```

→ This code is to configure the hosts. Here, we mentioned all. If you have any specific hosts mentioned in the inventory file, provide the name here in hosts.
→ Become: yes -> Means it will become root user
→ Vars : (means defining the variable here)
  - -variable-name : value

```
=============================================================================================
  tasks:
    - name : update and upgrade apt packages
      apt :
        upgrade : yes
        update_cache : yes
```

→ Tasks, These task are mentioned on what are the packages we are planning to install or setup through ansible
→ -name : (Provide the name of the task)
→ For ububtu VM , we are generally updating the apt packages
→ FOR THIS COMMAND : ***$sudo apt update***

```
=============================================================================================
- name : install python
  apt :
    name : python3-pip
    state: latest
```

→ This task will install PYTHON PIP
→ TASK FOR THIS COMMAND : ***$ sudo apt install python3-pip -y***
→ State: latest means, it will install latest python package

```
==========================================================================================
- name : install sqlite3
    apt :
     name : sqlite3
     state : latest
```

→ This task will install SQLITE3 database required for airflow
→ COMMAND FOR THIS TASK : ***$ sudo apt install sqlite3 -y***

```
==========================================================================================
- name : install airflow
       pip :
         name : apache-airflow
```

→ Installing APACHE-AIRFLOW using PIP
→ COMMAND FOR THIS TASK IS : ***$ pip install "apache-airflow[postgres]==2.5.1***

```
==========================================================================================
- name : install postgre
    apt :
     name : postgresql-contrib
     state : latest
```

→ This will install postgre sql database
→ COMMAND FOR THIS TASK IS : ***$*** sudo apt install postgresql postgresql-contrib -y

```
==========================================================================================
  - name : set the initilisation of startup script
    block :
    - name : copy init-script file
     copy :
       src : "{{ remote_install_path }}/init-script.sh"
       dest : /tmp/
       owner : root
       group : root
       mode : '0775'
```

→ This will copy the init-script.sh file in the path mentioned in the variable {{remote_install_path}} to destination "/tmp/" in in the VM
→ Providing the permissions 0775
[The chmod 775 command will grant the read, write, and execute permissions to the owner of the file or directory and to the group associated to it. The '5' in this command will grant both read and execute permissions (4+1) to other users.]

```
==========================================================================================
  - name : setting up the rc-local. service file
    block :
    - name : copy rc-local.service
     copy :
       src : "{{ remote_install_path }}/rc-local.service"
       dest : /etc/systemd/system/
       owner : root
       group : root
       mode : '0775'
```

→ This will copy The rc-local.service file in the path mentioned in the variable {{remote_install_path}} to destination "/etc/systemd/system/" in in the VM
→ Providing the permissions 0775

```
==========================================================================================
  - name : Create a rc.local file
    copy :
     dest : /etc/rc.local
     content: |
      #!/bin/bash
```

→ This will create a file (rc.local) in the path /etc/ and will paste the content #!/bin/bash

```
==========================================================================================
  - name : Changing the permissions for rc.local
    file :
     path : /etc/rc.local
     state : file
     owner : root
     group  : root
     mode : '0755'
```

→ This will Provide the permission for the file rc.local

```
- name : Enable a service
  service :
   name : rc-local
   enabled : yes
```

→ Enable the rc-local service

```
- name : start the rc.local service
  service :
   name : rc-local
   state: started
```

→ Start the rc-local service

```
- name : Copy the conent of init scipt to rc.local
  copy :
   src : "{{ remote_install_path }}/init-script.sh"
   dest: /etc/rc.local
   owner : root
   group : root
   mode  : '0775'
```

→ This will copy the content in the init-script.sh to the /etc/rc.local file
→ Providing the permissions for it

```
- name : Message
  debug :
    msg : "Now, This machine will reboot. SSH connection will be disconnected, ignore the below error"
```

→ This will just print the message as we are rebooting the machine in next task, VM will get disconnected

```
- name : rebooting the machine
  command : reboot
```

→ This will reboot the machine

→ INIT-SCRIPT.SH

```
#!/bin/bash

#IT WILL INITILIZE THE AIRFLOW DATABASE
airflow db init


#THESE  ARE COMMANDS JUST FOR REFERENCE PURPOSE TO CREATE DB AND USERS (IGNORE THESE)
#sudo -i -u postgres
#sudo -i -u postgres psql -c "CREATE DATABASE airflow;"
#sudo -i -u postgres psql -c "CREATE USER airflow WITH PASSWORD 'airflow';"
#sudo -i -u postgres psql -c "GRANT ALL PRIVILEGES ON DATABASE airflow TO airflow;"


#CHANGING THE DIRECTIORY. ALL AIRFLOW CONFIGURATION FILES WILL BE STORE IN THIS FOLDER
cd /root/airflow

#SEARCHING THE SQL_ALCHEMY_CONNECTION IN airflow.cfg file
grep sql_alchemy_conn airflow.cfg
#OUTPUT FORO THIS COMMAND
#sqlite:////root/airflow/airflow.db

#DECLARING IN THE BELOW VARIABLE FOR THIS COMMAND :  grep "sql_alchemy_conn =" /root/airflow/airflow.cfg
aiflow_initial_db=`grep "sql_alchemy_conn =" /root/airflow/airflow.cfg`

#REPLACING THE ALCHEMY DATABASE
echo "$aiflow_initial_db" | sed 's/sql_alchemy_conn = //g'
database_value=`echo "$aiflow_initial_db" | sed 's/sql_alchemy_conn = //g'`

sed -i 's#$database_value#postgresql+psycopg2://admin123:admin123a@aairflow-db.cyn1tccpnjs5.us-
east-1.rds.amazonaws.com/airflowdata#g' airflow.cfg
```

```
sed -i 's#sqlite:////home/ubuntu/airflow/airflow.db#postgresql+psycopg2://admin_123a:admin_123a@airflow-db.cj0jajqgnaw0.us-east-1.rds.amazonaws.com/airflowdata#g' airflow.cfg

#sed -i 's#SequentialExecutor#LocalExecutor#g' airflow.cfg

airflow db init

airflow users create -u airflow-1234 -f airflow -l airflow -r Admin -p airflow -e airflowi@gmail.com

airflow webserver &

airflow scheduler
```

RC LOCAL

Monday, June 26, 2023 11:34 PM

The rc. local script in some Linux distributions and Unix systems is a superuser startup script, usually located under the directory /etc/etc/rc. d. The file name rc refers to Run Control.

For UBUBTU, to run in RC LOCAL, you need to add this or configure this in your machine
Create a file with name : "rc-local.service"
Add this below code and save it

```
[Unit]
 Description=/etc/rc.local Compatibility
 ConditionPathExists=/etc/rc.local
[Service]
 Type=forking
 ExecStart=/etc/rc.local start
 TimeoutSec=0
 StandardOutput=tty
 RemainAfterExit=yes
 SysVStartPriority=99
[Install]
 WantedBy=multi-user.target
```

MANUAL COMMANDS TO EXECUTE OR SETUP THE RC.LOCAL :

```
$ ls
```

```
$ sudo systemctl enable rc-local

● rc-local.service - /etc/rc.local Compatibility
    Loaded: loaded (/lib/systemd/system/rc-local.service; static)
   Drop-In: /usr/lib/systemd/system/rc-local.service.d
            └─debian.conf
    Active: inactive (dead)
      Docs: man:systemd-rc-local-generator(8)
root@ip-172-31-94-84:/home/ubuntu# sudo systemctl enable rc-local
The unit files have no installation config (WantedBy=, RequiredBy=, Also=,
Alias= settings in the [Install] section, and DefaultInstance= for template
units). This means they are not meant to be enabled using systemctl.
```

```
$ sudo vim /etc/systemd/system/rc-local.service

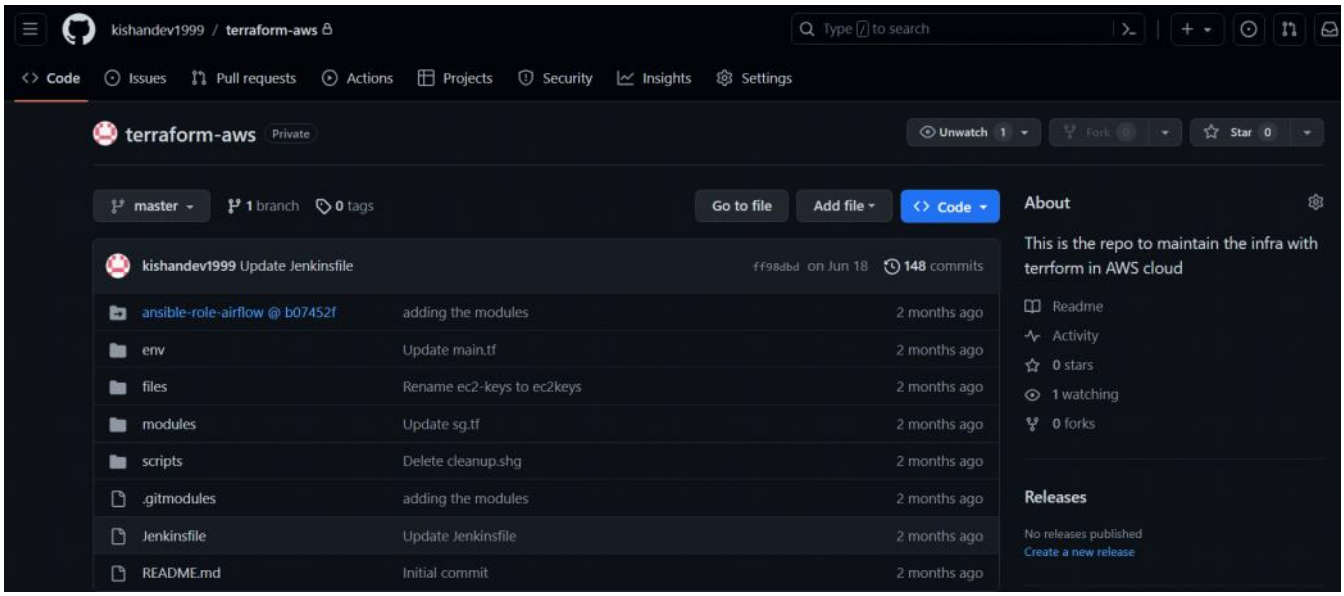
[Unit]
 Description=/etc/rc.local Compatibility
 ConditionPathExists=/etc/rc.local
[Service]
 Type=forking
 ExecStart=/etc/rc.local start
 TimeoutSec=0
 StandardOutput=tty
 RemainAfterExit=yes
```

SysVStartPriority=99 *[Install]* *WantedBy=multi-user.target*
$ printf '%s\n' '#!/bin/bash' 'exit 0' | sudo tee -a /etc/rc.local
$ sudo chmod +x /etc/rc.local
$ sudo systemctl enable rc-local *Created symlink /etc/systemd/system/multi-user.target.wants/rc-local.service → /etc/systemd/system/rc-local.service.*
$ sudo systemctl start rc-local.service
$ sudo systemctl status rc-local.service *● rc-local.service - /etc/rc.local Compatibility* *Loaded: loaded (/etc/systemd/system/rc-local.service; enabled; vendor pres>* *Drop-In: /usr/lib/systemd/system/rc-local.service.d* *└─debian.conf* *Active: active (exited) since Sun 2023-06-11 15:48:16 UTC; 14s ago* *Process: 1416 ExecStart=/etc/rc.local start (code=exited, status=0/SUCCESS)* *CPU: 2ms* *Jun 11 15:48:16 ip-172-31-94-84 systemd[1]: Starting /etc/rc.local Compatibilit>* *Jun 11 15:48:16 ip-172-31-94-84 systemd[1]: Started /etc/rc.local Compatibility.* *lines 1-10/10 (END)*
$ Reboot

REFERENCE :
https://linuxhint.com/use-rc-local-on-ubuntu/
https://www.linuxbabe.com/linux-server/how-to-enable-etcrc-local-with-systemd
How To Run A Command On StartUp in Linux| How to run a startup script

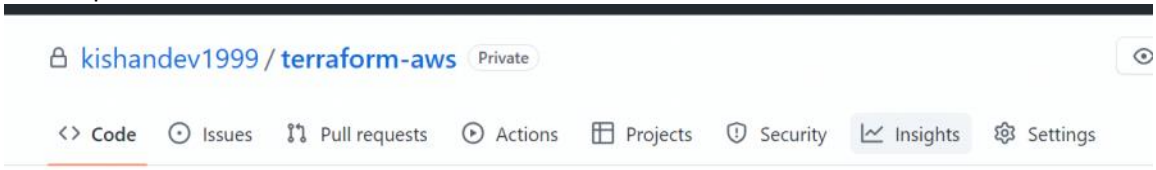
SUBMODULES IN GIT

Monday, June 26, 2023 11:34 PM

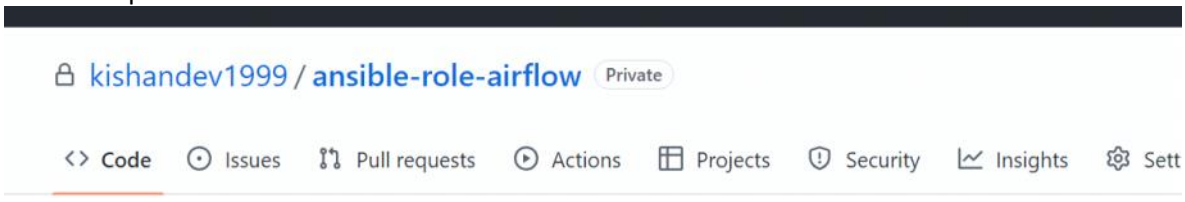


To create a submodule, follow the below commands :
Consider 2 repositories :

1. First Repo is



2. Second Repo is



Now we need to link 2nd repo (ansible-role-airflow) to the first repo (terraform-aws)

1. Create a folder in your local machine :
2. Git clone the 1st repo (terraform-aws)

asus@KishanDev MINGW64 /y/terraform aws latest

$ git clone https://github.com/kishandev1999/terraform-aws.git
Cloning into 'terraform-aws'...
remote: Enumerating objects: 276, done.
remote: Counting objects: 100% (155/155), done.
remote: Compressing objects: 100% (155/155), done.
remote: Total 276 (delta 88), reused 0 (delta 0), pack-reused 121
Receiving objects: 100% (276/276), 65.71 KiB | 1.88 MiB/s, done.
Resolving deltas: 100% (115/115), done.
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead

hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git submodule add https://github.com/kishandev1999/ansible-role-airflow.git
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"

Cloning into 'Y:/terraform aws latest/terraform-aws/ansible-role-airflow'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
warning: LF will be replaced by CRLF in .gitmodules.
The file will have its original line endings in your working directory

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git add .
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git commit -m "adding the submodule ansible-role-airflow"
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
[master a5cf2ba] adding the submodule ansible-role-airflow
 2 files changed, 4 insertions(+)
 create mode 100644 .gitmodules
 create mode 160000 ansible-role-airflow

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git push origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 432 bytes | 432.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/kishandev1999/terraform-aws.git
  ff7d01b..a5cf2ba master -> master

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)

Check the above commands :
The repo will automatically link to your git

REMOVING THE SUBMODLE :
$ git clone https://github.com/kishandev1999/terraform-aws.git
Cloning into 'terraform-aws'...

```
remote: Enumerating objects: 449, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 449 (delta 0), reused 0 (delta 0), pack-reused 446
s:  90% (405/449)
Receiving objects: 100% (449/449), 101.97 KiB | 1.32 MiB/s, done.
Resolving deltas: 100% (213/213), done.
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"


asus@KishanDev MINGW64 /y/terraform aws latest
$ ls
terraform-aws/

asus@KishanDev MINGW64 /y/terraform aws latest
$ cd terraform-aws/

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ ls
Jenkinsfile  README.md  ansible-role-airflow/  env/  files/  modules/  scripts/

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git submodule deinit -f ansible-role-airflow/
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"

error: could not lock config file .git/modules/ansible-role-airflow/config: No such file or directory
warning: Could not unset core.worktree setting in submodule 'ansible-role-airflow'
Cleared directory 'ansible-role-airflow'

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git rm -rf ansible-role-airflow/
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
rm 'ansible-role-airflow'

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ ls
Jenkinsfile  README.md  env/  files/  modules/  scripts/

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ rm -rf .git/modules/ansible-role-airflow/

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git status
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitmodules
        deleted:    ansible-role-airflow
```

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git add .
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git commit -m "removing the modules"
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
[master b325842] removing the modules
 2 files changed, 4 deletions(-)
 delete mode 160000 ansible-role-airflow

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 284 bytes | 284.00 KiB/s, done.
Total 3 (delta 1), reused 1 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/kishandev1999/terraform-aws.git
   6bd3486..b325842  master -> master

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git submodule add https://github.com/kishandev1999/ansible-role-airflow.git
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"

Cloning into 'Y:/terraform aws latest/terraform-aws/ansible-role-airflow'...
remote: Enumerating objects: 38, done.
remote: Counting objects: 100% (38/38), done.
remote: Compressing objects: 100% (29/29), done.
remote: Total 38 (delta 10), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (38/38), 8.21 KiB | 4.10 MiB/s, done.
Resolving deltas: 100% (10/10), done.
warning: LF will be replaced by CRLF in .gitmodules.
The file will have its original line endings in your working directory

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ ls
Jenkinsfile  README.md  ansible-role-airflow/  env/  files/  modules/  scripts/

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git add .
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git commit -m "Adding the modules"
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
[master 1bf6890] Adding the modules
 2 files changed, 4 insertions(+)
 create mode 160000 ansible-role-airflow
```

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git push orign master
fatal: 'orign' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 417 bytes | 417.00 KiB/s, done.
Total 3 (delta 1), reused 1 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/kishandev1999/terraform-aws.git
   b325842..1bf6890  master -> master

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
```

JENKINS PIPELINE SCRIPT

Monday, June 26, 2023 11:34 PM

```
pipeline{
   agent any
   tools {
      terraform 'terraform-11'
   }
  parameters {
  choice(
  name : 'environment',
   choices : [
    'dev',
    'testing',
    'prod'
   ],
   description : 'Choose the environment you want to deploy: ')
  choice (
  name : 'application',
   choices : [
   'airflow',
   'tom-cat',
   'jenkins'
   ],
   description : 'Choose the application')
  }

  //environment variables
  environment {
   ENVIRONMENT = "${params.environment}"
   APP = "${params.application}"
   WORKSPACE = "${env.WORKSPACE}"
  }
stages {

   // stage: 0
  stage("Clean up") {
      steps {
       withCredentials([[
           $class: 'AmazonWebServicesCredentialsBinding',
            credentialsId: 'aws-jenkins-creds',
            ACCESS_KEY: 'AWS_ACCESS_KEY_ID',
            SECRET_KEY: 'AWS_SECRET_ACCESS_KEY'
            ]]) {
            dir("${WORKSPACE}/scripts") {
               sh label: '', script: 'chmod +x cleanup.sh'
               sh label: '', script: './cleanup.sh'
               }
            }
          }
       }

   // stage: 0.1
```

```
  stage("Updating the submodule") {
     steps {
       withCredentials([gitUsernamePassword(credentialsId: 'git-cred-id',
            gitToolName: 'git-tool')]) {
            sh 'git submodule update --init --recursive'
           }
       //sh 'git config --global user.name "kishandev1999"'
       //sh 'git config --global user.email "kishandev005@gmail.com"'
       //sh  'git config -l'

     }
     }

  //stage: 1
  stage("Terraform Init") {
     steps {
      withCredentials([[
          $class: 'AmazonWebServicesCredentialsBinding',
           credentialsId: 'aws-jenkins-creds',
           ACCESS_KEY: 'AWS_ACCESS_KEY_ID',
           SECRET_KEY: 'AWS_SECRET_ACCESS_KEY'
           ]]) {
         dir("${WORKSPACE}/env/${ENVIRONMENT}") {
           sh "terraform init "
             }
           }
        }
     }
   //stage: 2
  stage("Terraform apply") {
     steps {
      withCredentials([[
          $class: 'AmazonWebServicesCredentialsBinding',
           credentialsId: 'aws-jenkins-creds',
           ACCESS_KEY: 'AWS_ACCESS_KEY_ID',
           SECRET_KEY: 'AWS_SECRET_ACCESS_KEY'
           ]]) {
           dir("${WORKSPACE}/env/${ENVIRONMENT}") {
            sh "terraform destroy --auto-approve"
              }
           //input(message: "Deploy to ${env.ENVIRONMENT} with application ${env.APP}?", ok :
'Deploy')
           }
         }
      }
     //stage : 3
     stage("Configuring Selected Application") {
        steps {
         withCredentials([[
          $class: 'AmazonWebServicesCredentialsBinding',
           credentialsId: 'aws-jenkins-creds',
           ACCESS_KEY: 'AWS_ACCESS_KEY_ID',
           SECRET_KEY: 'AWS_SECRET_ACCESS_KEY'
           ]]) {
            script {
           if ("${APP}"== "airflow") {
               dir("${WORKSPACE}/scripts") {
```

```
                sh "chmod +x ansible-airflow-config.sh"
                sh "./ansible-airflow-config.sh"
            }
        }
        else {
            echo "Select the application"
        }
    }
   }
  }
 }
}

}
```