Case study

Integration of Devops tools with Jenkins

NAME: PRADEEPA K

EMAIL: KPRADEEPACHARYA0@GMAIL.COM

You have been hired as a DevOps Engineer in XYZ Software Company. They want to implement a CI/CD pipeline in their company. You have been asked to implement this lifecycle as fast as possible. As this is a product-based company, their product is available on this GitHub link.

https://github.com/hshar/website.git

Following are the specifications of the Continuous integration:

1. Git Workflow hasto be implemented

2. Code Build should automatically be triggered once commit is made to master branch

or develop branch.

If commit is made to master branch, build and publish website on

port 82. If commit is made to develop branch, just build the

product, do not publish.

3. Create a pipeline for the above tasks.

4. Create a container with Ubuntu and apache installed in it and use that container to

build the code and the code should be on '/var/www/html'.

Introduction

This case study focuses on implementing a Continuous Integration (CI) pipeline using Jenkins for a product hosted on GitHub. The objective is to establish an efficient CI workflow that integrates with the existing Git repository. Key tasks include implementing a Git workflow, automating builds triggered by commits to specific branches, and creating a Docker container with Ubuntu and Apache to manage the build process. Commits to the master branch will trigger a full build and deployment, while commits to the develop branch will result in a build only, without deployment.

Project Background

The product, hosted on GitHub, is regularly updated by a team of developers. However, the current process of building and deploying code is manual, which leads to several issues:

- Slow Development: Manually building and deploying code takes a lot of time, slowing down the development process.
- Inconsistent Environments: Developers sometimes face problems because the build environments aren't always the same.
- Human Errors: Manual processes are more likely to have mistakes, which can cause bugs and other issues.

To solve these problems, a Continuous Integration and Continuous Deployment (CI/CD) pipeline using Jenkins is being implemented. This will automate the build, test, and deployment processes, making the workflow faster, more consistent, and less prone to errors.

Objectives

The primary objectives of this case study are:

- Implement Git Workflow: Establish a structured Git workflow that integrates with Jenkins, ensuring that code changes are managed effectively across different branches.
- Automate Build Triggers: Configure Jenkins to automatically trigger builds based on commits to specific branches:
- Master Branch: Trigger a full build and deploy the product on port 82.
- Develop Branch: Trigger a build without deploying the product.
- Create a Jenkins Pipeline: Develop a Jenkins pipeline that automates the entire process, from code integration to testing and deployment.

Containerize the Build Environment: Use Docker to create a container with Ubuntu and Apache, which will serve as the build environment. Ensure that the code is built and placed in the /var/www/html directory within the container.

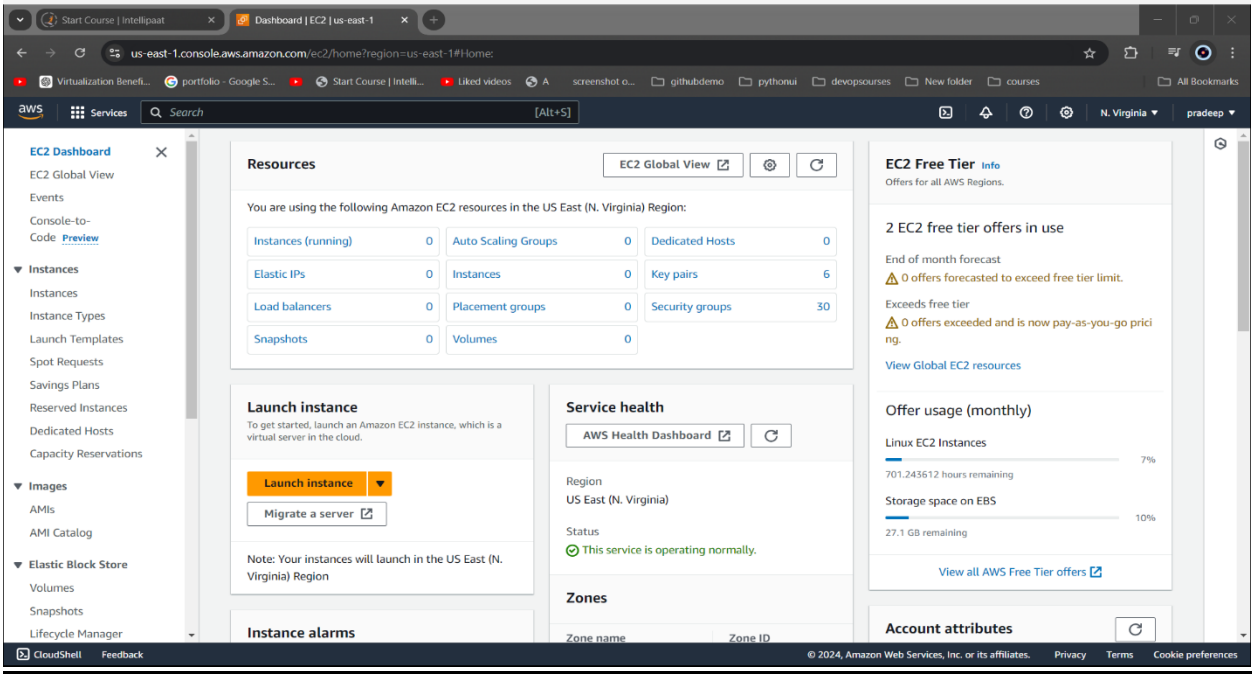
Resources Used

The following tools and technologies were utilized in the implementation of the CI/CD pipeline:

- Version control system used to manage and track changes in the codebase, ensuring effective collaboration among developers.
- AWS: Cloud platform used to host the Jenkins server and other necessary infrastructure for the CI/CD pipeline.
- Jenkins: Automation server used to create and manage the CI/CD pipeline, automating the build, test, and deployment processes.
- Docker: Containerization tool used to create a consistent and isolated build environment, ensuring that the code is built and deployed in a controlled manner.

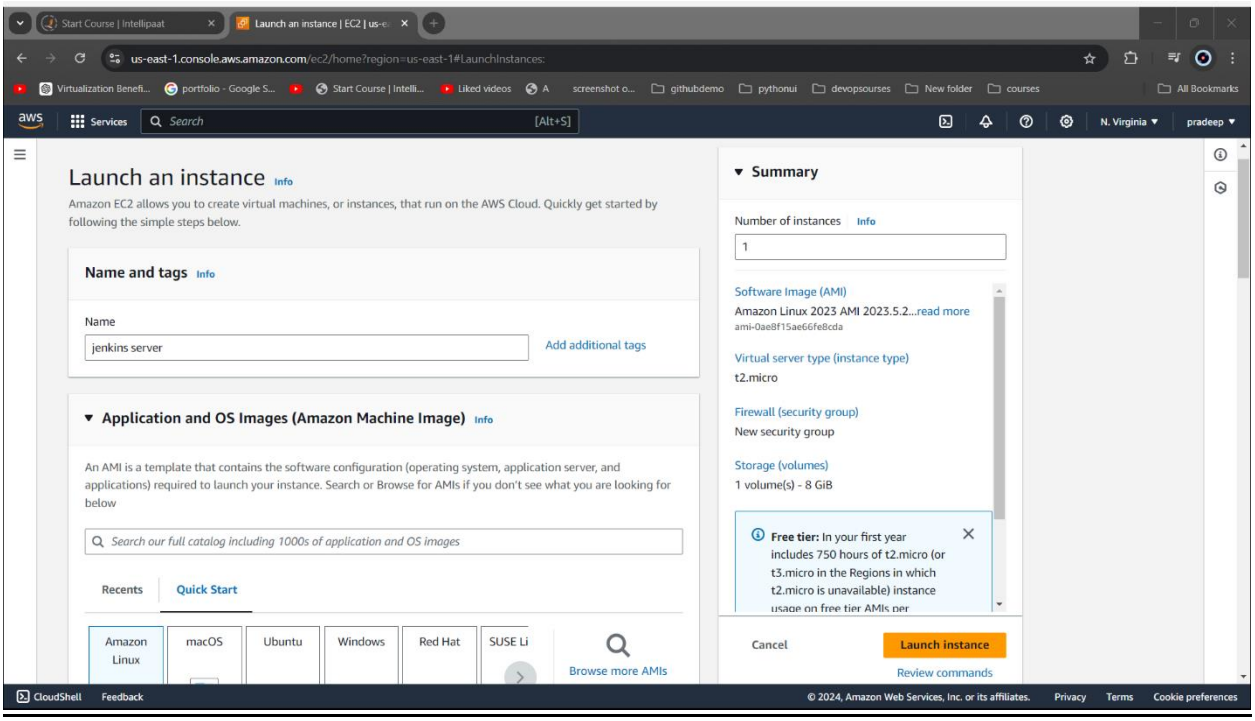
Implementation Steps:

Setup EC2 Instance:

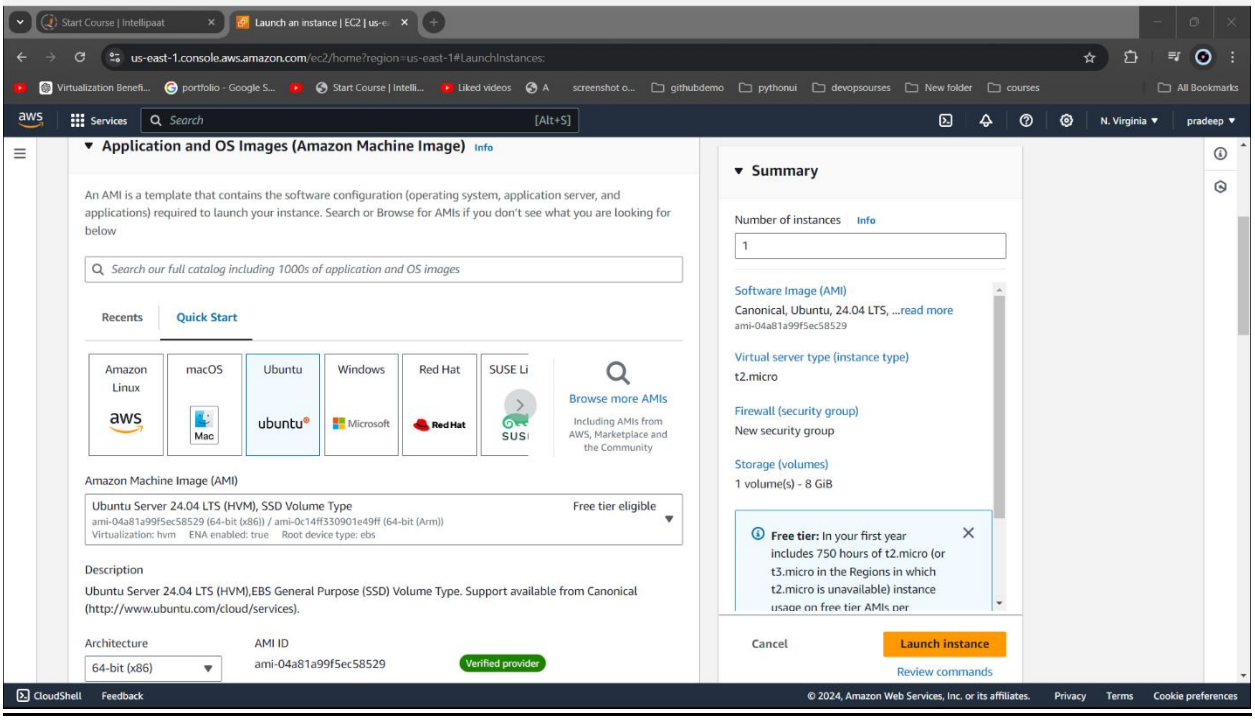


Creating a Jenkins sever:

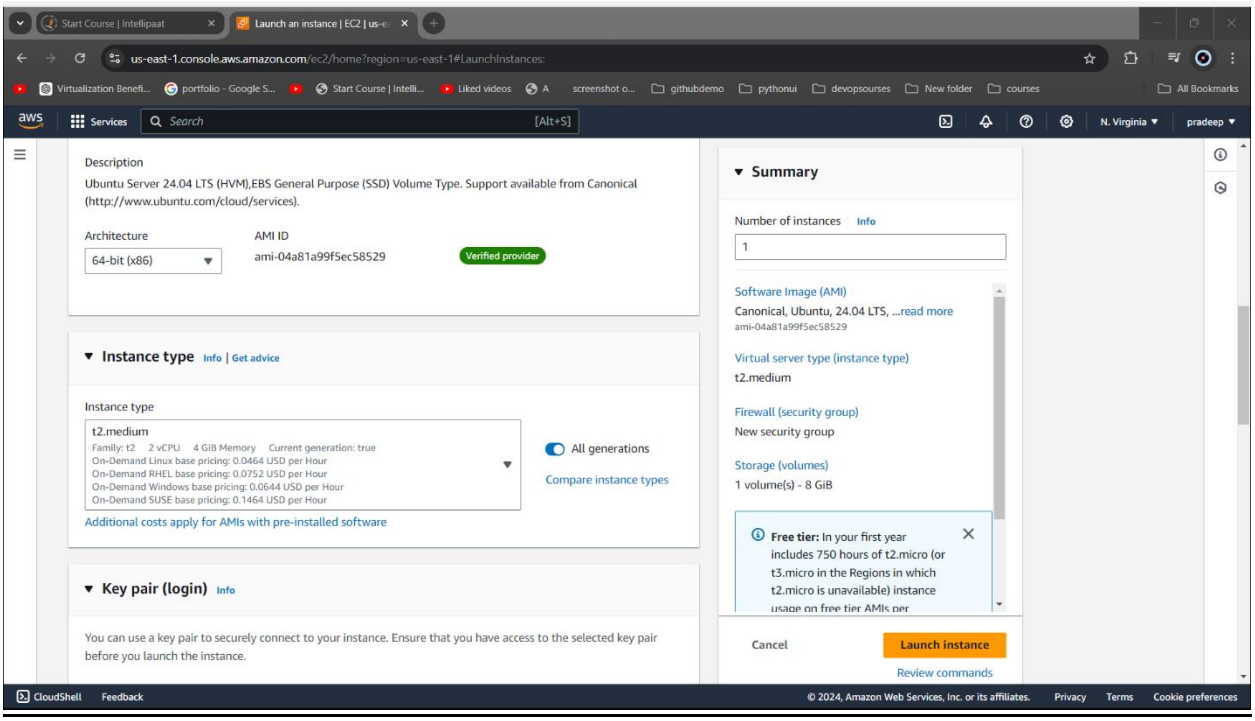
Name and tags:



Application and OS Images (Amazon Machine Image):

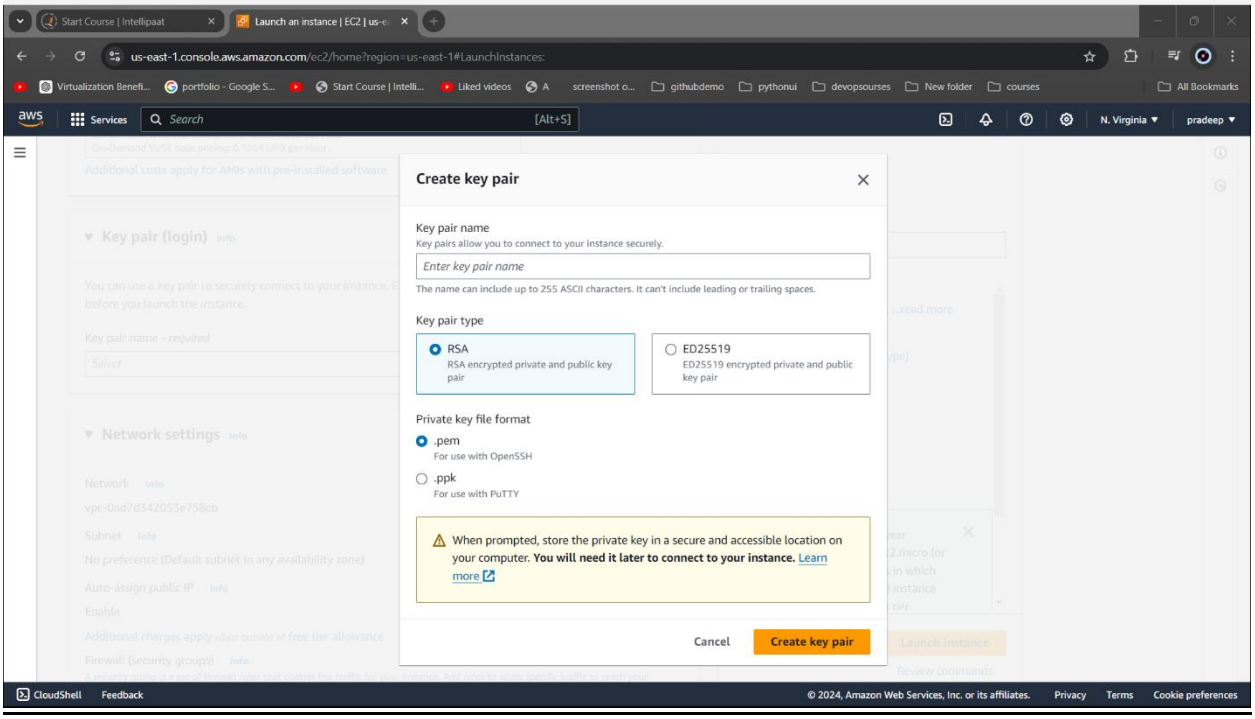


Instance type:



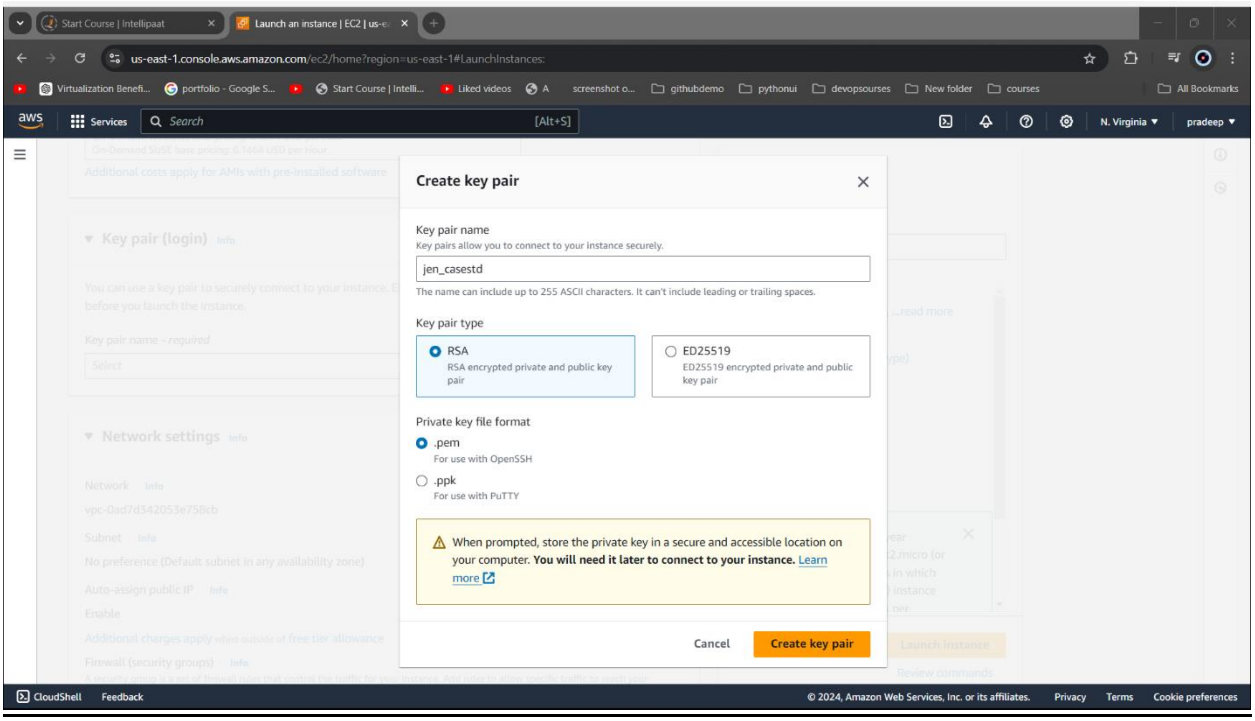
Key pair (login):

-> click on Create new key pair.



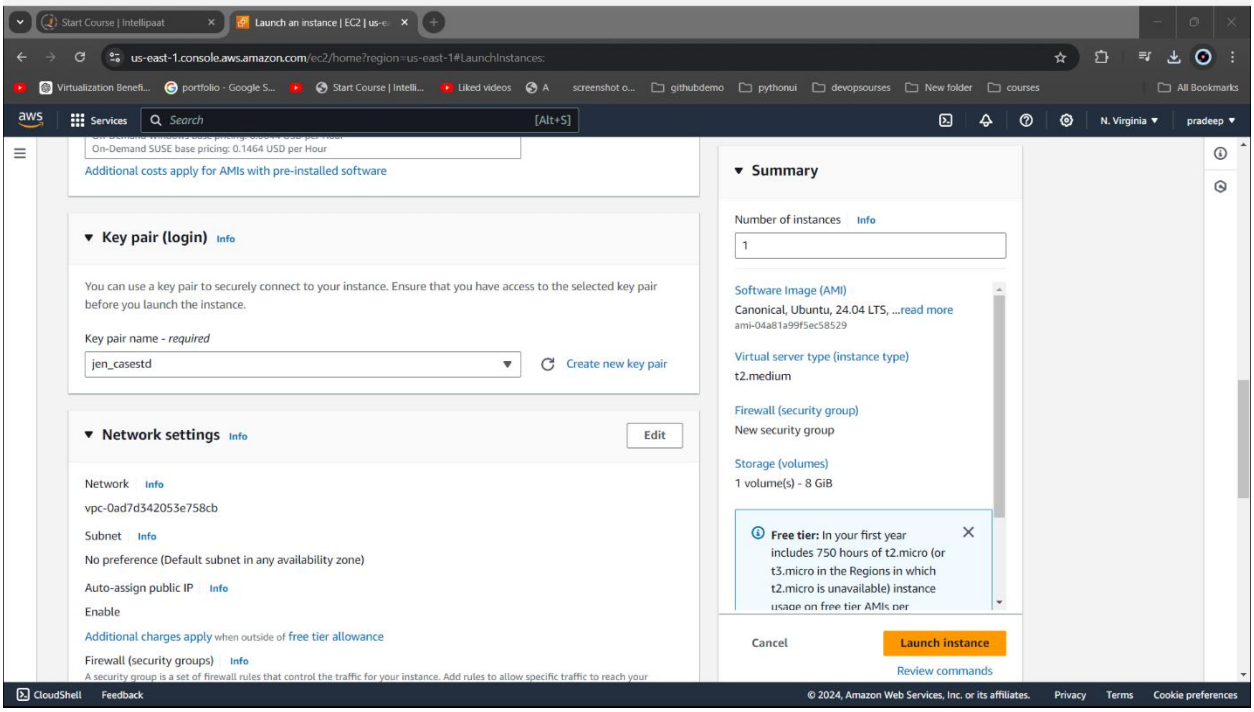
Create key pair:

Key pair name:



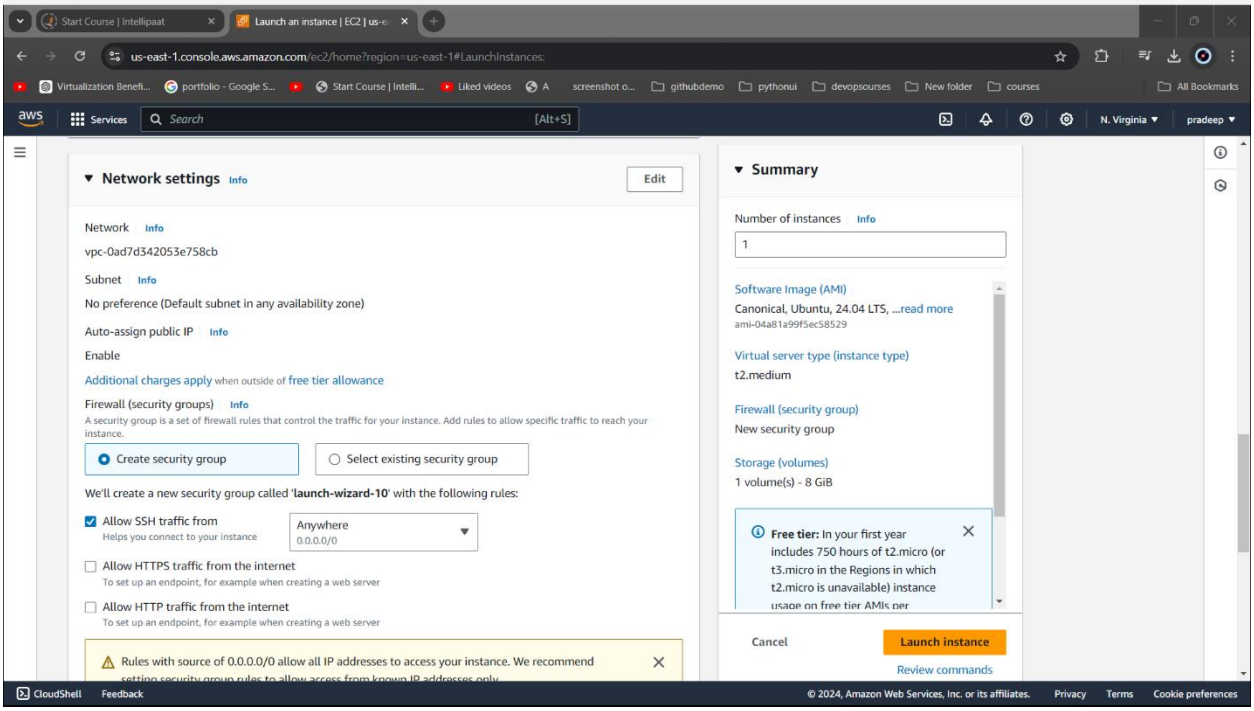
Private key file format:

-> pem.



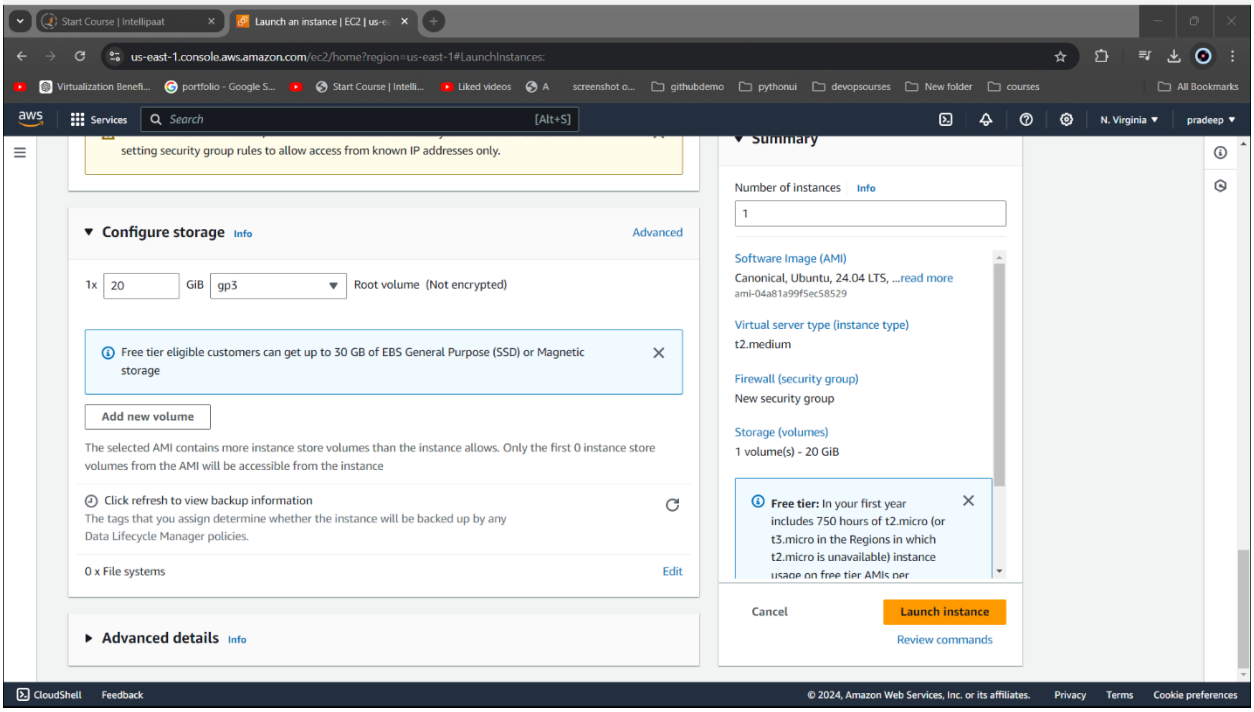
Keypair is created.

Network settings is set to be default:

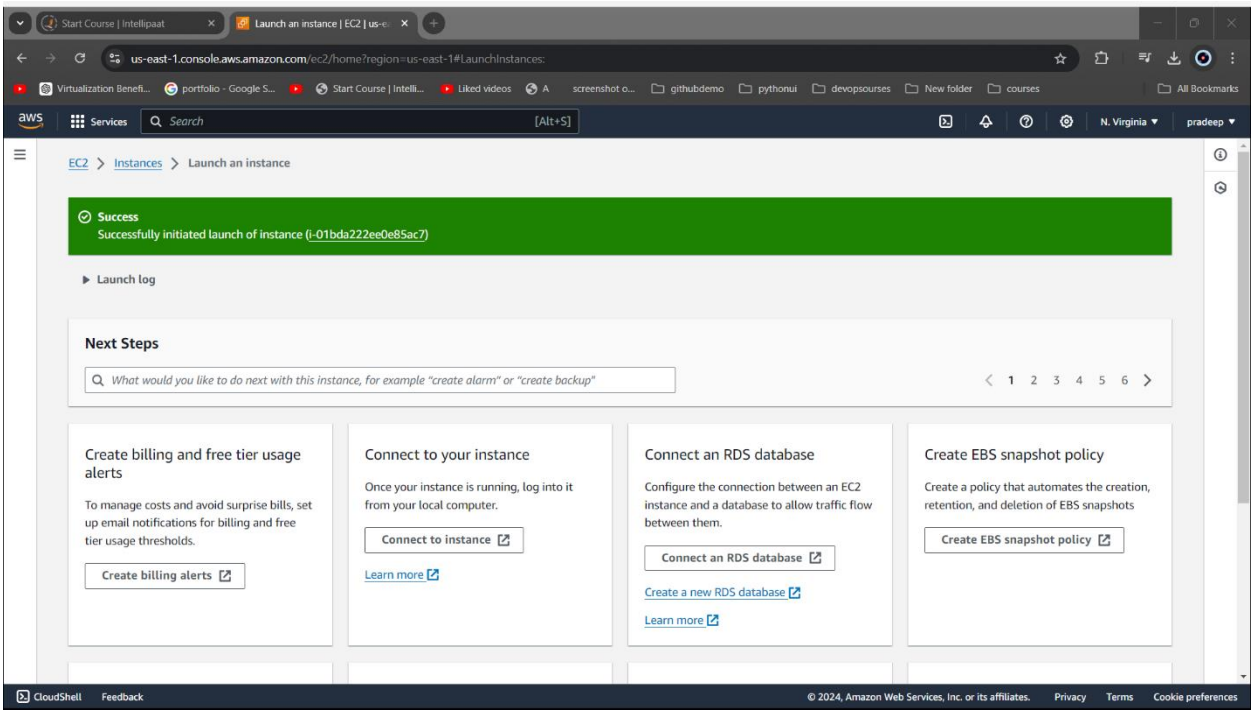


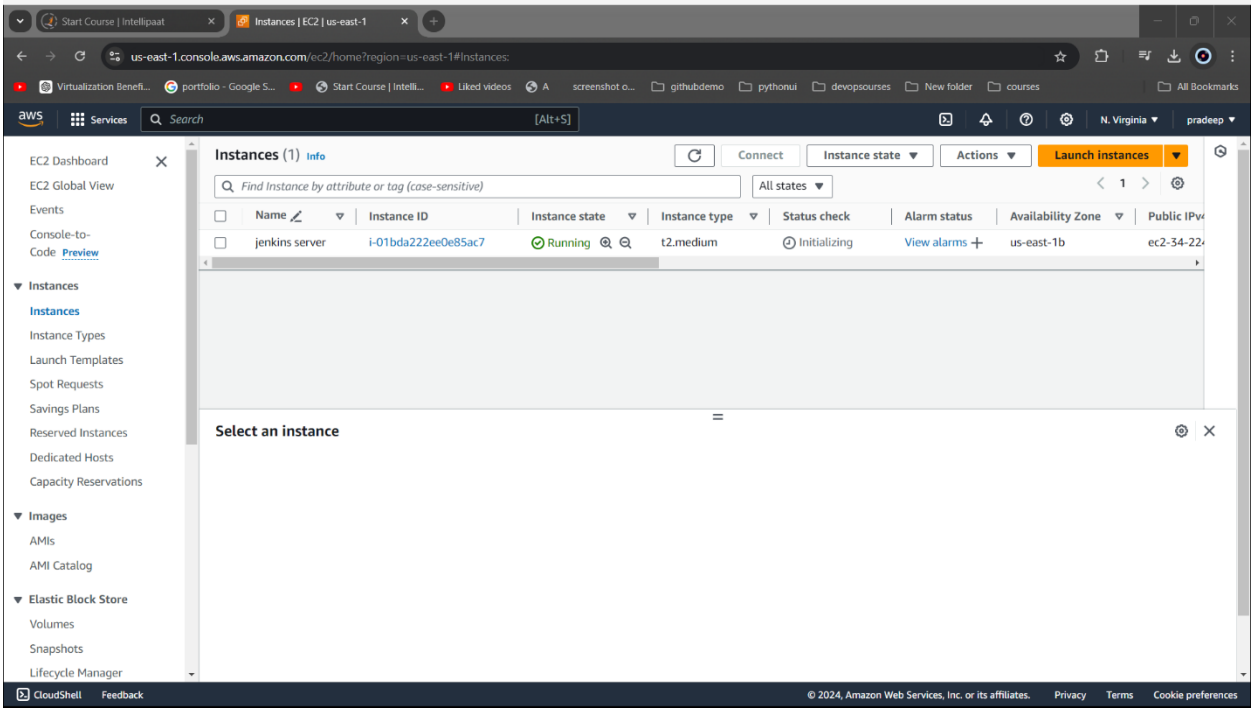
Configure storage:

storage is set to be 20 gb.



Launching the instance.



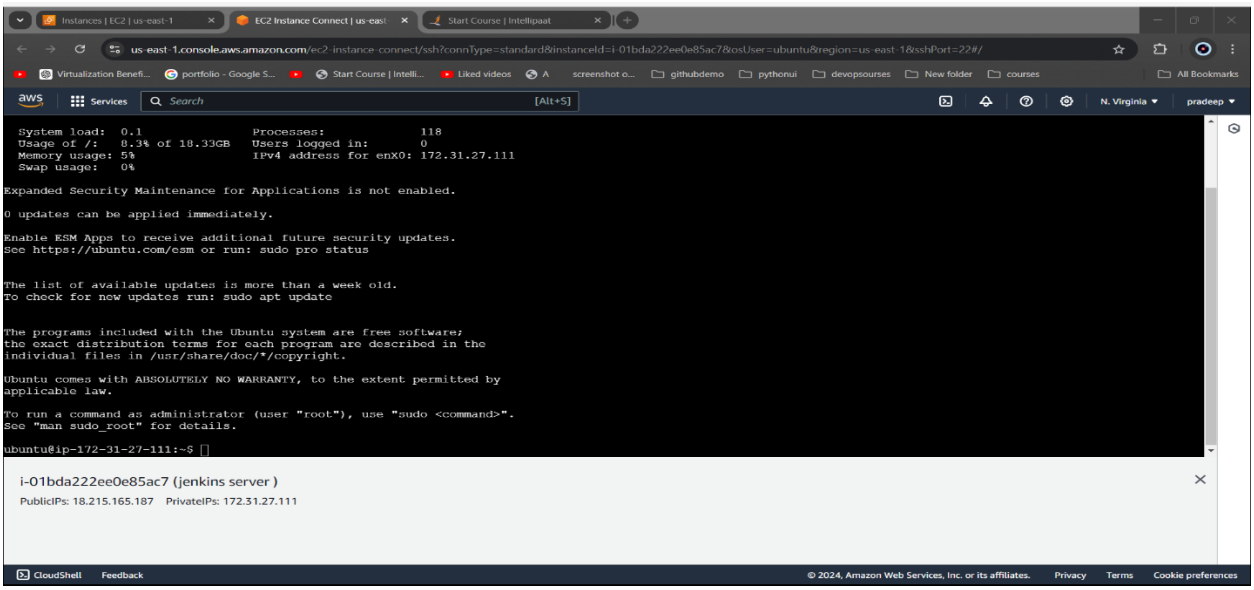


Jenkins server in created.

So, the creation of the EC2 instance for Jenkins is complete. Now, Jenkins needs to be installed, and the Jenkins dashboard needs to be set up."

Connecting to Jenkins server:

Select the jenkins server - > connect.



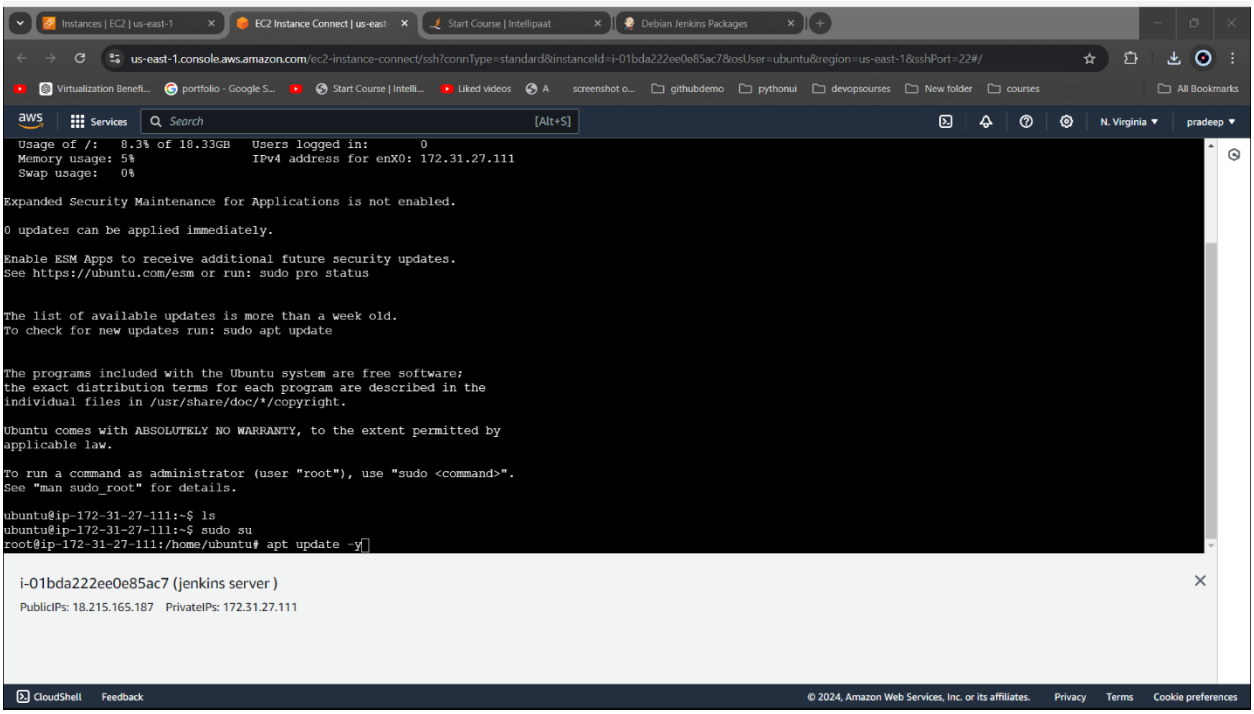
<u>To install and set up a Jenkins server, follow these steps:</u>

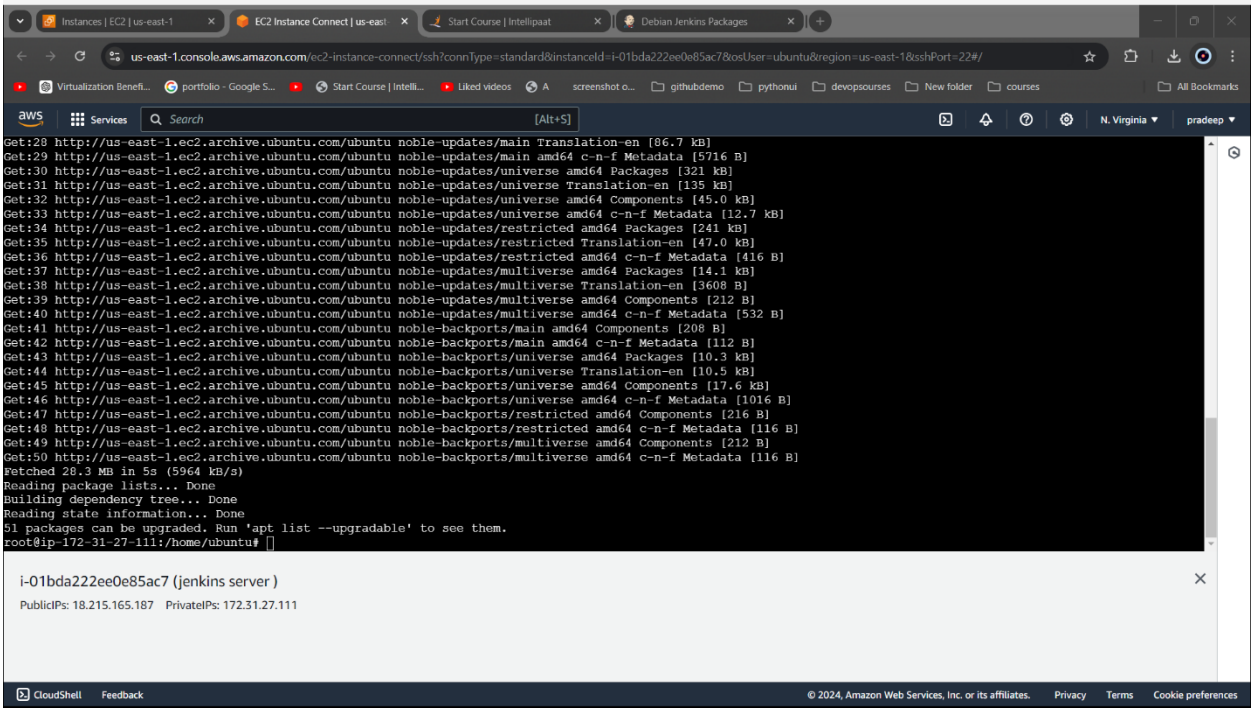
<u>Install Java:</u>

Jenkins requires Java to run, so you need to install it first.

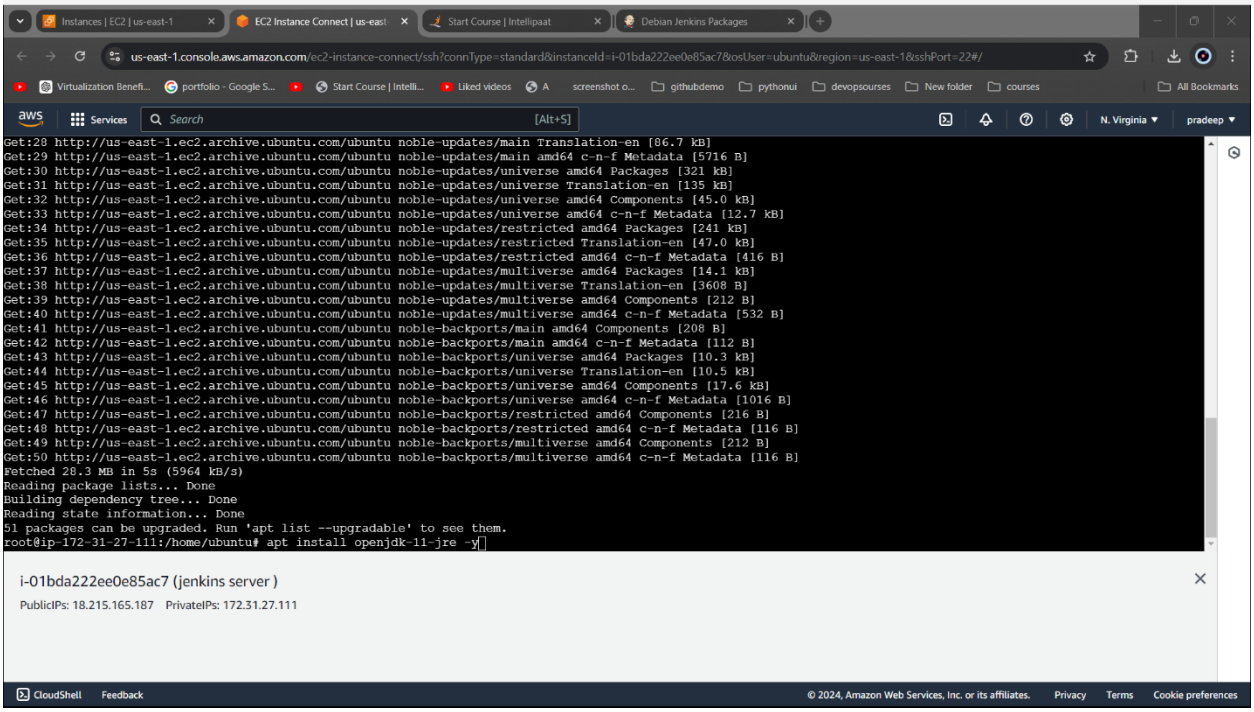
Commands:

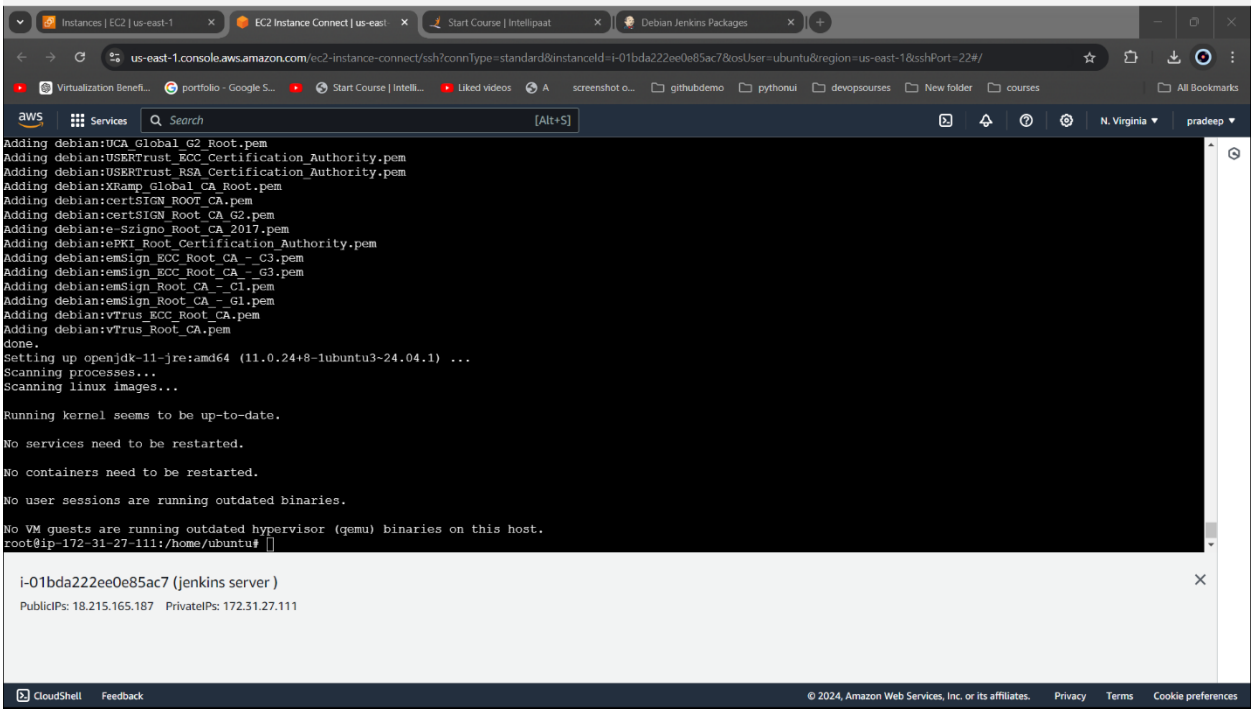
- **➔ Sudo apt update -y**





➔ **Sudo apt install openjdk-11-jre**

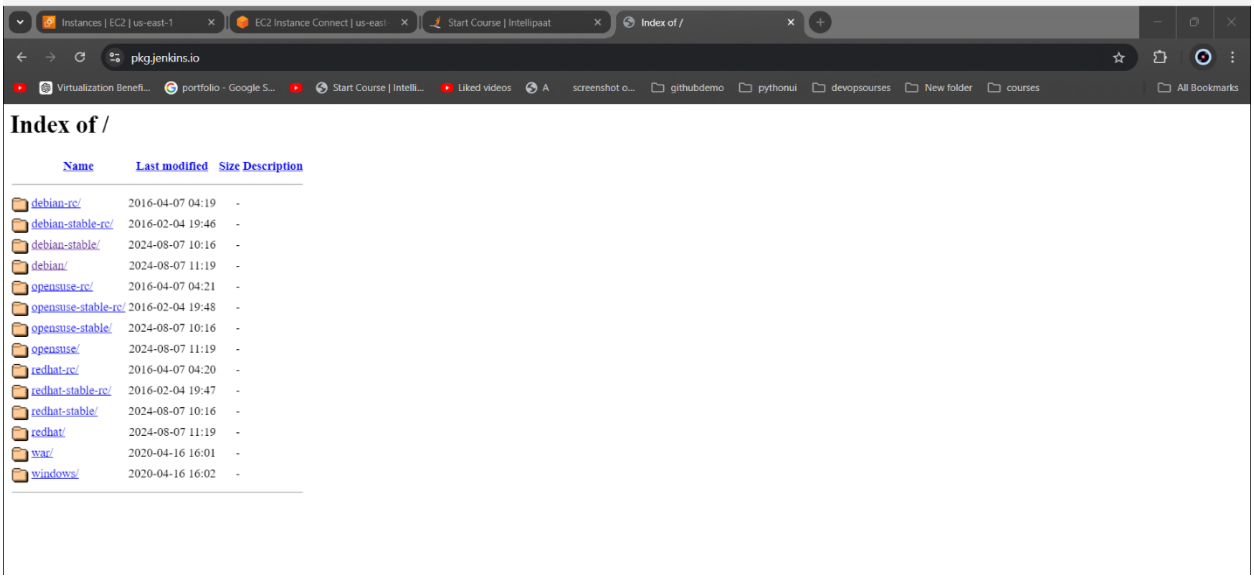




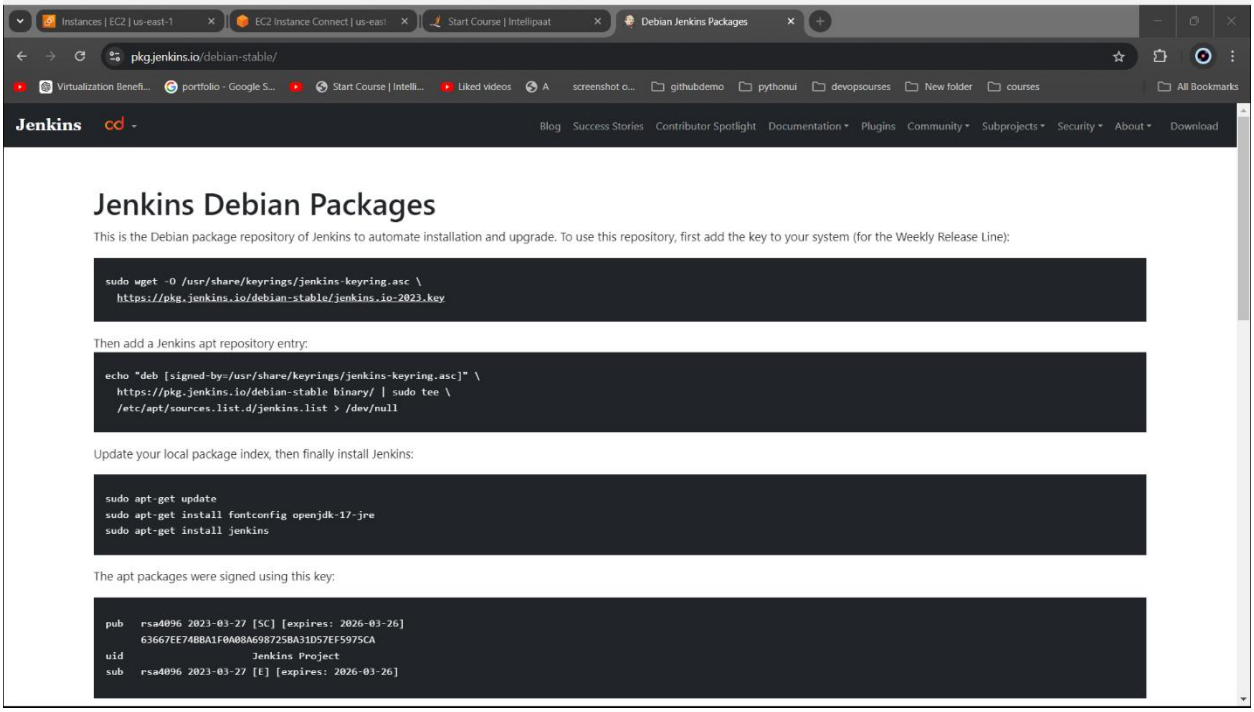
Add Jenkins Repository:

We need to add the Jenkins repository to install the latest version.

We can use the “pkg.jenkins.io” this site for installation for repo.



We are using ubuntu so it is from Debian family of linux. And we can need select the option called.
-> “debian-stable/”



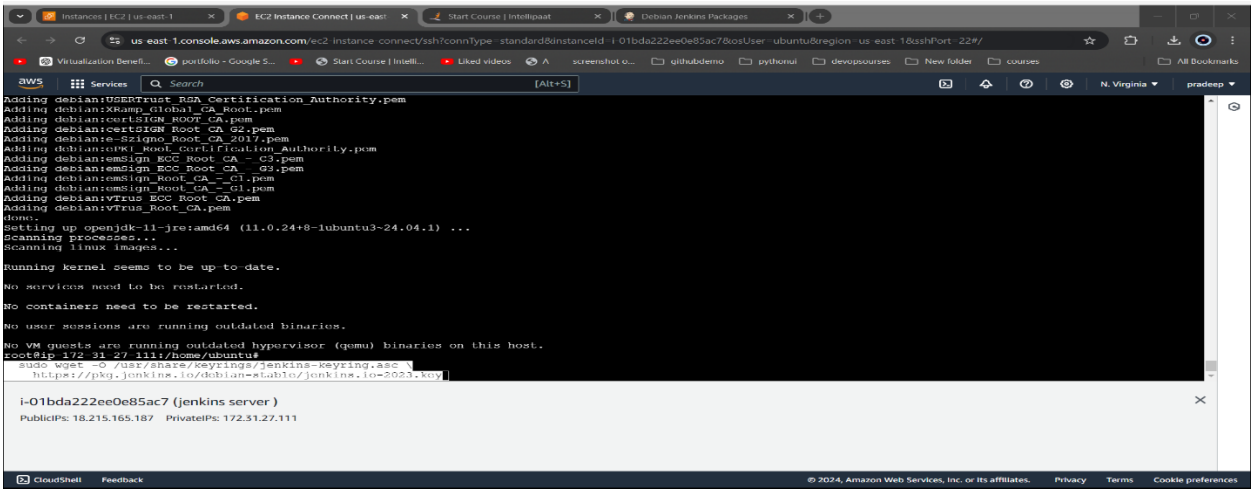
This is the Debian package repository of Jenkins to automate installation and upgrade.

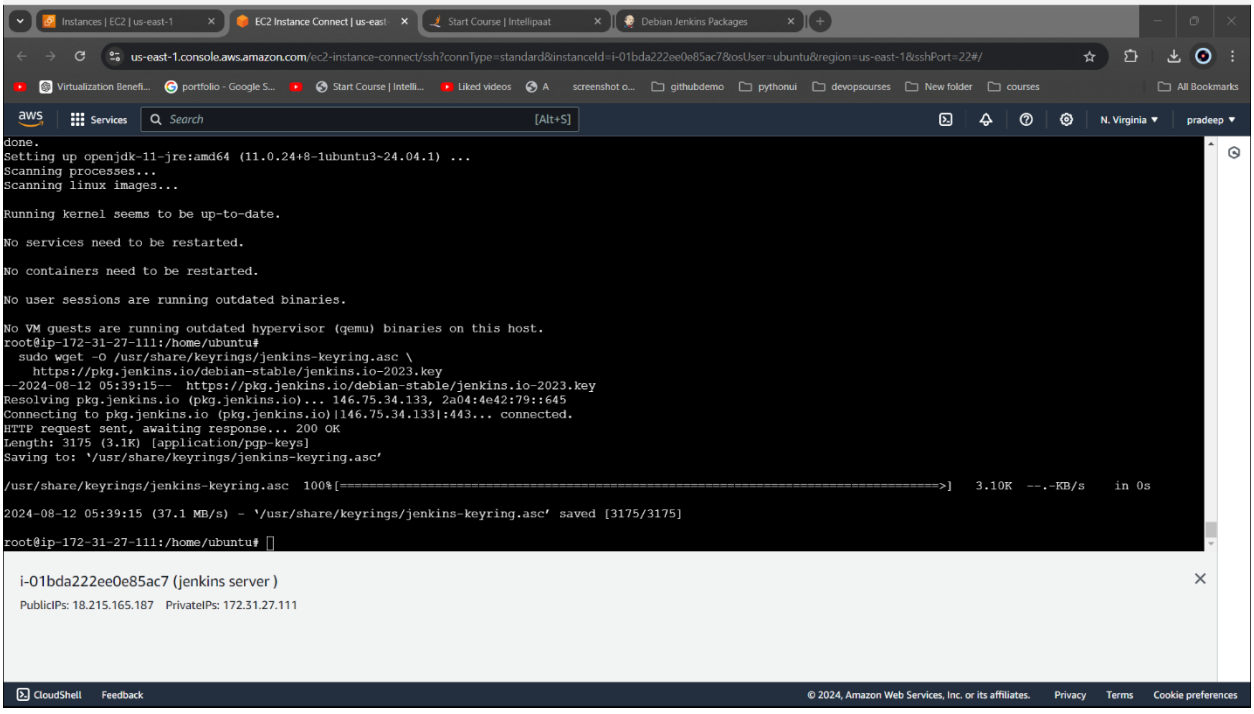
➔ sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \

**sudo wget -O /usr/share/keyrings/jenkins-keyring.asc **

https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

This command is essential for adding the Jenkins package repository key to the system's keyring, enabling the system to verify the authenticity of the Jenkins packages during installation or updates.

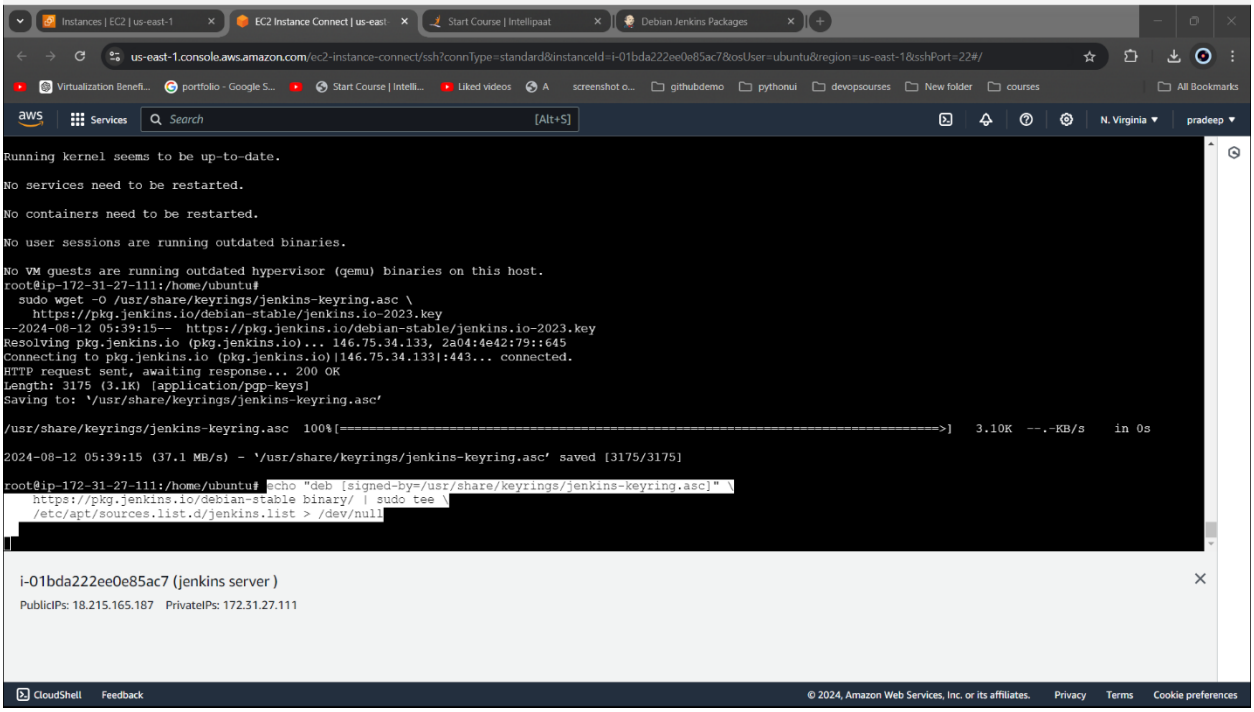


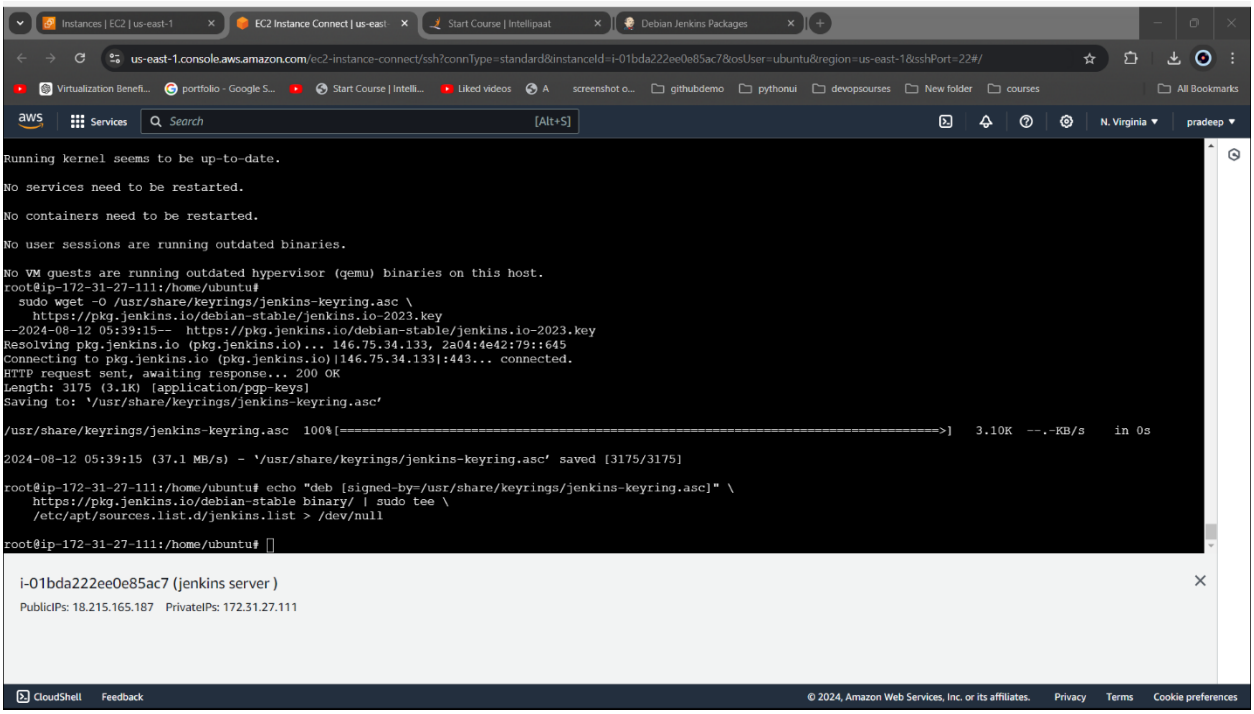


Then add a Jenkins apt repository entry:

➔ echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \

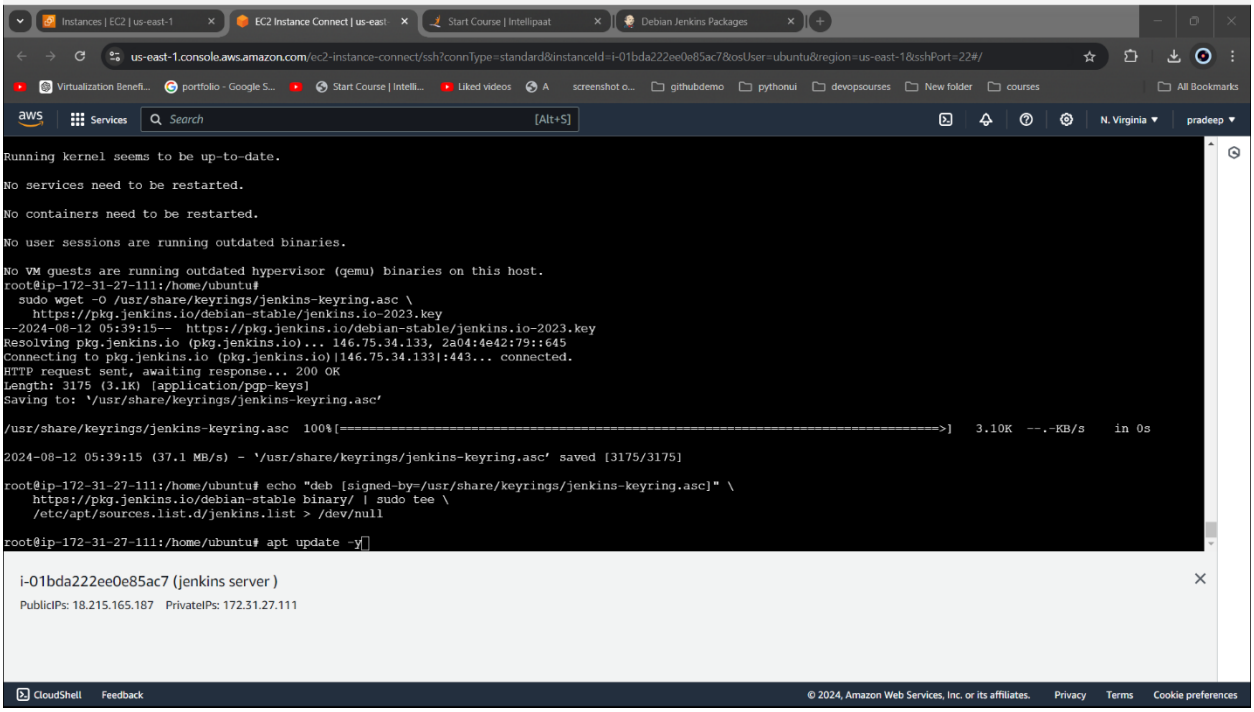
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null

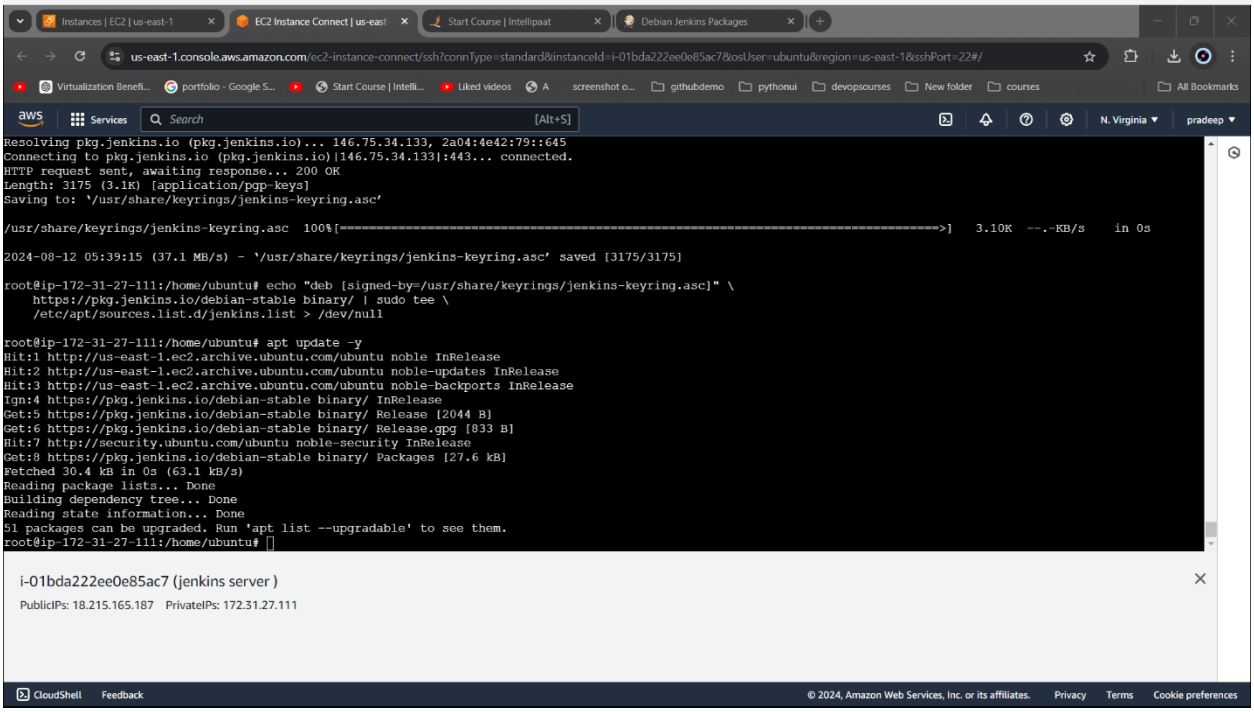




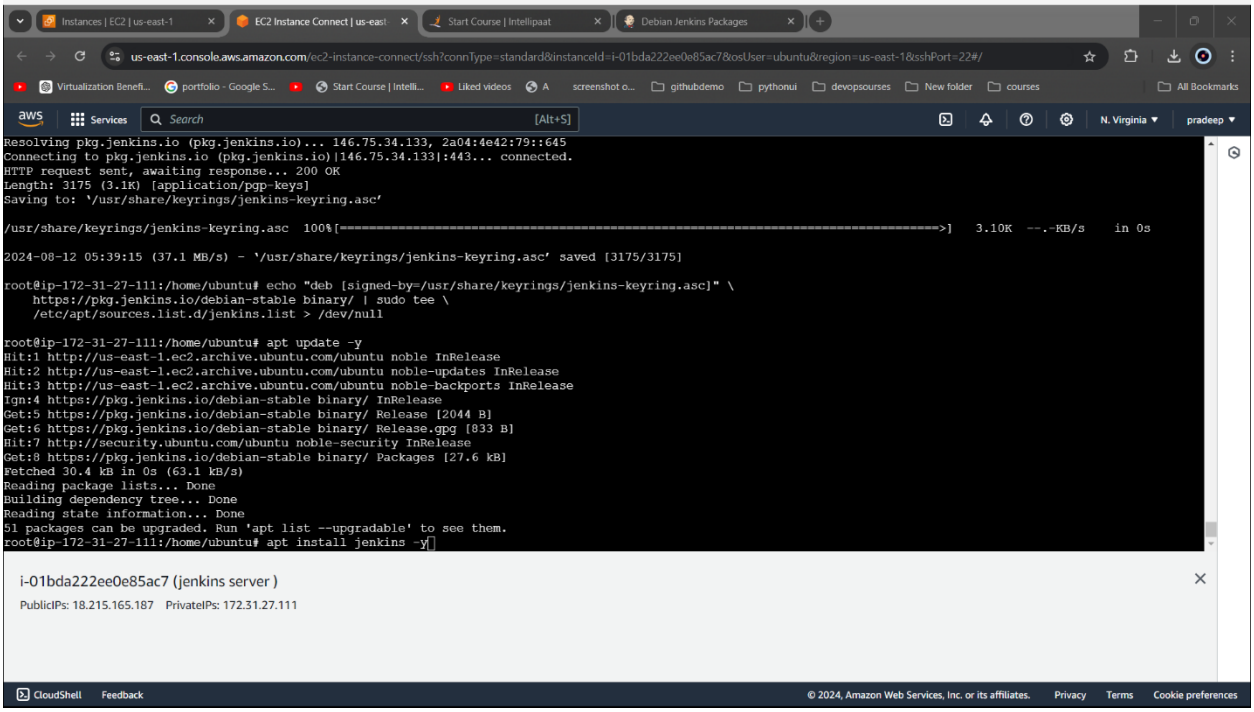
Update your local package index, then finally install Jenkins:

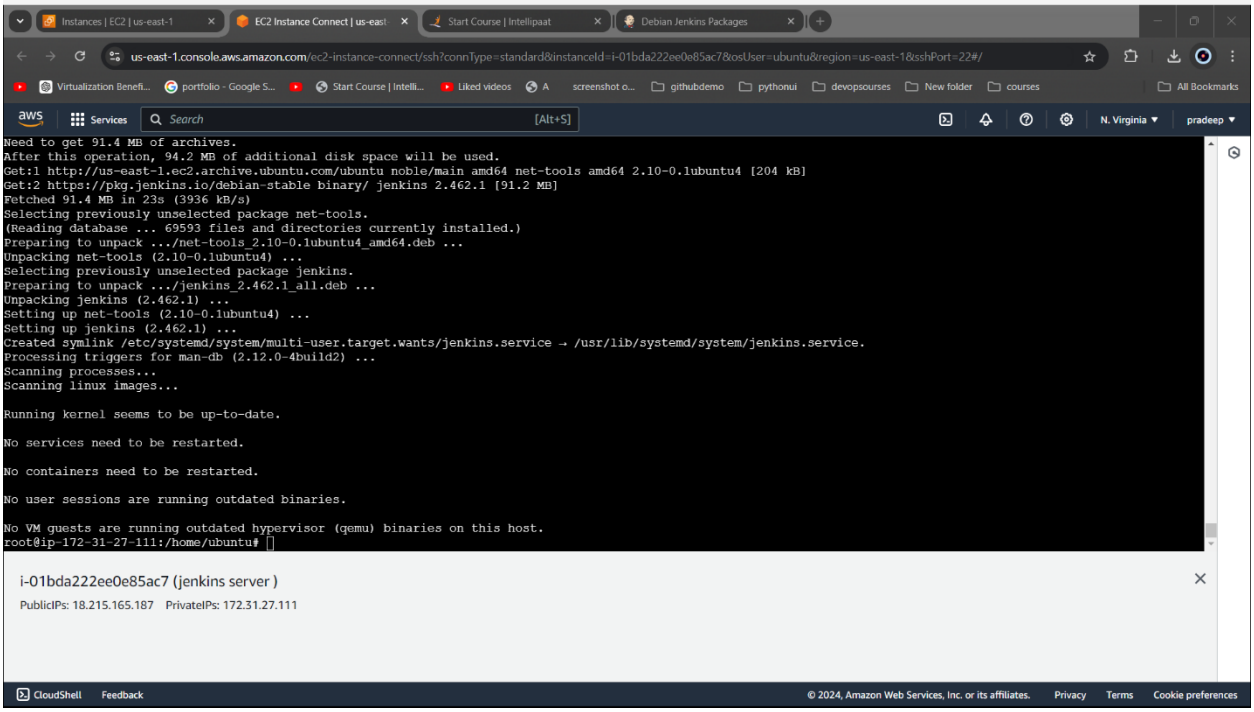
➔ sudo apt-get update





➔ sudo apt-get install jenkins

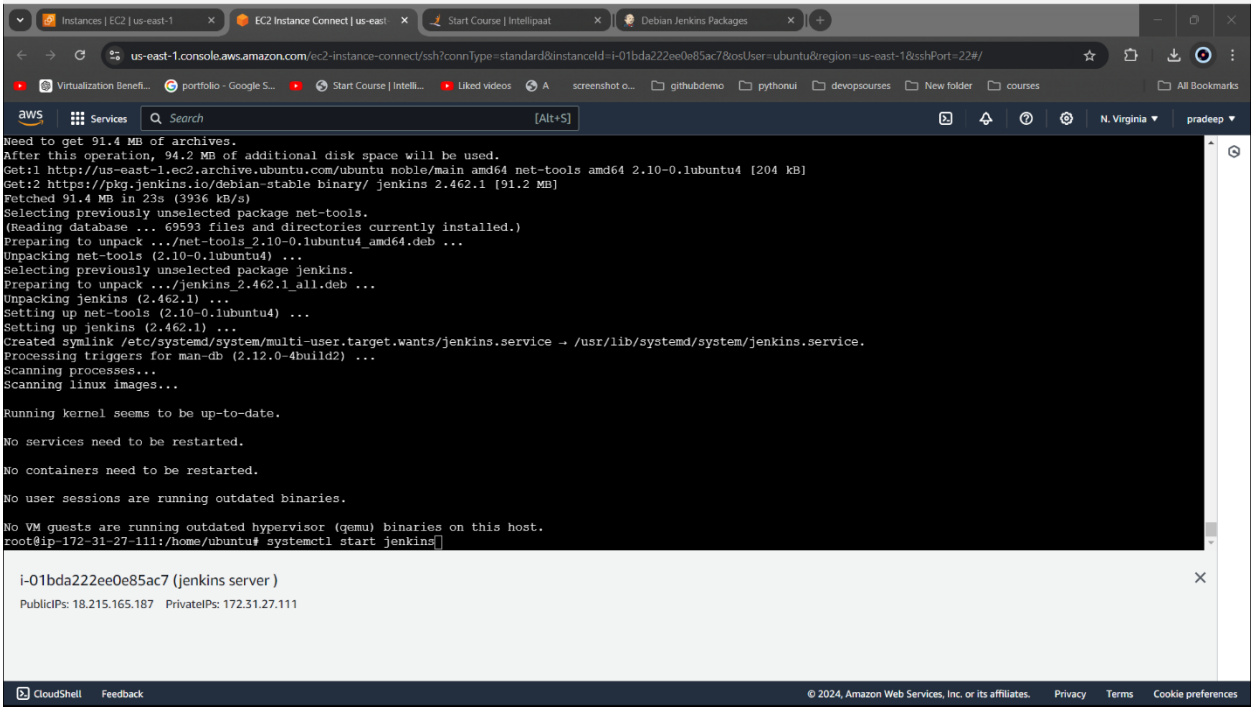


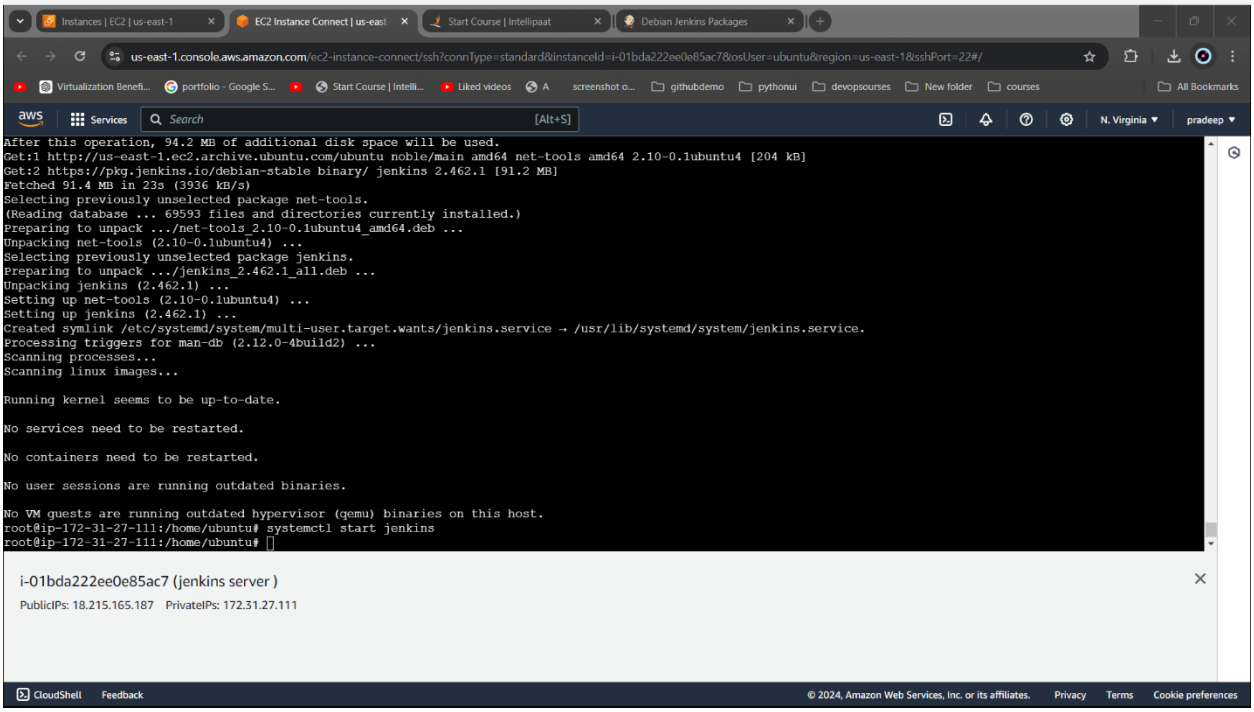


Start Jenkins:

Start the Jenkins service.

➔ sudo systemctl start jenkins

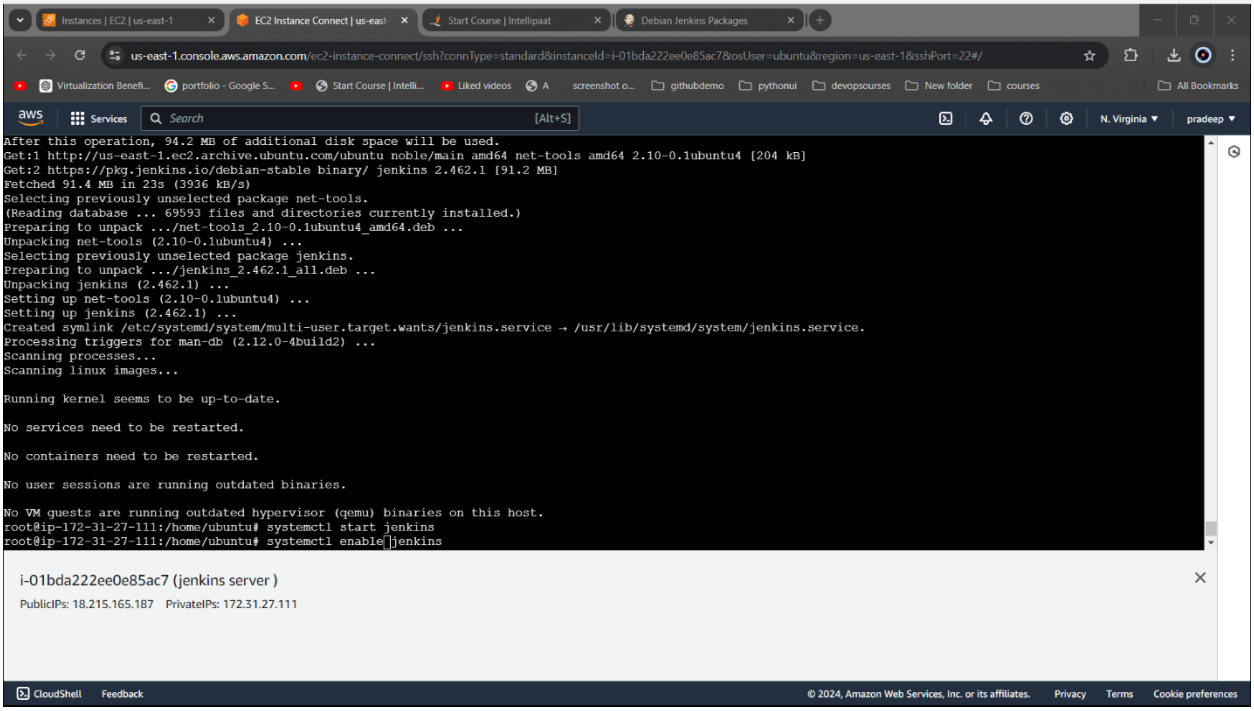


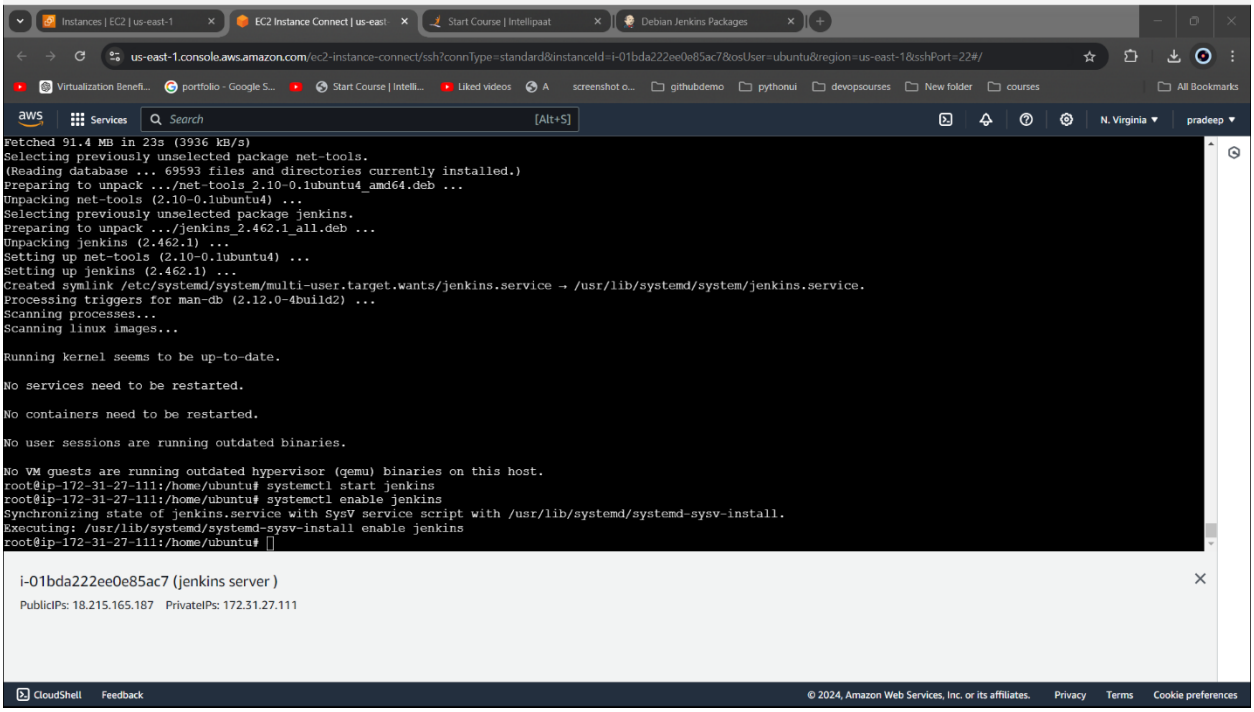


Enable Jenkins to Start on Boot:

To ensure Jenkins starts on boot, enable the service.

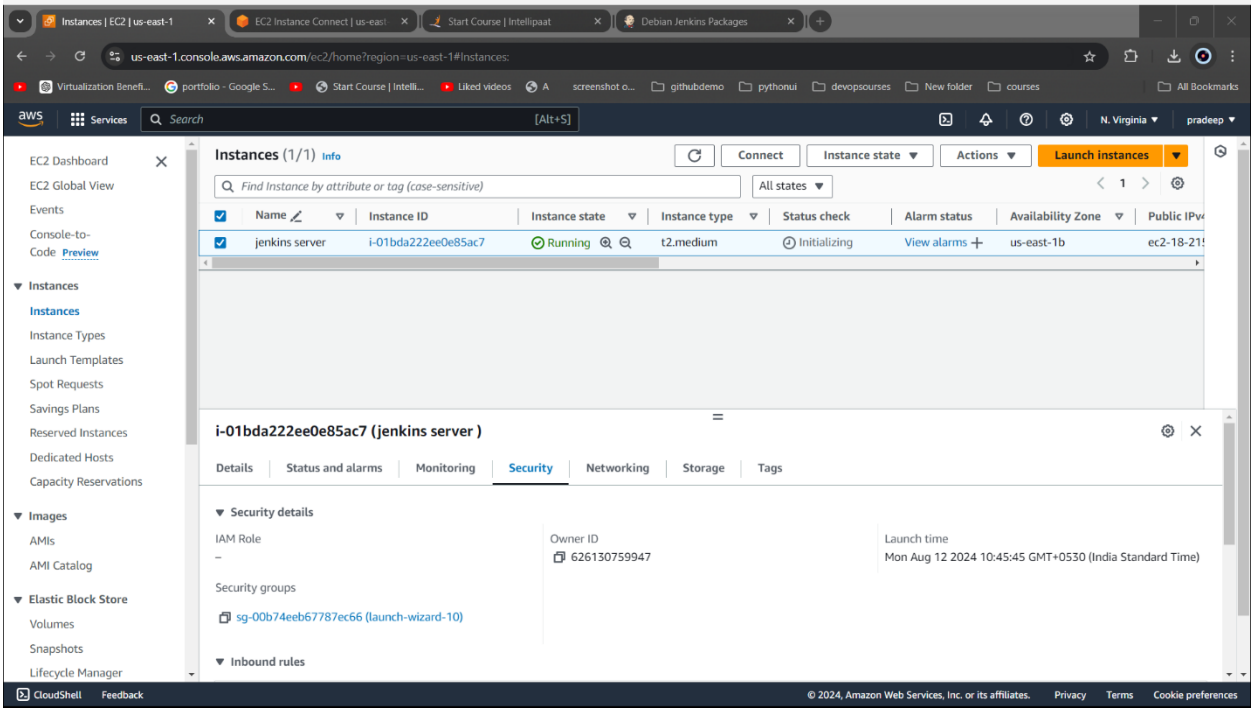
➔ sudo systemctl enable jenkins



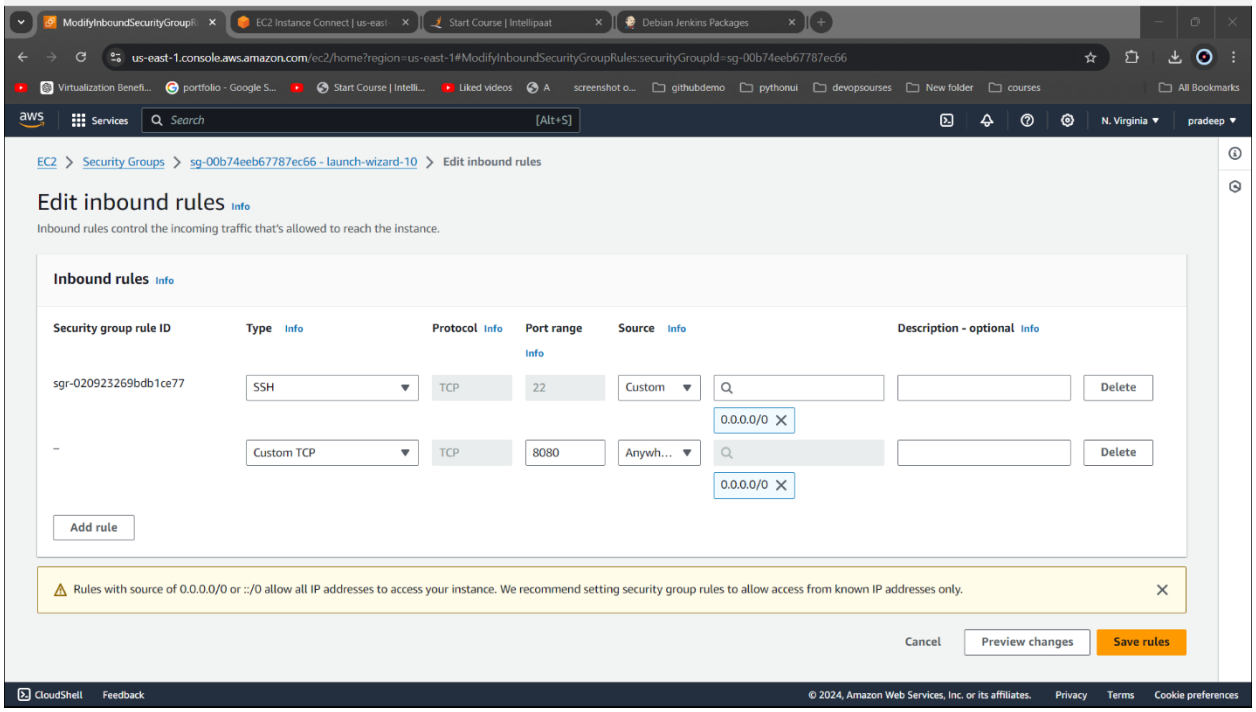


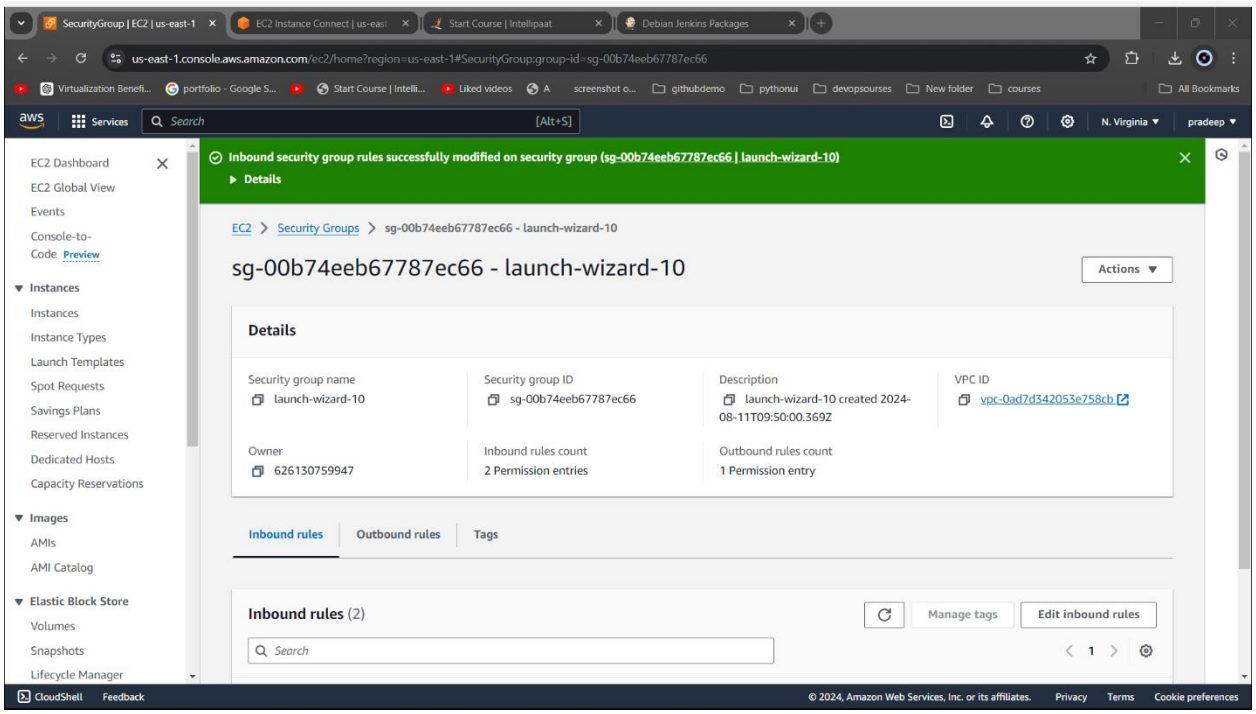
Adjust Firewall:

If you have a firewall running, you'll need to allow traffic on port 8080 in security group.



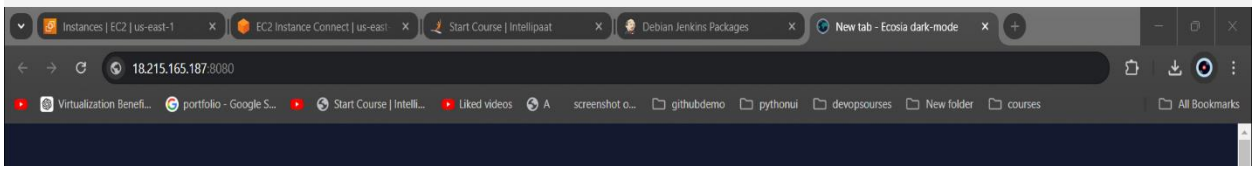
➔ **Inbound rules** -> Edit **Inbound rules** Inbound rules->add rule.

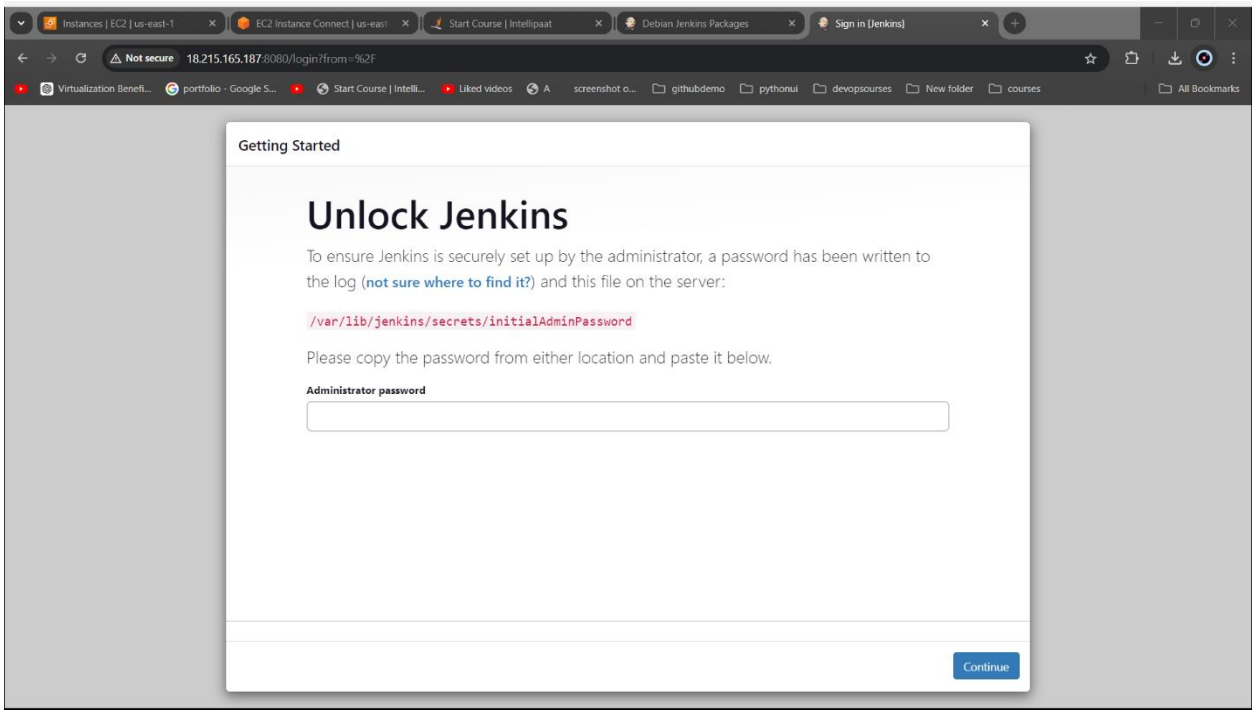




Access Jenkins:

Jenkins runs on port 8080 by default. You can access it by going to http://server_ip:8080.

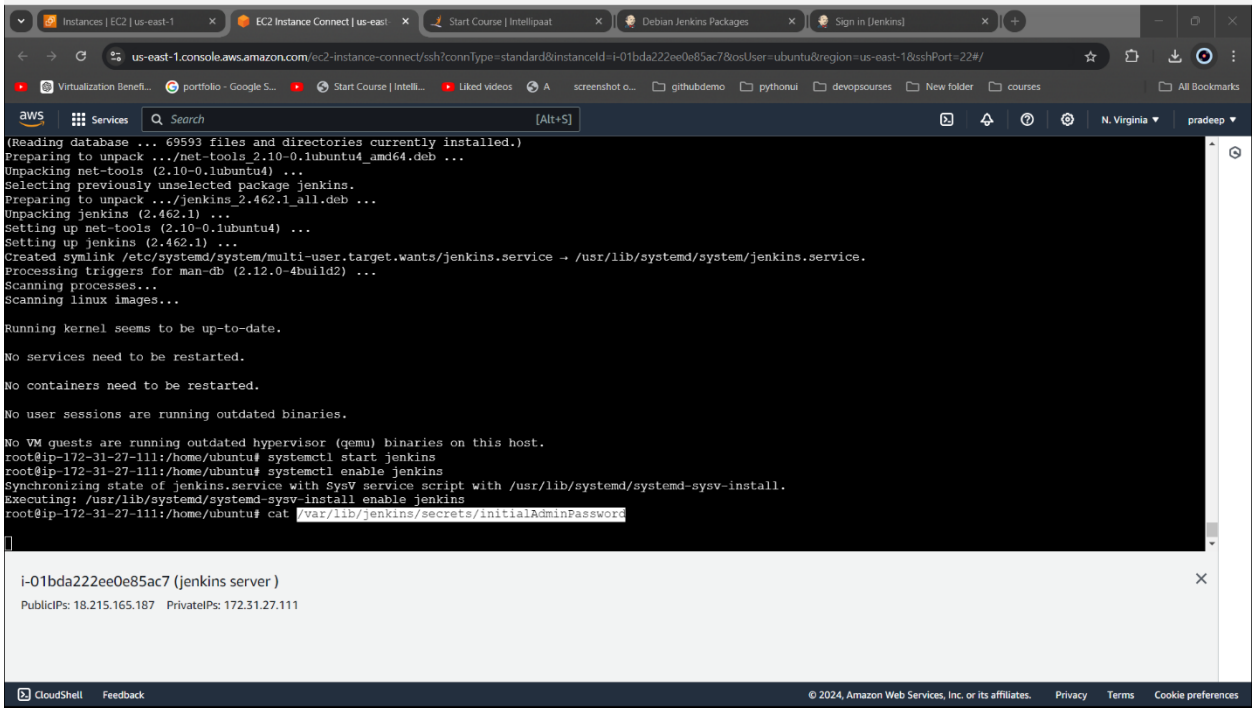


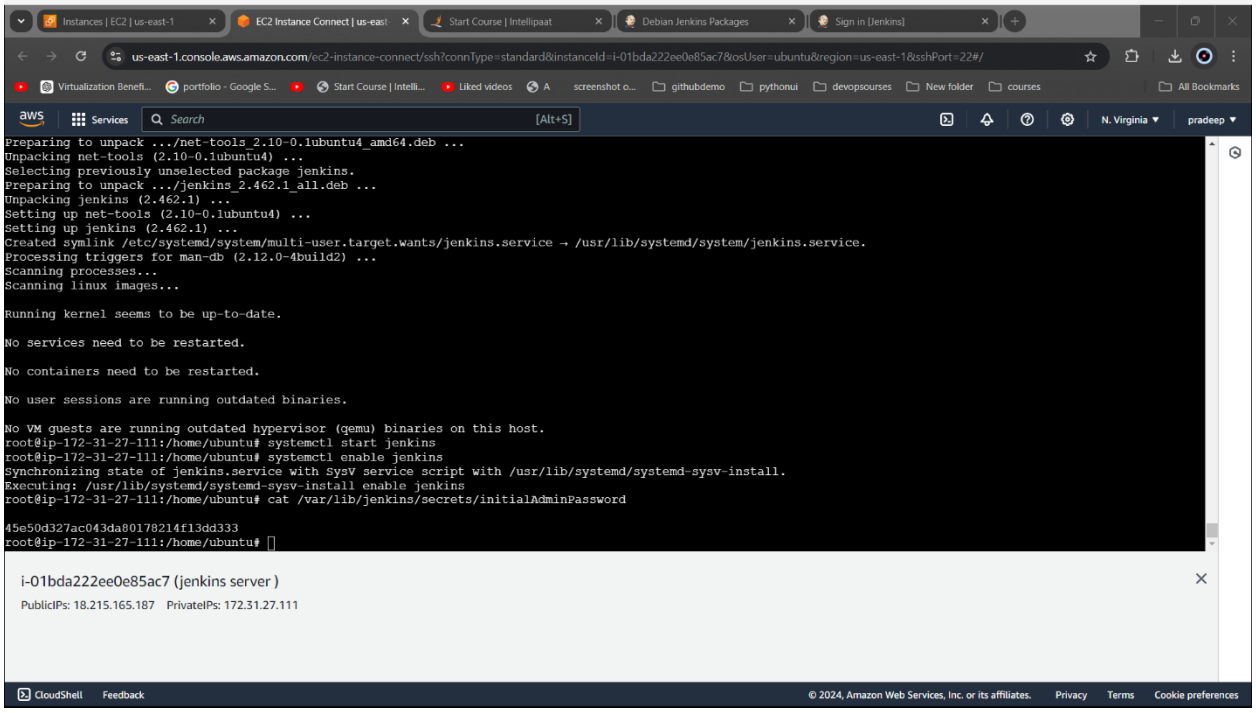


Unlock Jenkins:

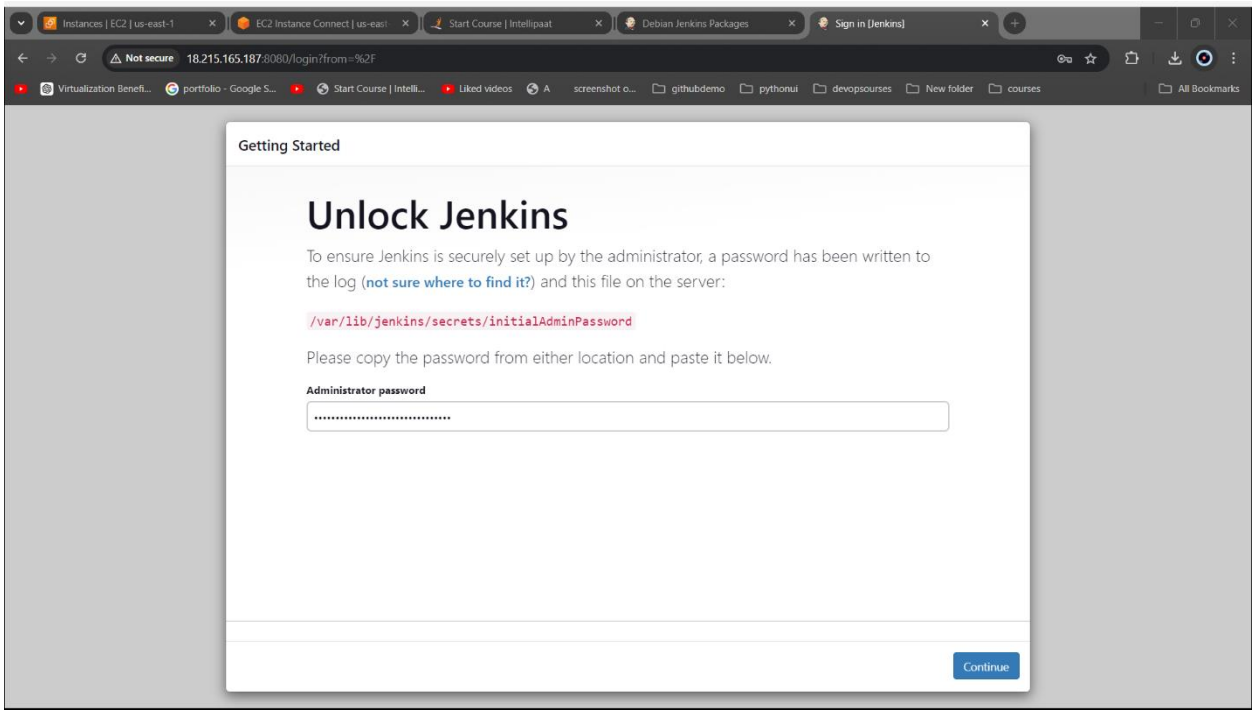
When we first access Jenkins, it will ask for an initial administrative password. We can find this password by running:

➔ sudo cat /var/lib/jenkins/secrets/initialAdminPassword

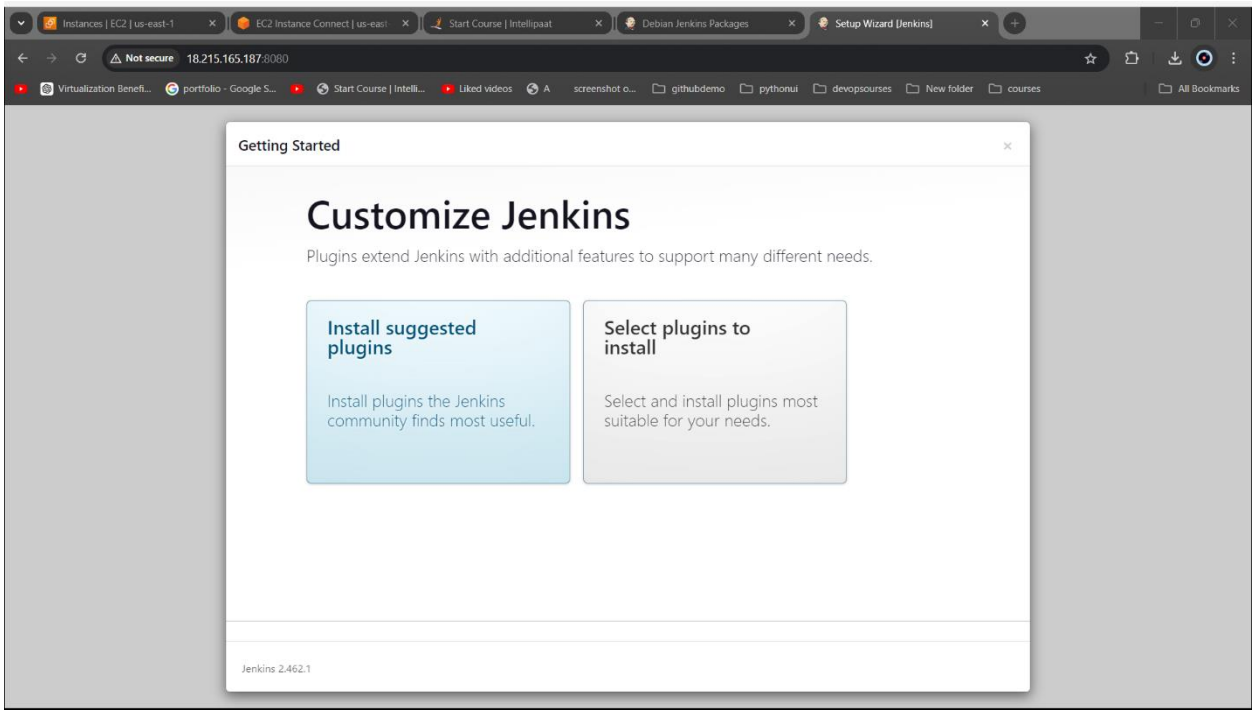

Instances | EC2 | us-east-1
EC2 Instance Connect | us-east
Start Course | Intellipaat
Debian Jenkins Packages
Sign in (Jenkins)
us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh?connType=standard&instanceId=i-01bda222ee0e85ac7&osUser=ubuntu&region=us-east-1&sshPort=22#/
Virtualization Benefi...
portfolio - Google S...
Start Course | Intelli...
Liked videos
A
screenshot o...
githubdemo
pythonui
devopsourses
New folder
courses
All Bookmarks
aws
Services
Search
[Alt+S]
N. Virginia
pradeep
(Reading database ... 69593 files and directories currently installed.)
Preparing to unpack .../net-tools_2.10-0.1ubuntu4_amd64.deb ...
Unpacking net-tools (2.10-0.1ubuntu4) ...
Selecting previously unselected package jenkins.
Preparing to unpack .../jenkins_2.462.1_all.deb ...
Unpacking jenkins (2.462.1) ...
Setting up net-tools (2.10-0.1ubuntu4) ...
Setting up jenkins (2.462.1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/jenkins.service → /usr/lib/systemd/system/jenkins.service.
Processing triggers for man-db (2.12.0-4build2) ...
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-172-31-27-111:/home/ubuntu# systemctl start jenkins
root@ip-172-31-27-111:/home/ubuntu# systemctl enable jenkins
Synchronizing state of jenkins.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable jenkins
root@ip-172-31-27-111:/home/ubuntu# cat /var/lib/jenkins/secrets/initialAdminPassword
i-01bda222ee0e85ac7 (jenkins server)
PublicIPs: 18.215.165.187 PrivateIPs: 172.31.27.111
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

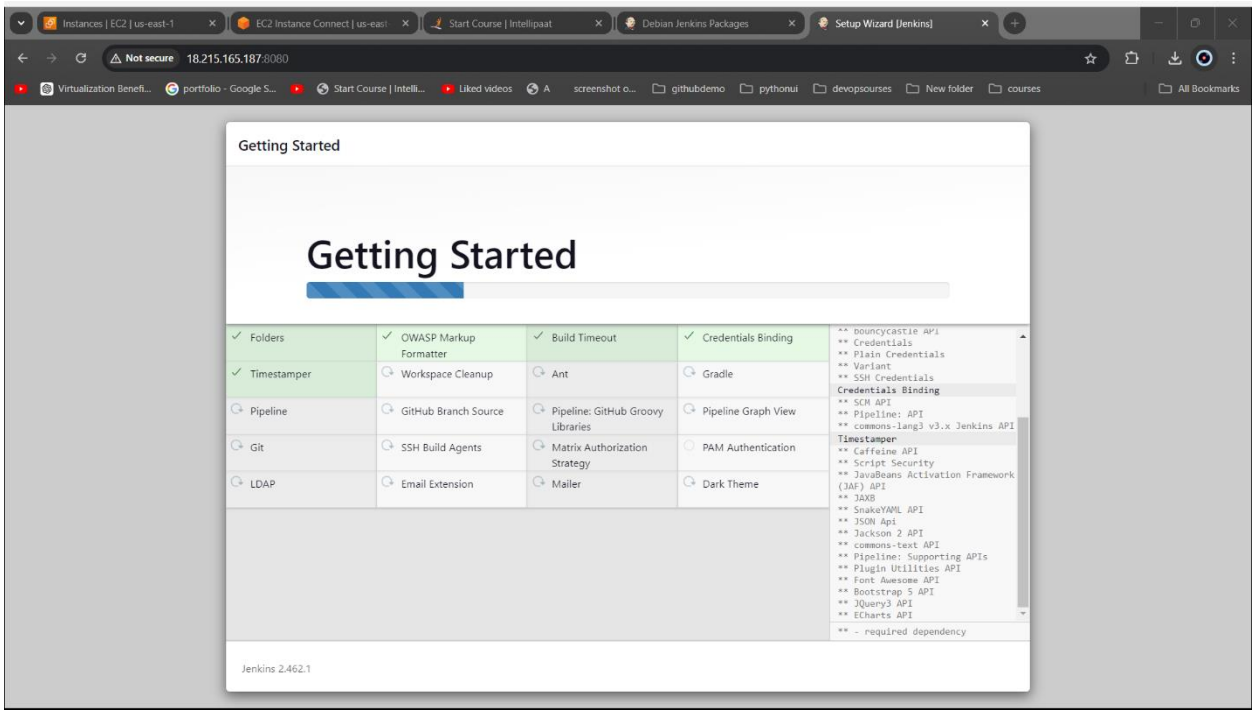

Instances | EC2 | us-east-1
EC2 Instance Connect | us-east
Start Course | Intellipaat
Debian Jenkins Packages
Sign in (Jenkins)
us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh?connType=standard&instanceId=i-01bda222ee0e85ac7&osUser=ubuntu&region=us-east-1&sshPort=22#/
Virtualization Benefi...
portfolio - Google S...
Start Course | Intelli...
Liked videos
A
screenshot o...
githubdemo
pythonui
devopsourses
New folder
courses
All Bookmarks
aws
Services
Search
[Alt+S]
N. Virginia
pradeep
Preparing to unpack .../net-tools_2.10-0.1ubuntu4_amd64.deb ...
Unpacking net-tools (2.10-0.1ubuntu4) ...
Selecting previously unselected package jenkins.
Preparing to unpack .../jenkins_2.462.1_all.deb ...
Unpacking jenkins (2.462.1) ...
Setting up net-tools (2.10-0.1ubuntu4) ...
Setting up jenkins (2.462.1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/jenkins.service → /usr/lib/systemd/system/jenkins.service.
Processing triggers for man-db (2.12.0-4build2) ...
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-172-31-27-111:/home/ubuntu# systemctl start jenkins
root@ip-172-31-27-111:/home/ubuntu# systemctl enable jenkins
Synchronizing state of jenkins.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable jenkins
root@ip-172-31-27-111:/home/ubuntu# cat /var/lib/jenkins/secrets/initialAdminPassword
45e50d327ac043da80178214f13dd333
root@ip-172-31-27-111:/home/ubuntu#
i-01bda222ee0e85ac7 (jenkins server)
PublicIPs: 18.215.165.187 PrivateIPs: 172.31.27.111
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Copy the password, paste it into the Jenkins interface

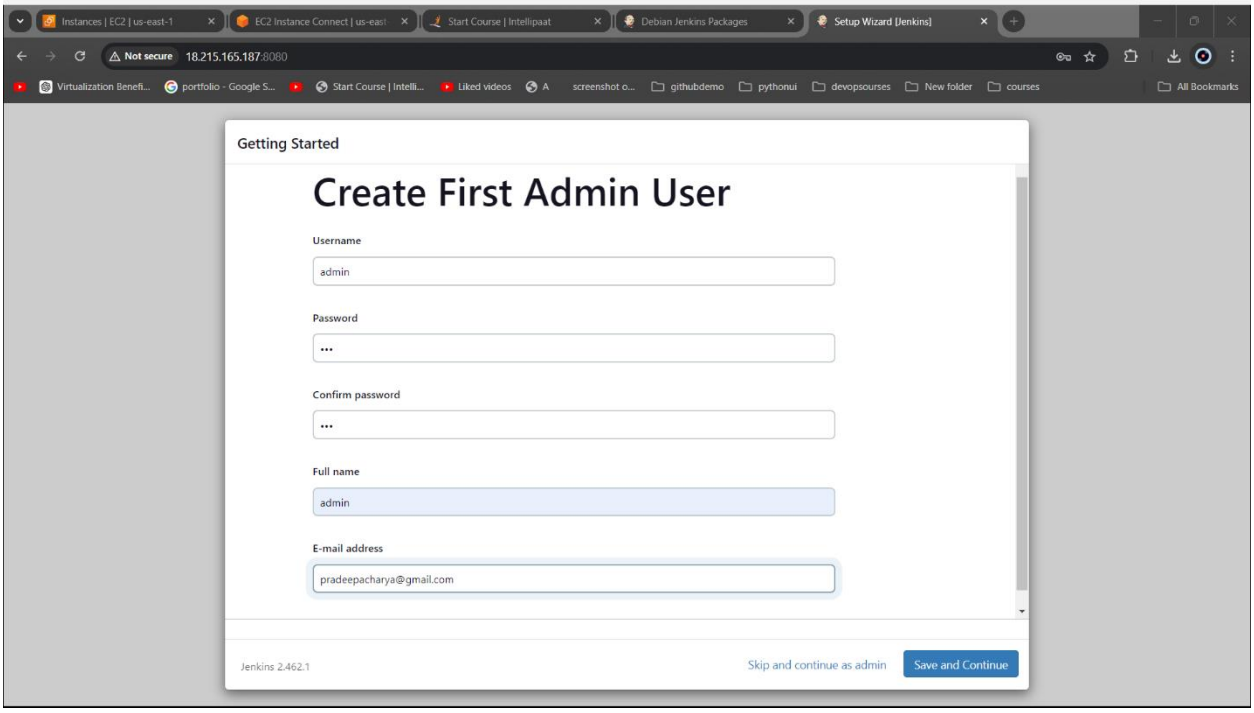


Choose customized jenkins:

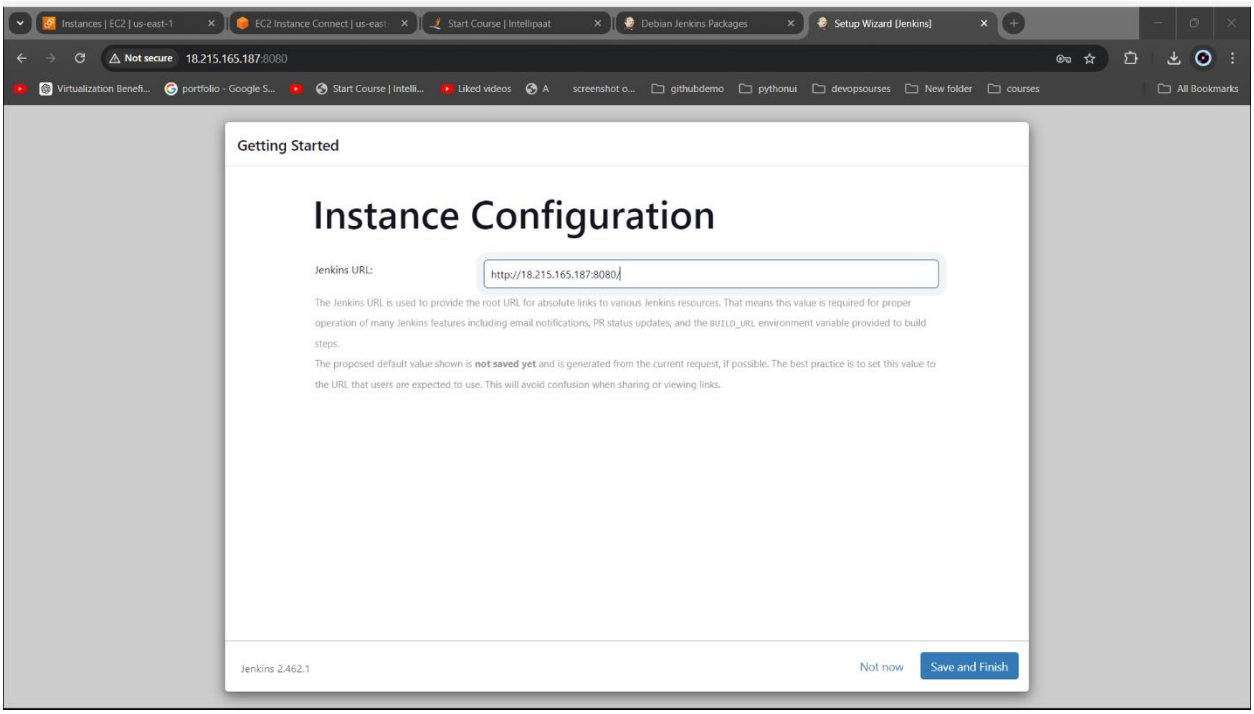


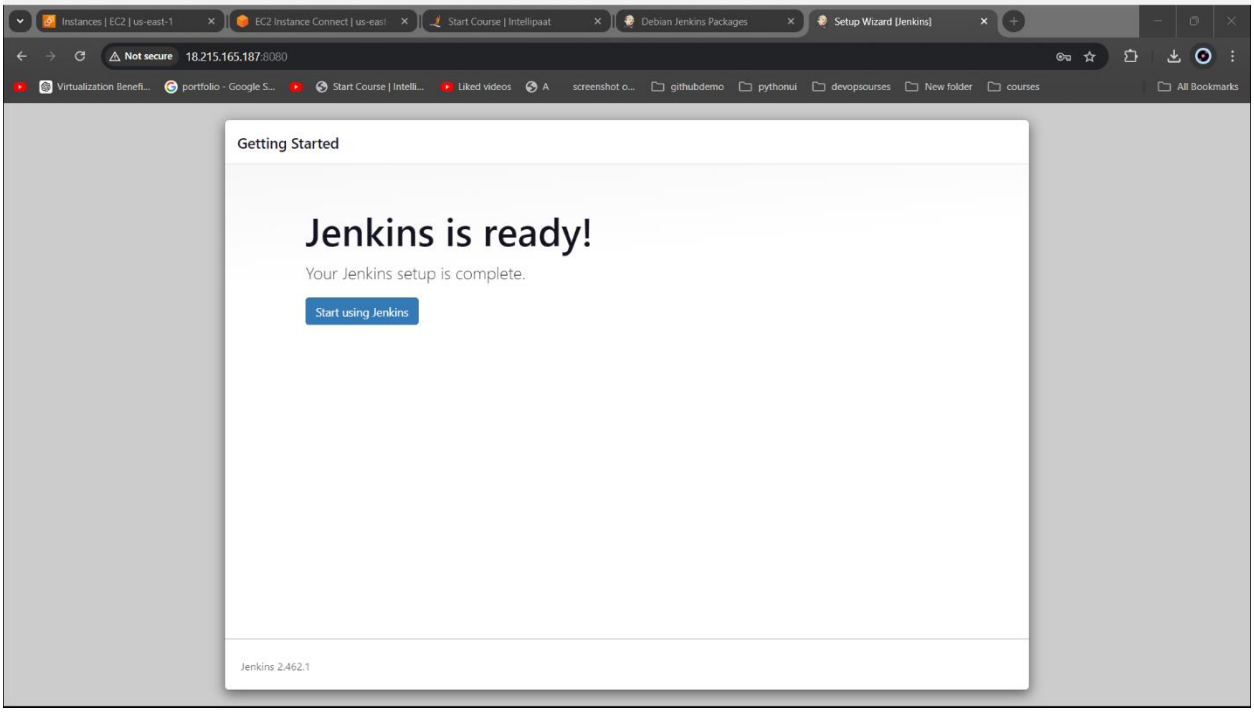


Creating admin user:



Save and continue.

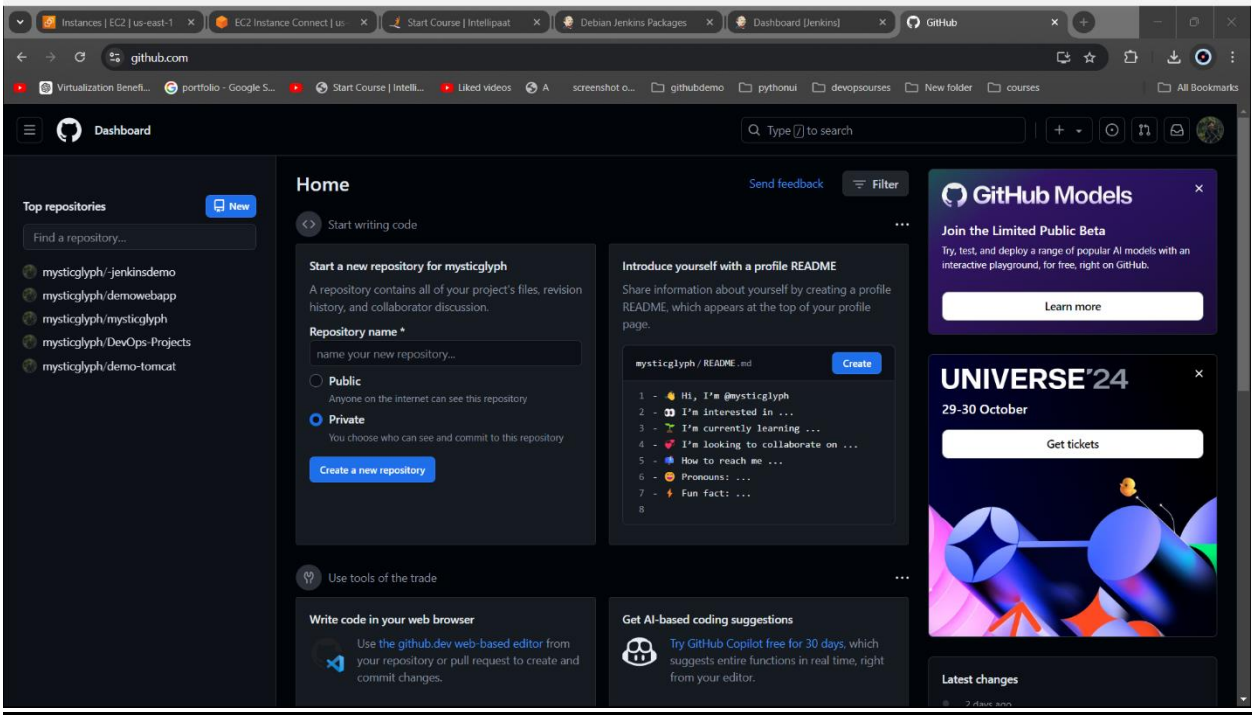


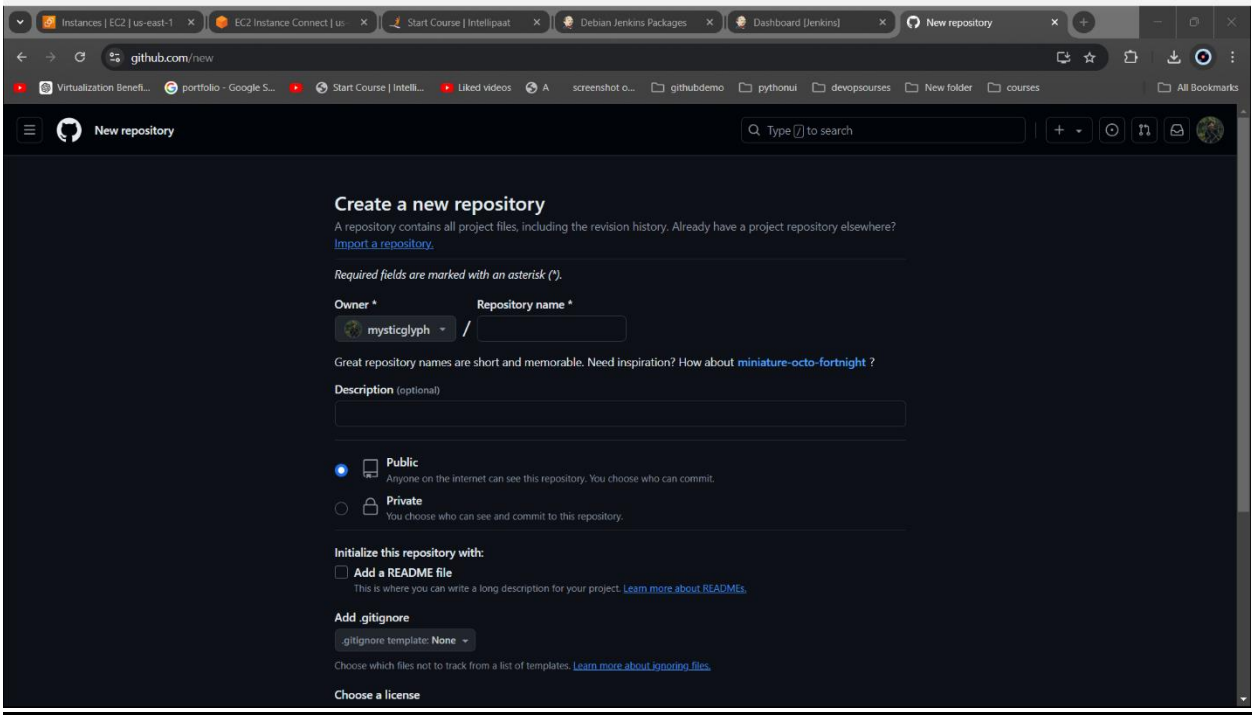


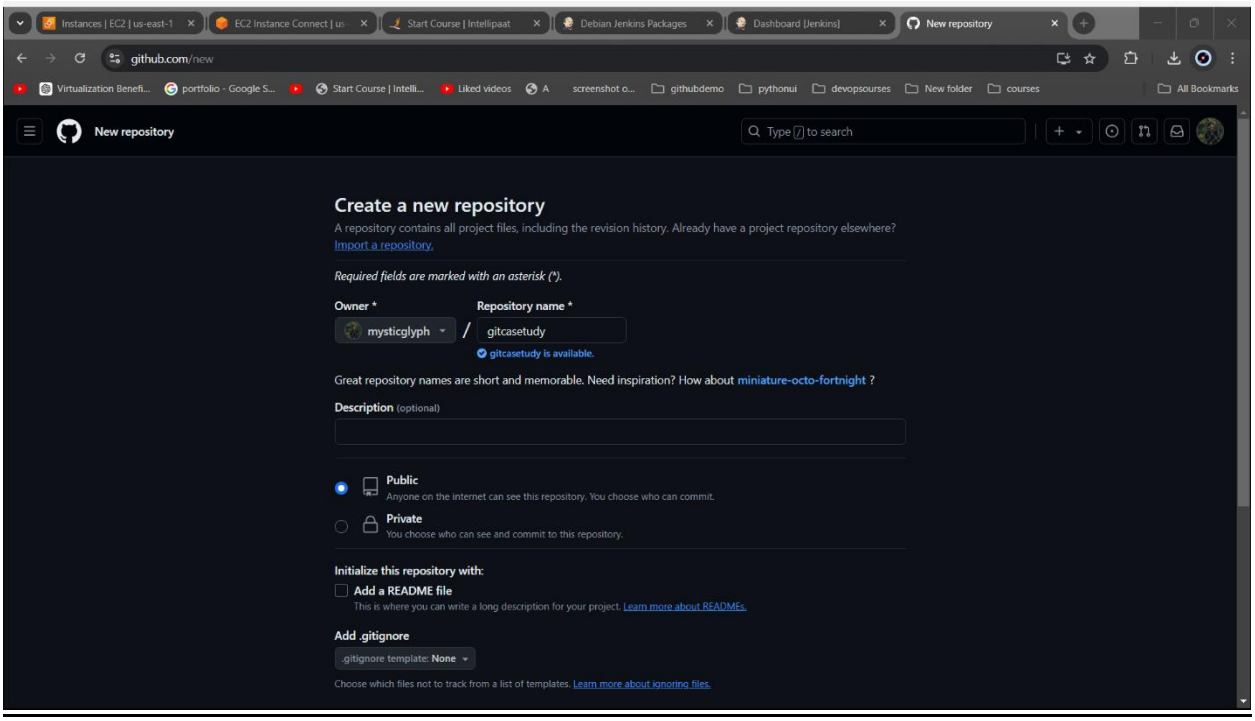
Jenkins server up and running.

Implement the Git Workflow for continuous integration (CI) in the product hosted at the GitHub repository.

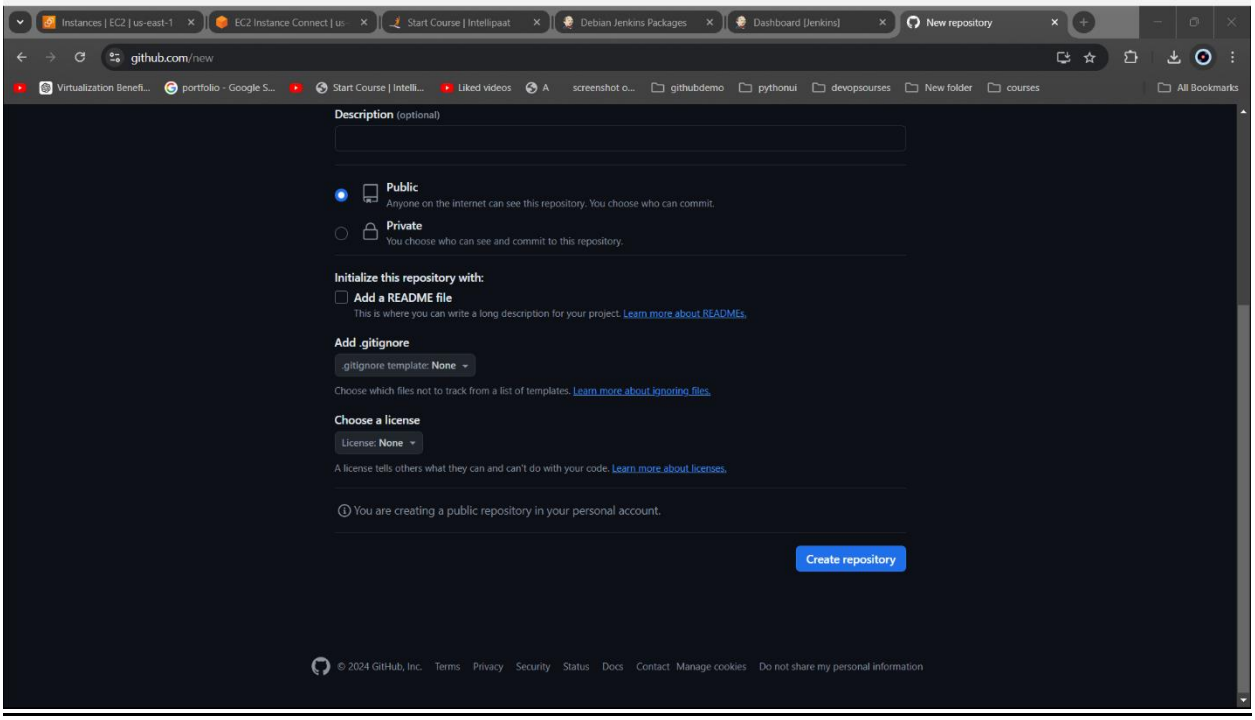
Creating a git repo in git hub:





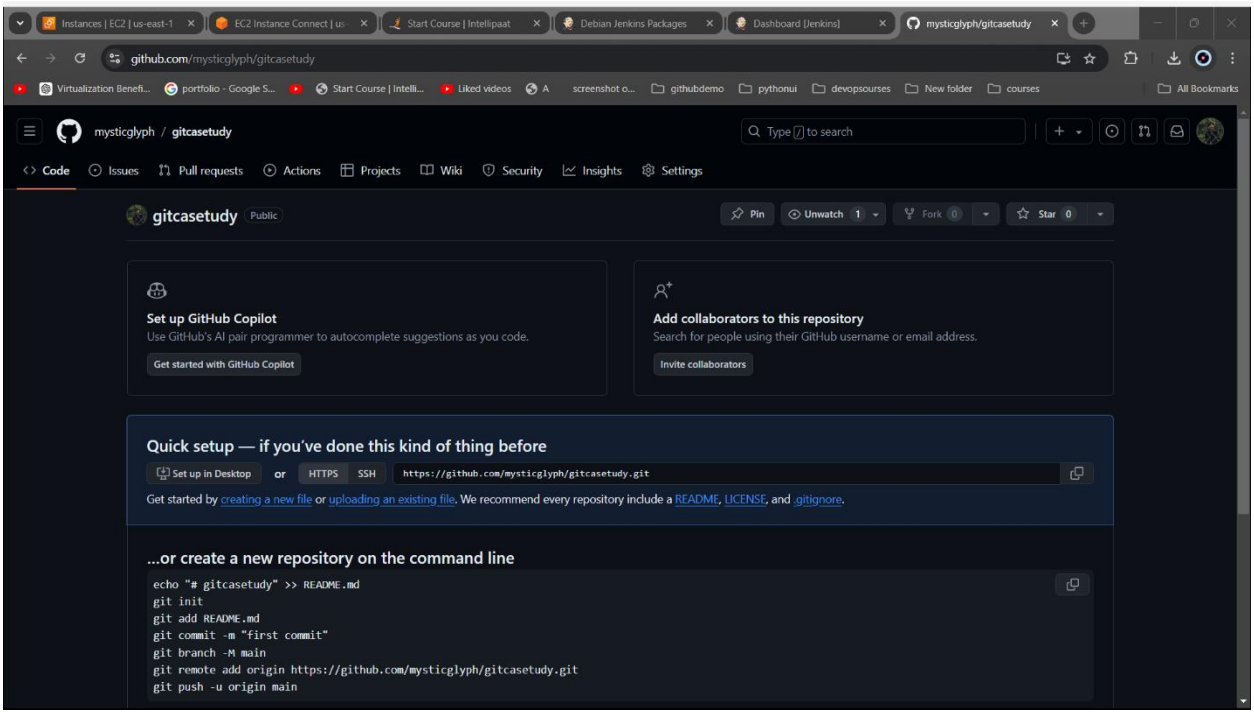


Repository is public:

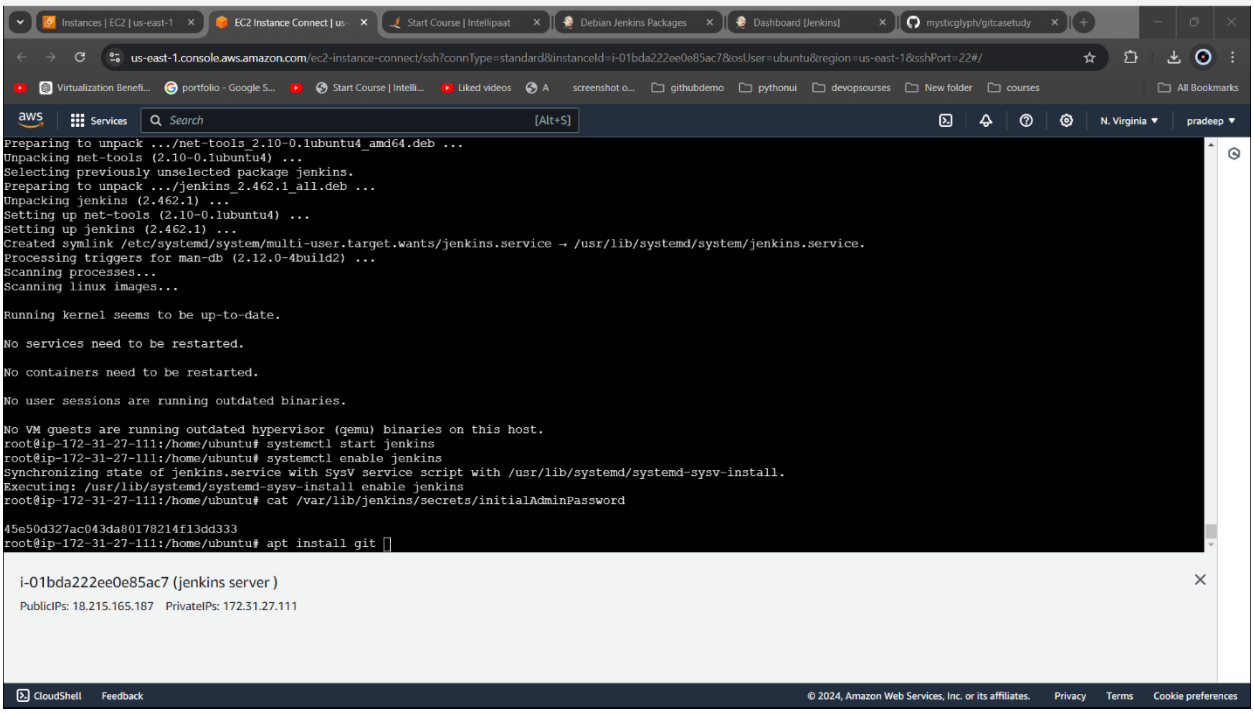


Click on create repo.

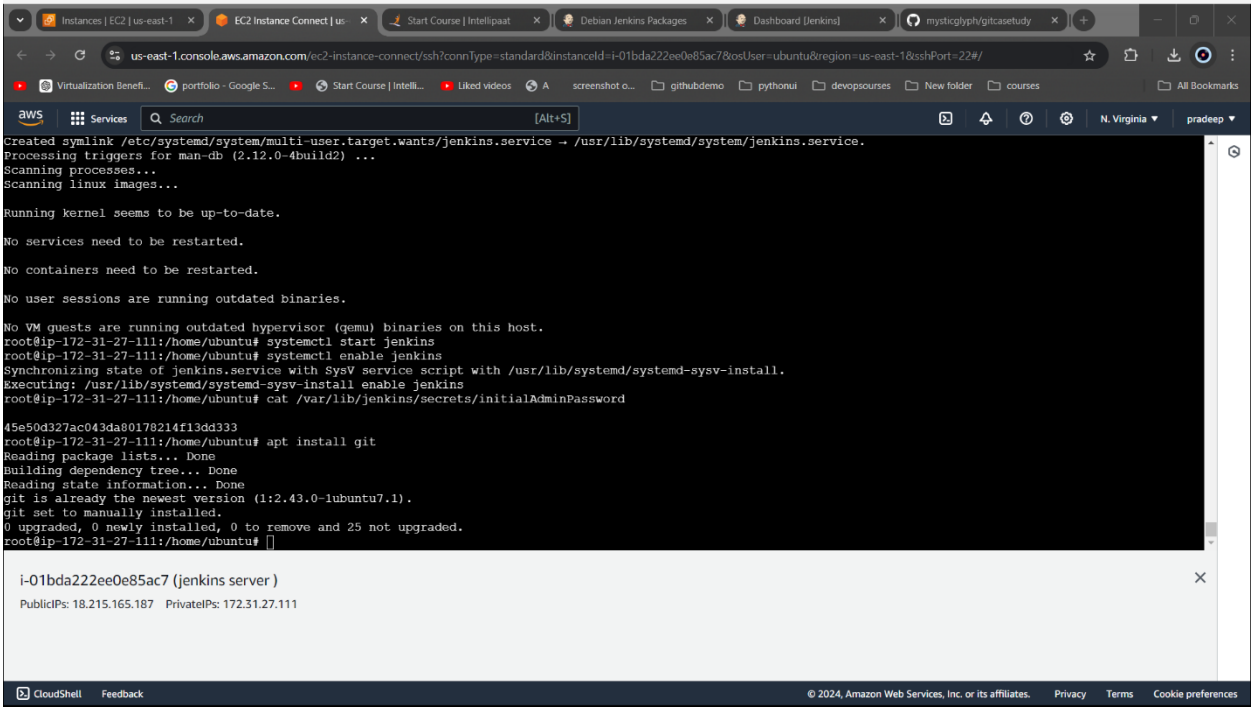
Repo is created:



Install git in local Machin:

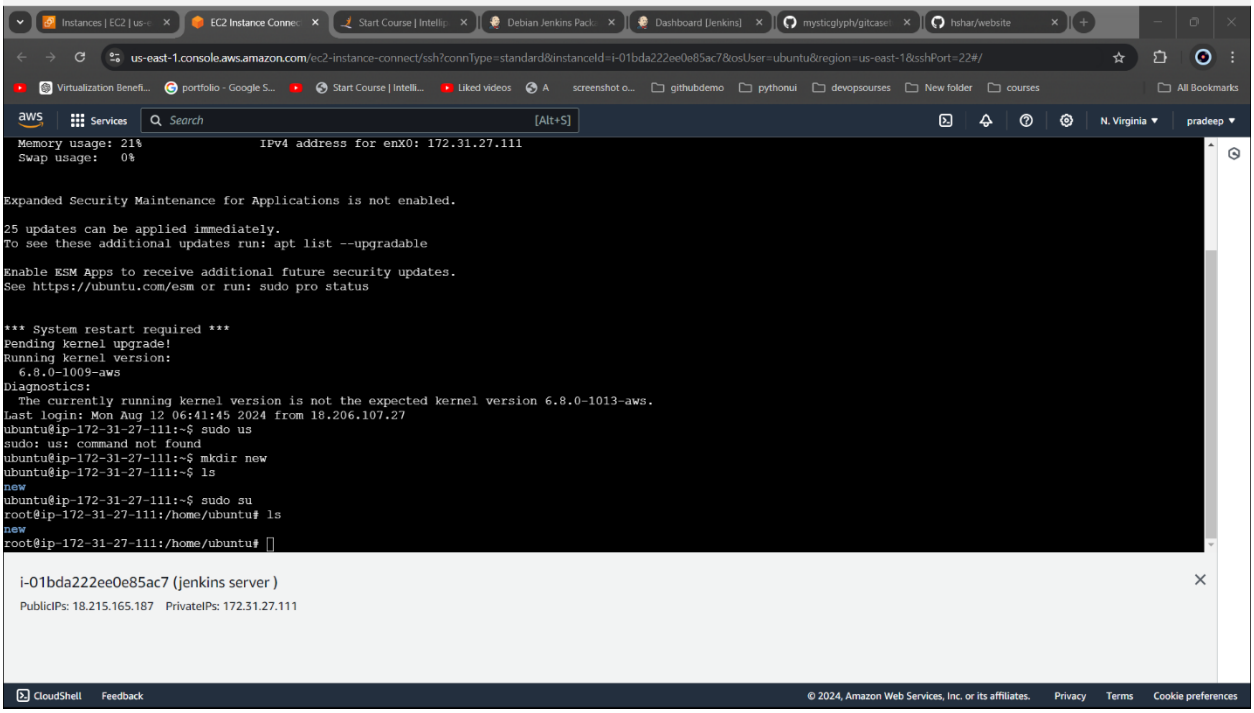


➔ apt install git



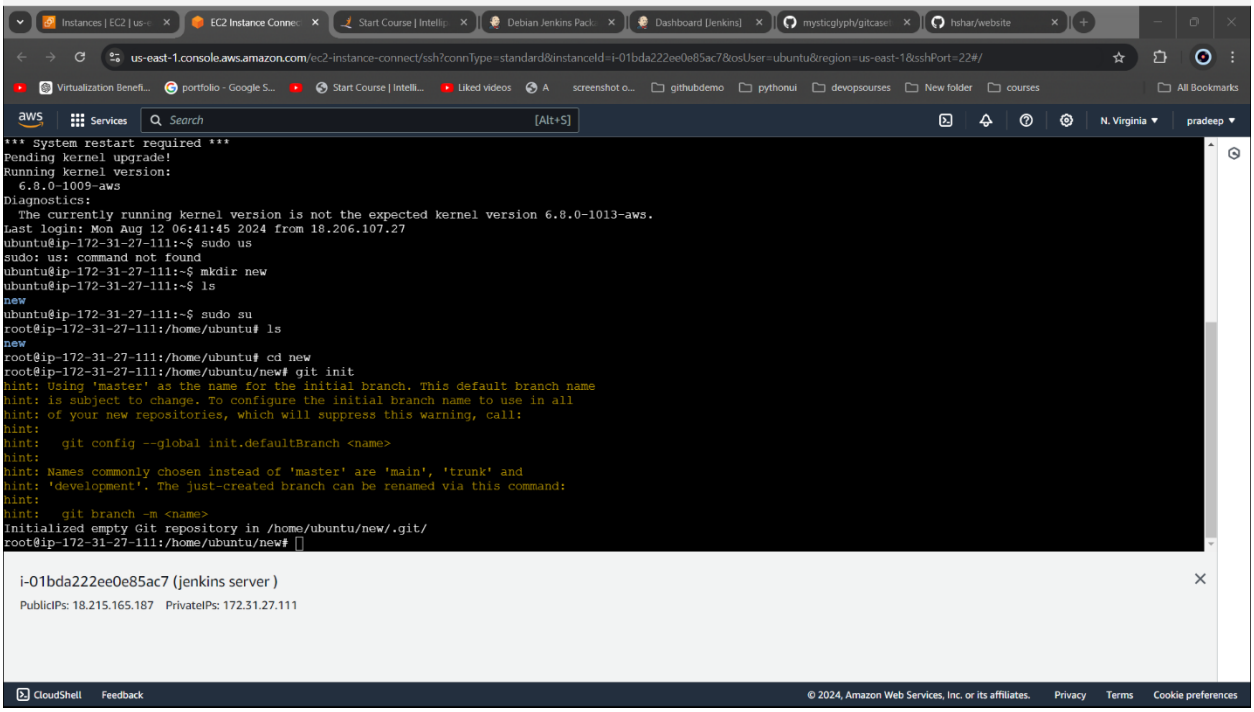
Create a folder to initializes local git repo:

➔ mkdir new



Initializing git

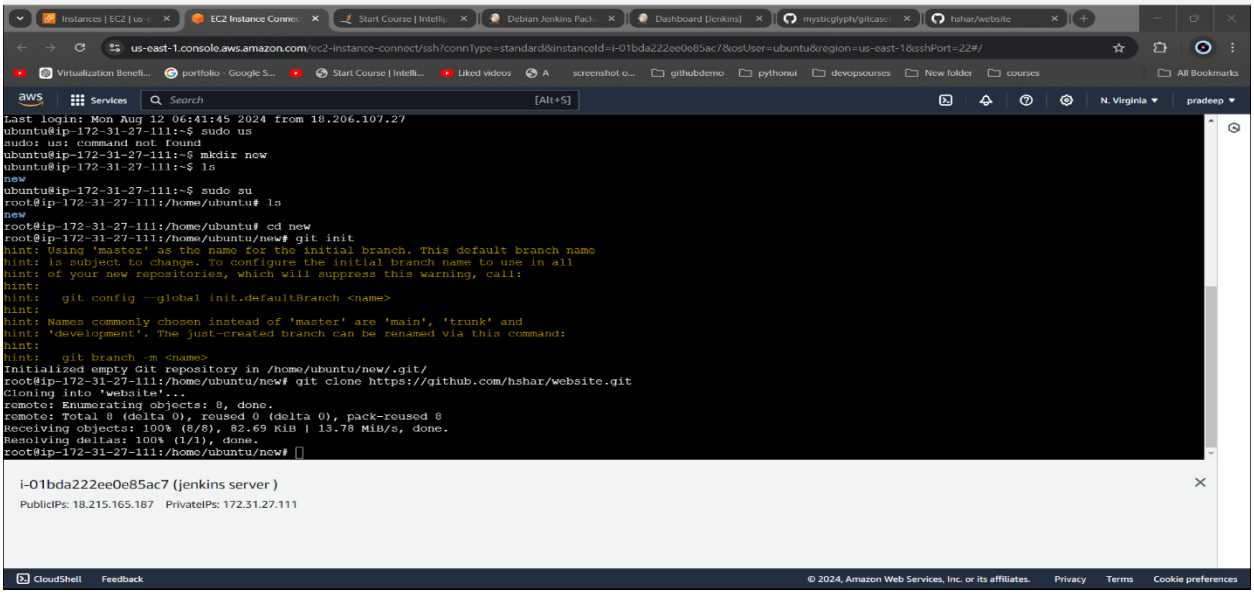
➔ git init

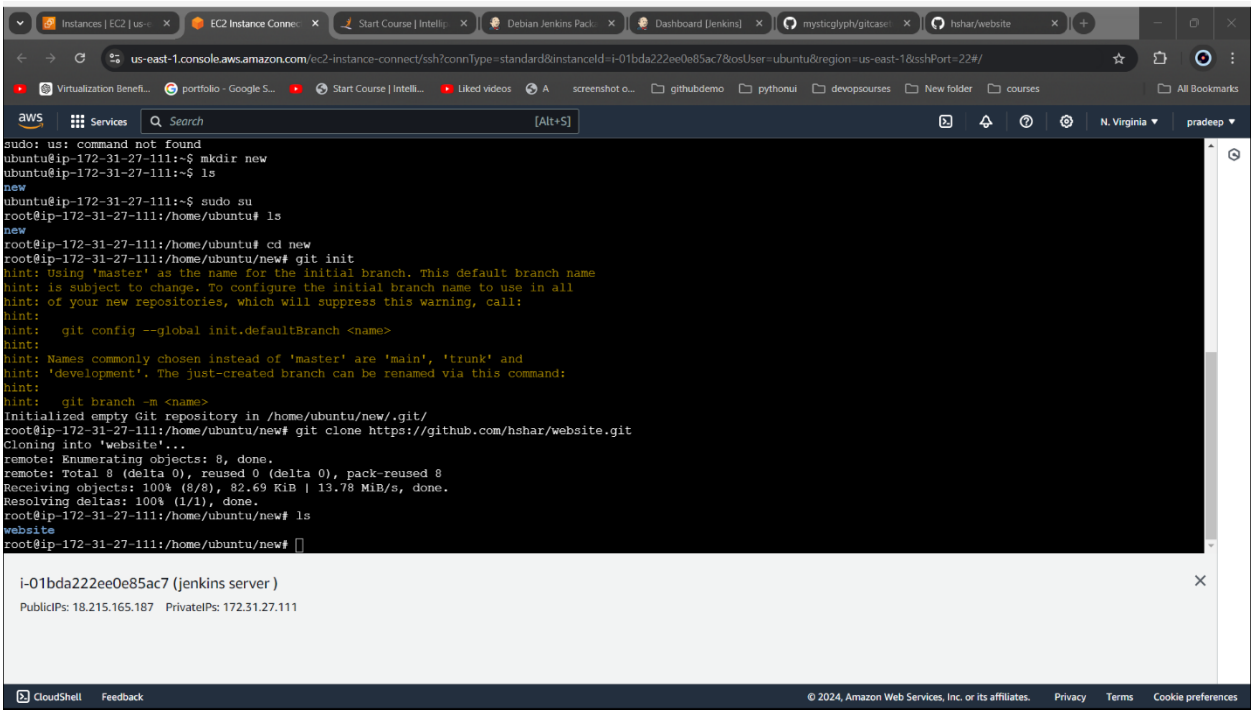


Clone the Repository:

Start by cloning the GitHub repository to the local machine:

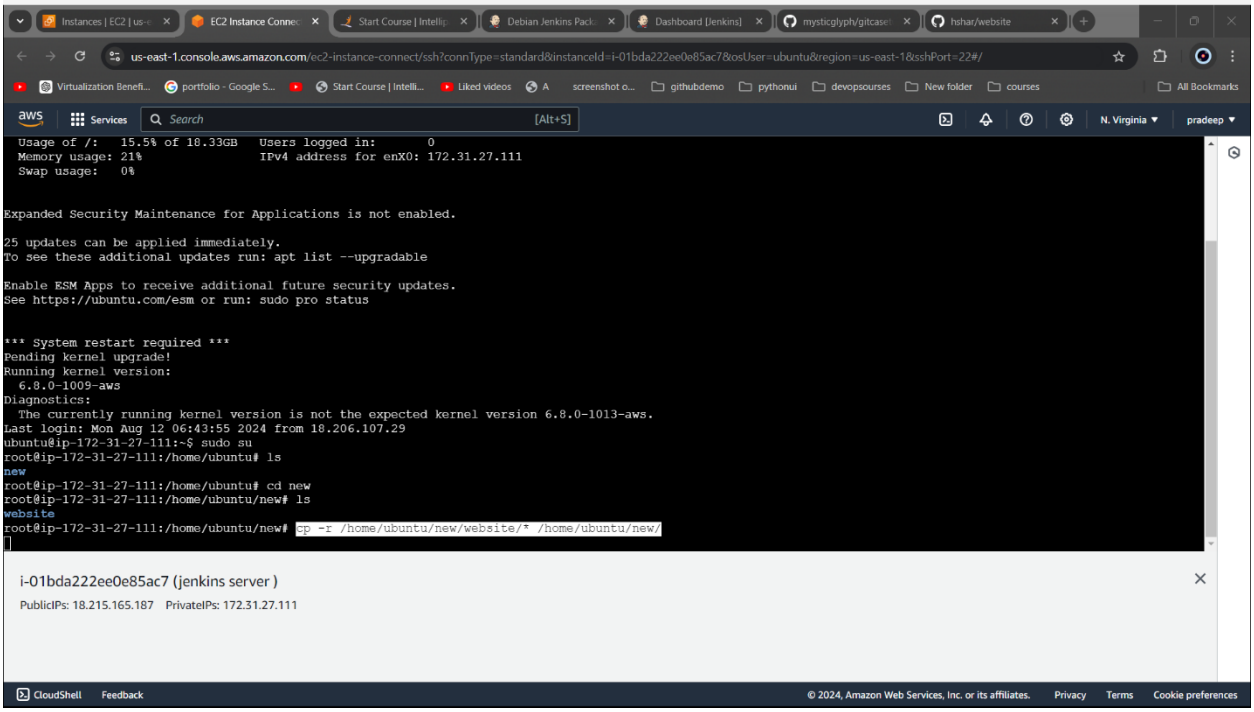
> git clone https://github.com/hshar/website.git



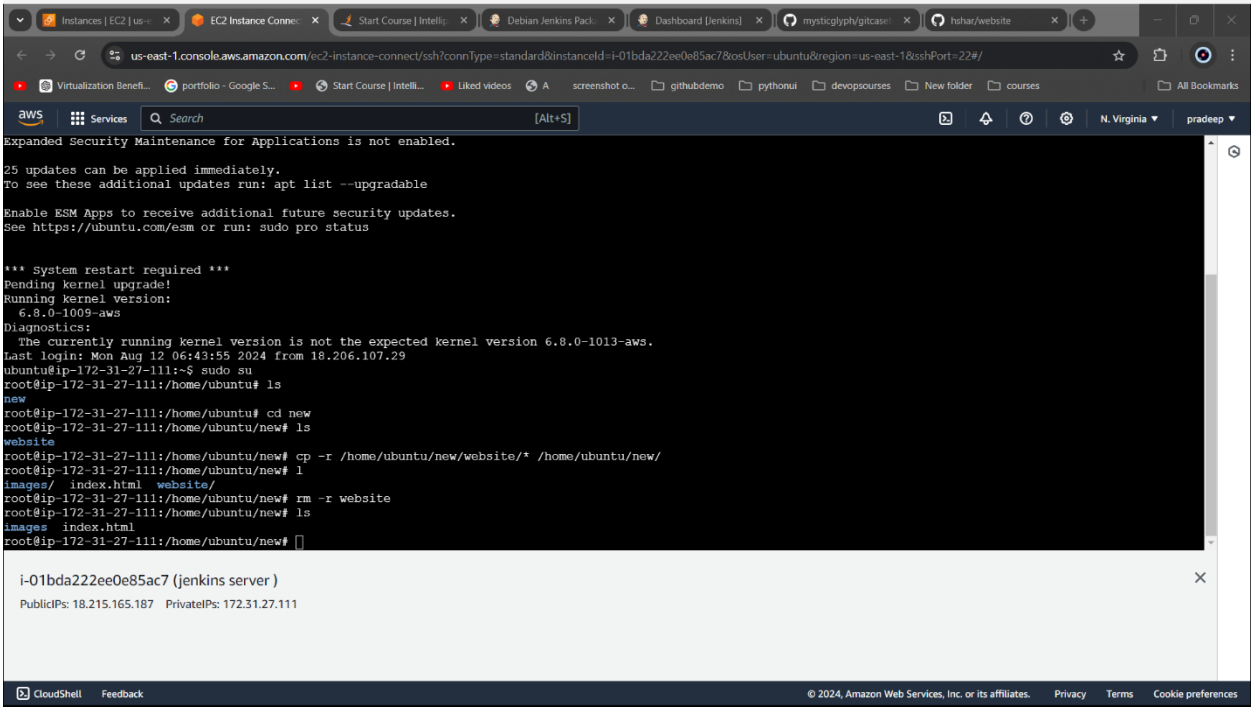


To copy the contents of the website directory to /home/ubuntu/new,:

➔ cp -r /home/ubuntu/new/website/* /home/ubuntu/new/



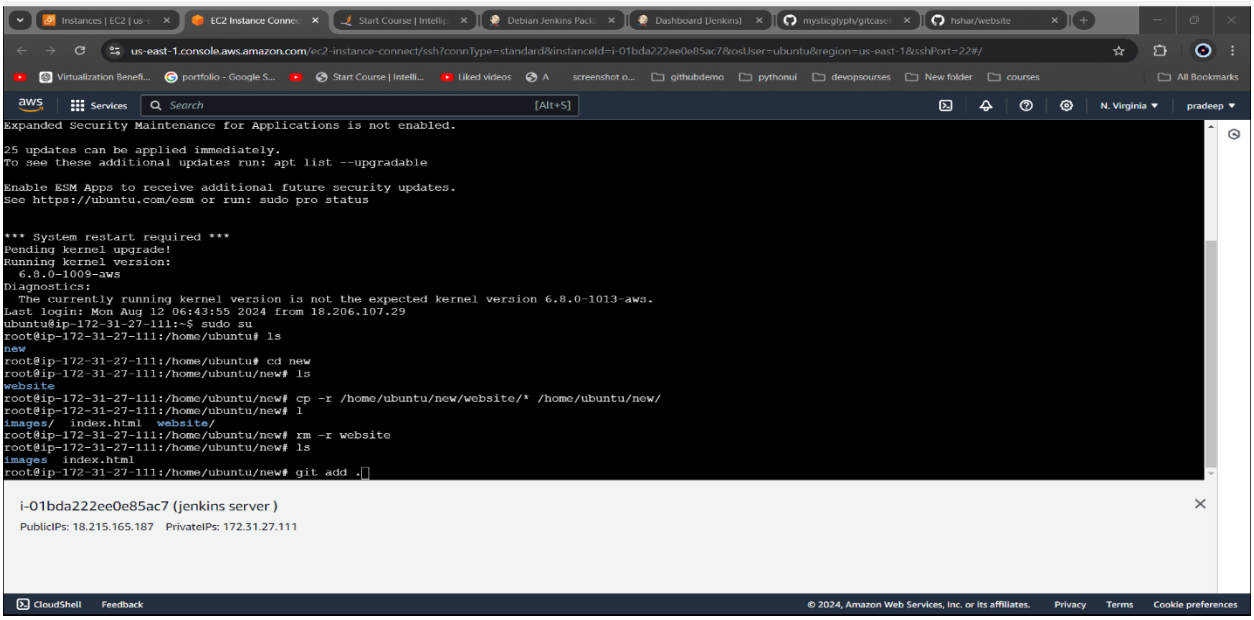
Removing the website folder cloned from git repo:

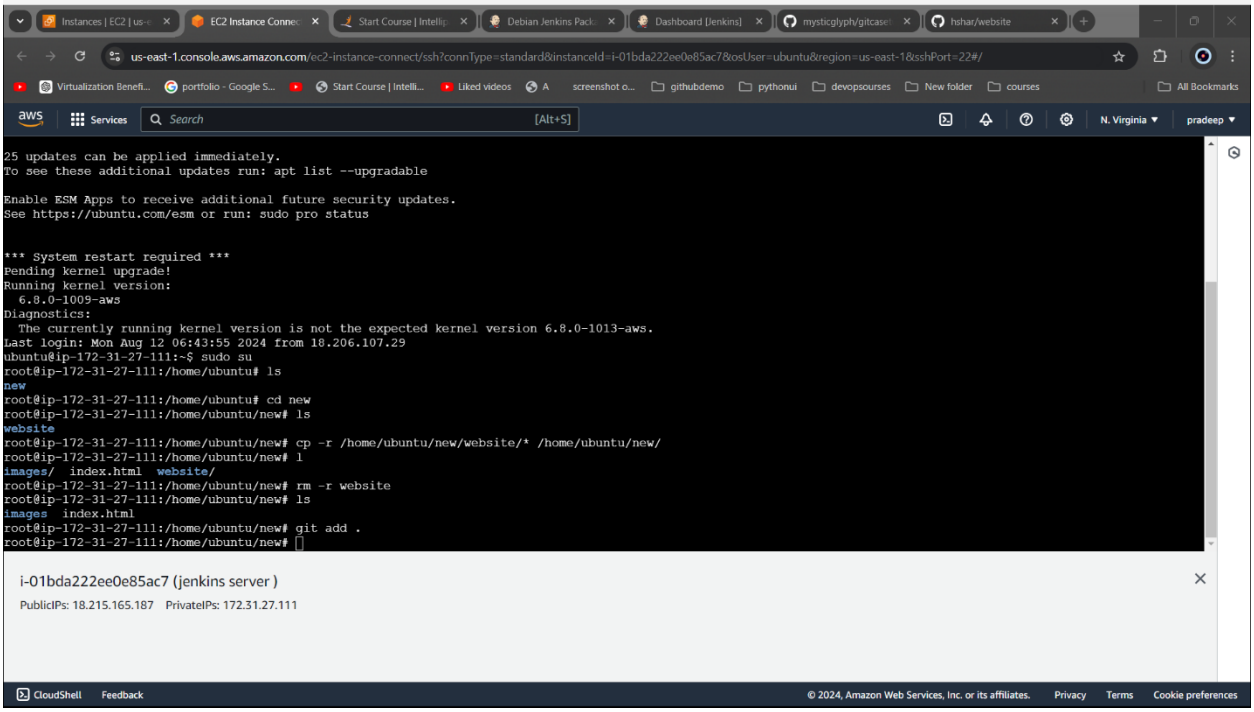


To push the contents of the new directory to a newly created GitHub repository:

Add the Files to the Staging Area:

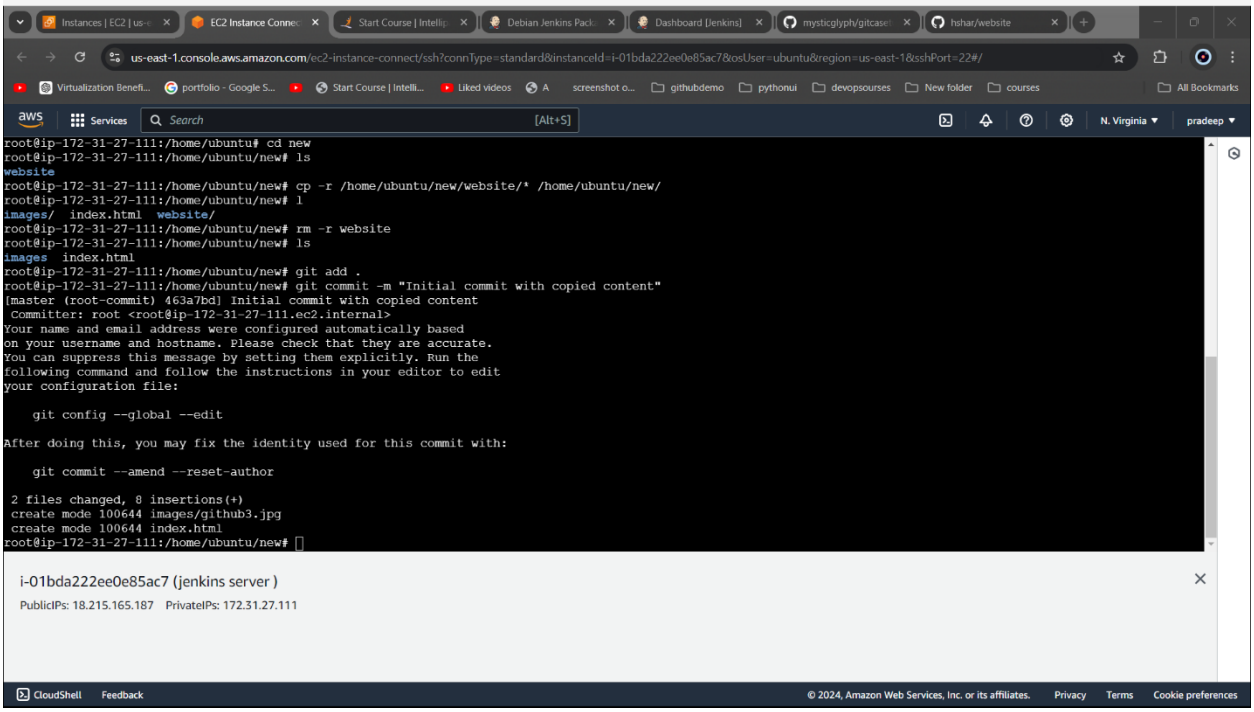
- ➔ Add all the files to the Git staging are:
- ➔ Git add .





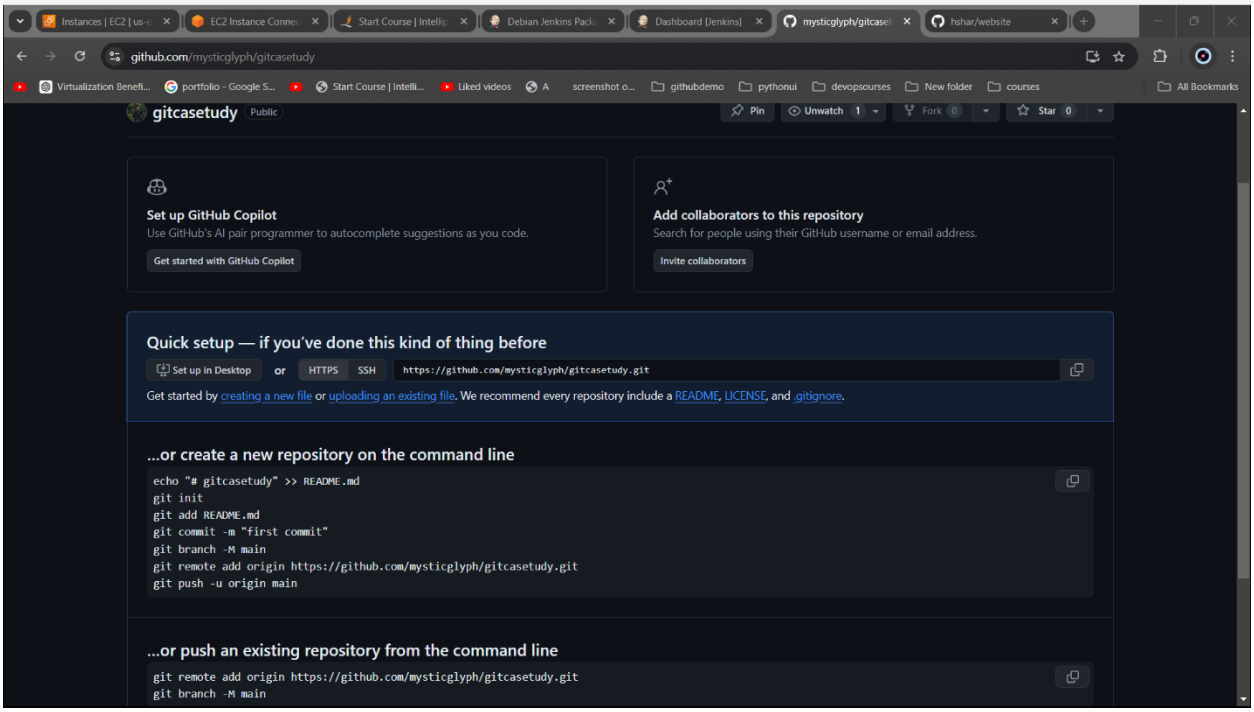
Commit the Files:

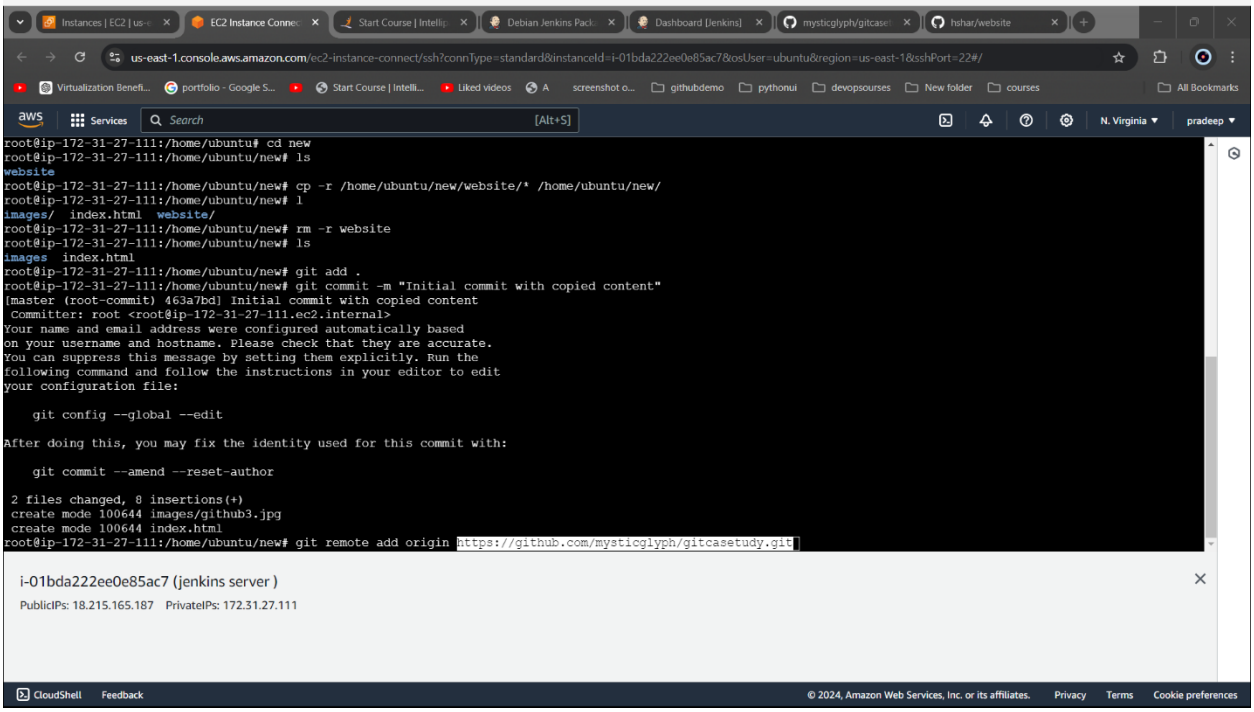
➔ git commit -m "Initial commit with copied content".

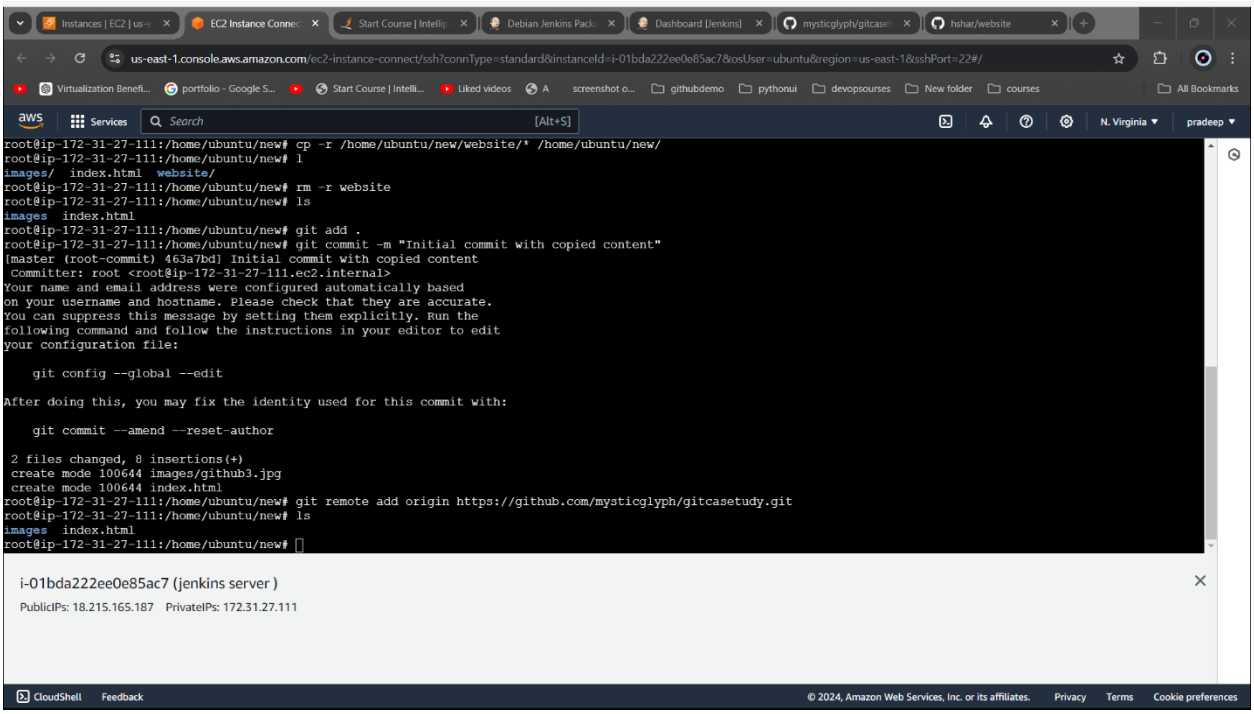


Connecting the remote repo to local repo :

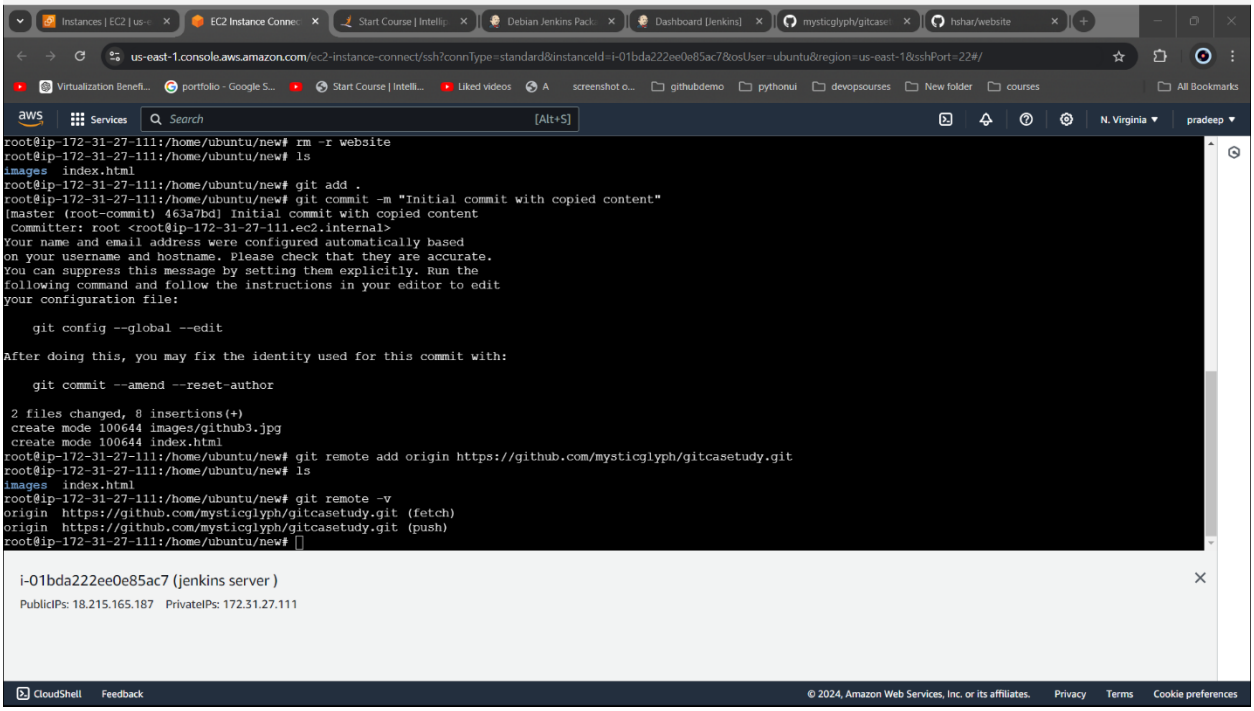
➔ git remote add origin remote repo url





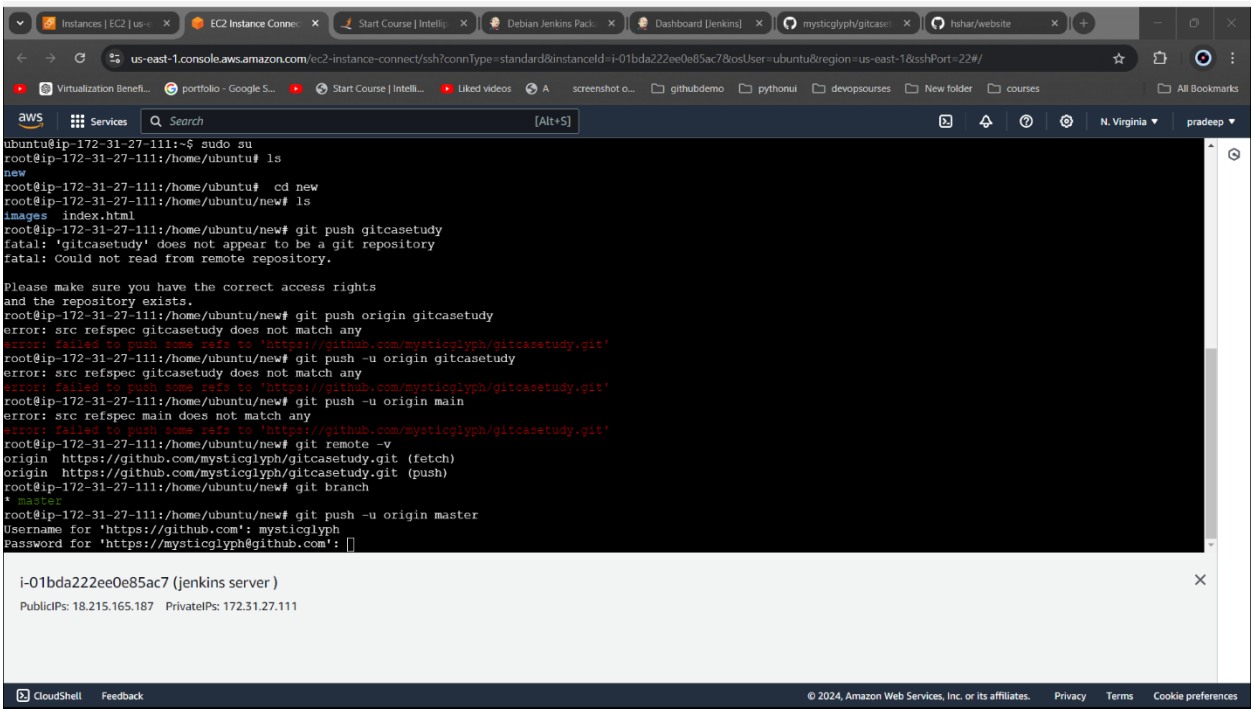


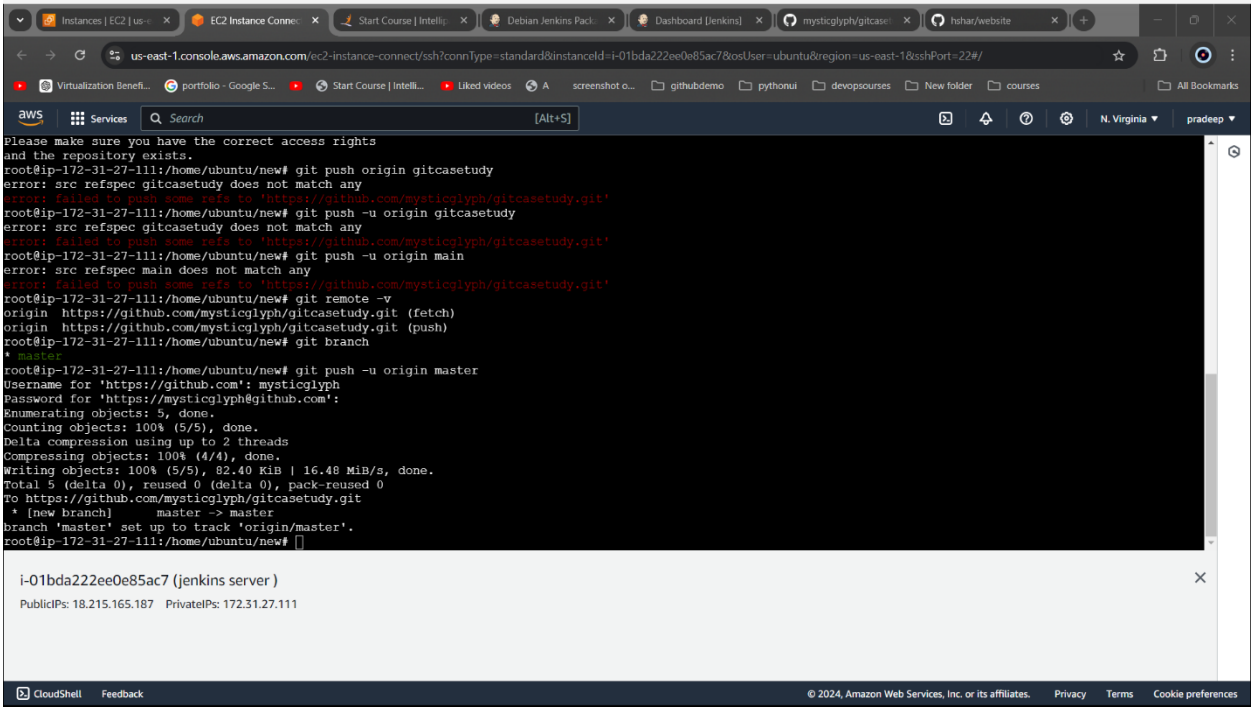
Connected to remote repo:

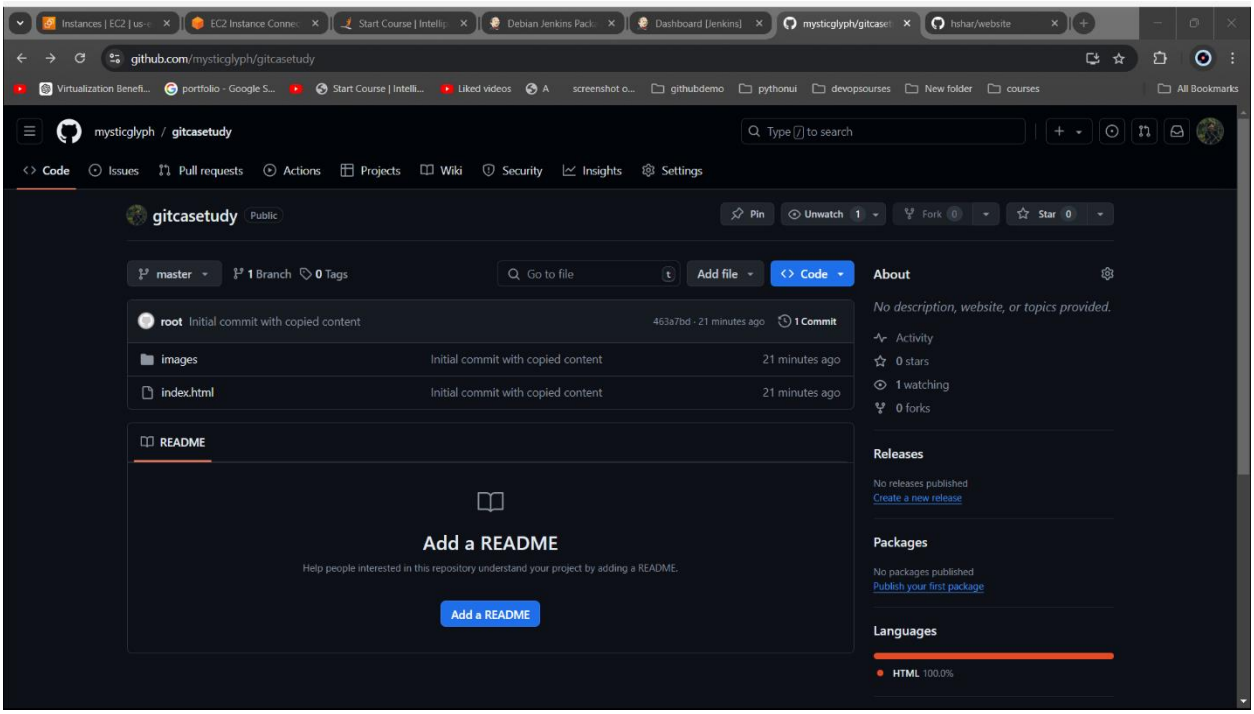


Push the Files to GitHub:

-> git push origin -u master



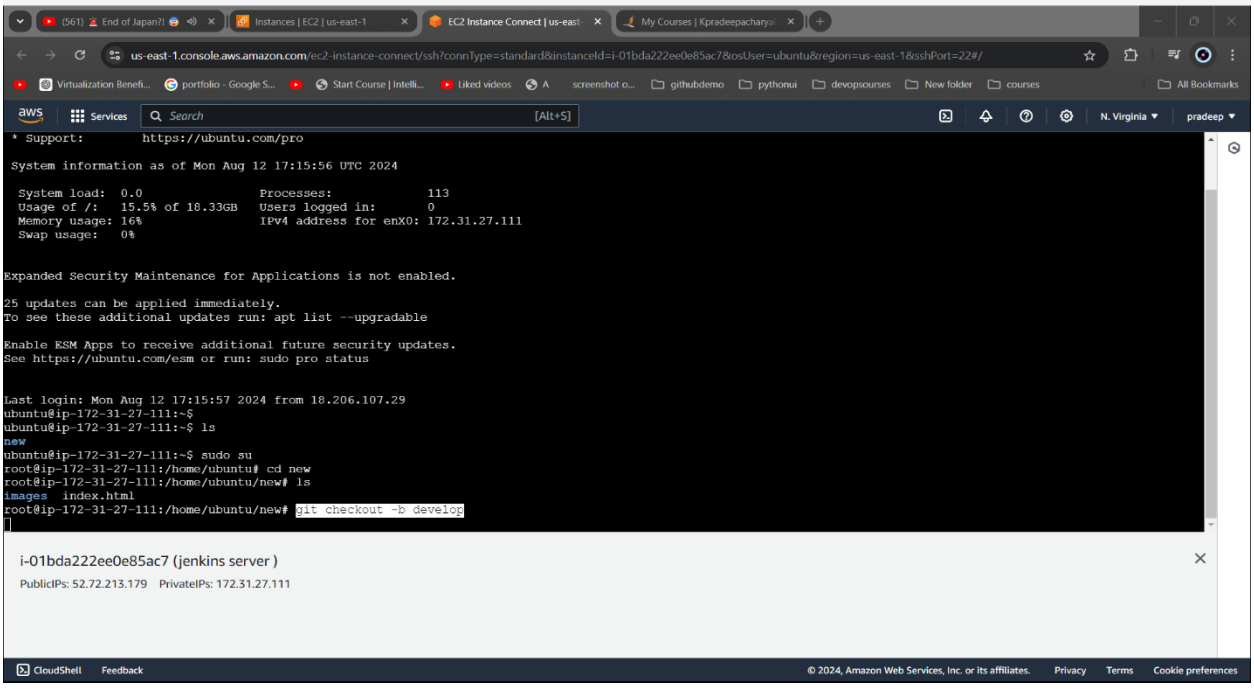


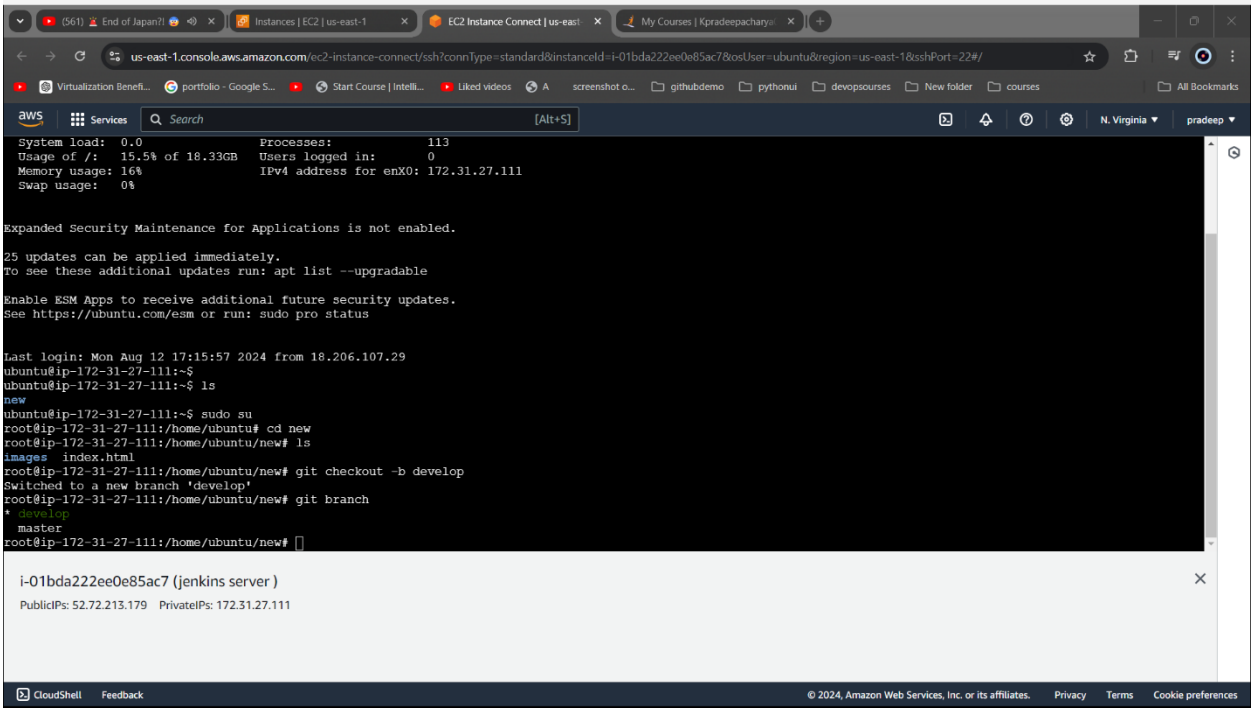


Successfully pushed to remote repo:

<u>creating develop branch:</u>

- ➔ git checkout -b branch_name
- ➔ git checkout -b develop

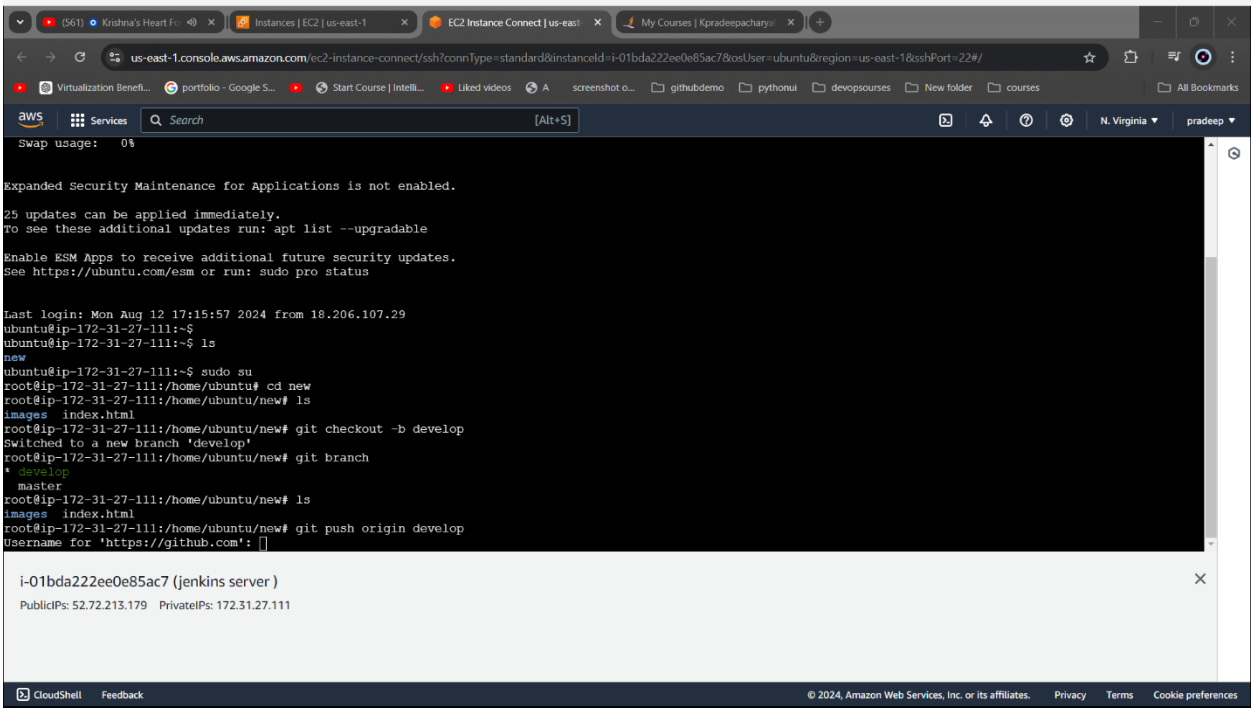


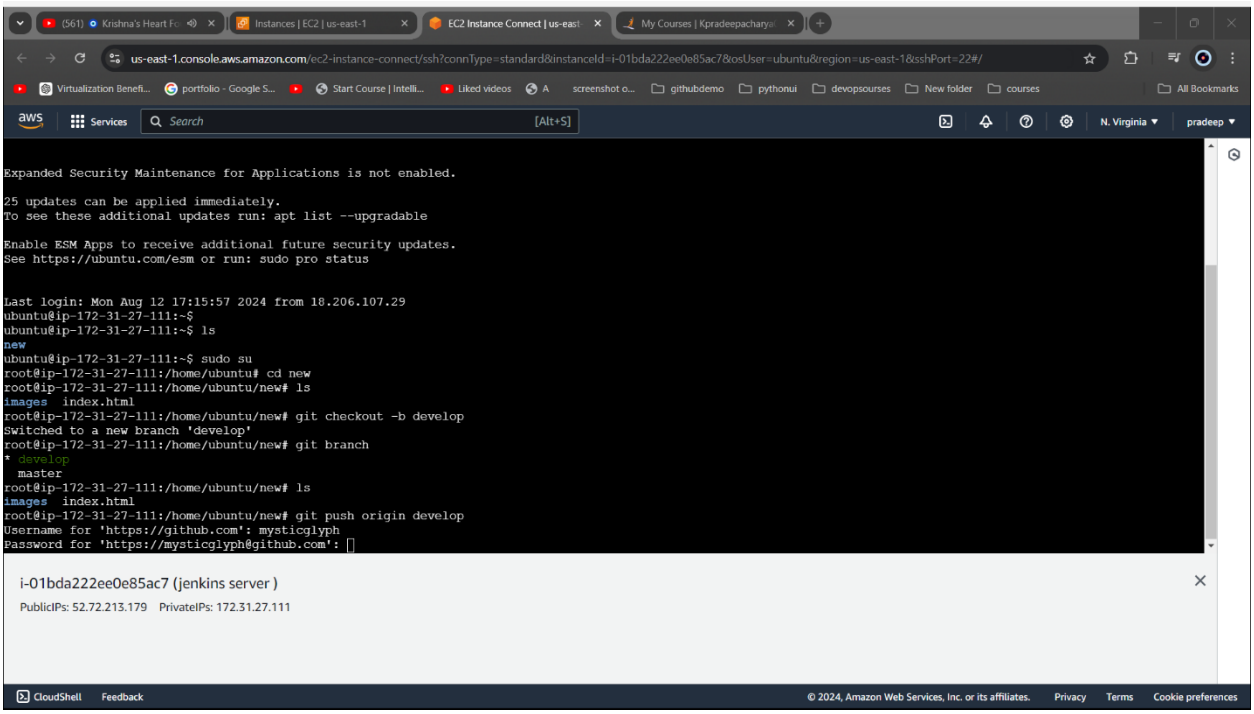


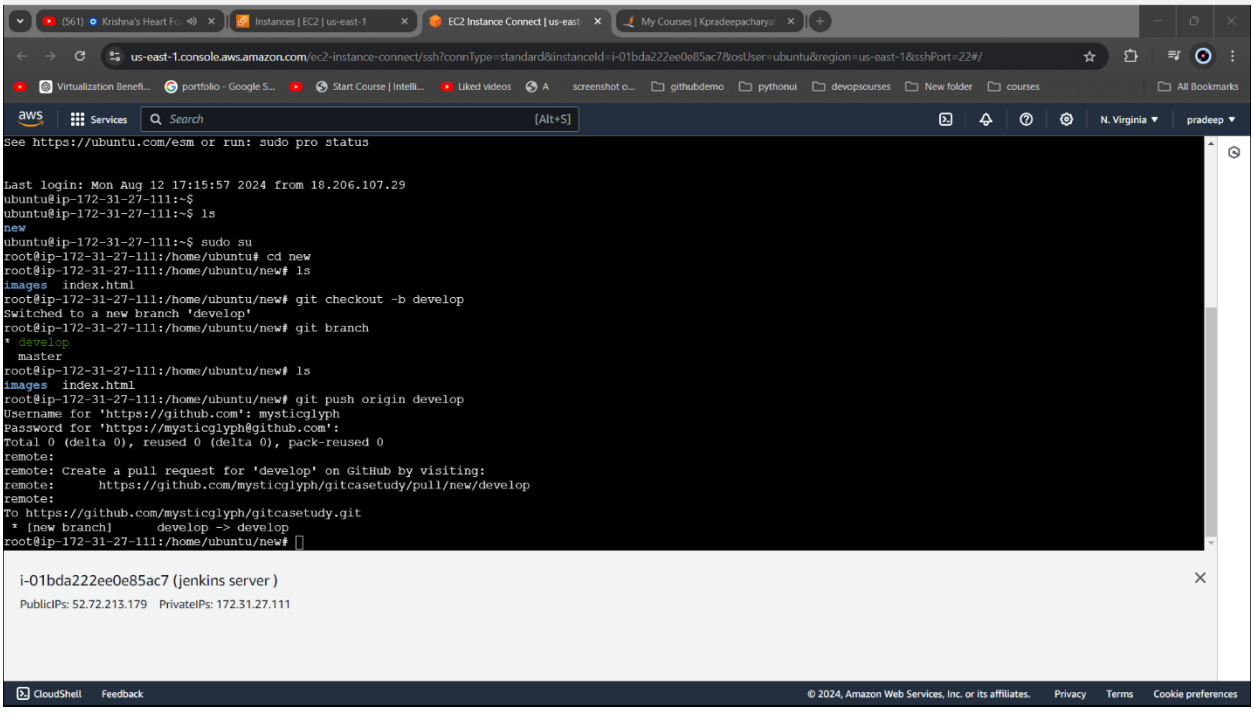
Created.

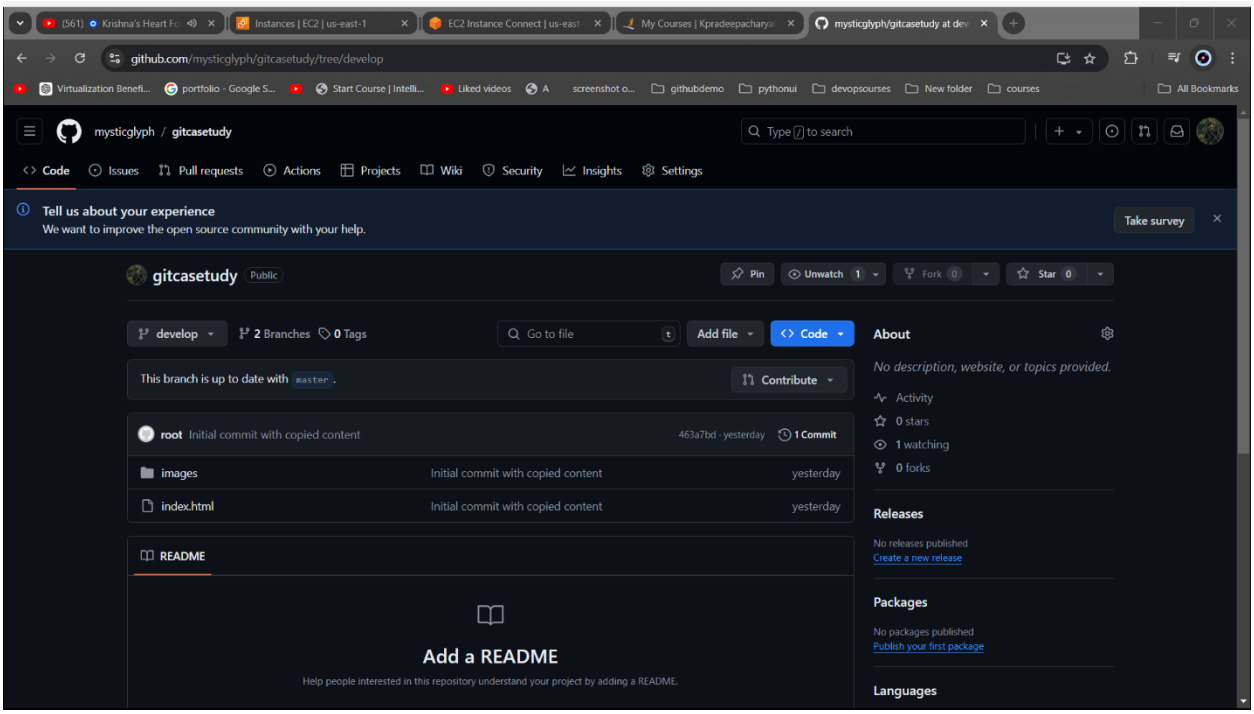
Push the new develop branch to GitHub:

➔ git push origin develop




us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh?connType=standard&instanceId=i-01bda222ee0e85ac7&osUser=ubuntu&region=us-east-1&sshPort=22#/
Expanded Security Maintenance for Applications is not enabled.
25 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: Mon Aug 12 17:15:57 2024 from 18.206.107.29
ubuntu@ip-172-31-27-111:~$
ubuntu@ip-172-31-27-111:~$ ls
new
ubuntu@ip-172-31-27-111:~$ sudo su
root@ip-172-31-27-111:/home/ubuntu# cd new
root@ip-172-31-27-111:/home/ubuntu/new# ls
images index.html
root@ip-172-31-27-111:/home/ubuntu/new# git checkout -b develop
Switched to a new branch 'develop'
root@ip-172-31-27-111:/home/ubuntu/new# git branch
* develop
master
root@ip-172-31-27-111:/home/ubuntu/new# ls
images index.html
root@ip-172-31-27-111:/home/ubuntu/new# git push origin develop
Username for 'https://github.com': mysticglyph
Password for 'https://mysticglyph@github.com':
i-01bda222ee0e85ac7 (jenkins server)
PublicIPs: 52.72.213.179 PrivateIPs: 172.31.27.111
CloudShell Feedback
© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences


us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh?connType=standard&instanceId=i-01bda222ee0e85ac7&osUser=ubuntu&region=us-east-1&sshPort=22#/
See https://ubuntu.com/esm or run: sudo pro status
Last login: Mon Aug 12 17:15:57 2024 from 18.206.107.29
ubuntu@ip-172-31-27-111:~$
ubuntu@ip-172-31-27-111:~$ ls
new
ubuntu@ip-172-31-27-111:~$ sudo su
root@ip-172-31-27-111:/home/ubuntu# cd new
root@ip-172-31-27-111:/home/ubuntu/new# ls
images index.html
root@ip-172-31-27-111:/home/ubuntu/new# git checkout -b develop
Switched to a new branch 'develop'
root@ip-172-31-27-111:/home/ubuntu/new# git branch
* develop
master
root@ip-172-31-27-111:/home/ubuntu/new# ls
images index.html
root@ip-172-31-27-111:/home/ubuntu/new# git push origin develop
Username for 'https://github.com': mysticglyph
Password for 'https://mysticglyph@github.com':
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote: https://github.com/mysticglyph/gitcasetudy/pull/new/develop
remote:
To https://github.com/mysticglyph/gitcasetudy.git
* [new branch] develop -> develop
root@ip-172-31-27-111:/home/ubuntu/new#
i-01bda222ee0e85ac7 (jenkins server)
PublicIPs: 52.72.213.179 PrivateIPs: 172.31.27.111
CloudShell Feedback
© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

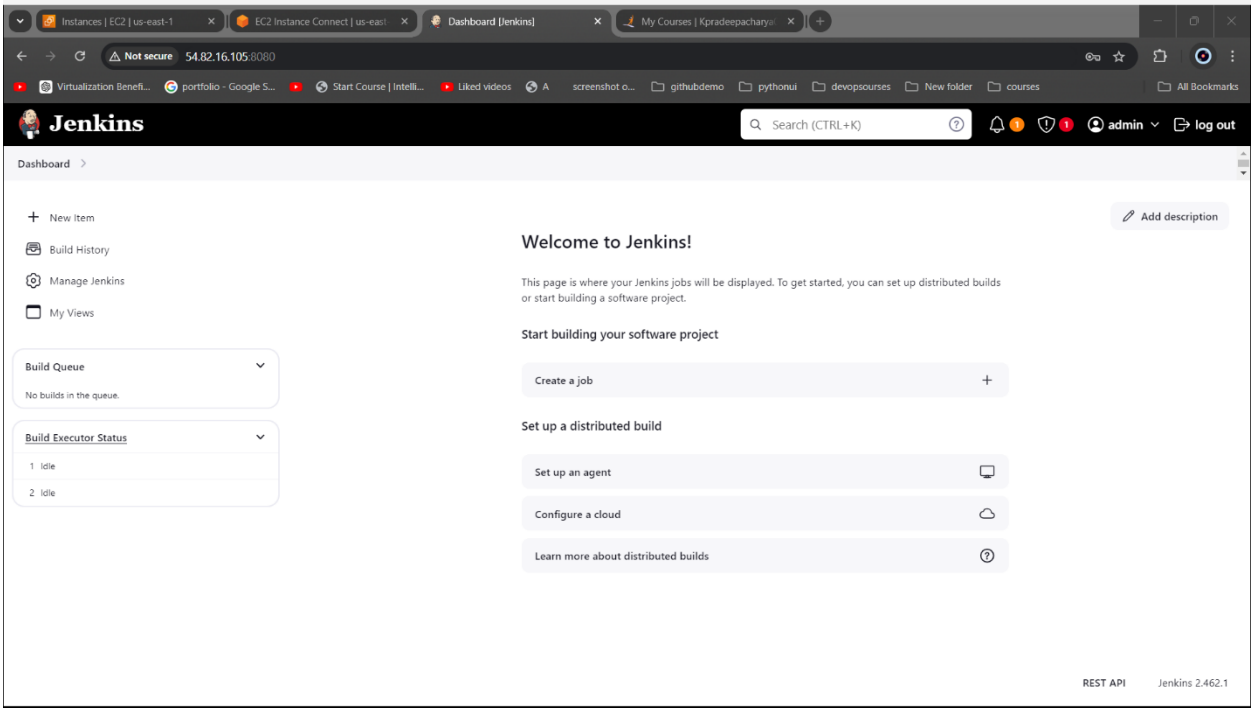


Successfully created.

Git Workflow has implemented.

Task: We now need to create a job that will automatically trigger a code build whenever a commit is made to the master or develop branch

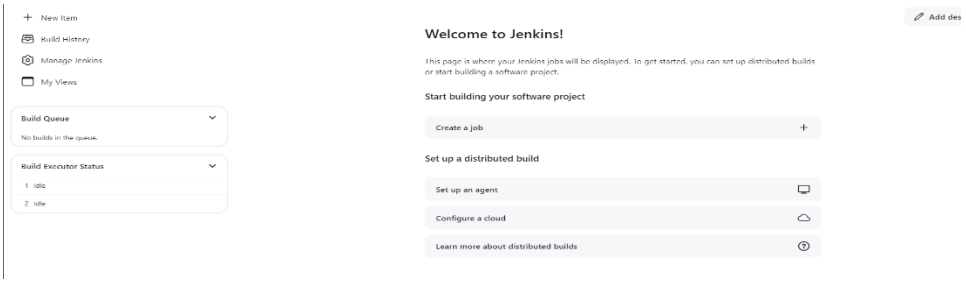
Let's create two separate jobs: one called “master-job” and another called “develop-job”.



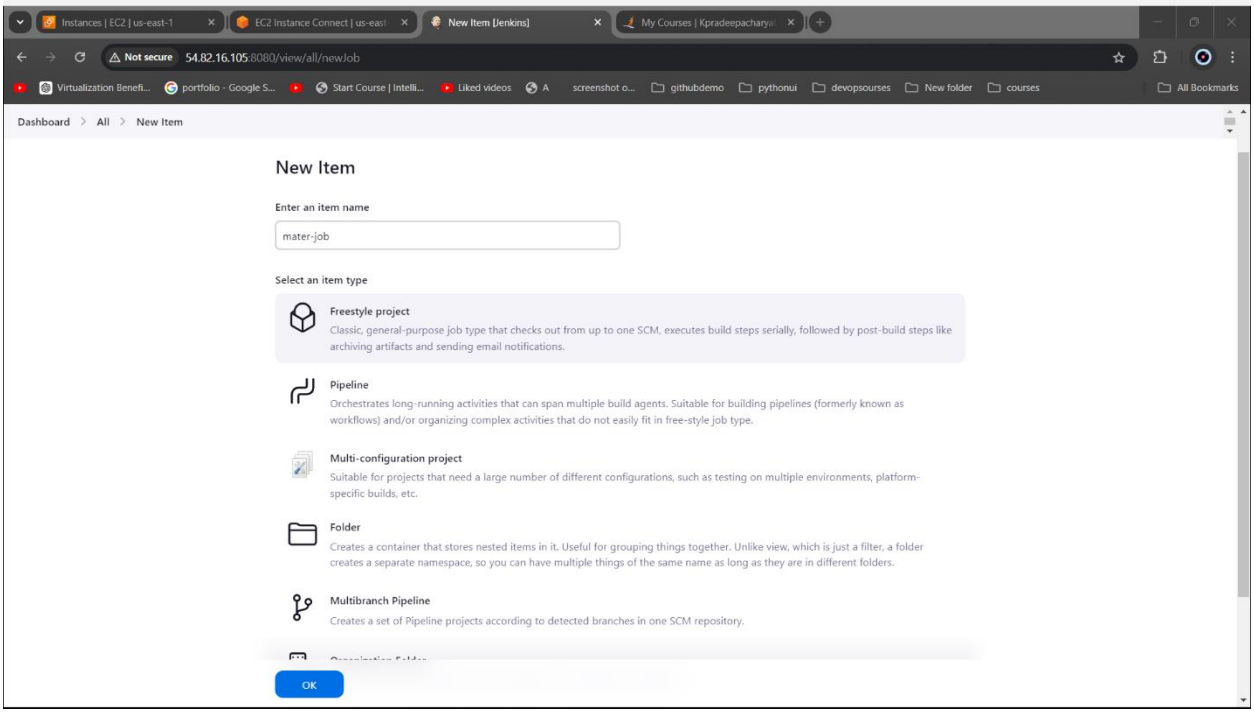
Creating a Master-Job:

The objective of this job, named Master Job, is to automatically trigger a code build whenever a commit is made to the master branch. If a commit is made to the master branch, the job will build and publish the website accordingly.

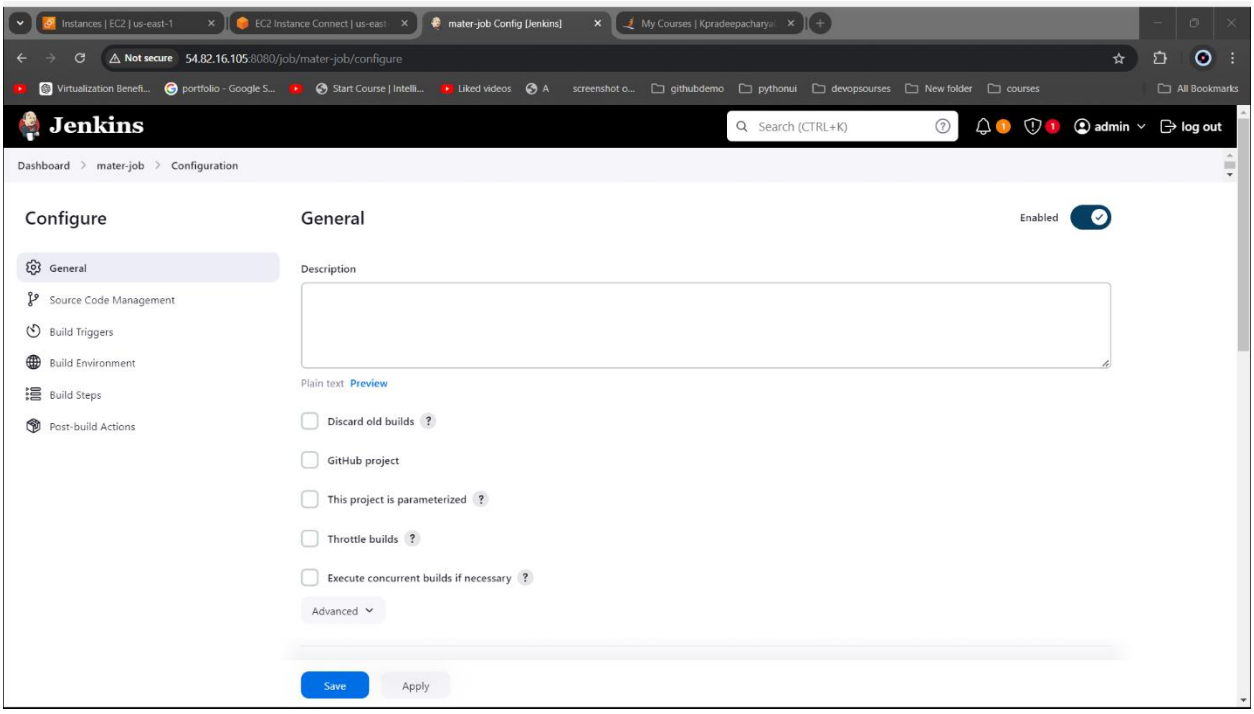
Click on new item or create a job for creating new job:



Enter an item name and select the job type:

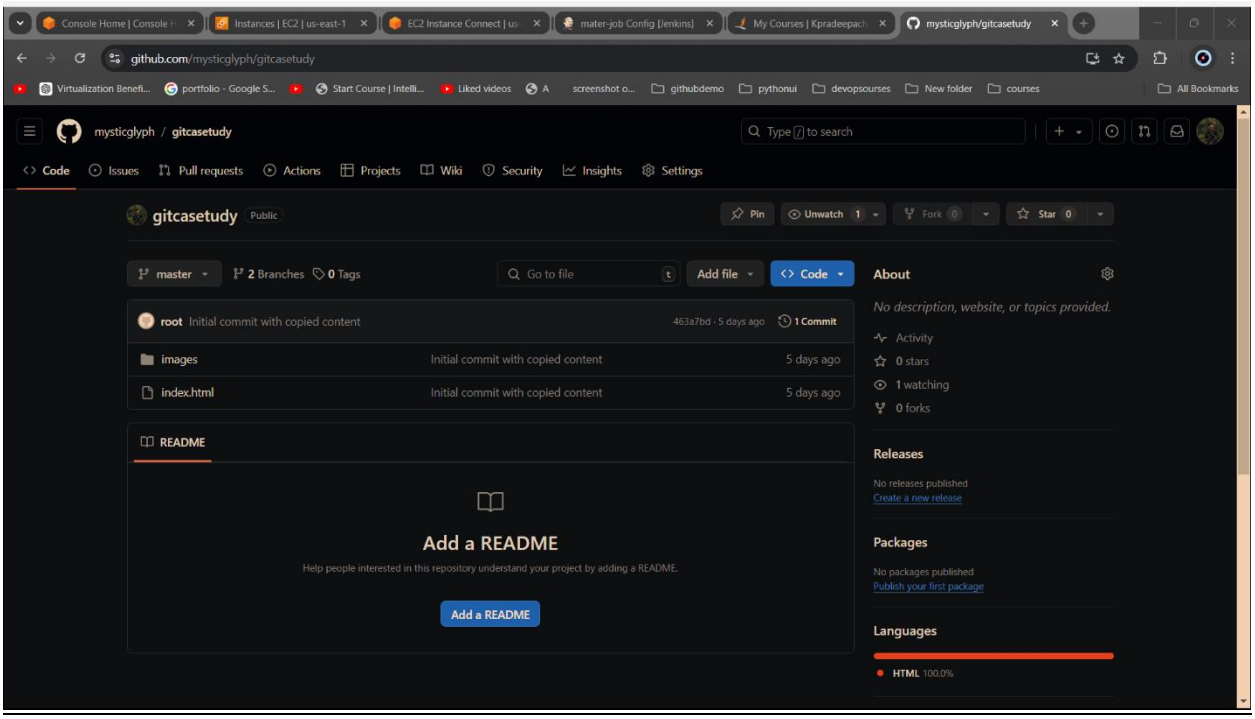


Click ok.

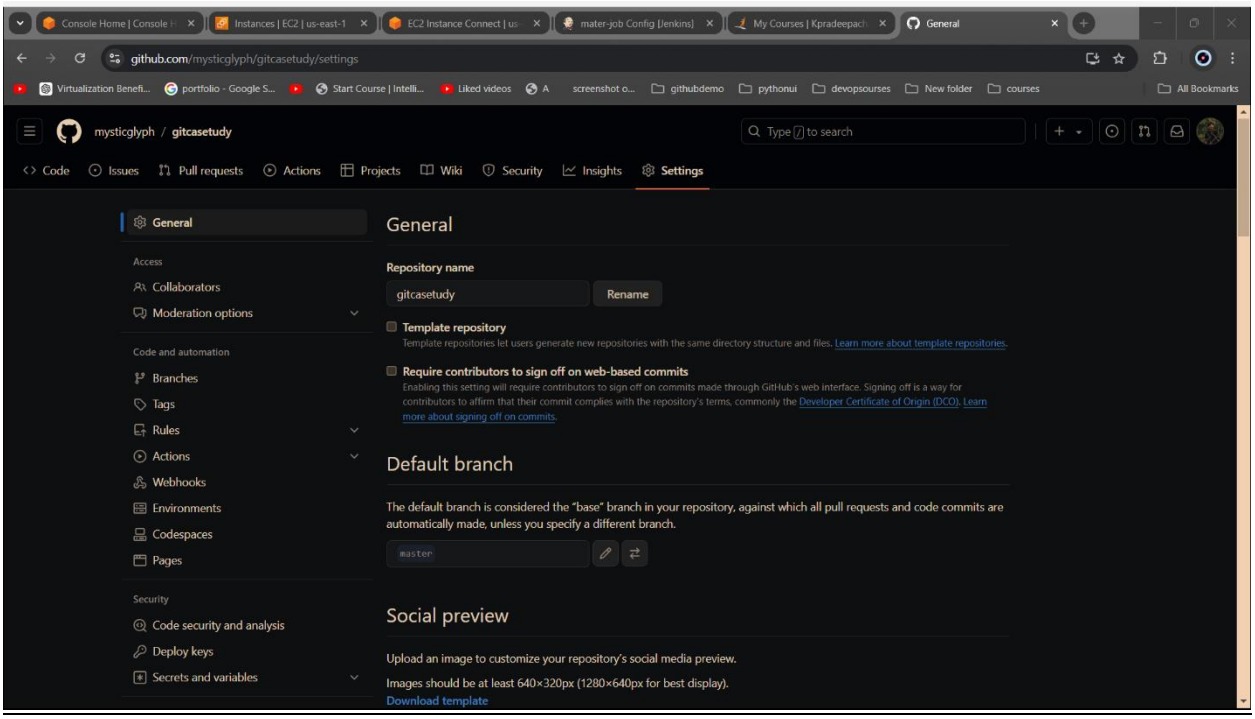


Job is created.

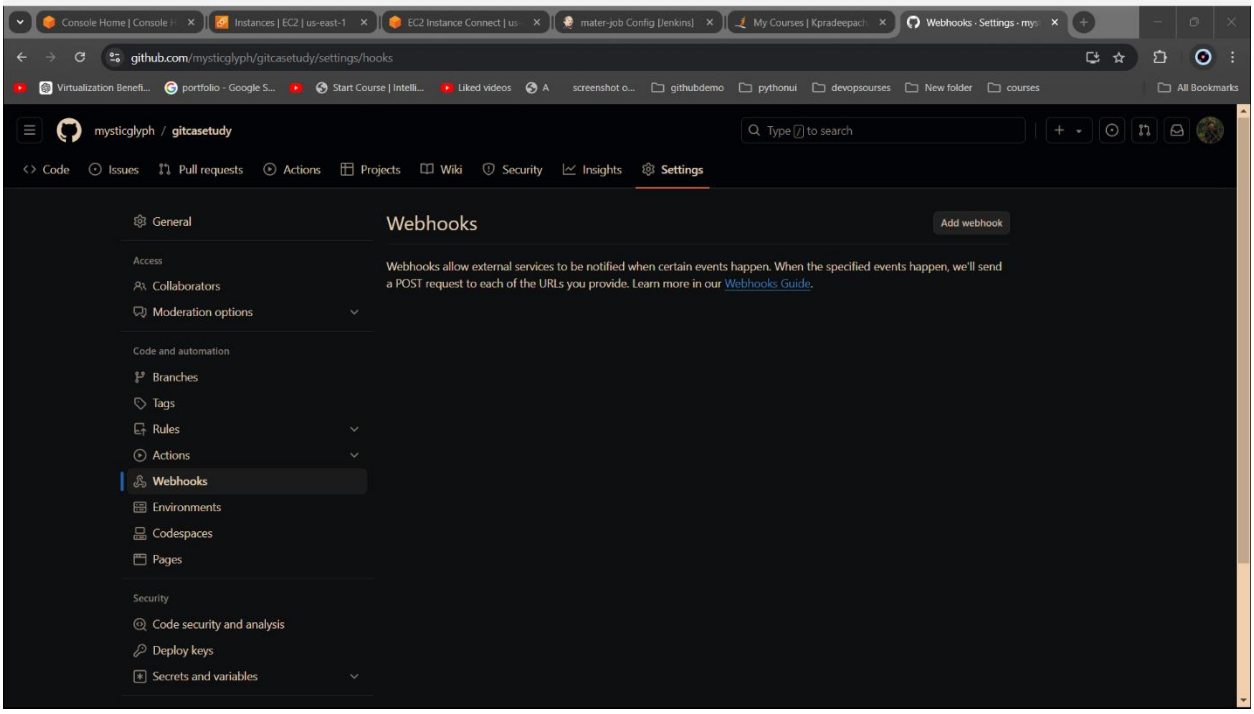
In order to trigger the job when a push event occurs, we need to create a webhook in GitHub.



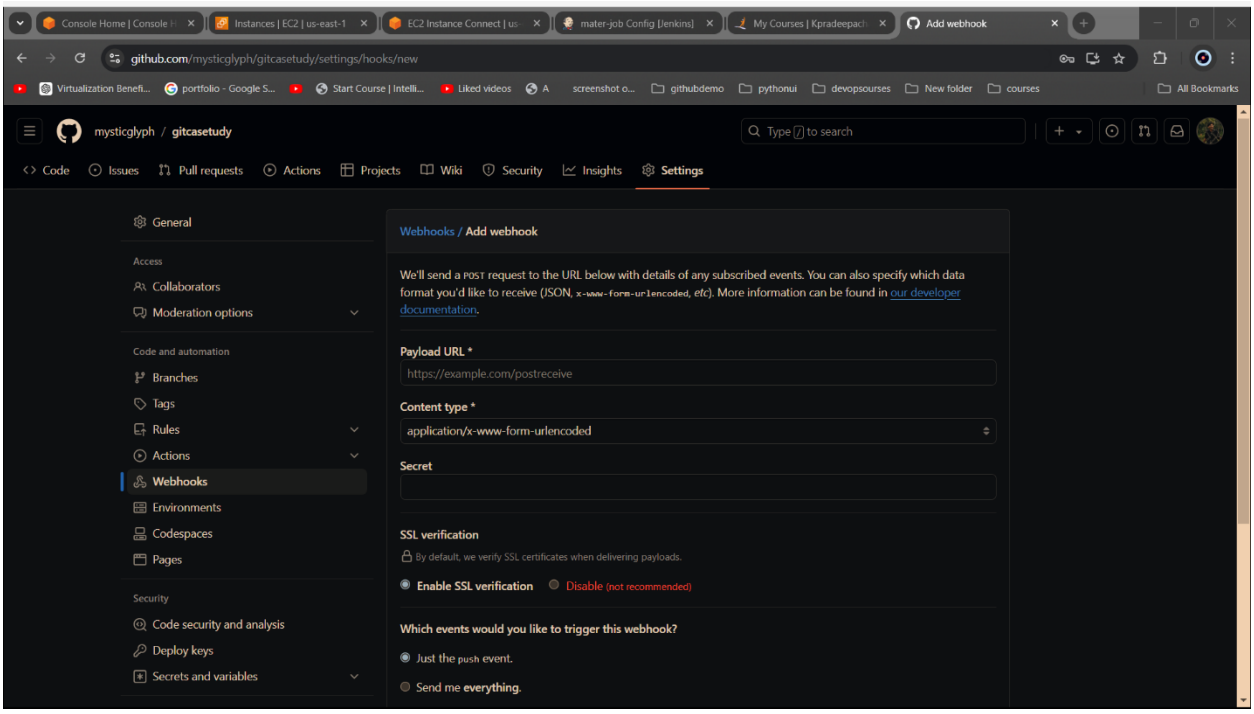
Goto settings:



select the option -> webhook



Add webhook:

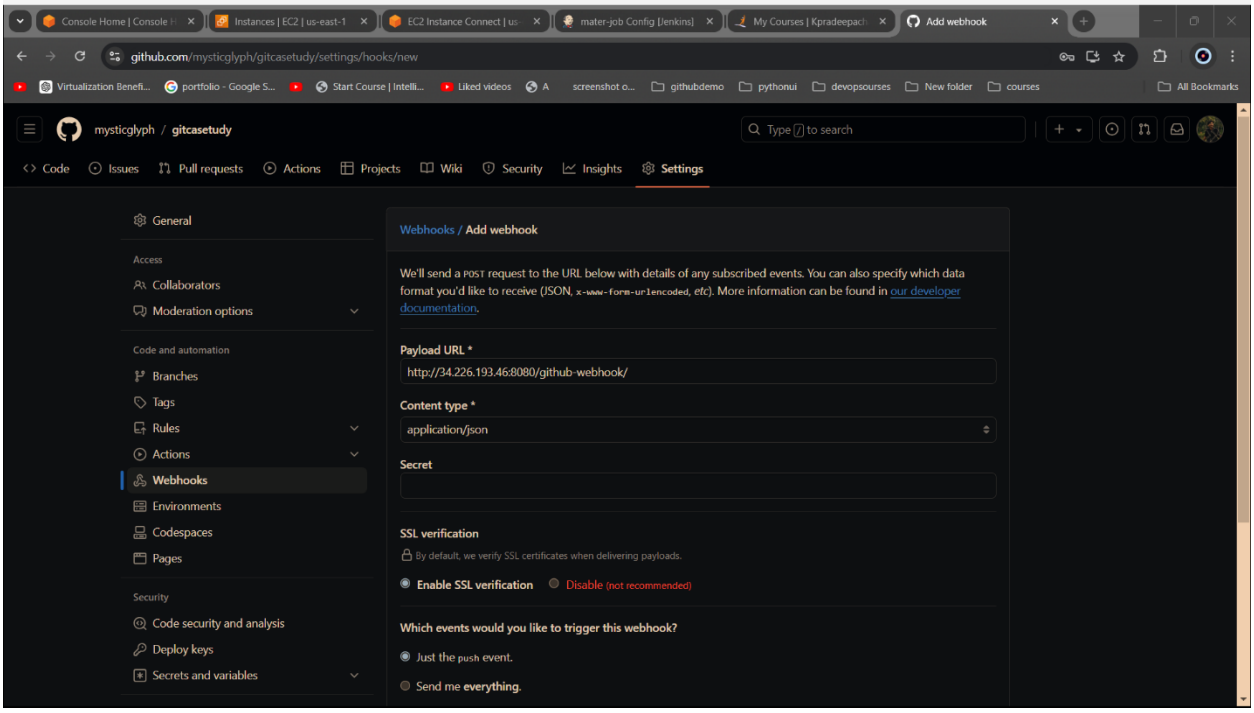


Payload URL:

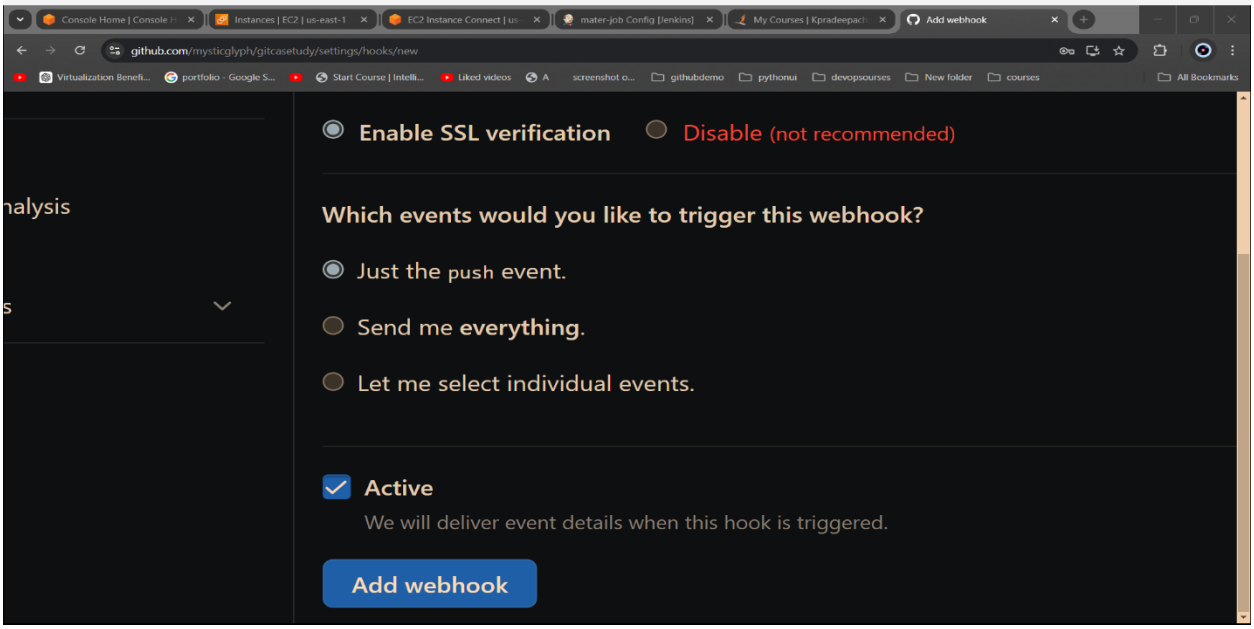
We need to provide the Jenkins URL in the webhook configuration.

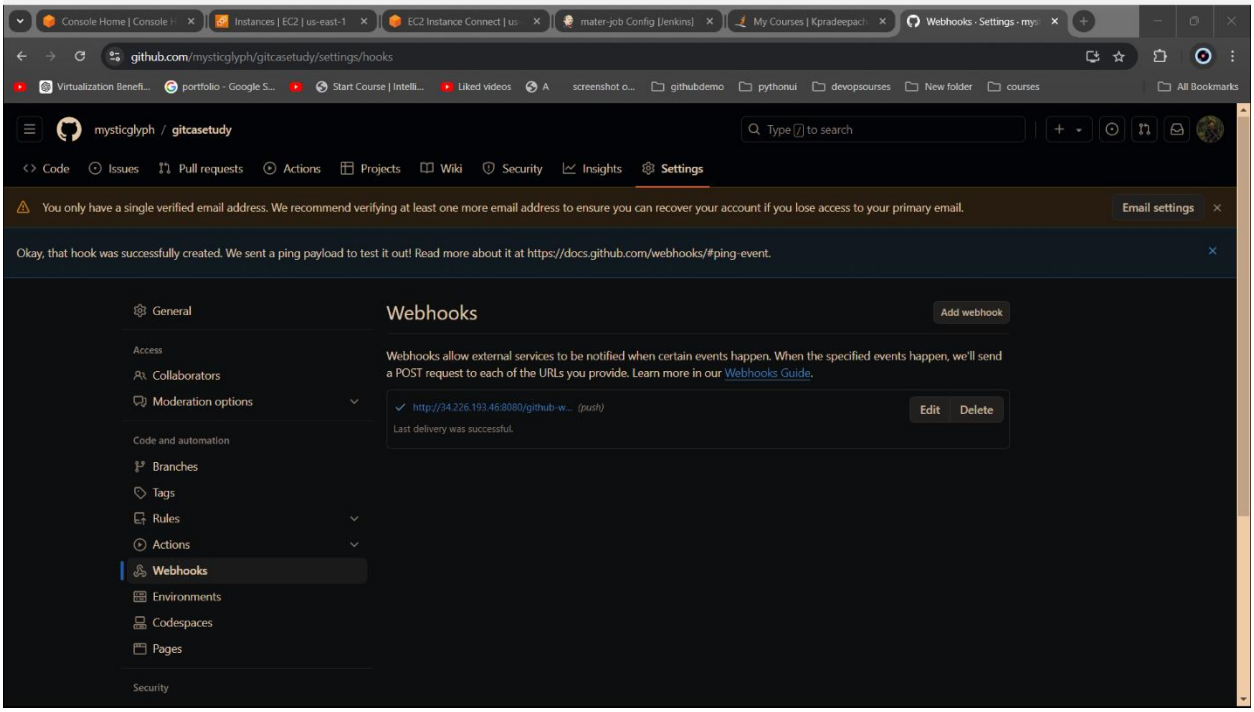
Content type:

Application/json



Which events would you like to trigger this webhook:
push



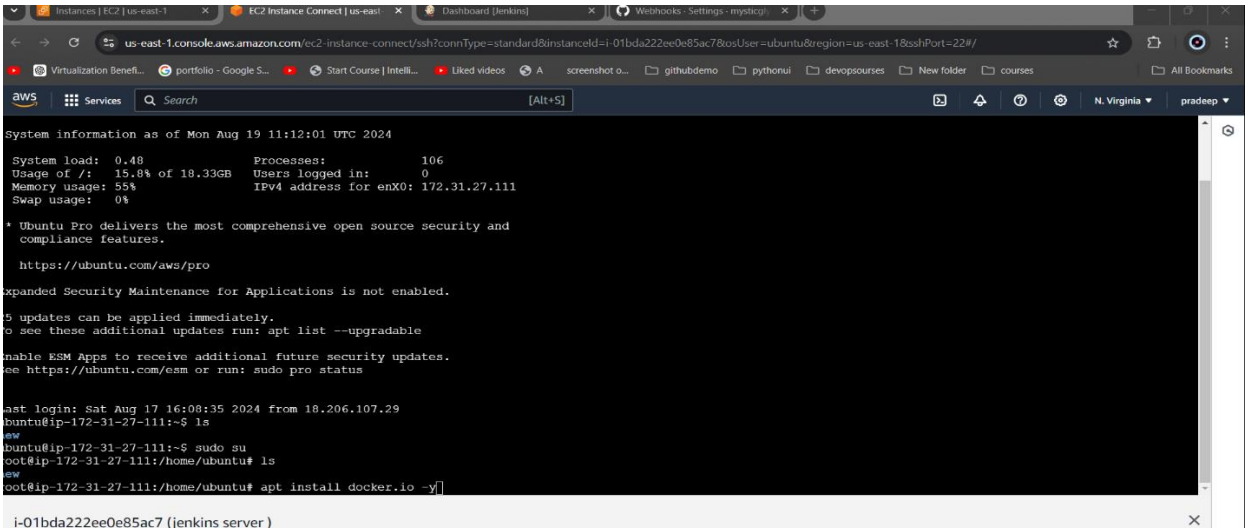


Webhook is created.

After setting up the webhook, the next step is to install Docker on the local EC2 instance. Once Docker is installed, start the Docker service. Following this, download the necessary plugins to create a Docker image and push it to Docker Hub. After successfully pushing the image, install a plugin to create a Docker container using the image stored in Docker Hub. Finally, deploy a Docker container using the pushed image.

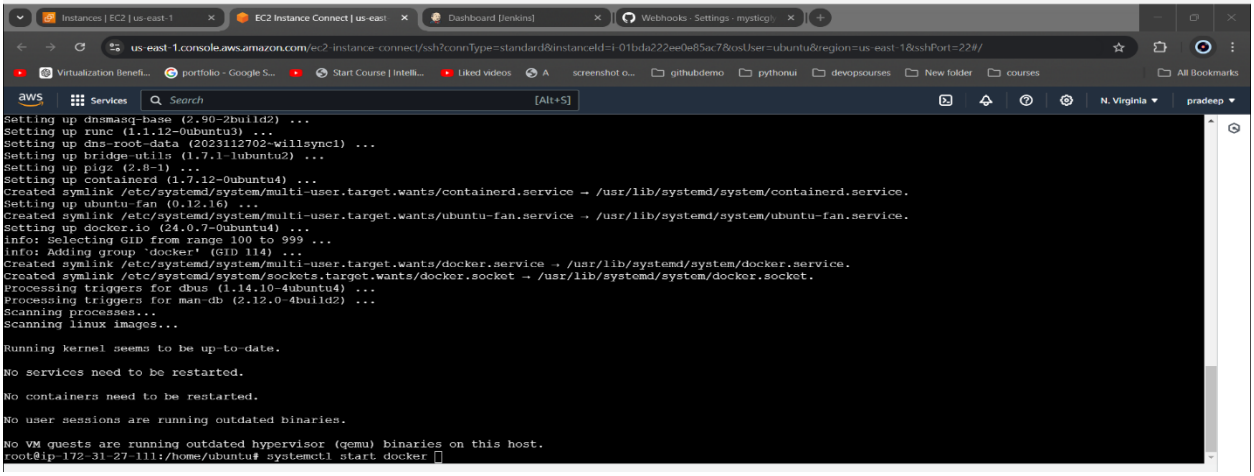
<u>Installing docker in local host:</u>

-> sudo apt install docker.io -y

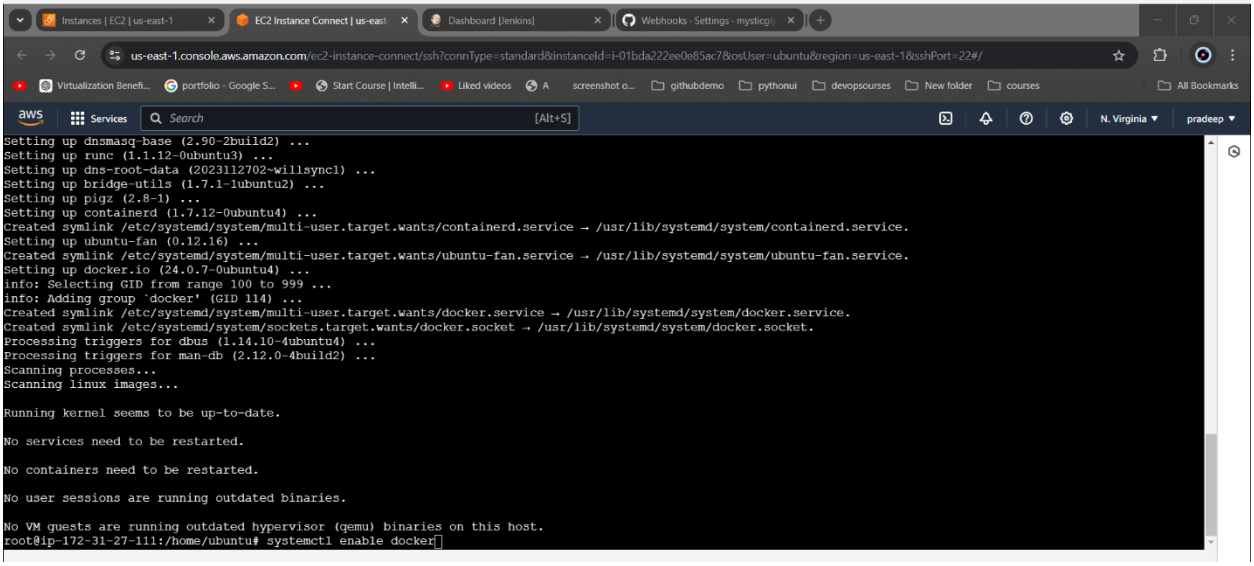


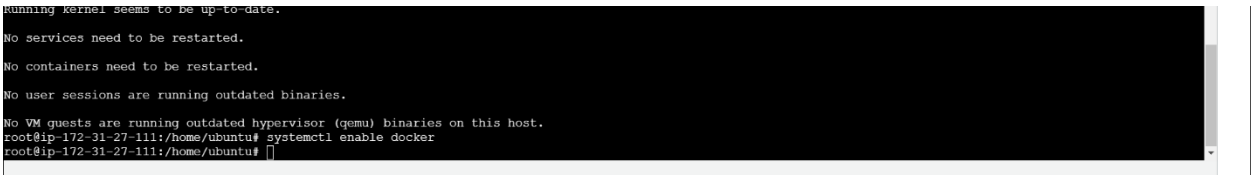
Start docker:

➔ **systemctl start docker**



➔ **systemctl enable docker**

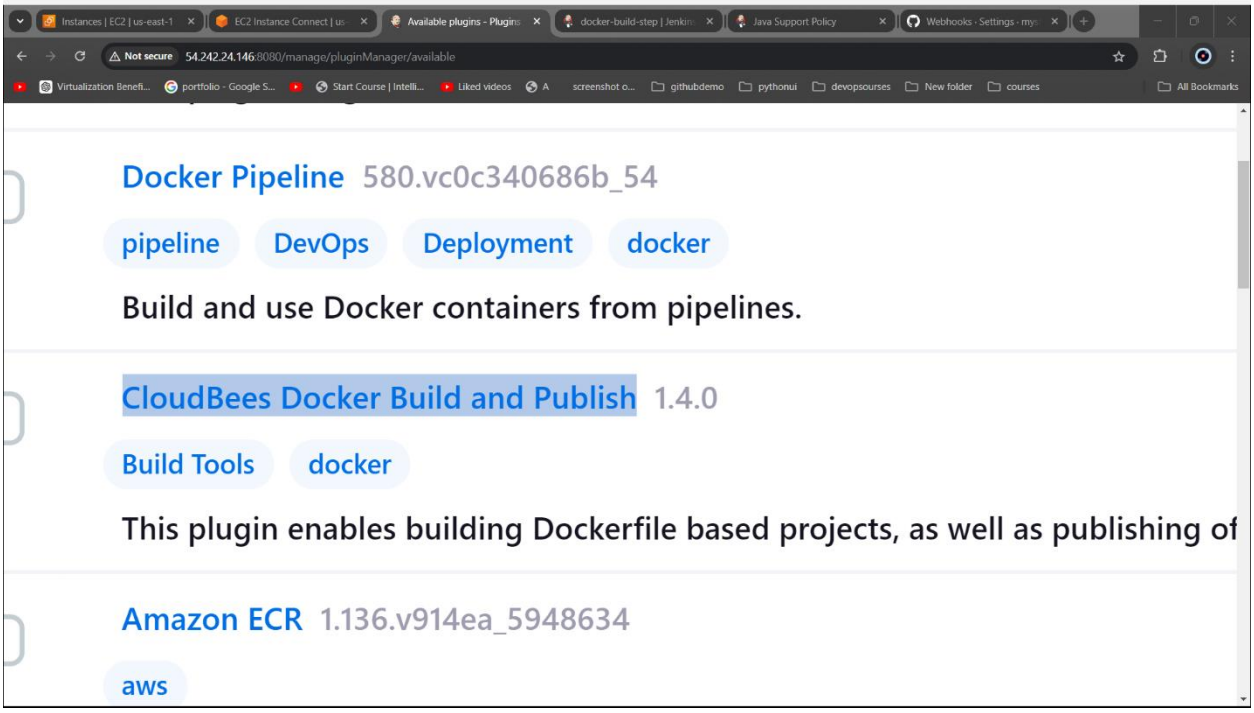




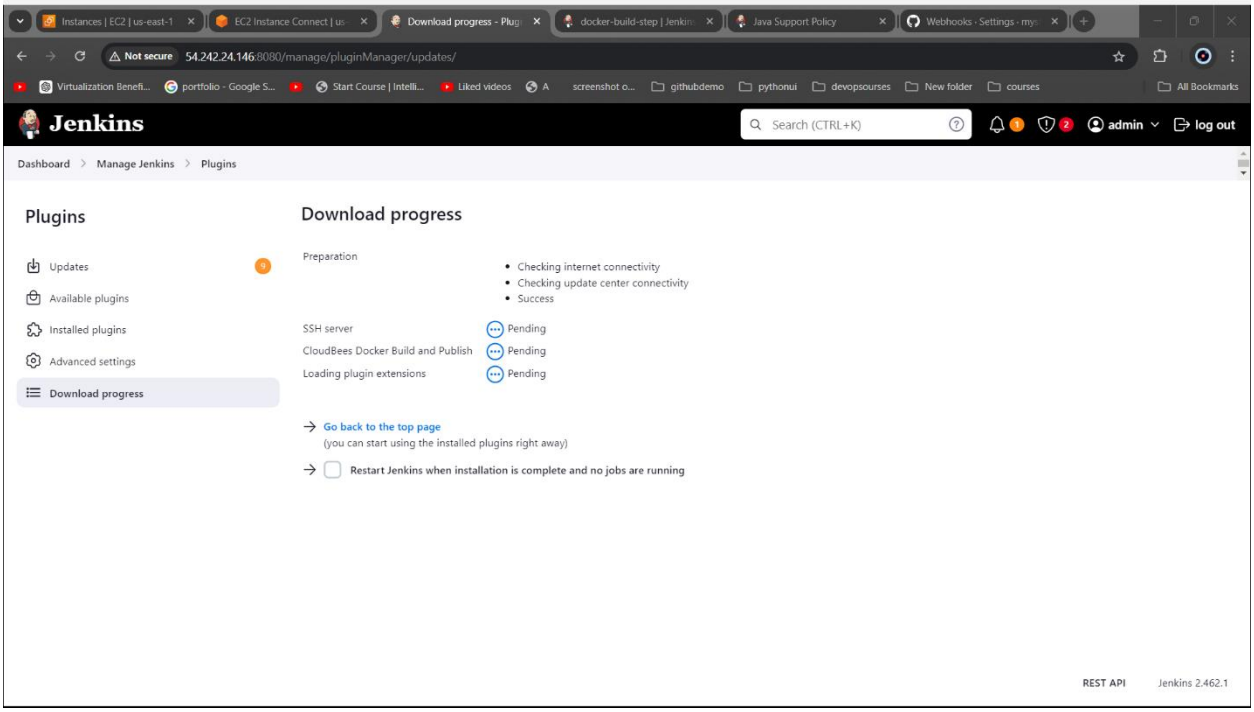
Installing plugins:

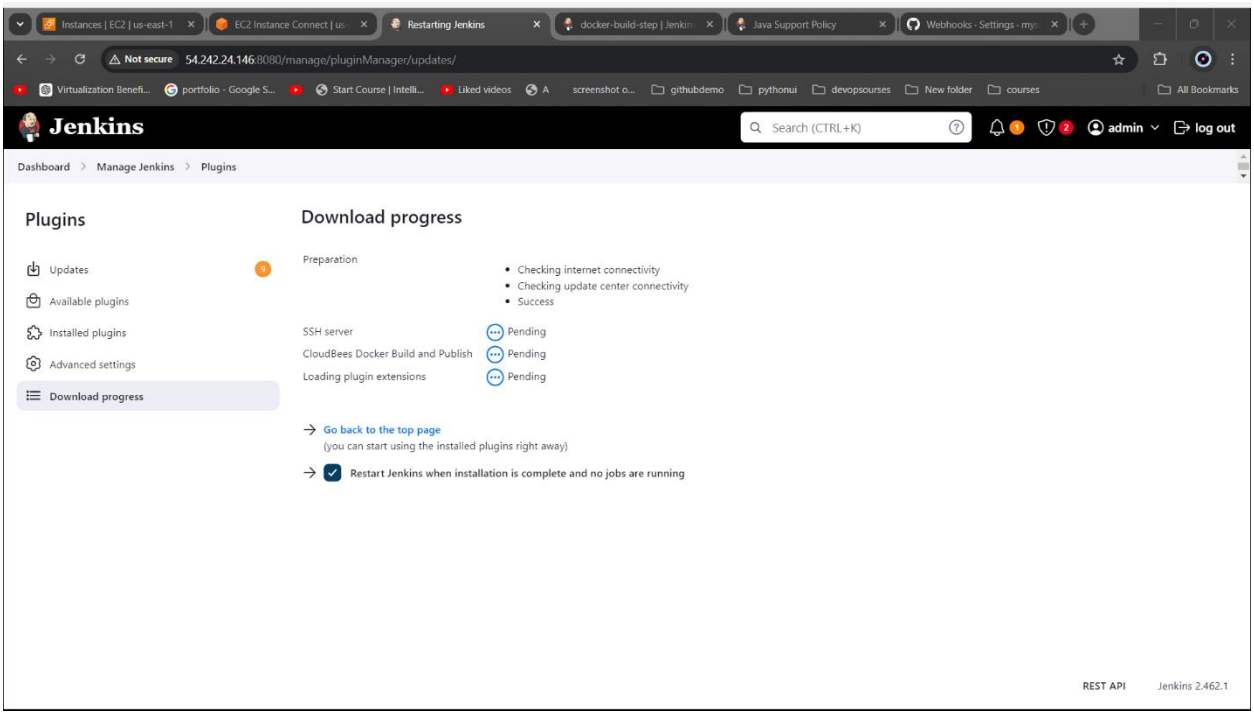
This plugin enables building Dockerfile based projects, as well as publishing of the built images/repos to the docker registry.

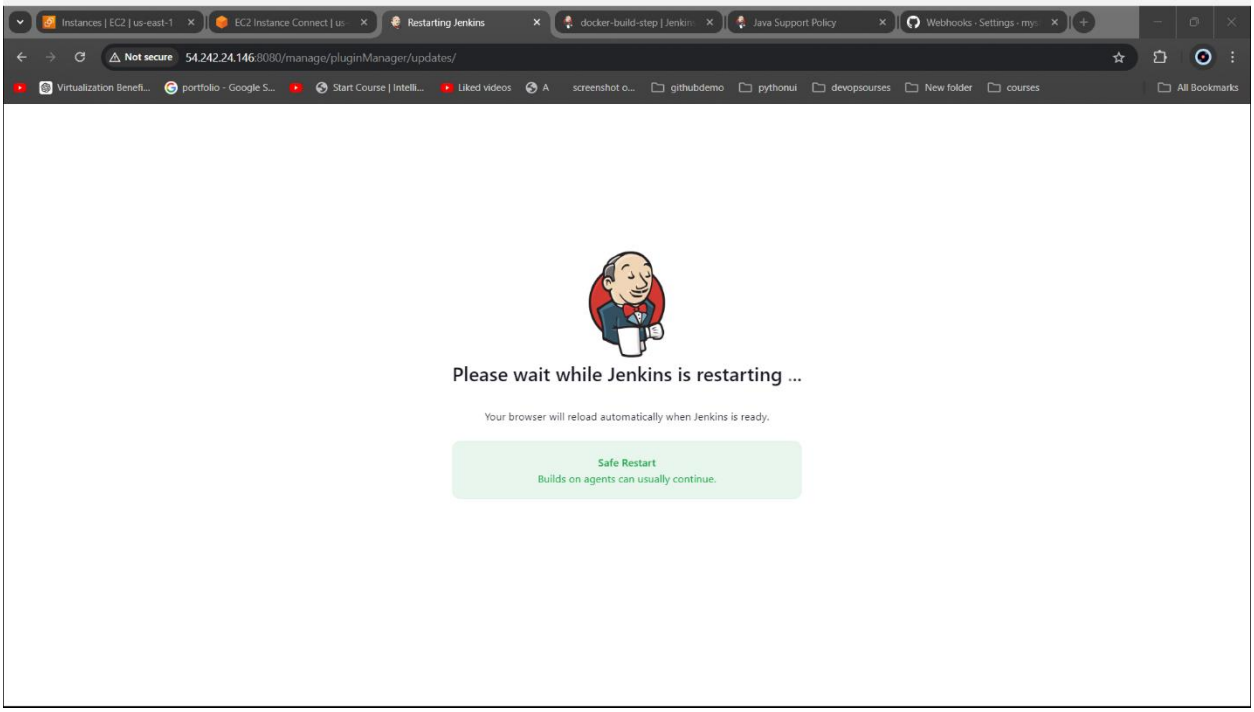
⇨ CloudBees Docker Build and Publish



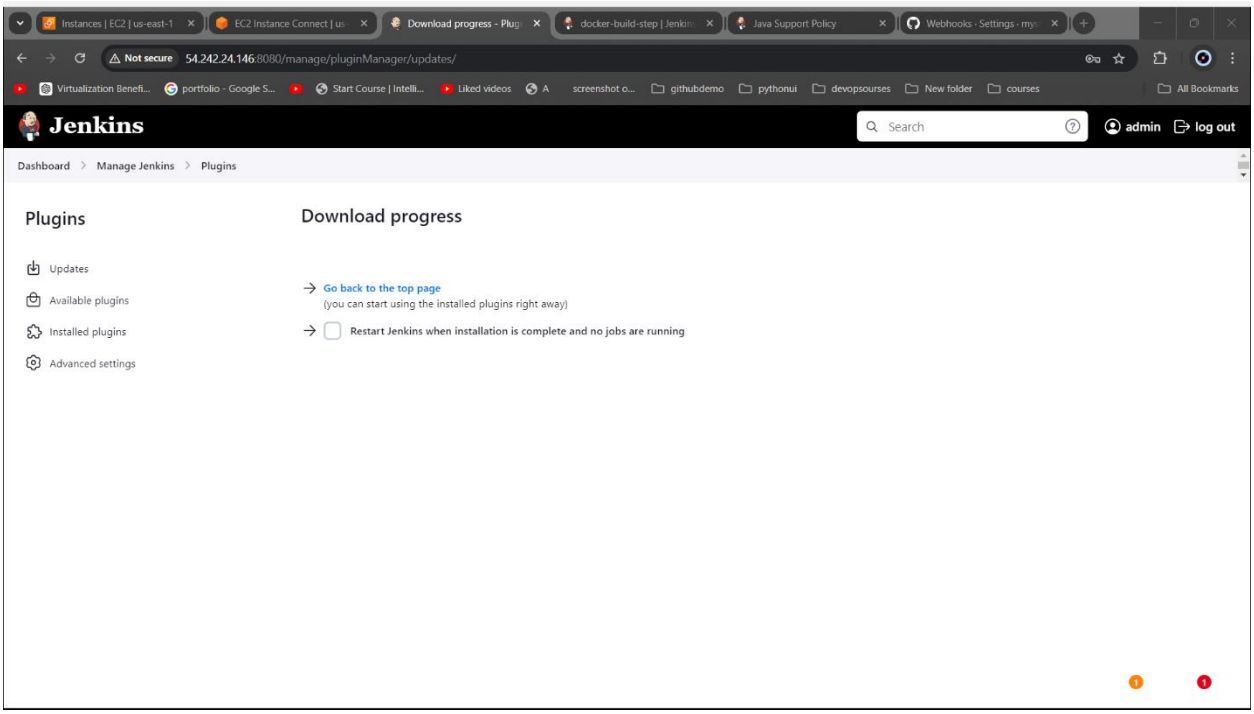
Installing this plugin:



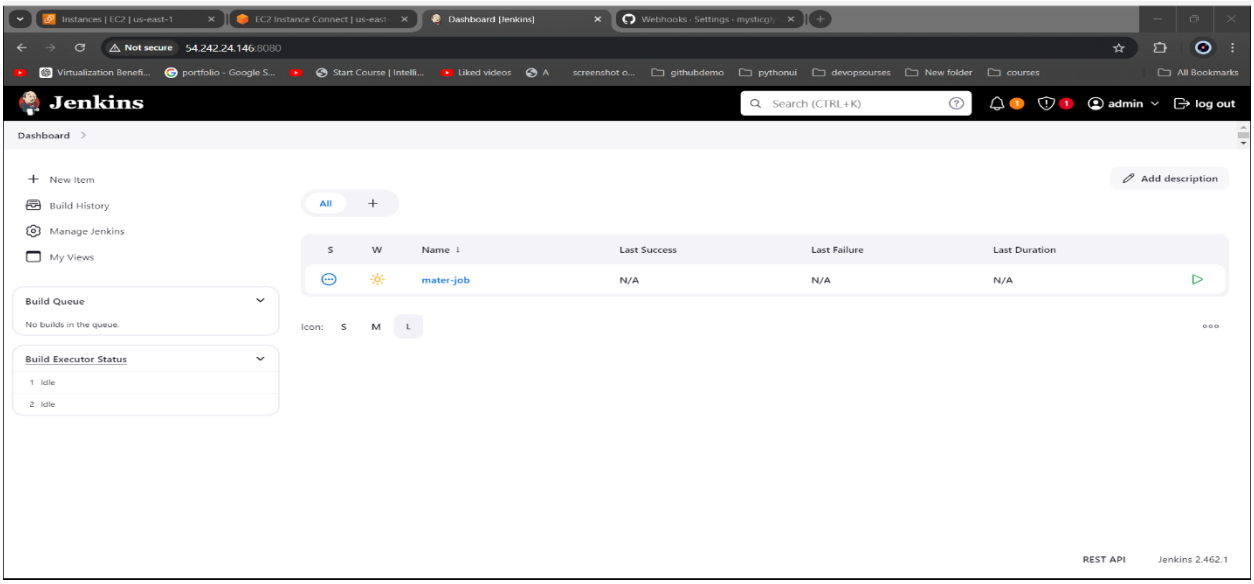




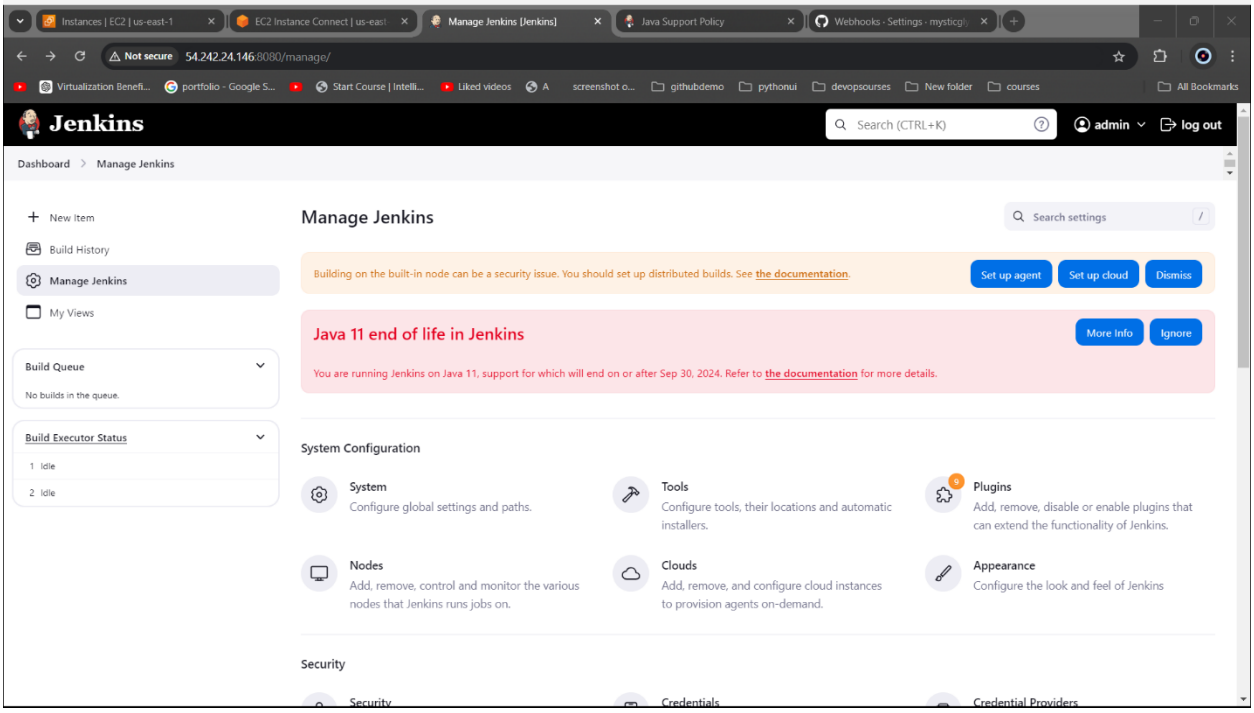
Plugin is installing.



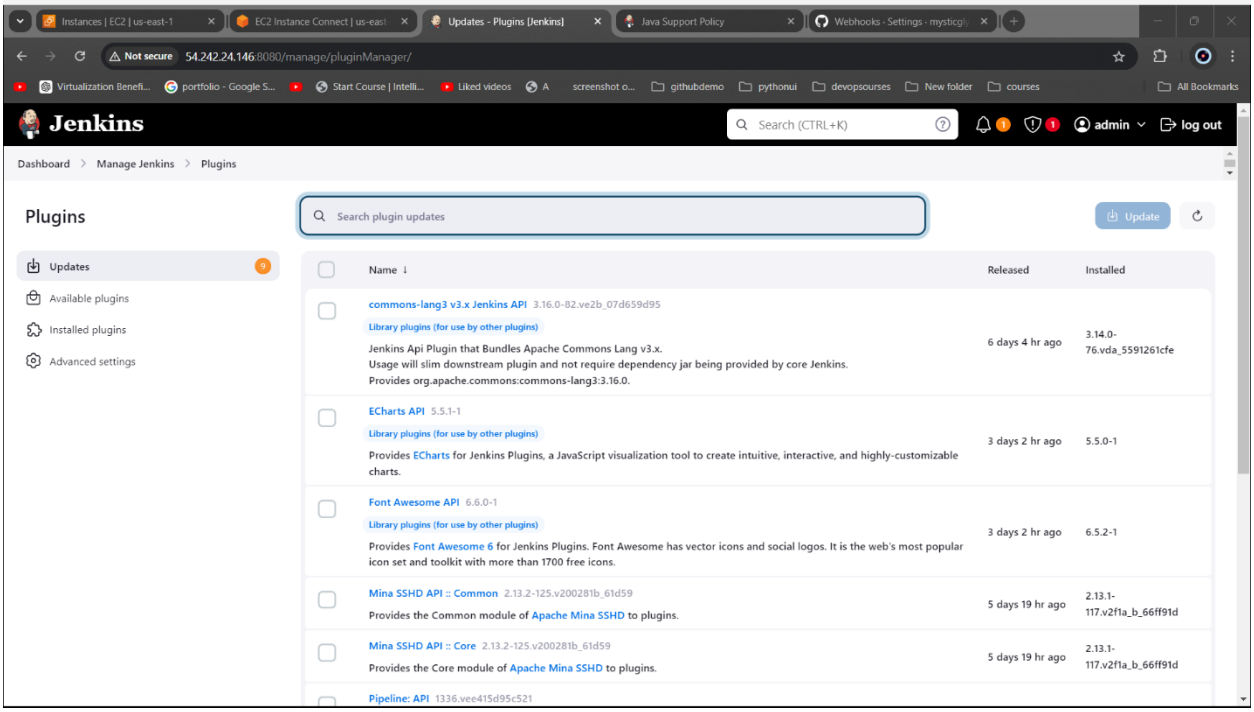
Plugin is installed.



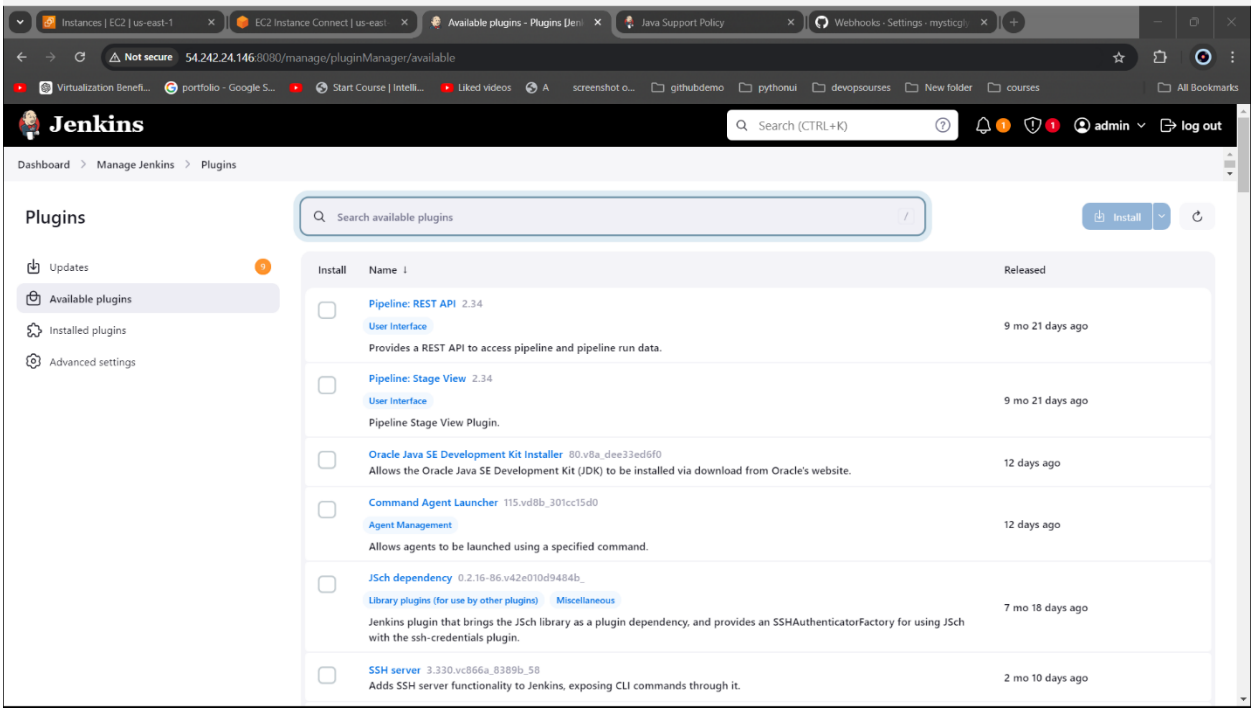
➔ manage Jenkins :



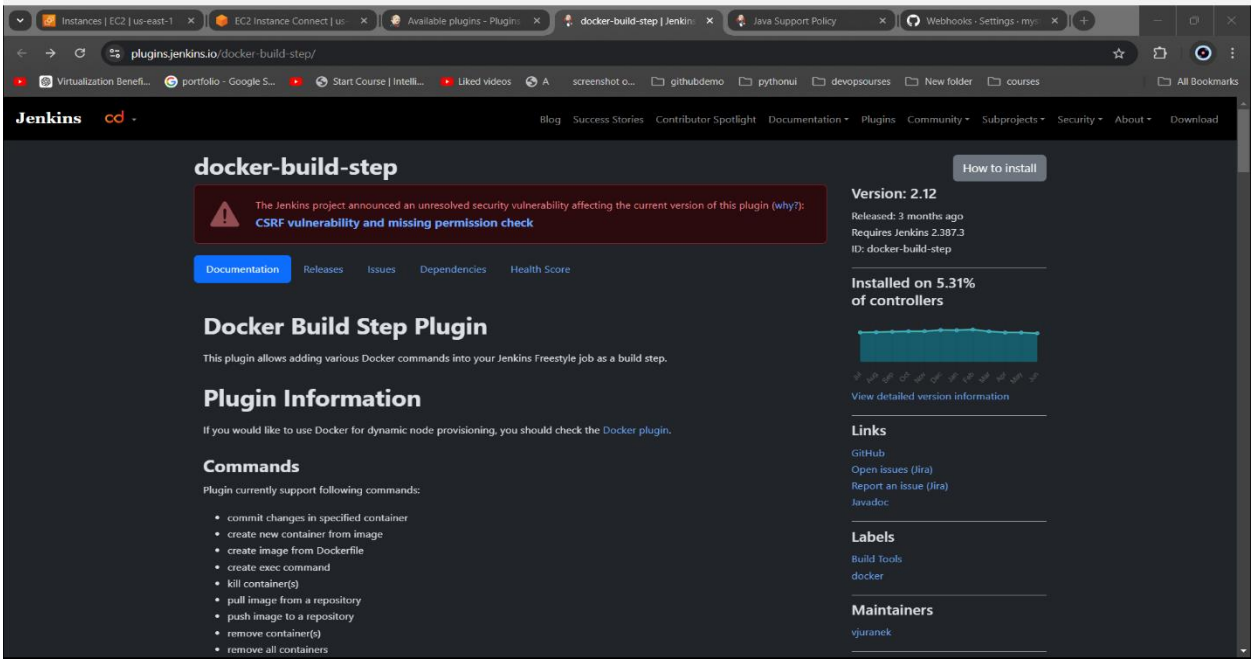
➔ plugins:



➔ available plugins:

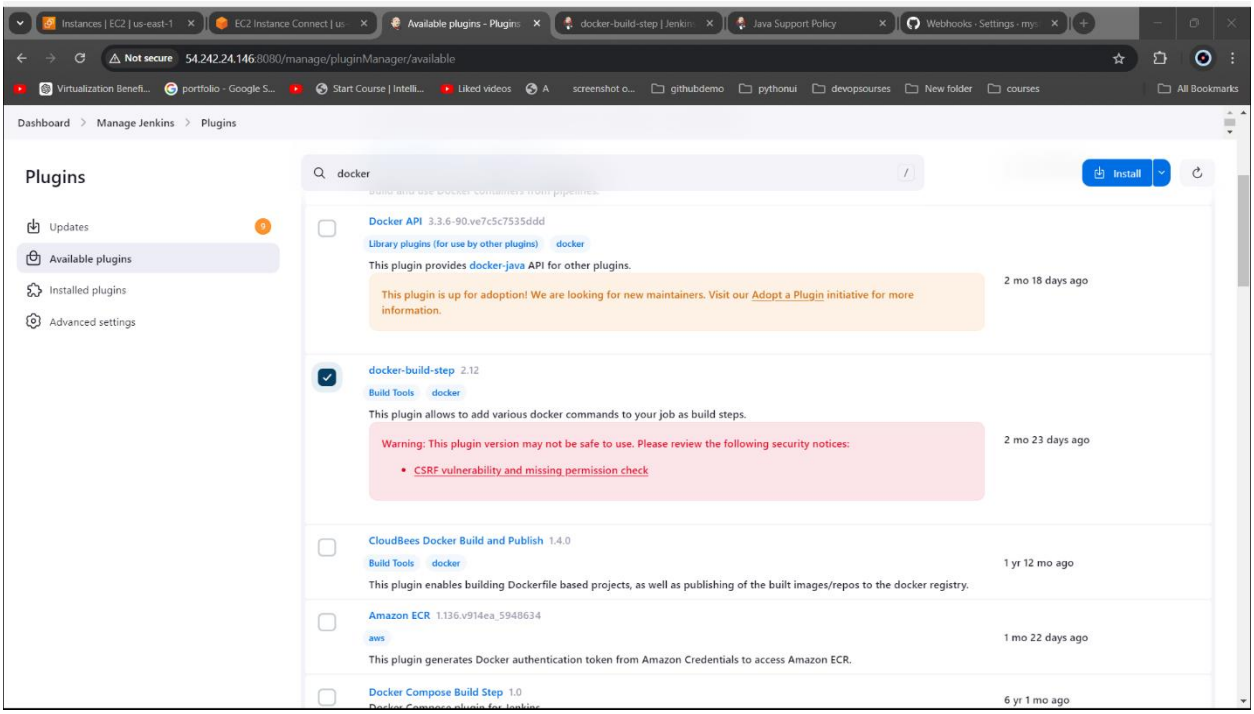


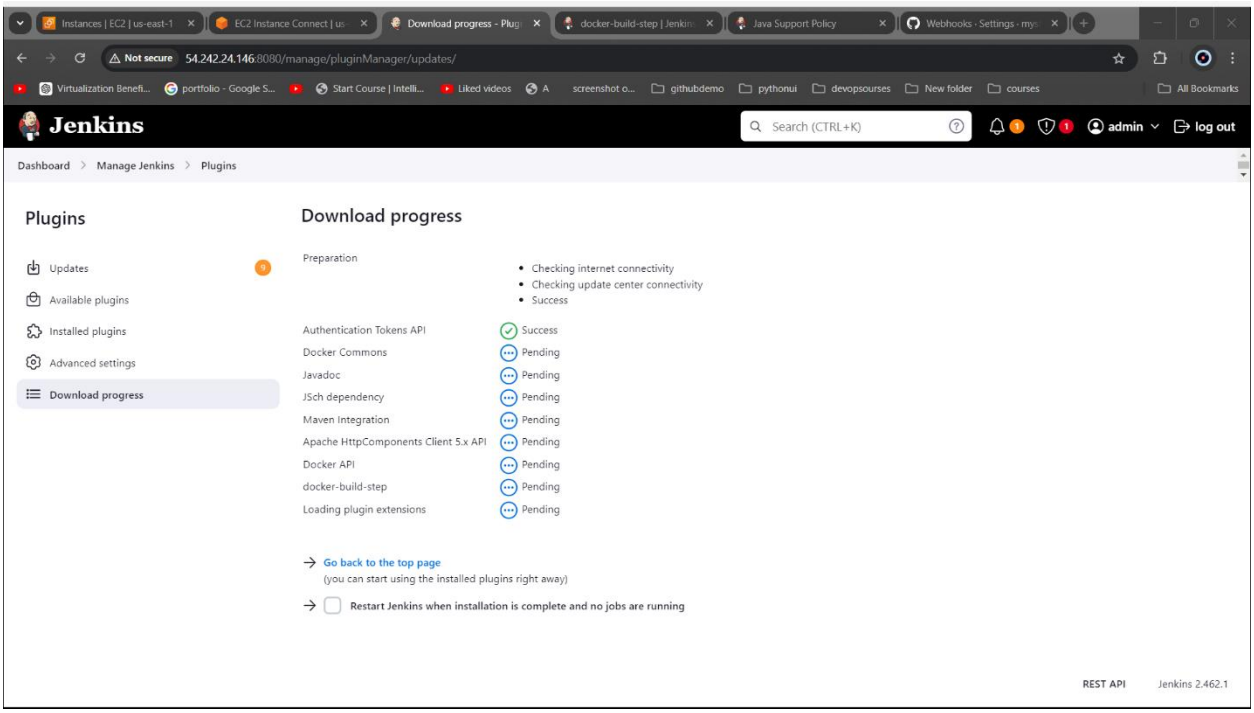
Downloading docker—build-step plugin:



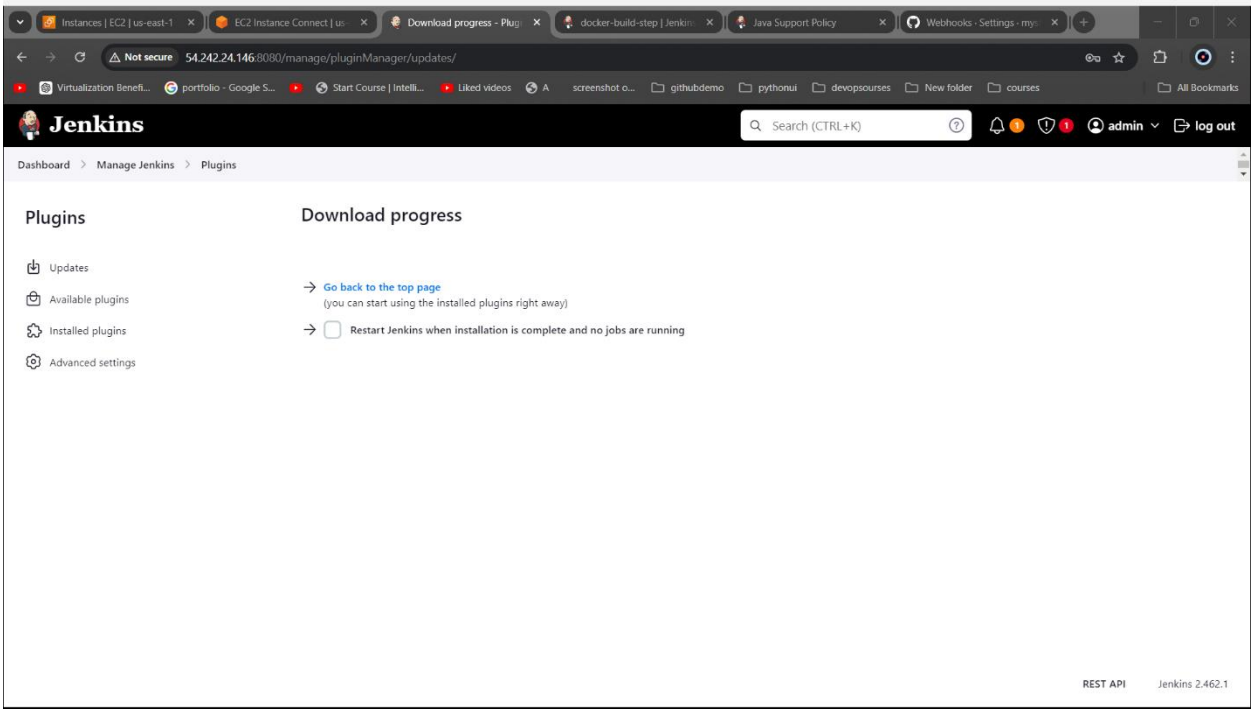
Plugin currently support following commands:

- commit changes in specified container
- create new container from image
- create image from Dockerfile
- create exec command
- kill container(s)
- pull image from a repository
- push image to a repository
- remove container(s)
- remove all containers
- restart container(s)
- start container(s)
- stop container(s)
- stop all containers
- start/stop all containers created from specified image
- start exec command


Not secure 54.242.24.146:8080/manage/pluginManager/available
Dashboard > Manage Jenkins > Plugins
Plugins
Updates
Available plugins
Installed plugins
Advanced settings
docker
Install
Docker API 3.3.6-90.ve7c5c7535ddd
Library plugins (for use by other plugins) docker
This plugin provides docker-java API for other plugins.
This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.
2 mo 18 days ago
docker-build-step 2.12
Build Tools docker
This plugin allows to add various docker commands to your job as build steps.
Warning: This plugin version may not be safe to use. Please review the following security notices:
CSRF vulnerability and missing permission check
2 mo 23 days ago
CloudBees Docker Build and Publish 1.4.0
Build Tools docker
This plugin enables building Dockerfile based projects, as well as publishing of the built images/repos to the docker registry.
1 yr 12 mo ago
Amazon ECR 1.136.v914ea_5948634
aws
This plugin generates Docker authentication token from Amazon Credentials to access Amazon ECR.
1 mo 22 days ago
Docker Compose Build Step 1.0
6 yr 1 mo ago


Not secure 54.242.24.146:8080/manage/pluginManager/updates/
Jenkins
Search (CTRL+K)
admin
log out
Dashboard > Manage Jenkins > Plugins
Plugins
Updates
Available plugins
Installed plugins
Advanced settings
Download progress
Download progress
Preparation
Checking internet connectivity
Checking update center connectivity
Success
Authentication Tokens API Success
Docker Commons Pending
Javadoc Pending
JSch dependency Pending
Maven Integration Pending
Apache HttpComponents Client 5.x API Pending
Docker API Pending
docker-build-step Pending
Loading plugin extensions Pending
Go back to the top page
(you can start using the installed plugins right away)
Restart Jenkins when installation is complete and no jobs are running
REST API
Jenkins 2.462.1

Plugin is downloaded:



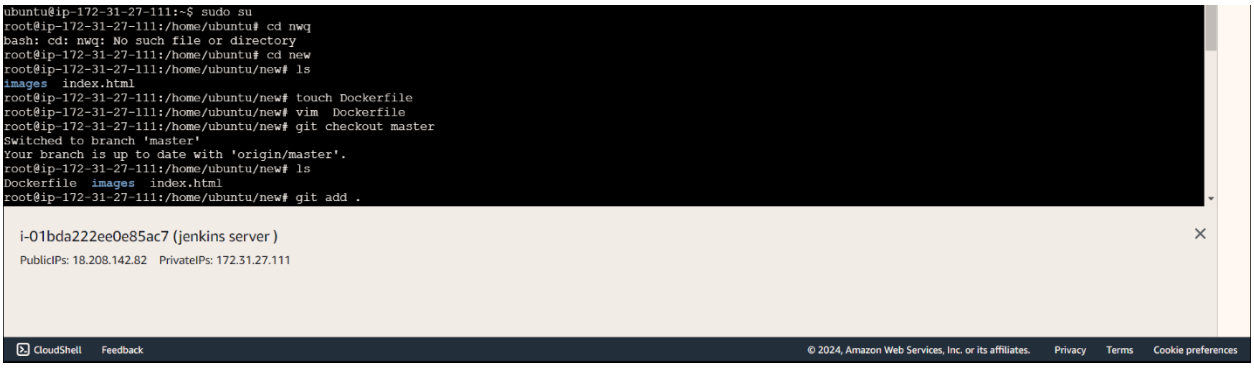
Writing and Pushing the Dockerfile to GitHub Repository:

We need to write a Dockerfile that defines the instructions to create a Docker container. Once the Dockerfile is complete, it should be pushed to the GitHub repository. This allows the Dockerfile to be used for automated builds and deployments as part of the CI/CD pipeline.

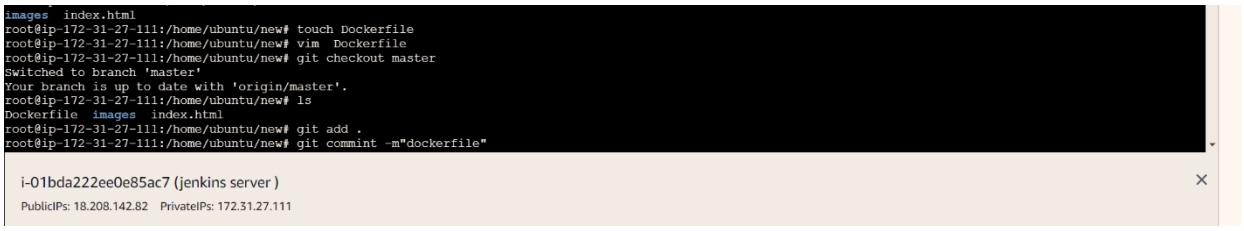
```
FROM ubuntu:latest
RUN apt update -y
RUN apt install apache2 -y
RUN apt install git -y
RUN cd /var/www/html && rm -rf * && git clone
https://github.com/mysticglyph/jenkinscasetudy.git
RUN mv /var/www/html/jenkinscasetudy/* /var/www/html
ENTRYPOINT apachectl -D FOREGROUND
```

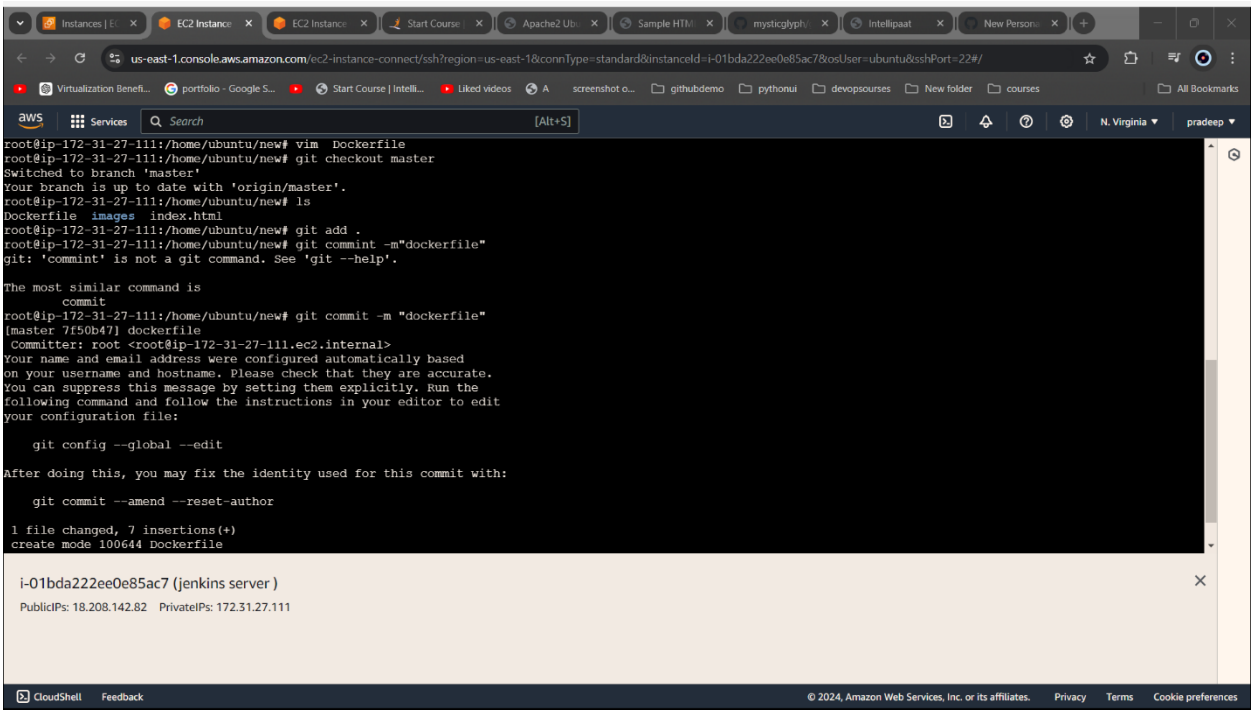
Adding Dockerfile to staging area:

➔ git add .

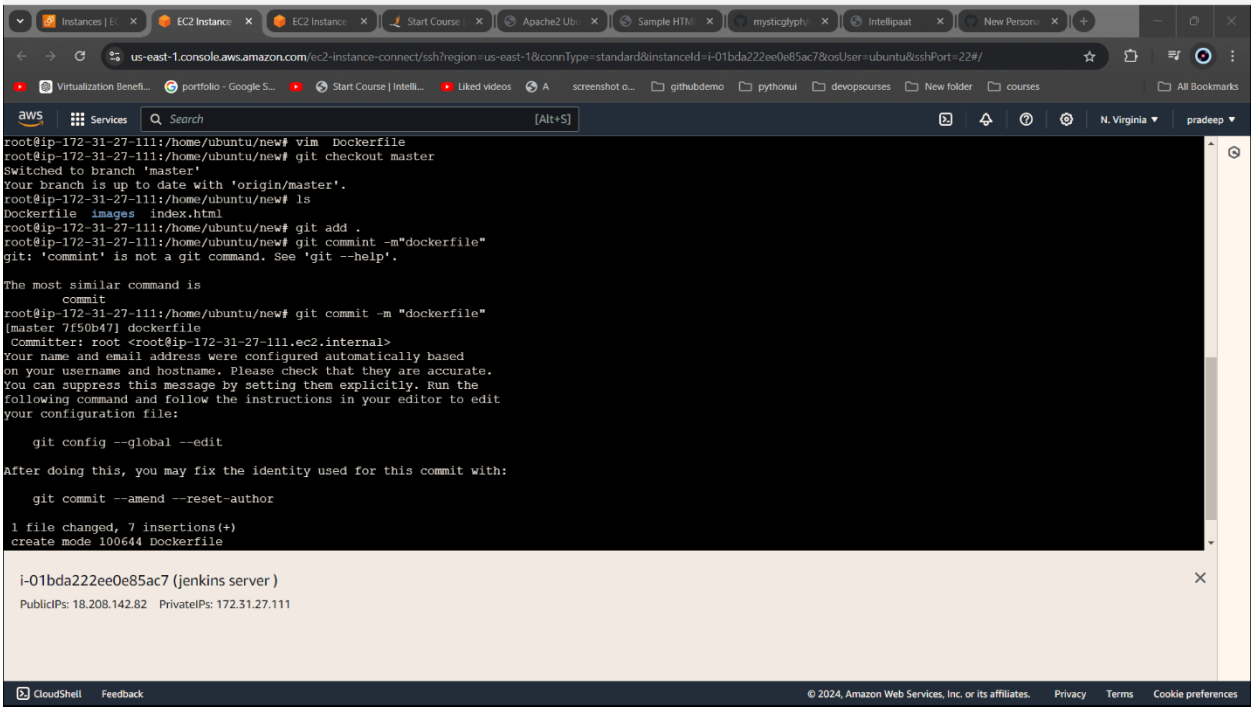


Committing the changes:



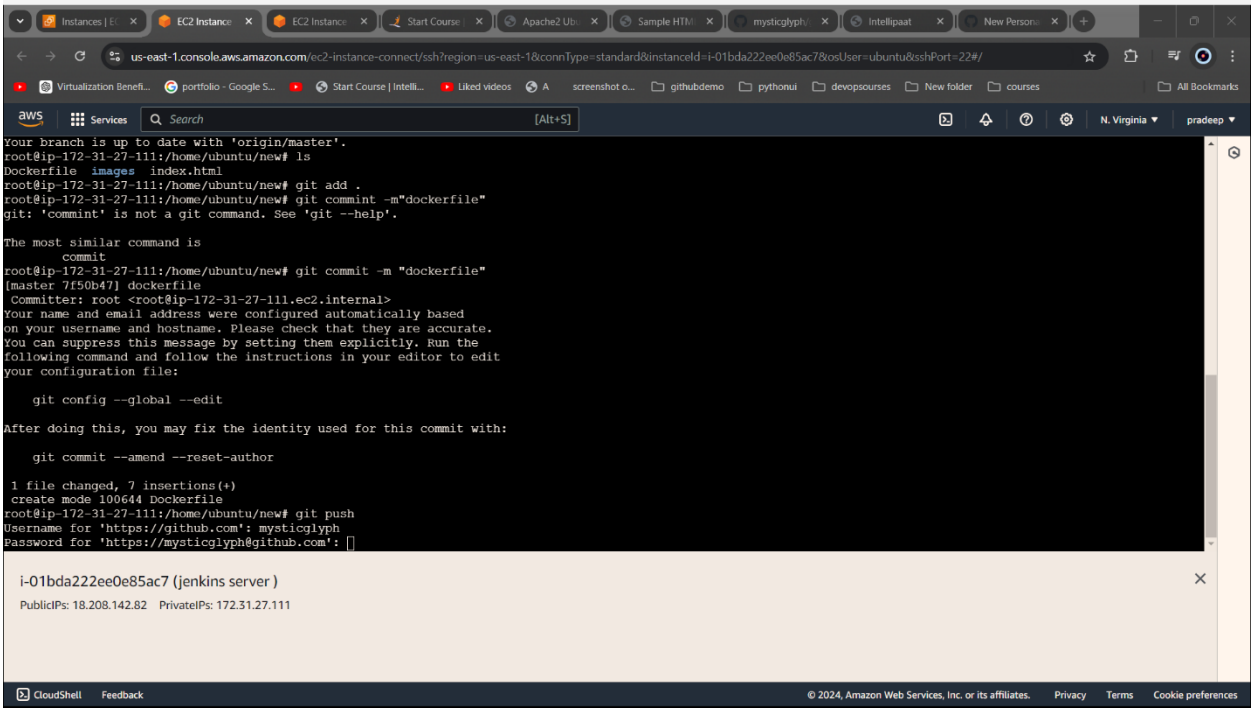


Changes successfully committed.

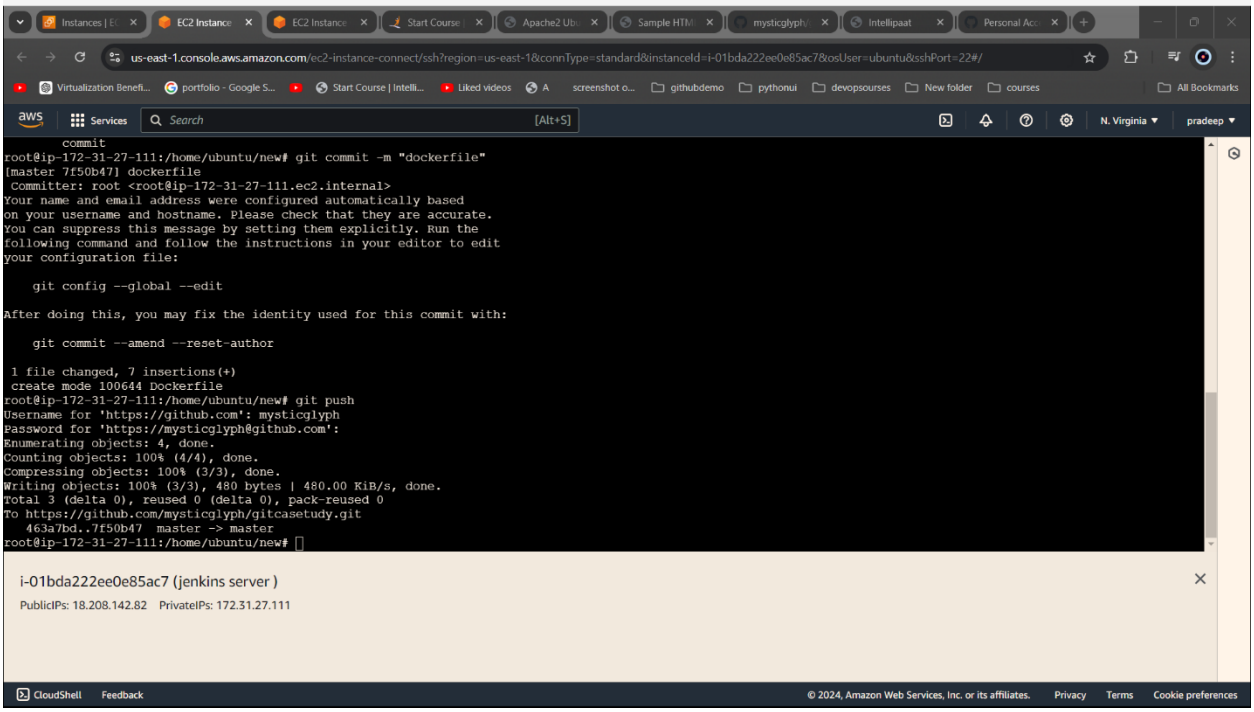


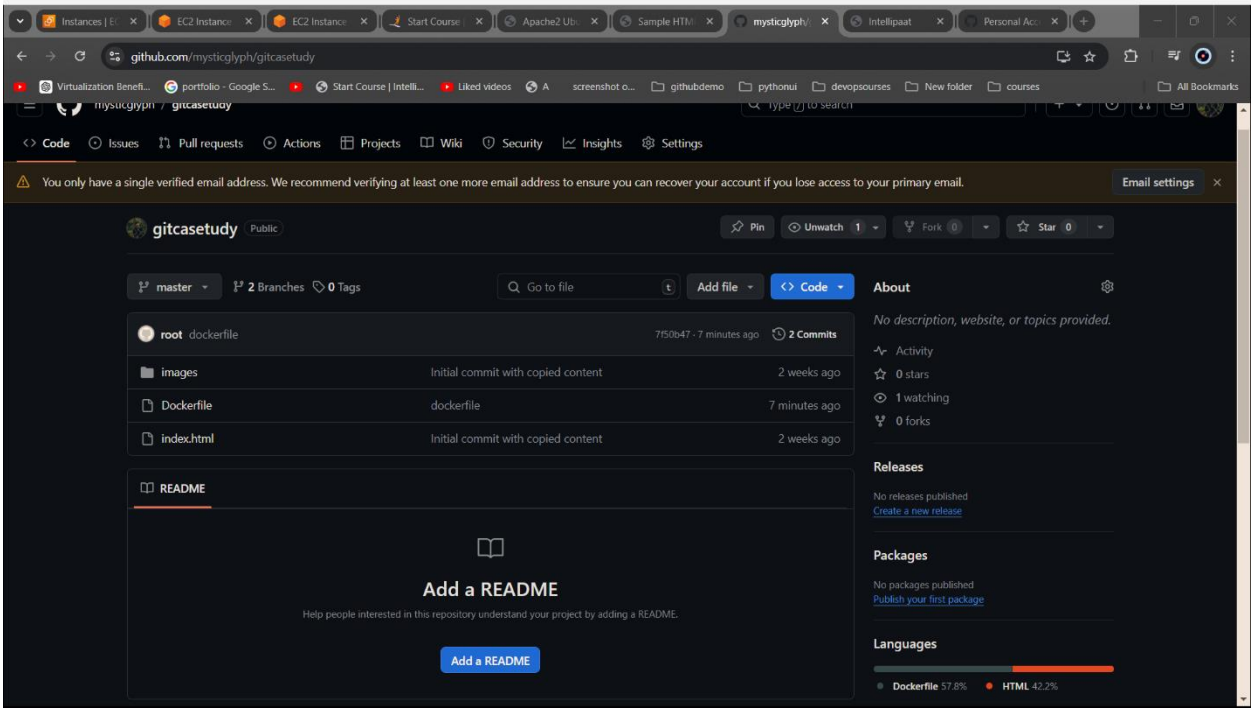
Pushing to remote repo:

➔ git push



Successfully pushed to master branch:

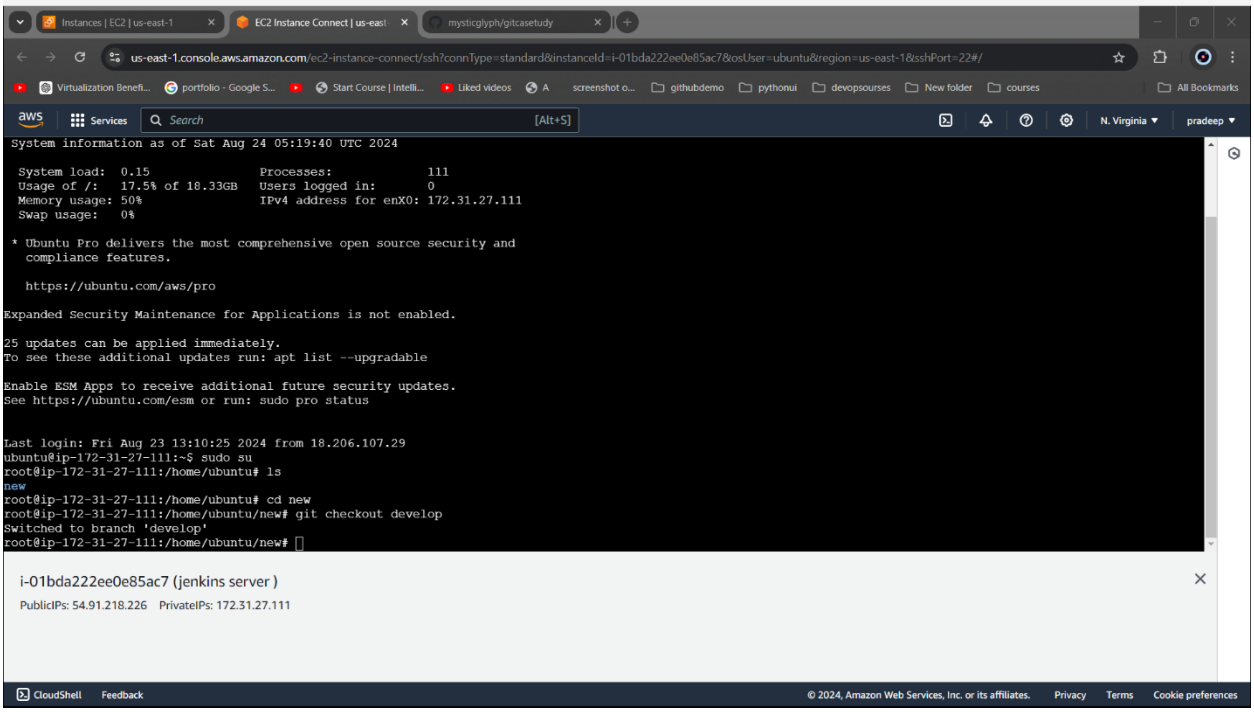




Dockerfile is pushed to remote repo.

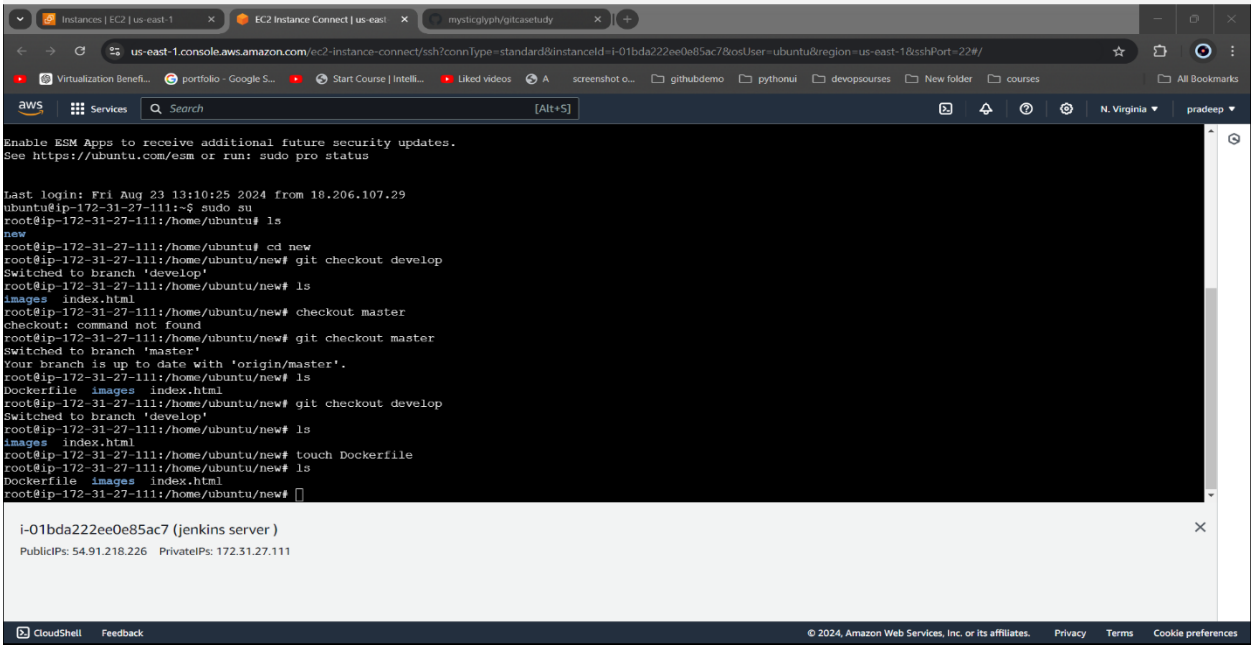
We pushed the Dockerfile to our master branch; we need to do the same for our develop branch as well.

➔ Git checkout develop

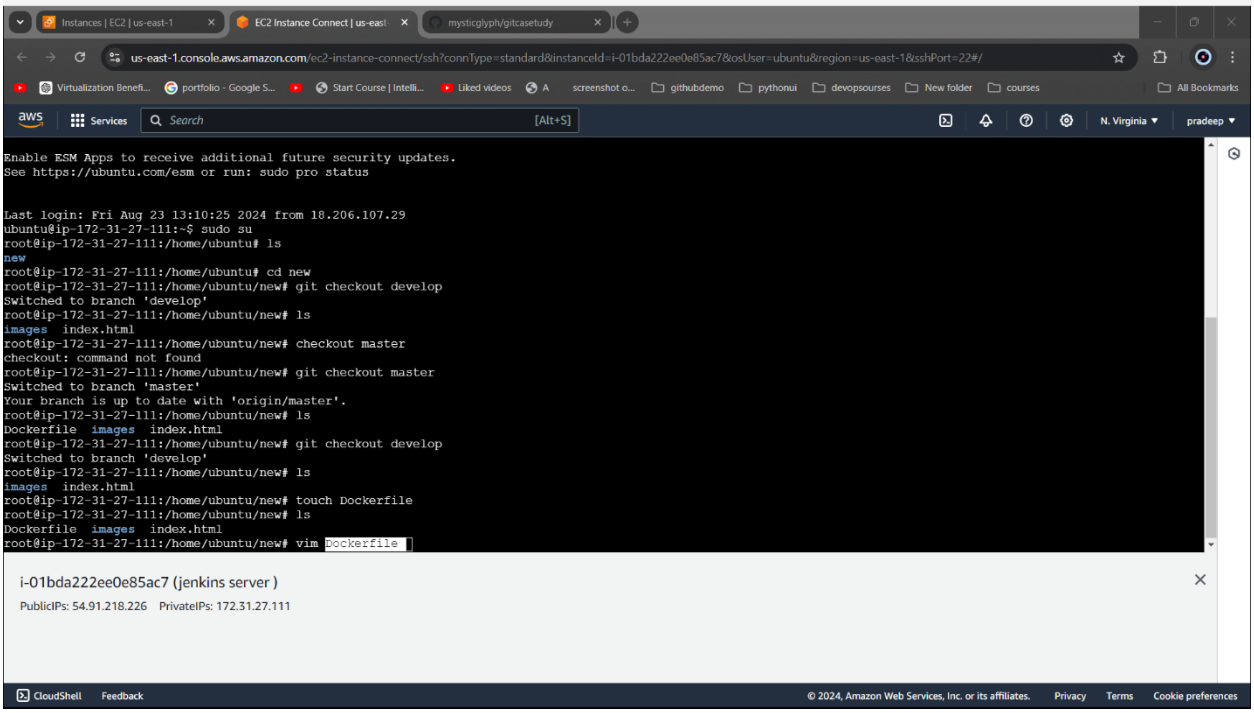


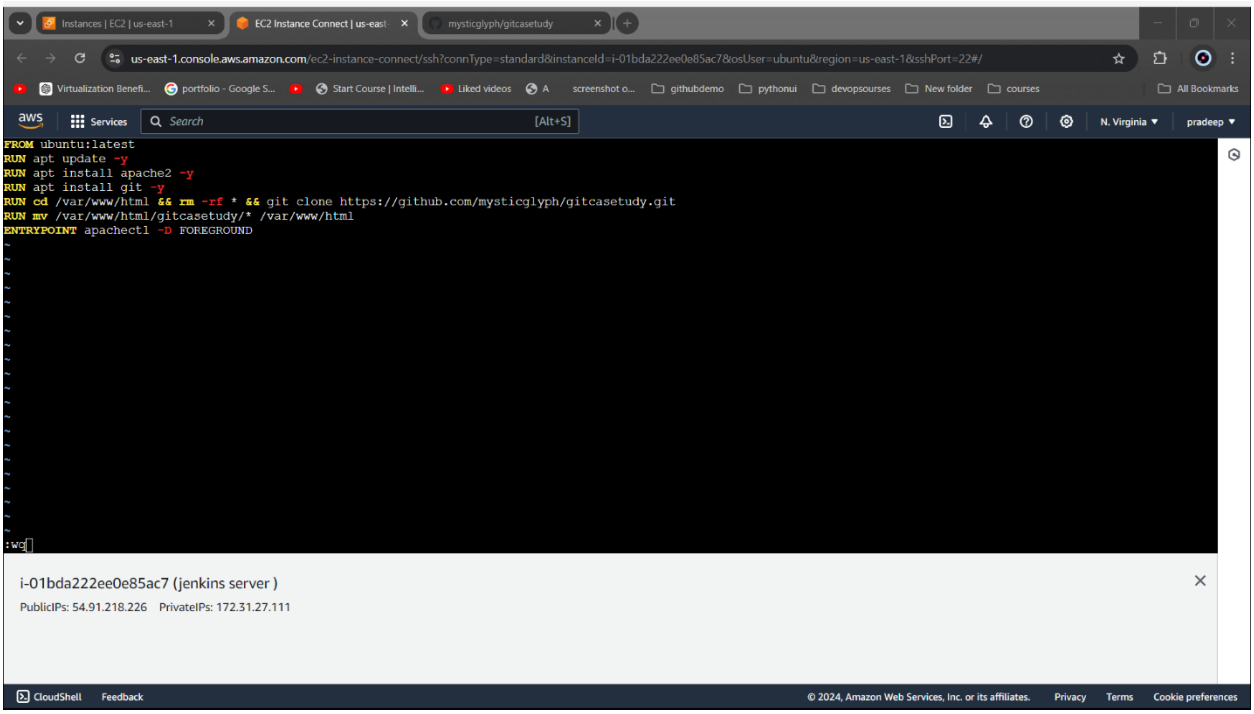
Creating dockerfile:

➔ Touch dockerfile

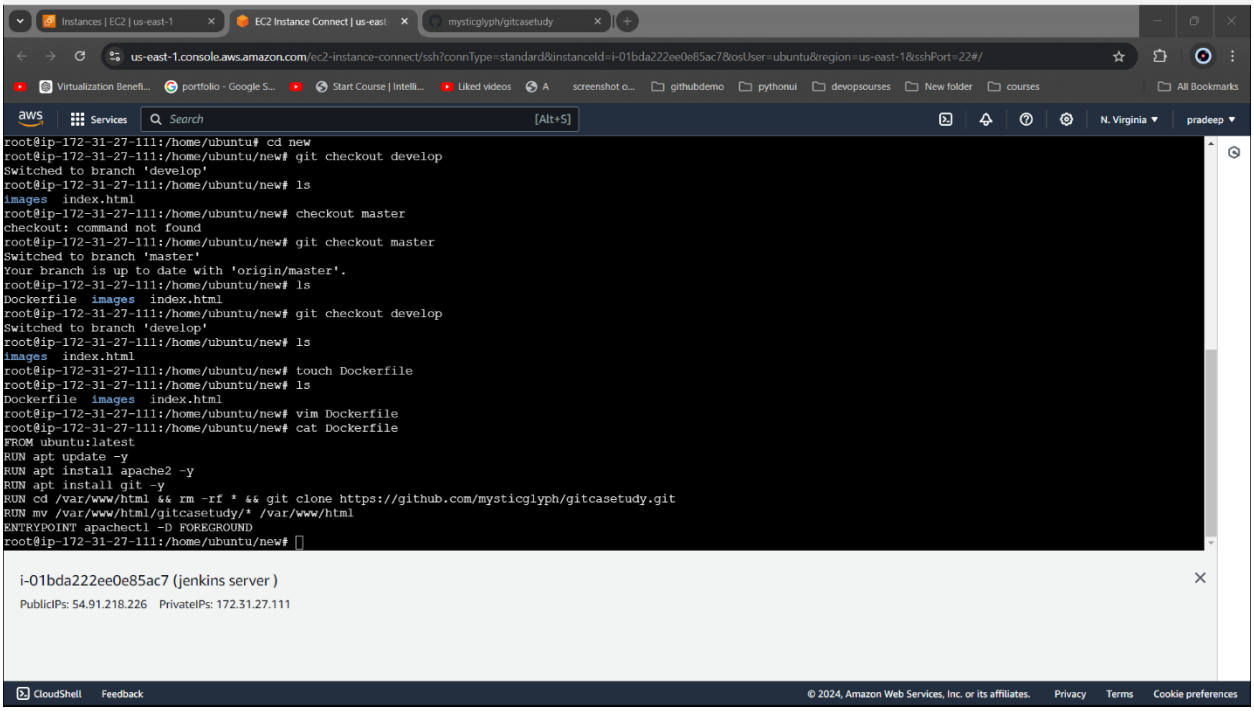


Open dockerfile with cim texteditor:

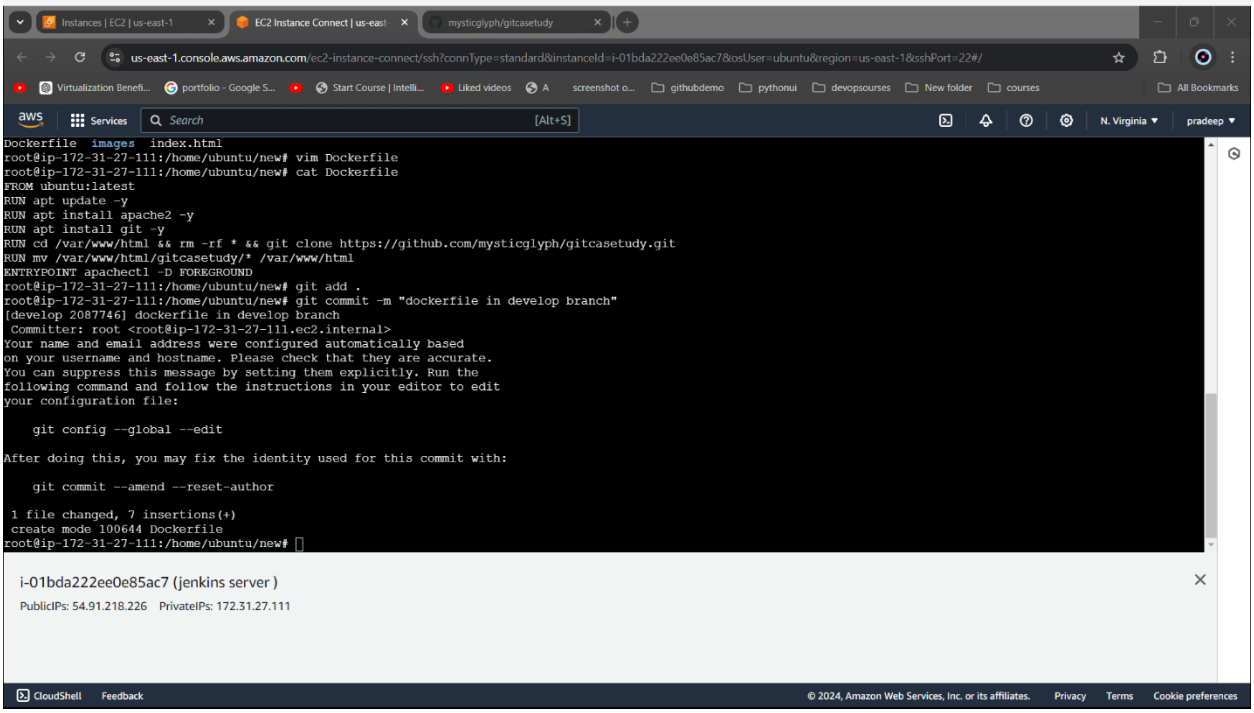




Save the docker file.



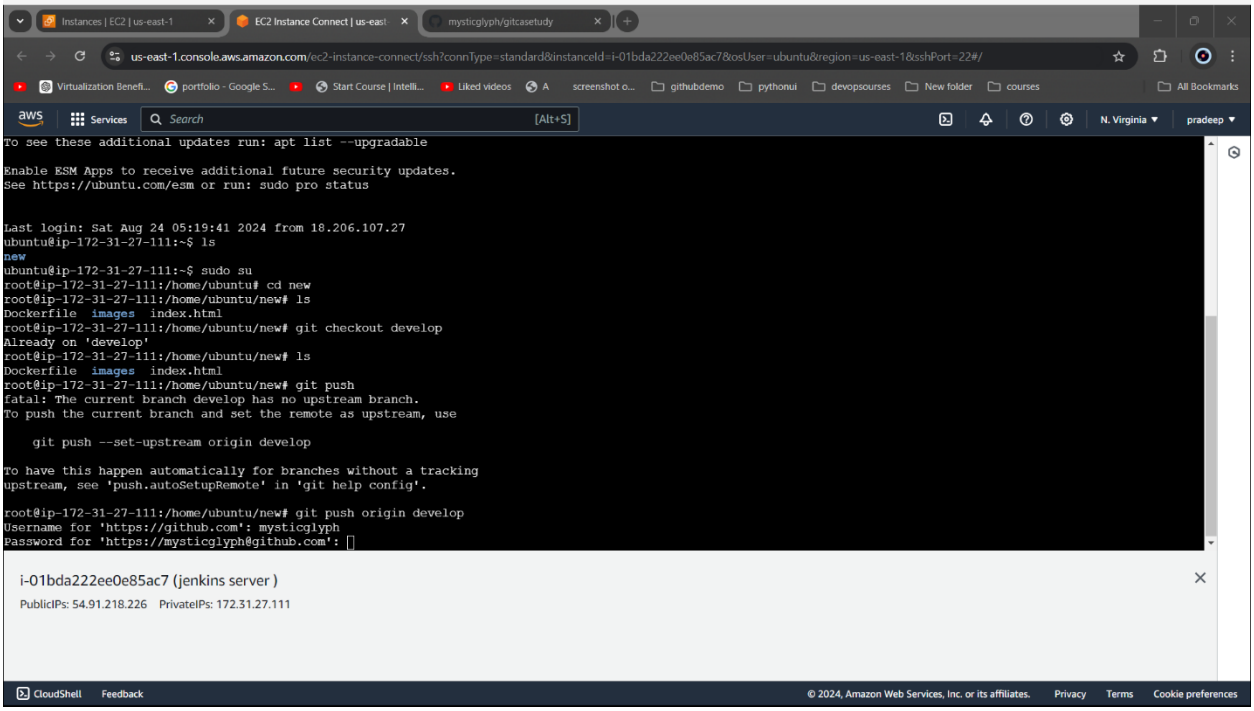
Add the file to the staging area, commit the changes, and then push the file to the develop branch.

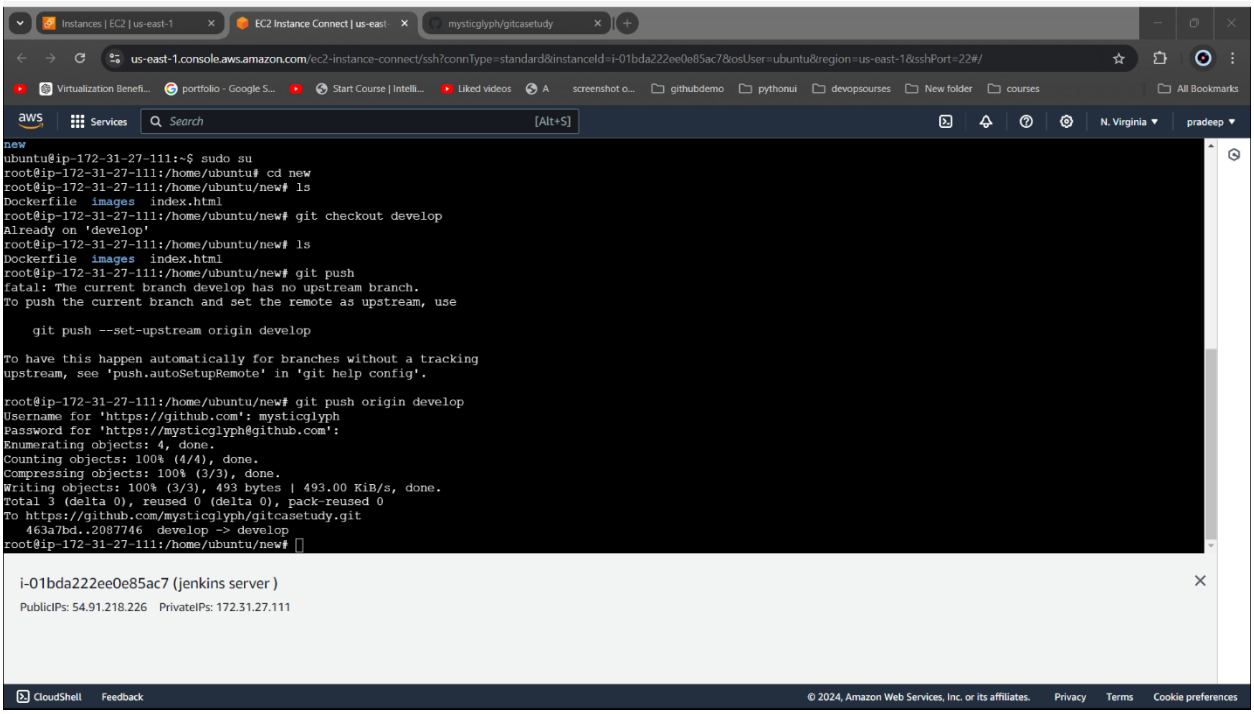


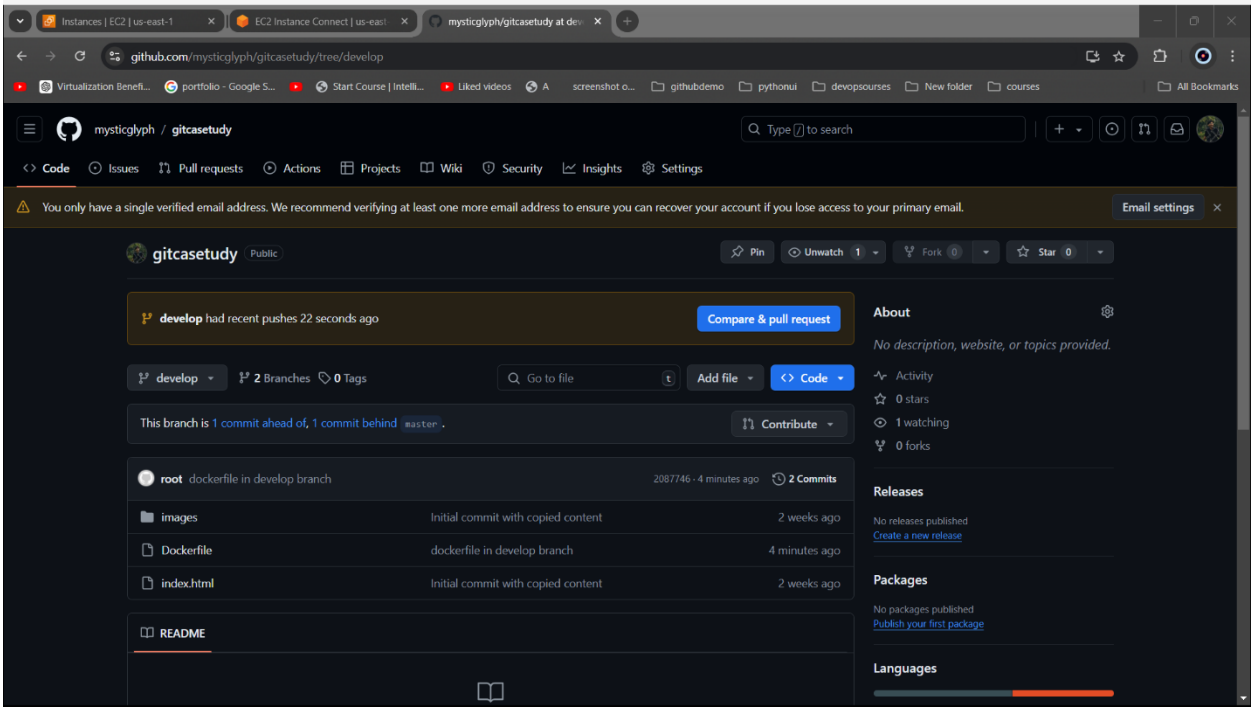
Added to staging area and changes are committed.

Pushing to remote repo:

➔ Git push



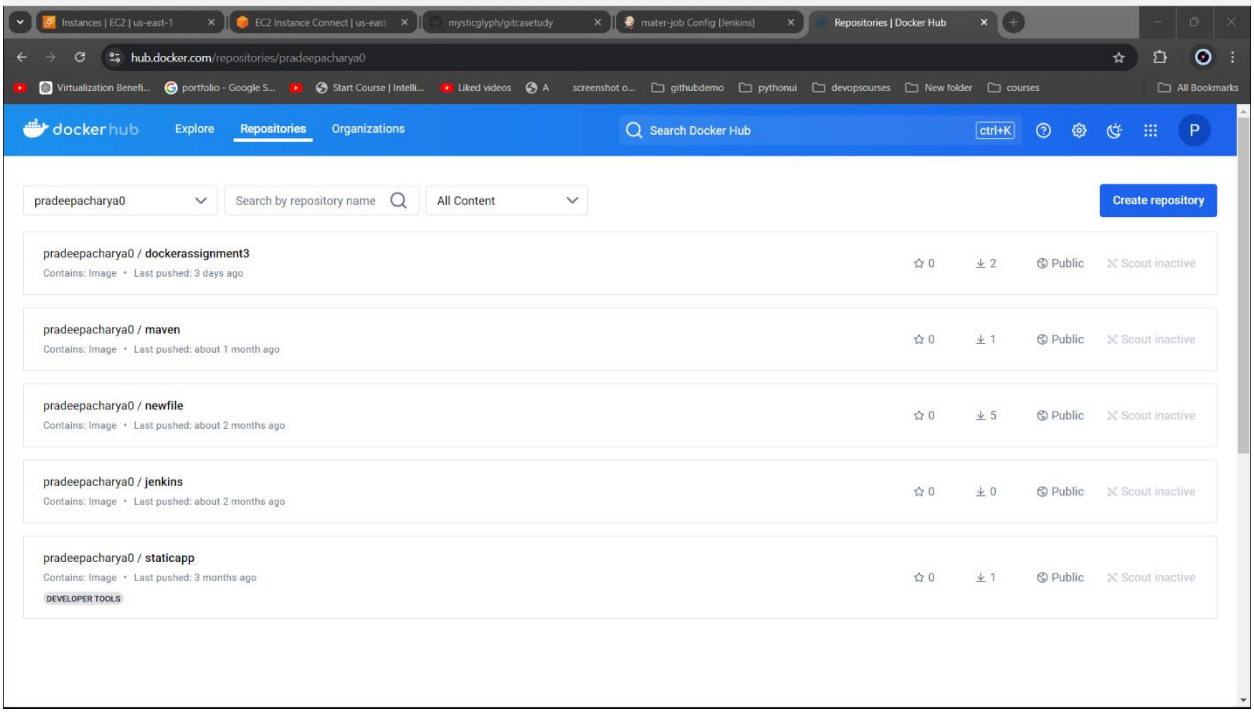




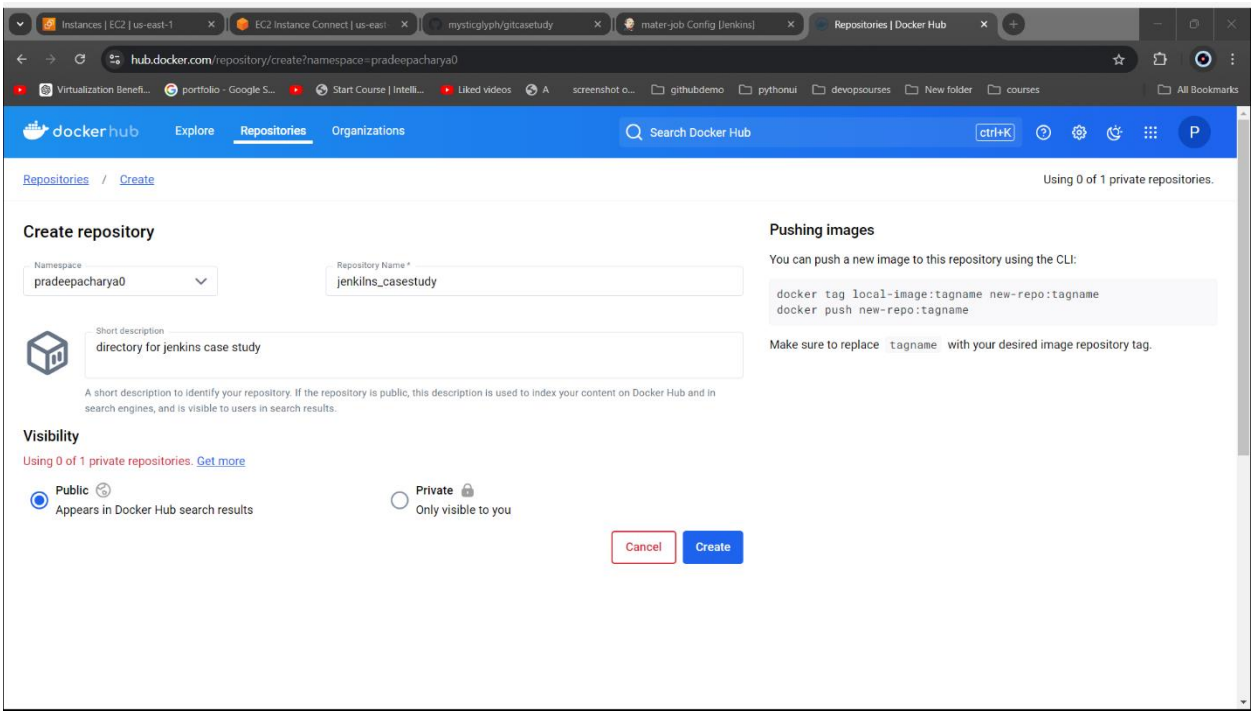
Push is successful.

We need to create a Docker repository for building the image and publishing it. We'll use Docker Hub for publishing.

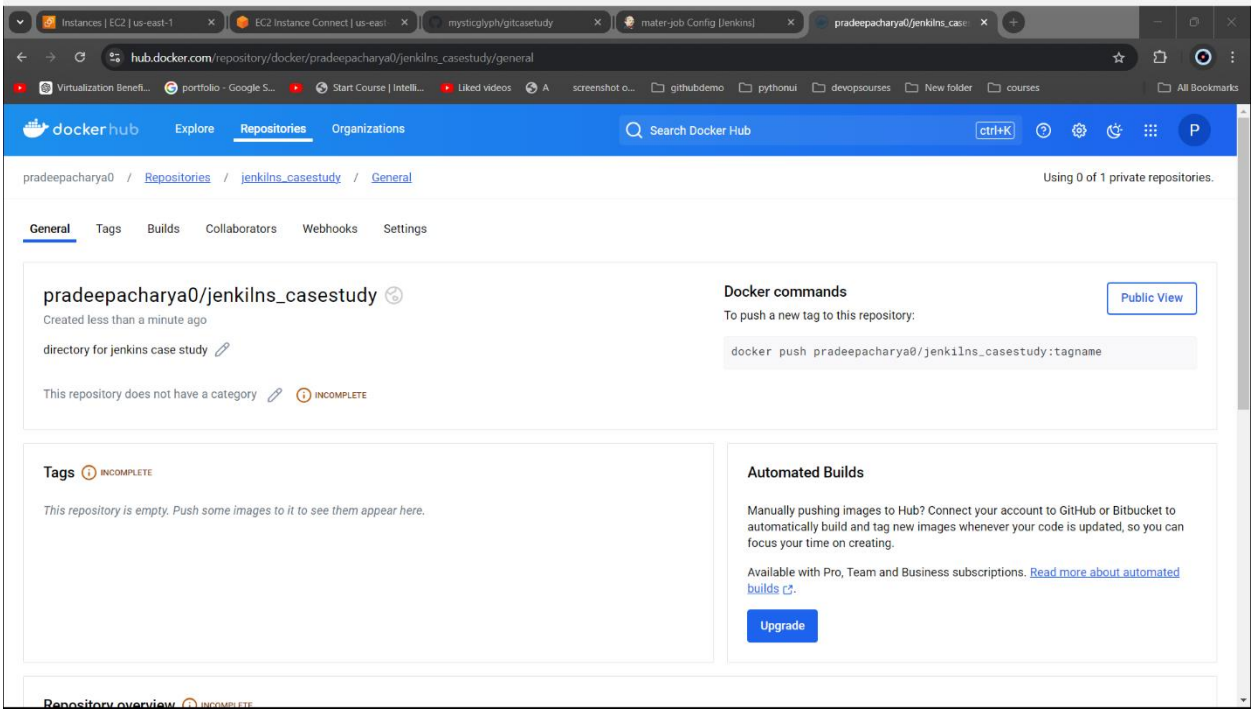
So we will create docker repository:



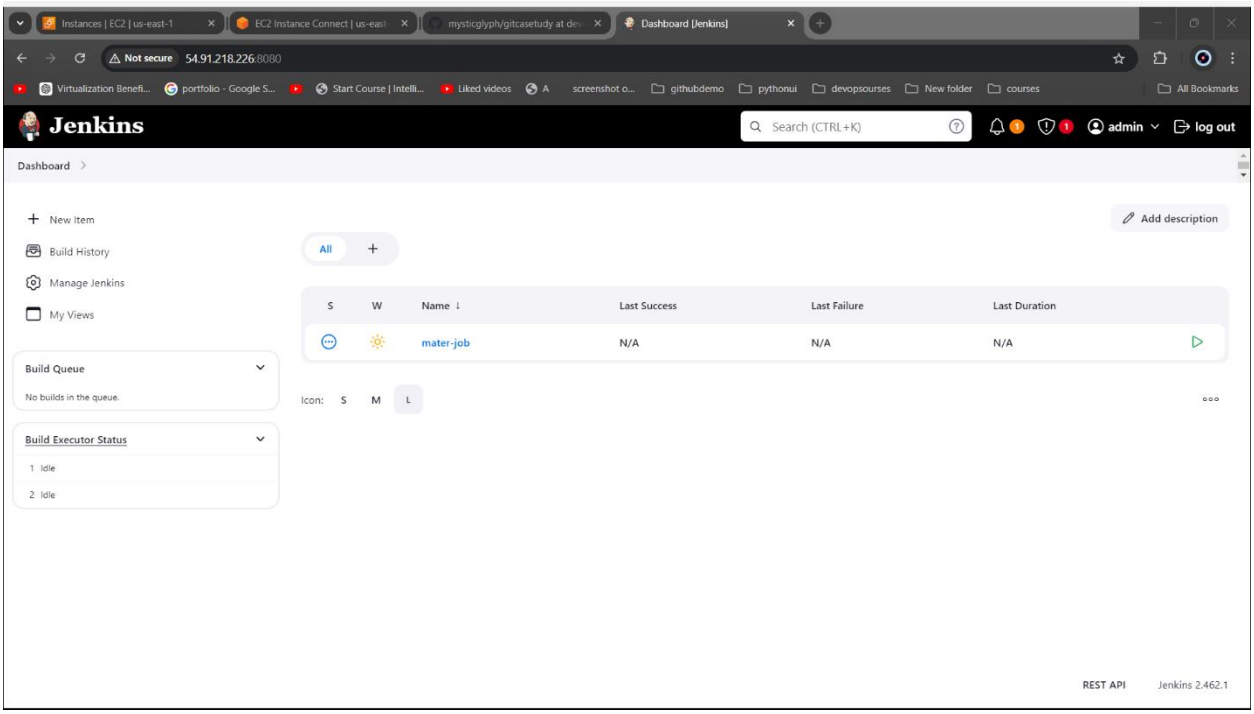
Creating directory:

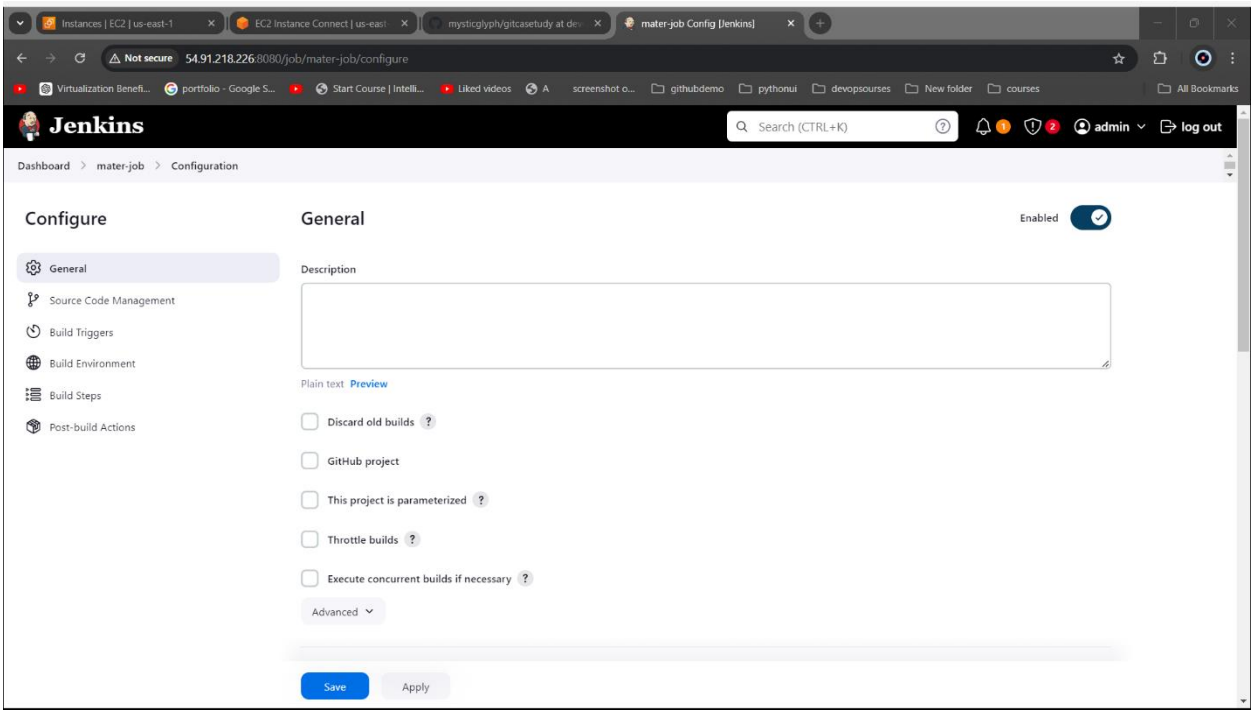


Directory is created.



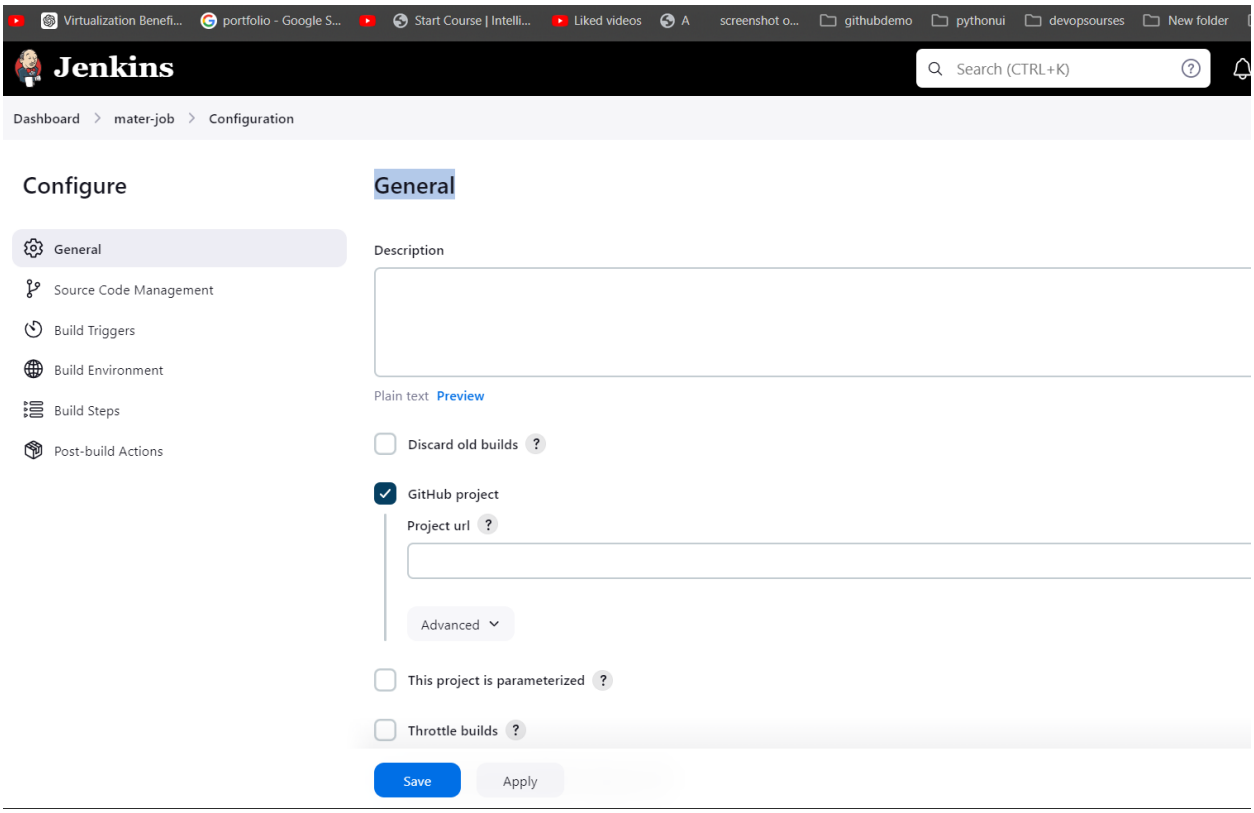
Now we need to configure our master job. The main objective of the master node is to automatically trigger a code build whenever a commit is made to the master or develop branch. If a commit is made to the master branch, the build should publish the website on port 82.

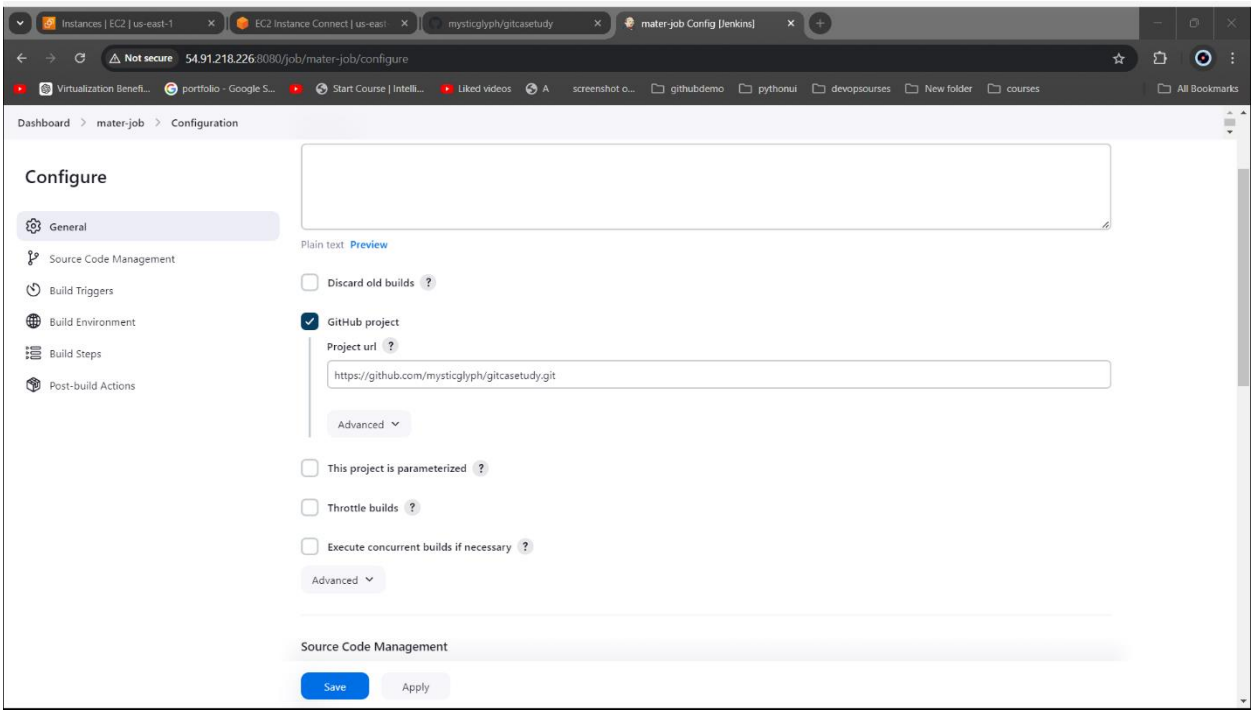




General:

Git project:

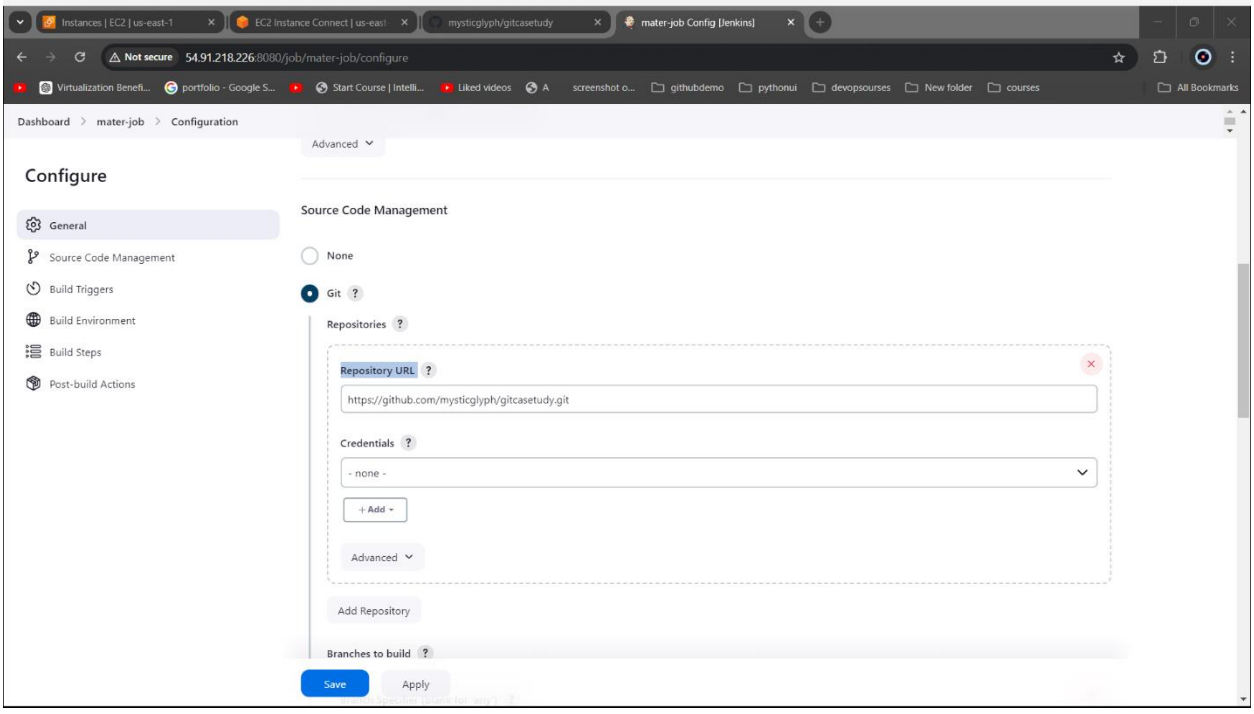




Please provide the GitHub URL.

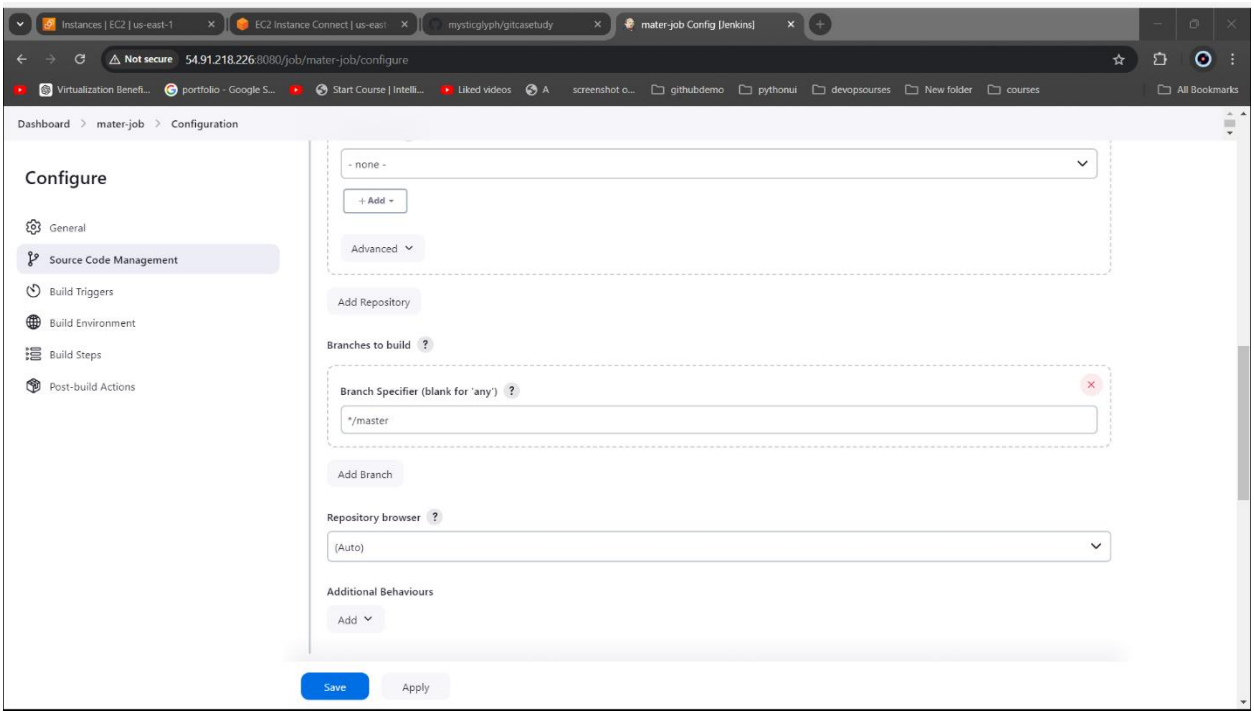
Source Code Management:

Repository URL:



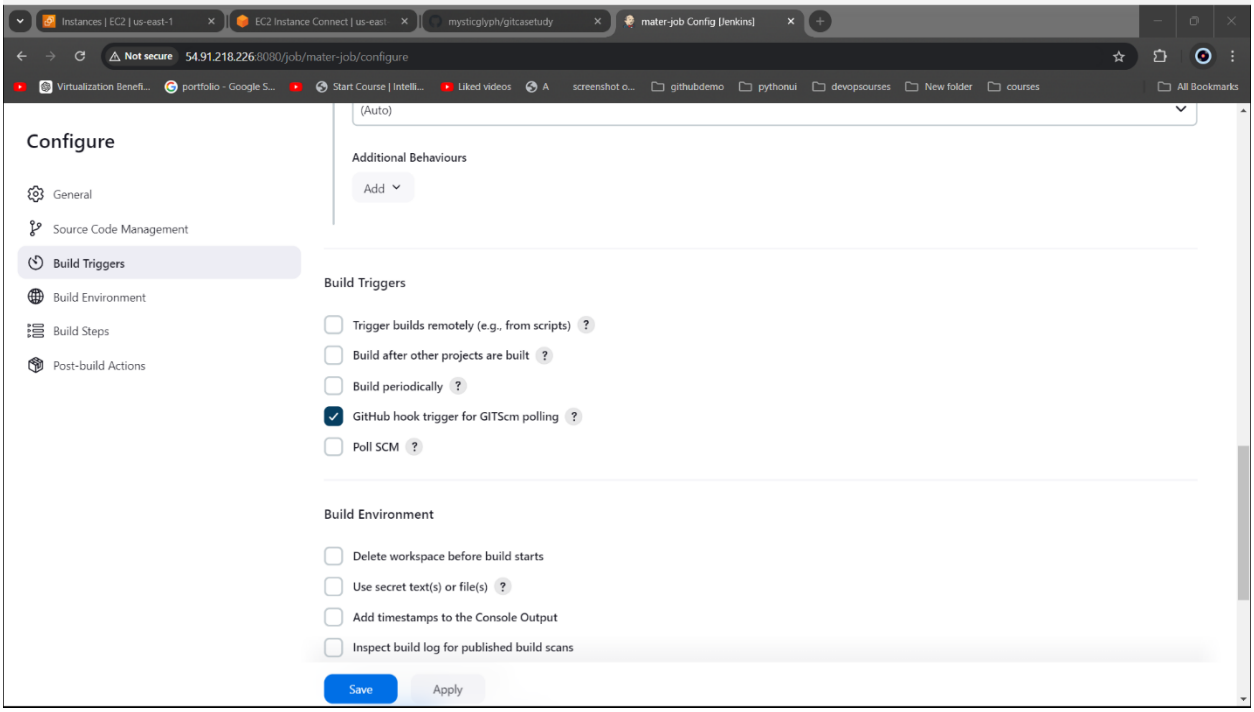
We don't need to provide credentials because our GitHub repository is public.

Branches to build is master branch:

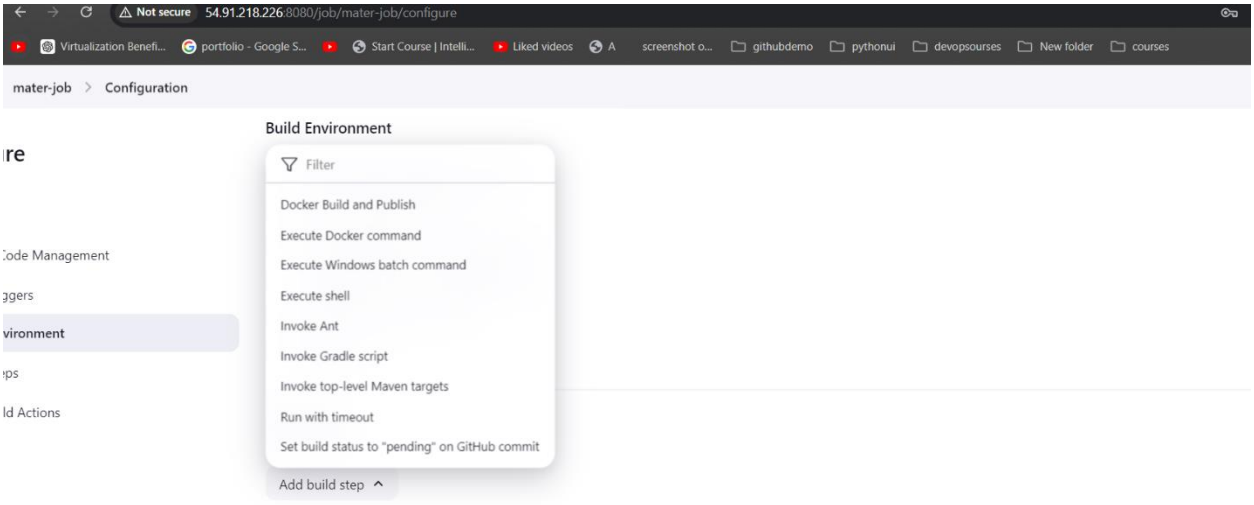


Build Triggers:

GitHub hook trigger for GITScm polling:

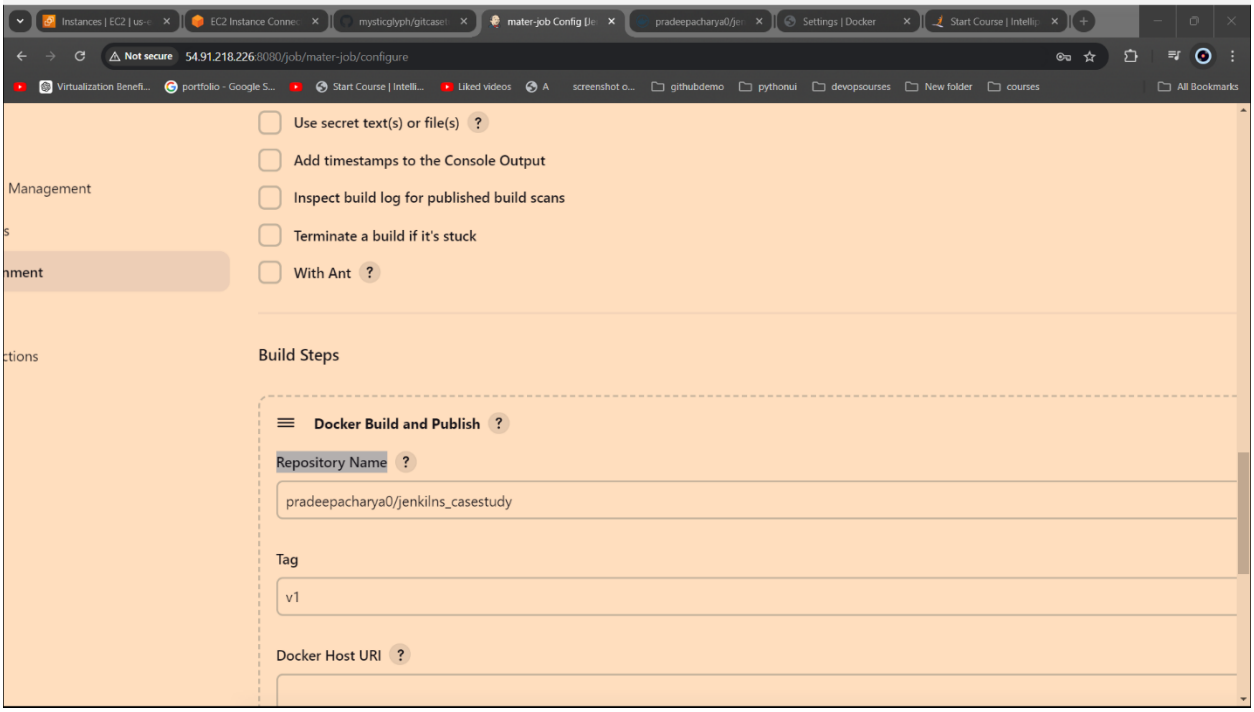


Build Steps:

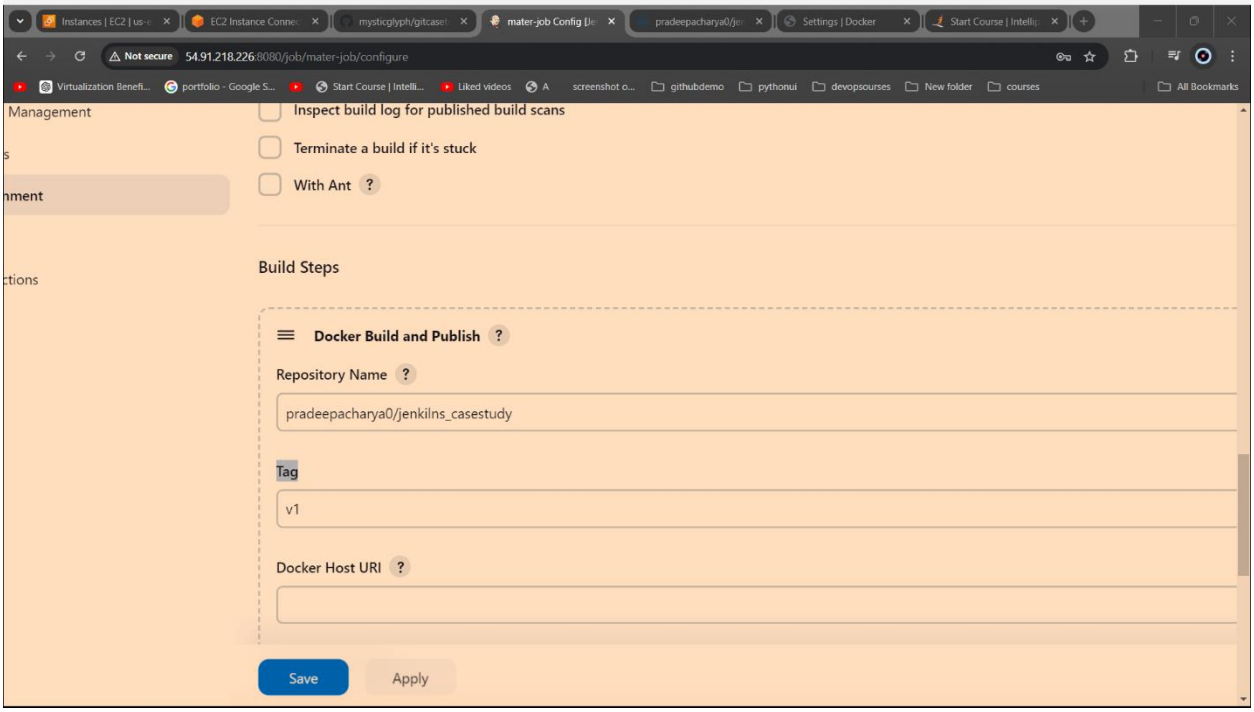


We need to select Docker Build and Publish.

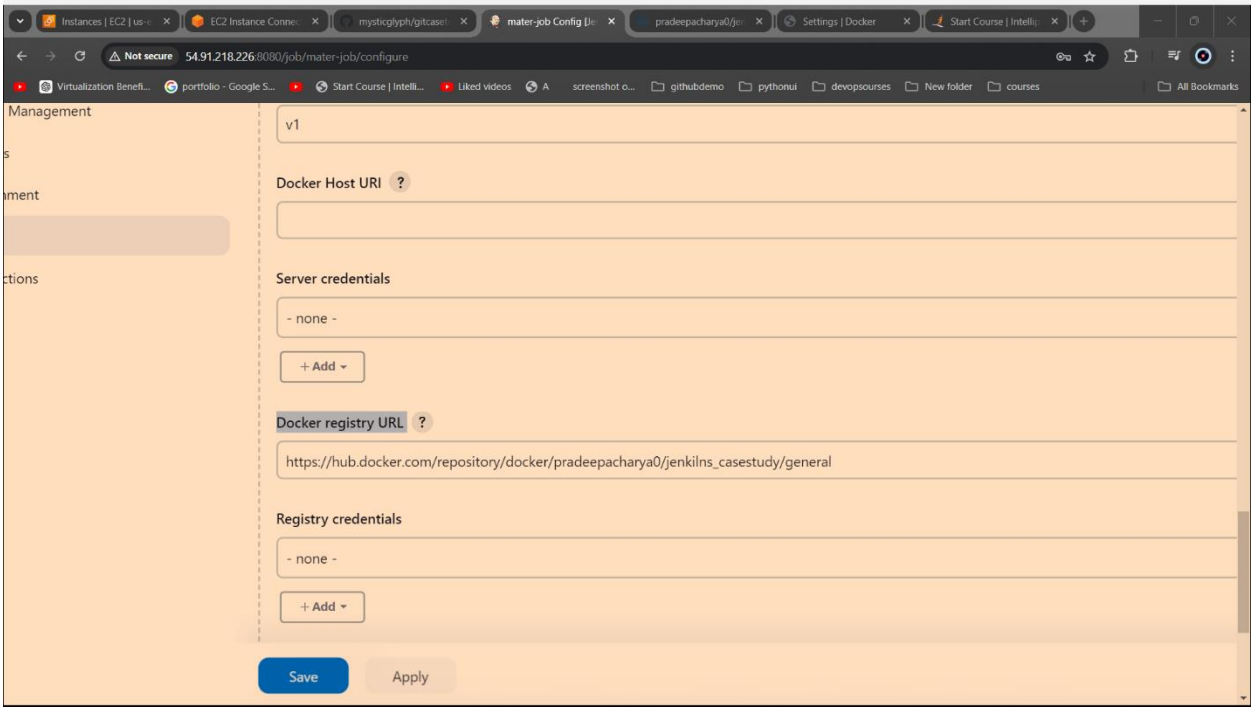
Repository Name:



Tag:

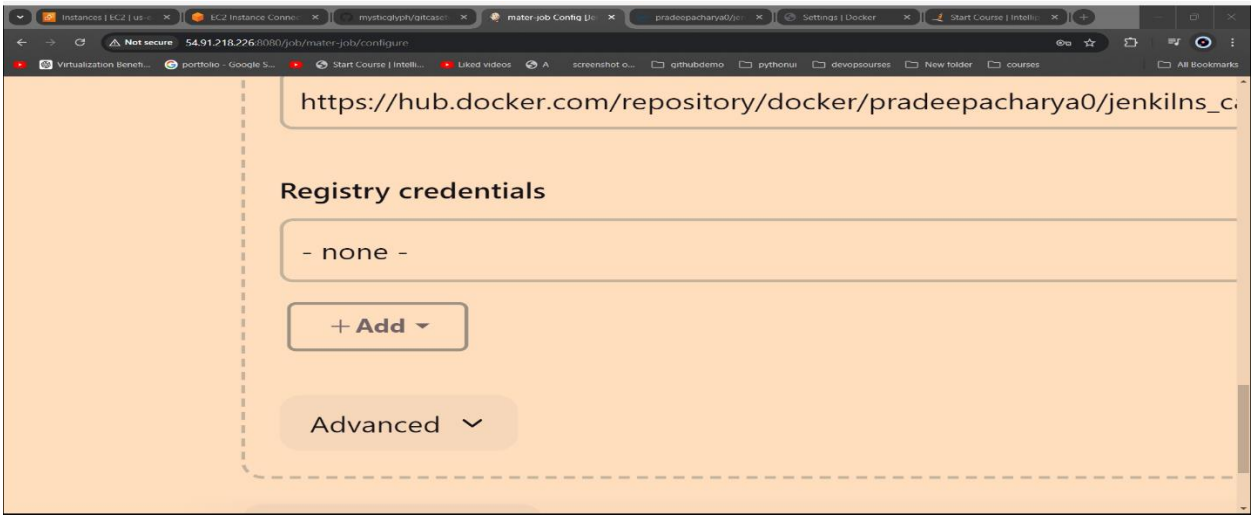


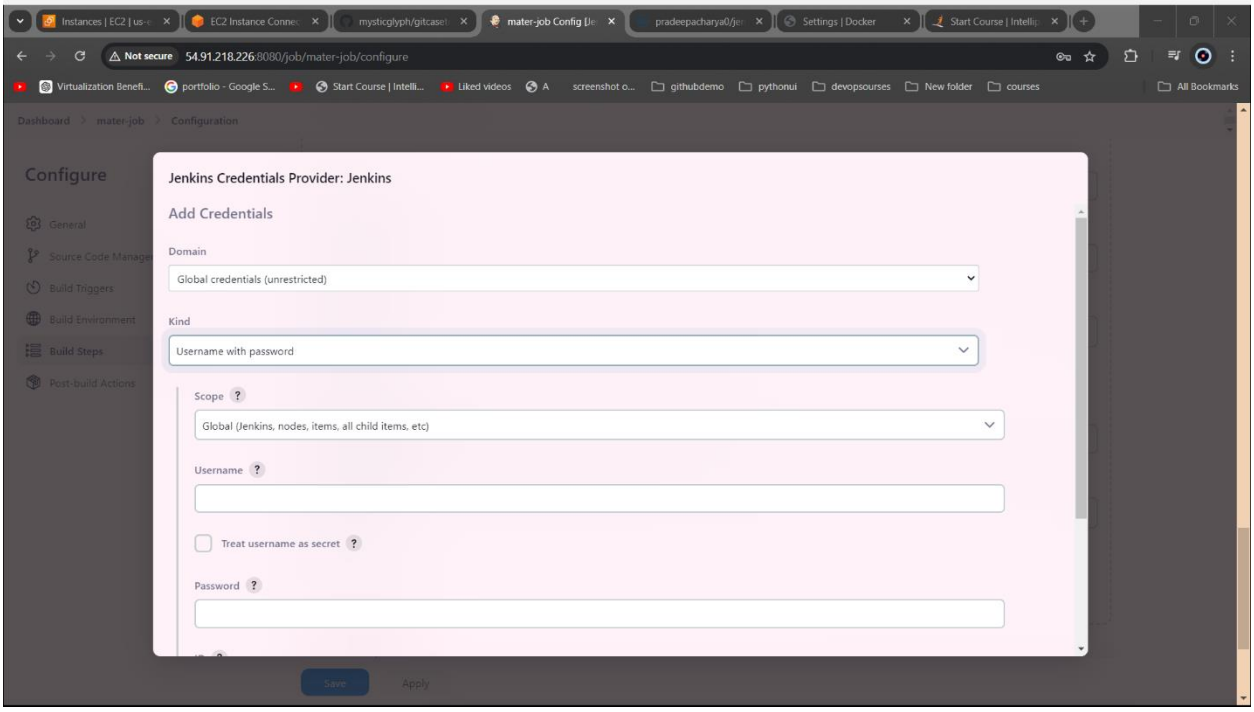
Docker registry URL:



Registry credentials:

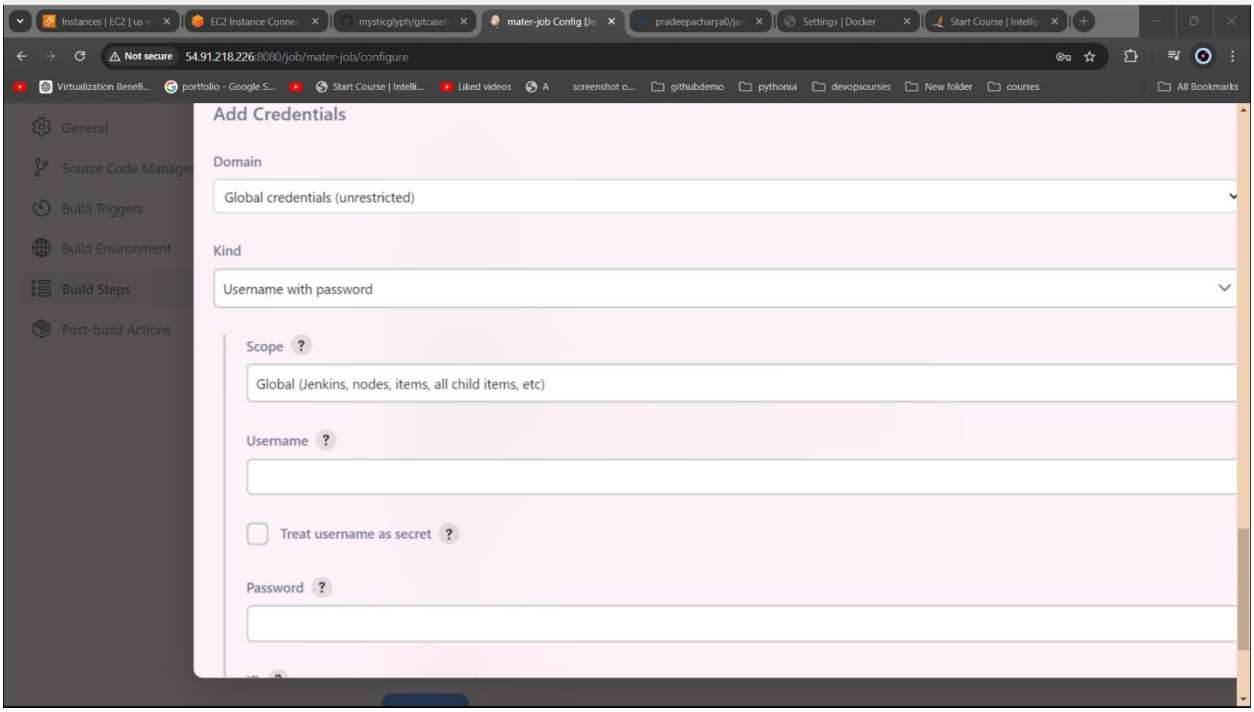
In order to add credentials, proceed with the 'Add' option.



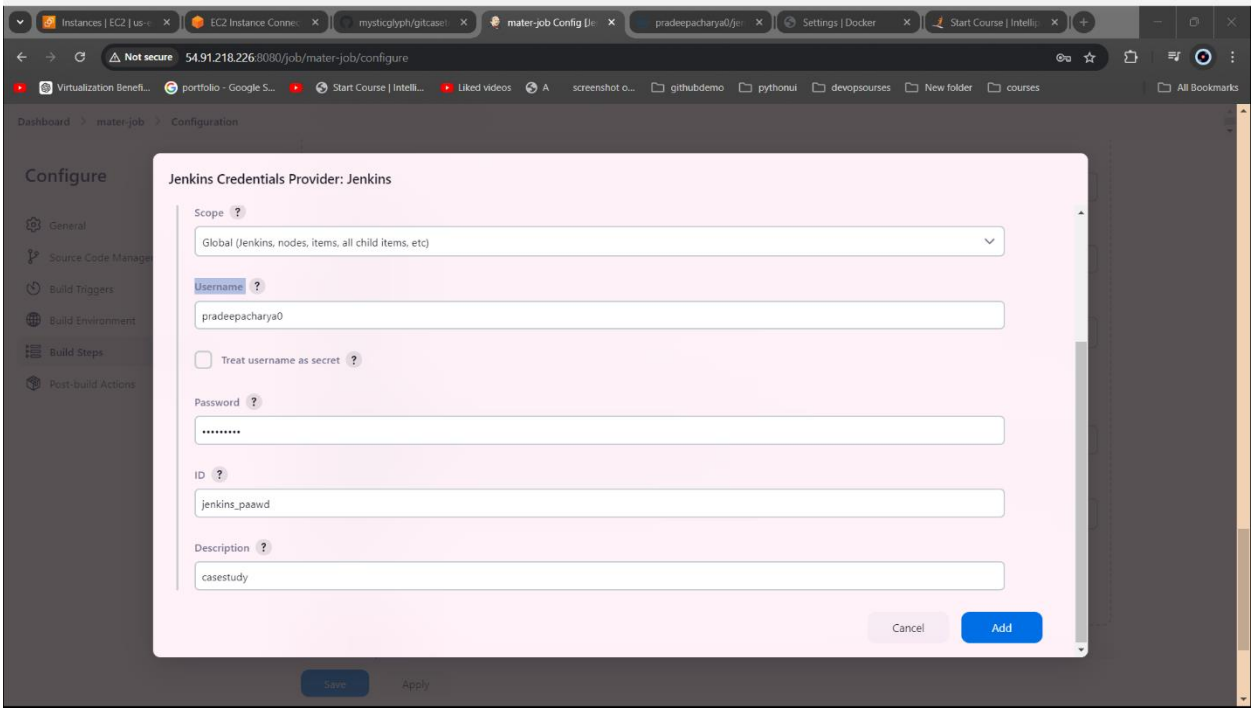


Kind:

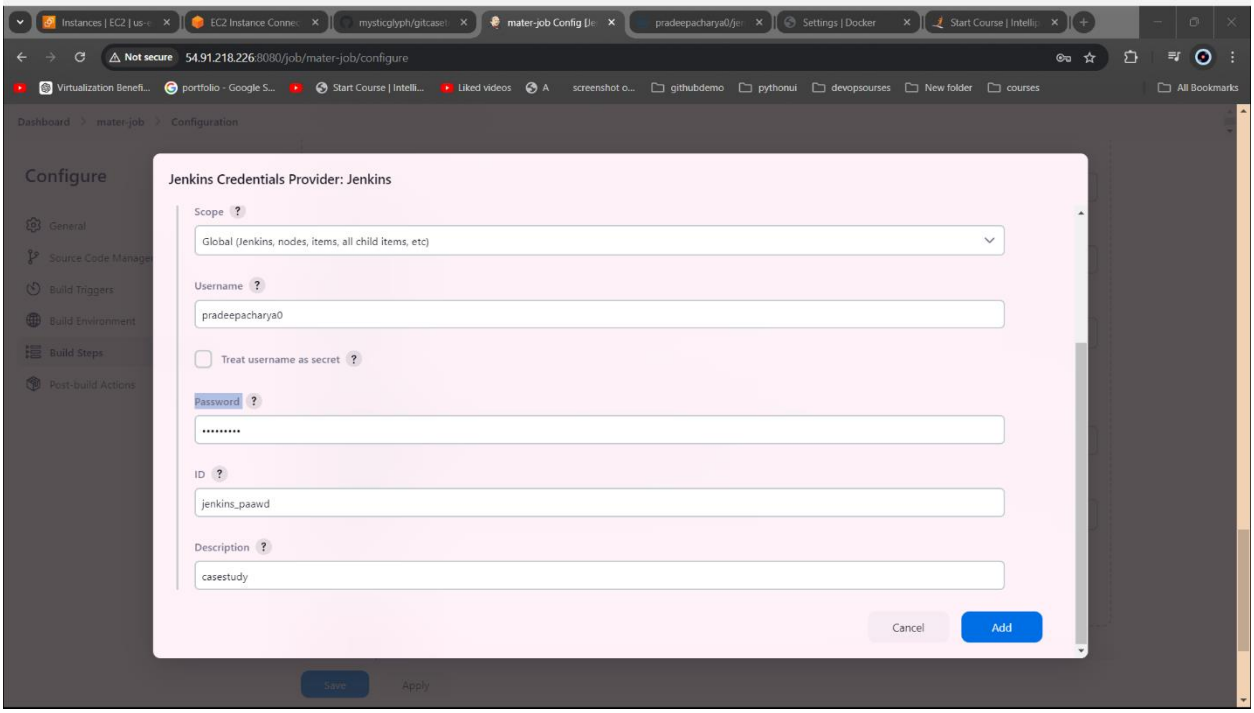
Username and password:



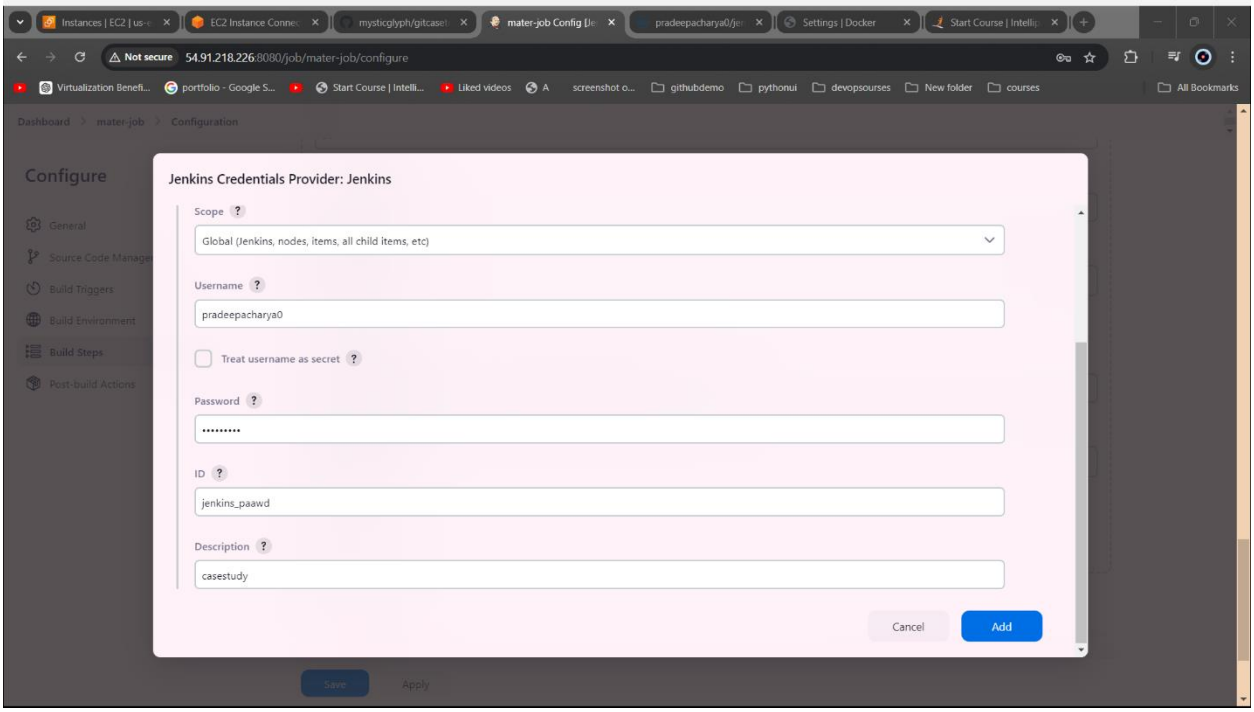
Username:



Password:

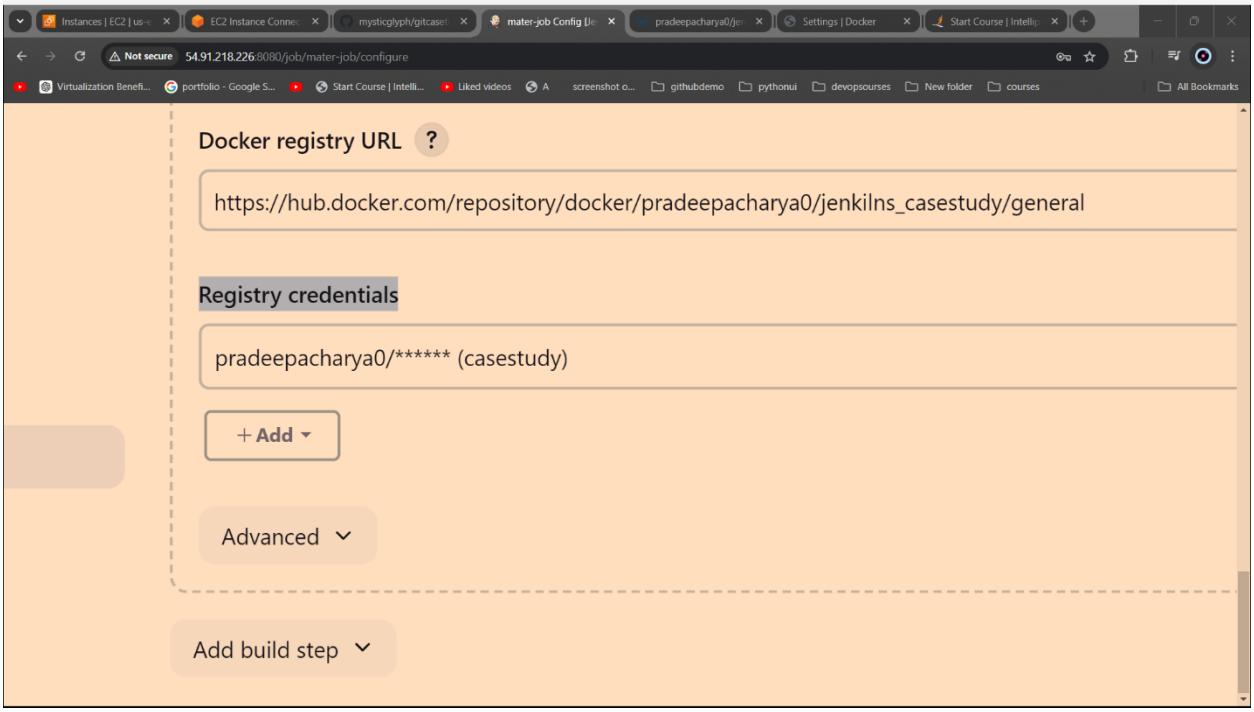


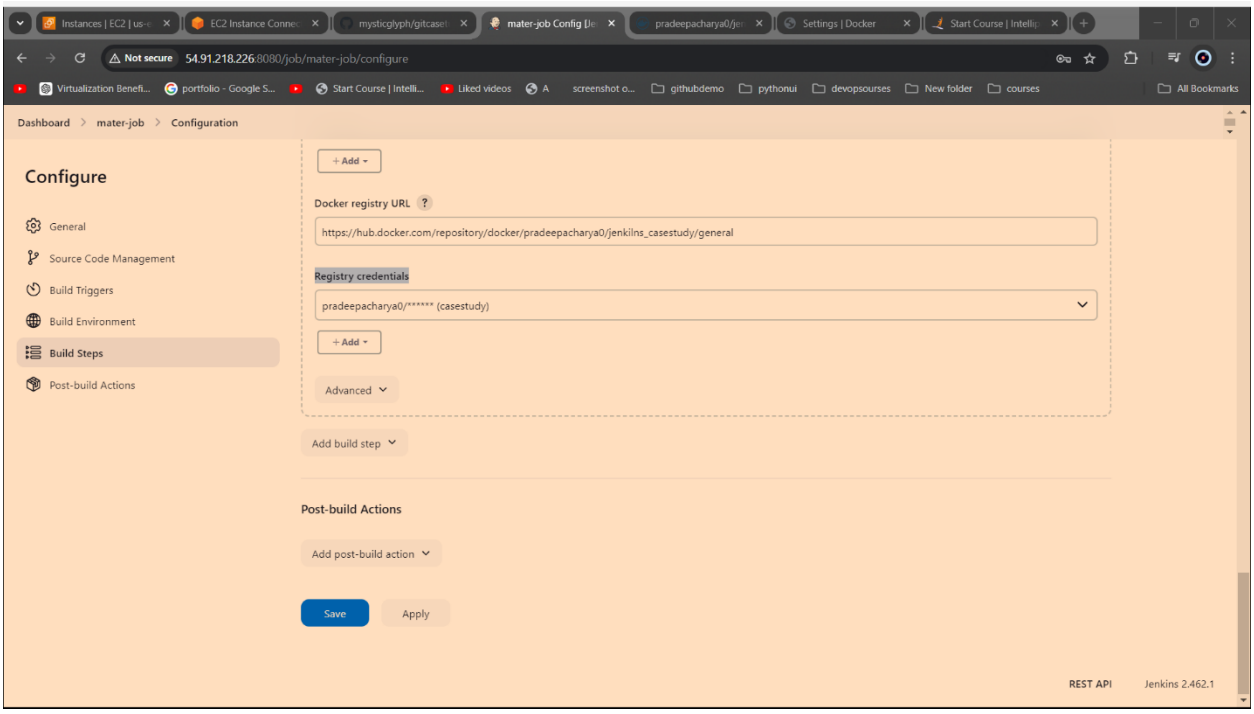
Description and id:



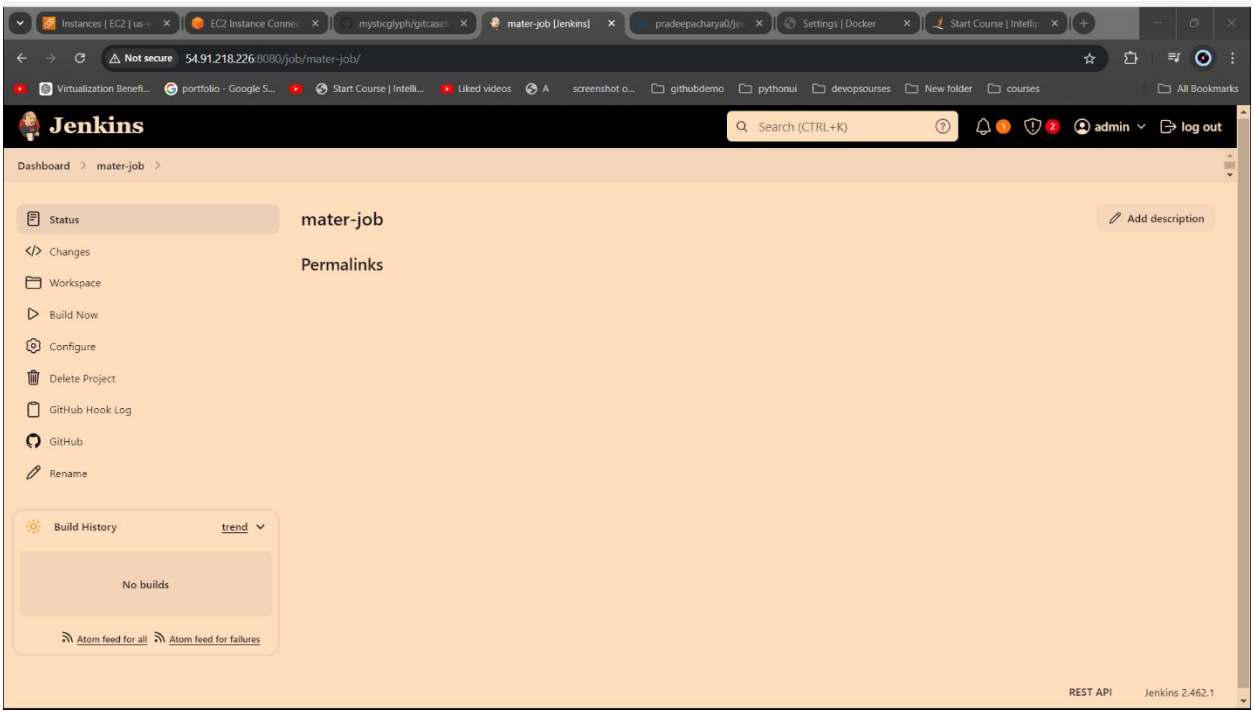
Registry credentials:

Choose the credentials that have been created.



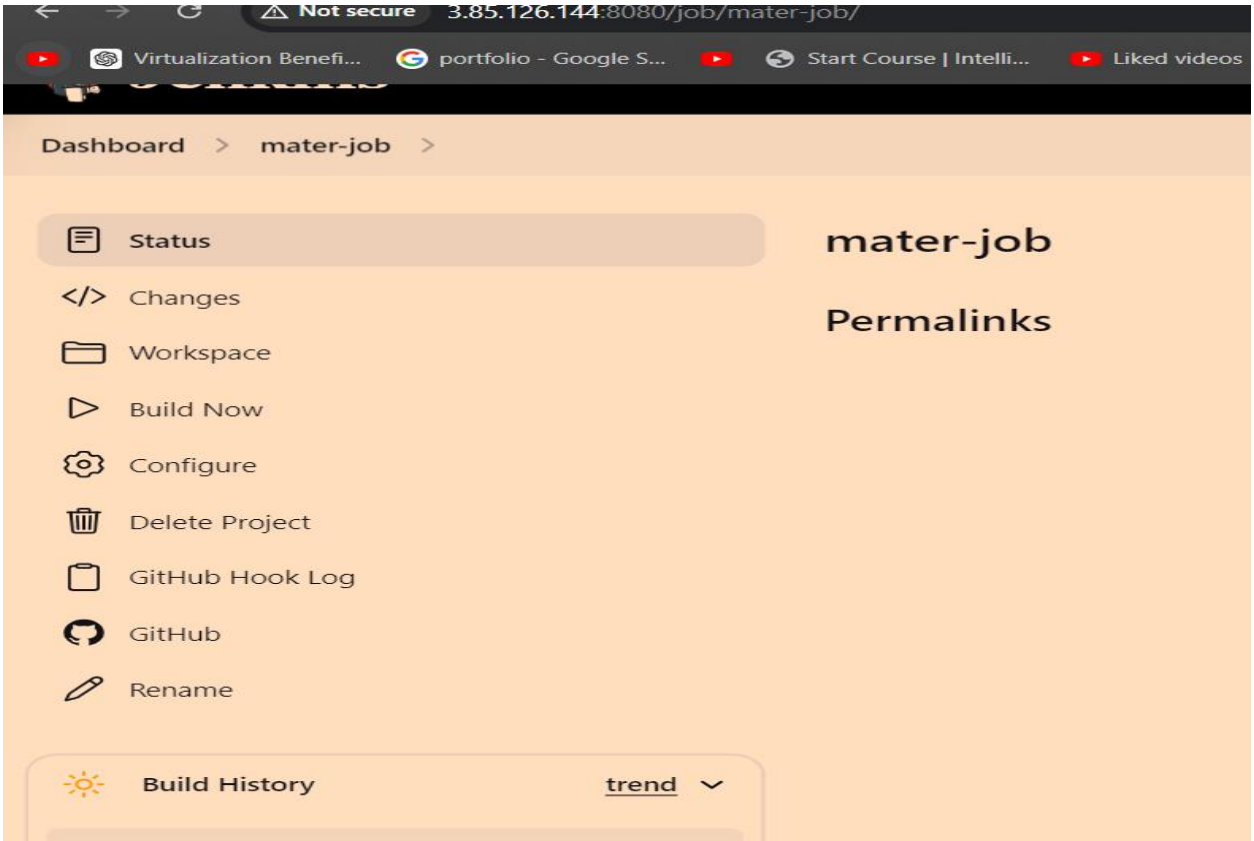


And save it.

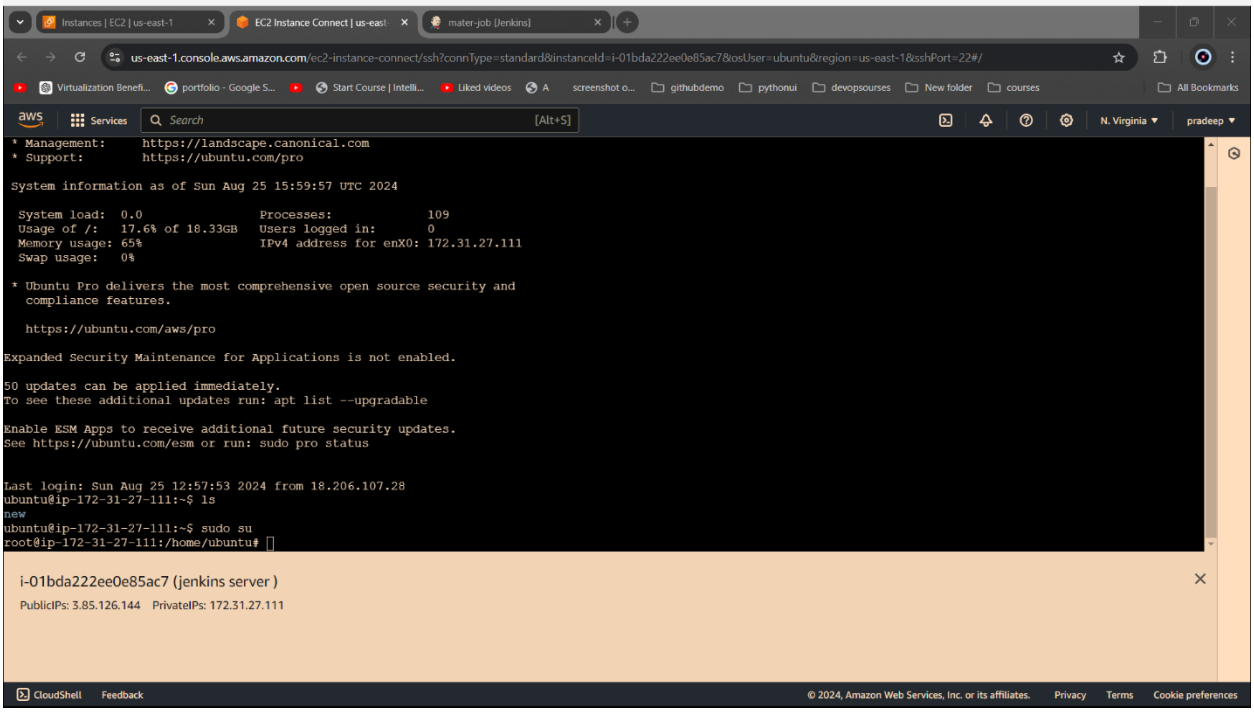


Job have been configured.

In order to execute the job, we need to choose the 'Build Now' option in the dashboard.

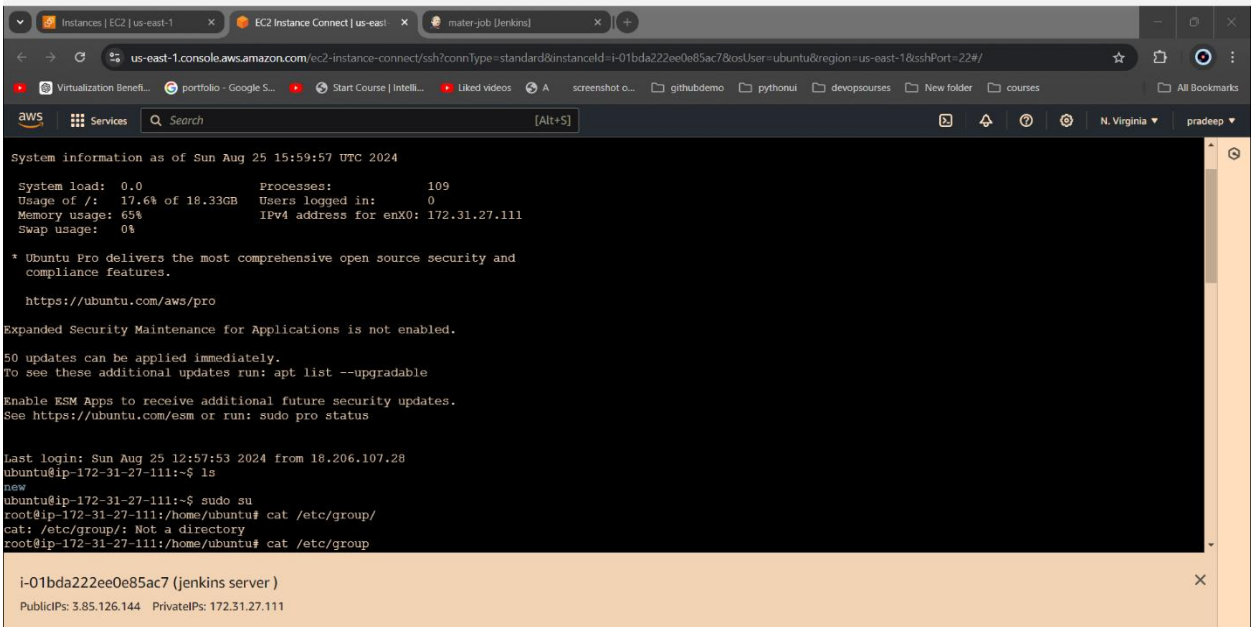


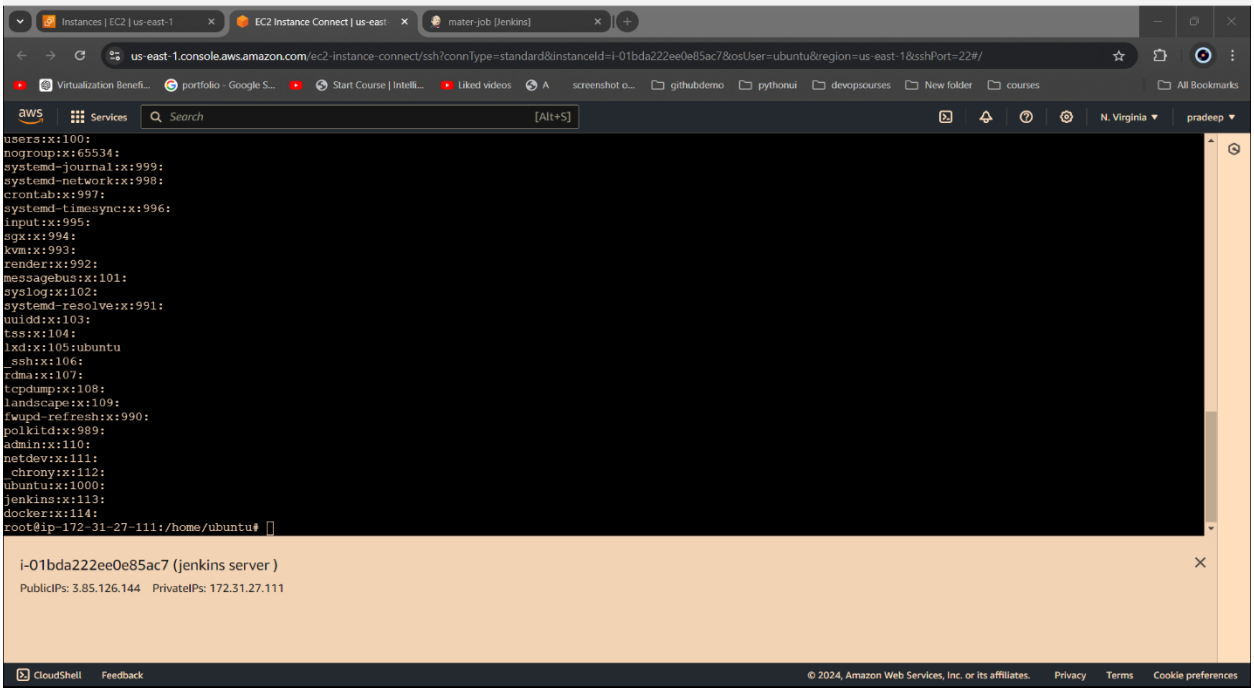
There is one more thing to do: we need to add the Jenkins user to the Docker group because the Jenkins user doesn't have the privileges to execute Docker commands. By adding the Jenkins user to the Docker group, Jenkins will be able to run Docker commands.



In order to check how many group are there we can run the command:

- ➔ **Cat /etc/group**

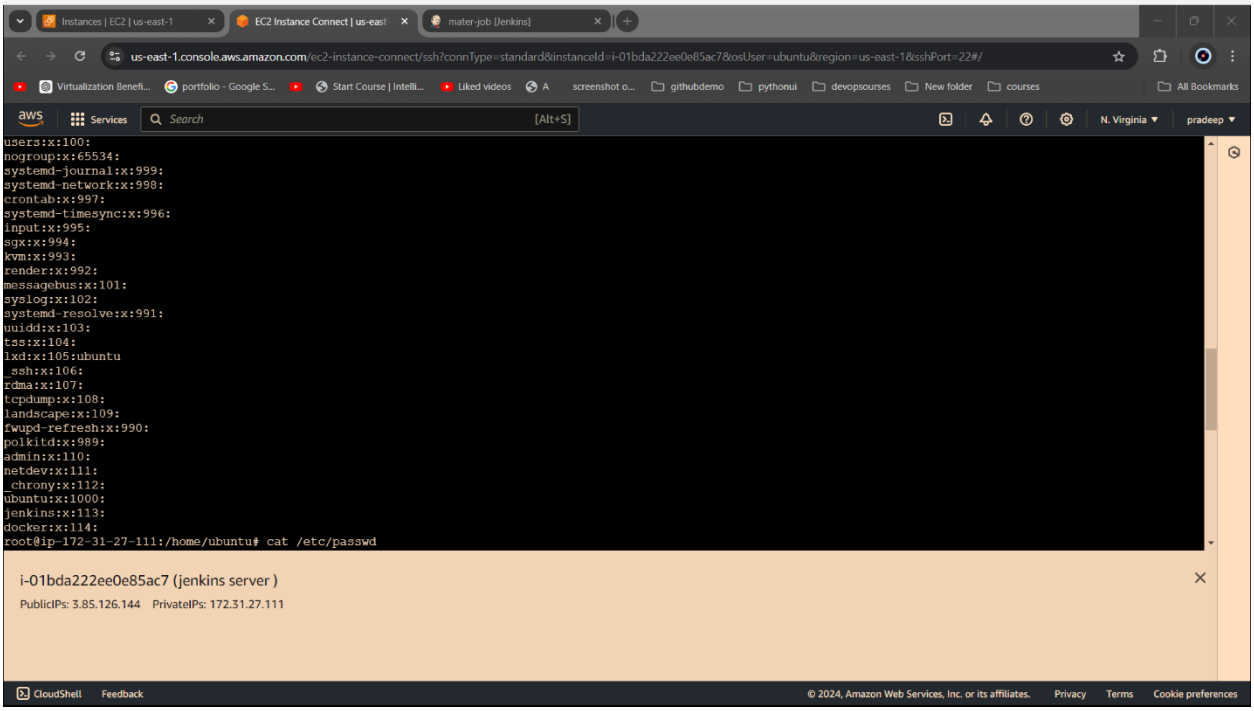


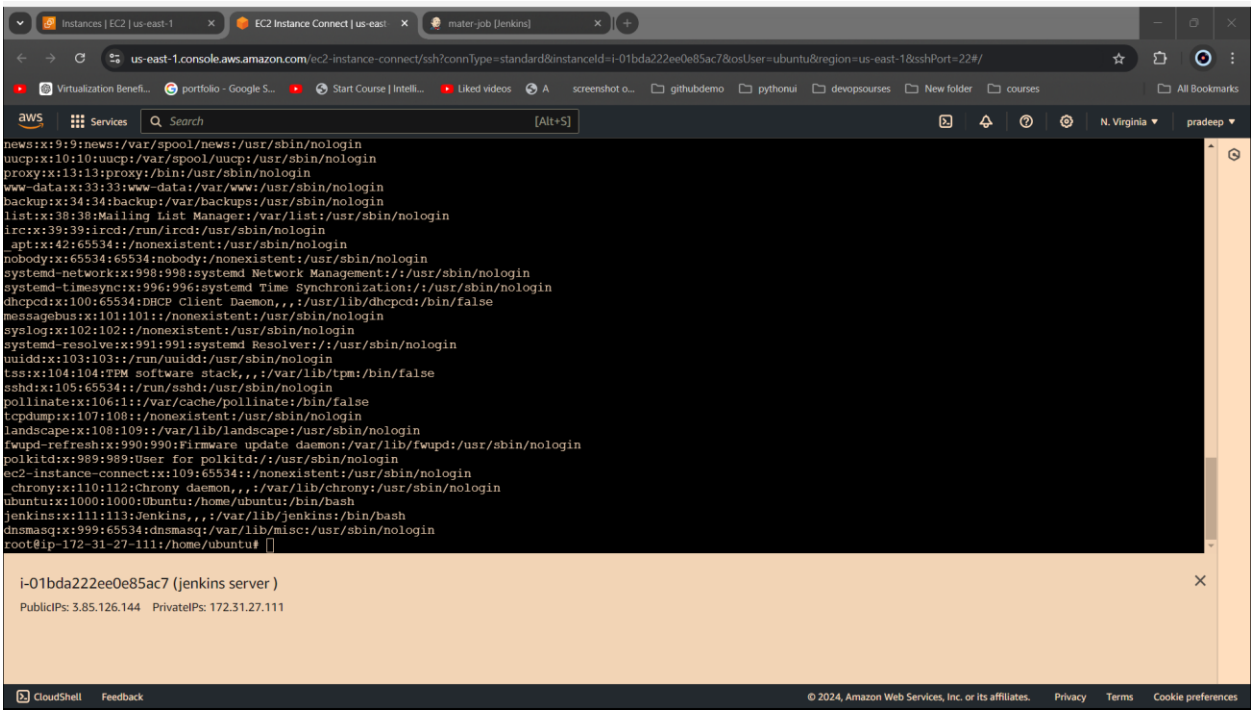


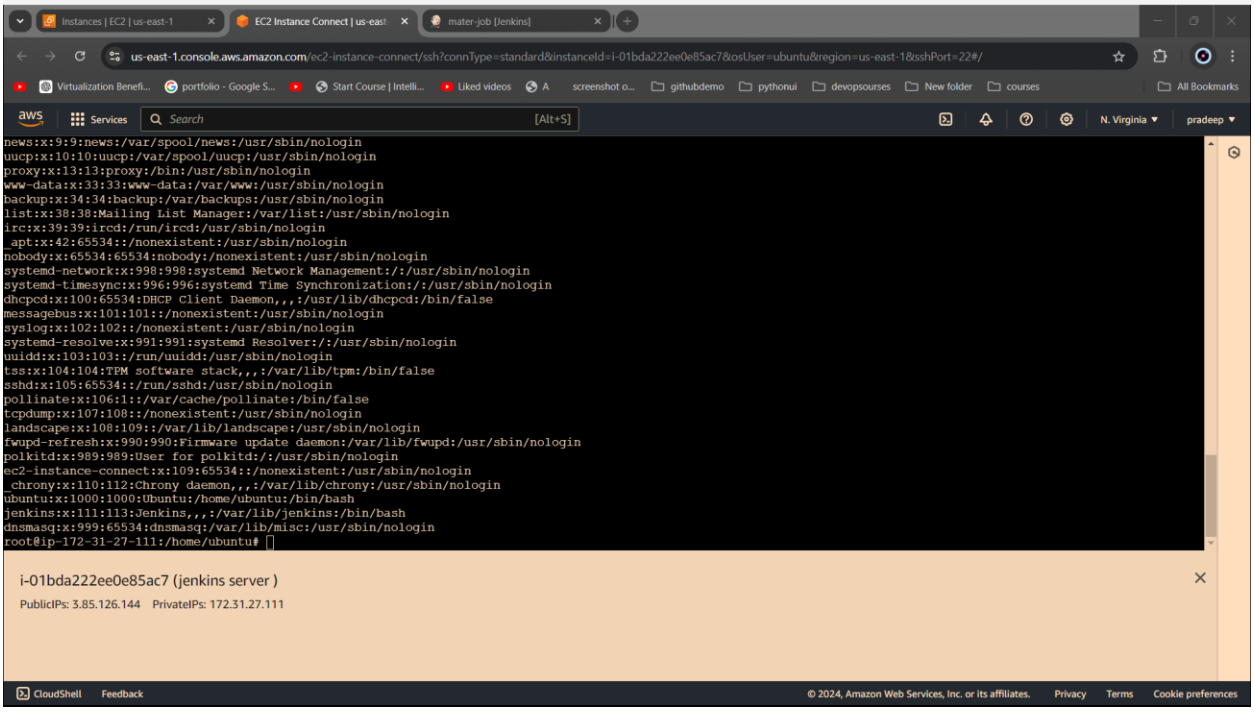
Now, it is evident that there is a group called 'docker'.

Identifying whether the Jenkins user is present or not.

➔ Cat /etc/passwd



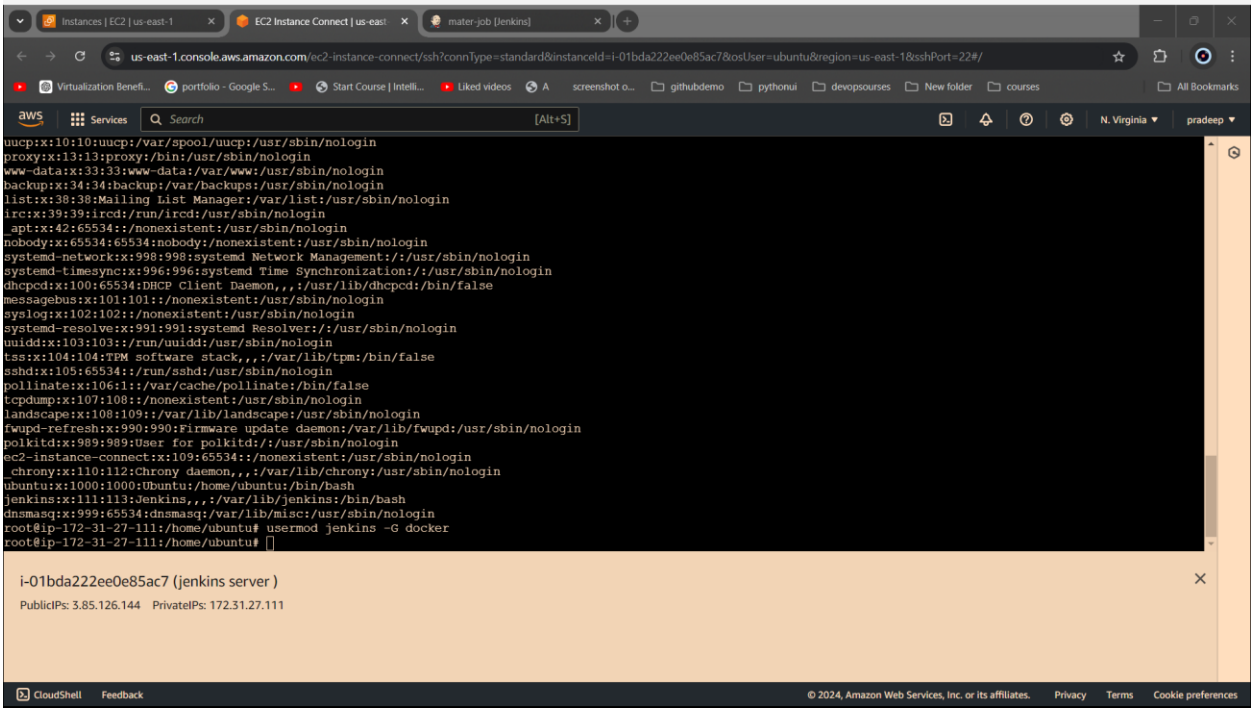




The Jenkins user is present on the server.

Now we need to add the Jenkins user to the Docker group. To do that, we need to run a command.

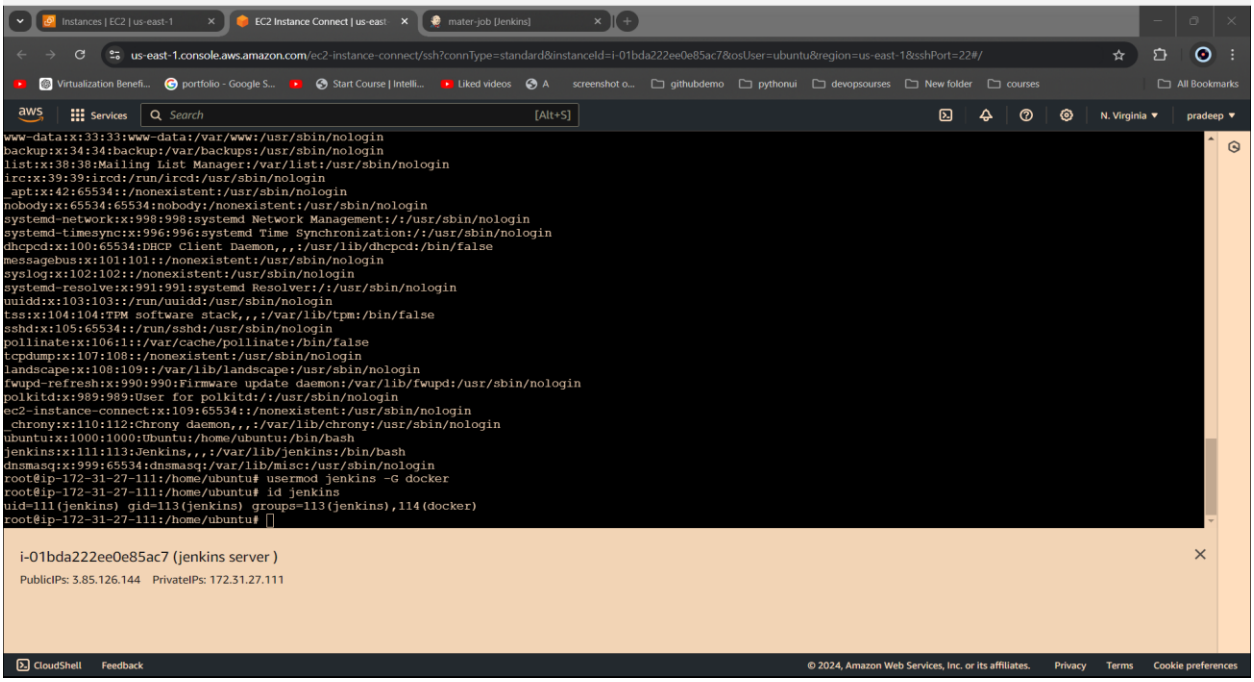
➔ Sudo usermod jenkins -G docker



Jenkins is added to docker group.

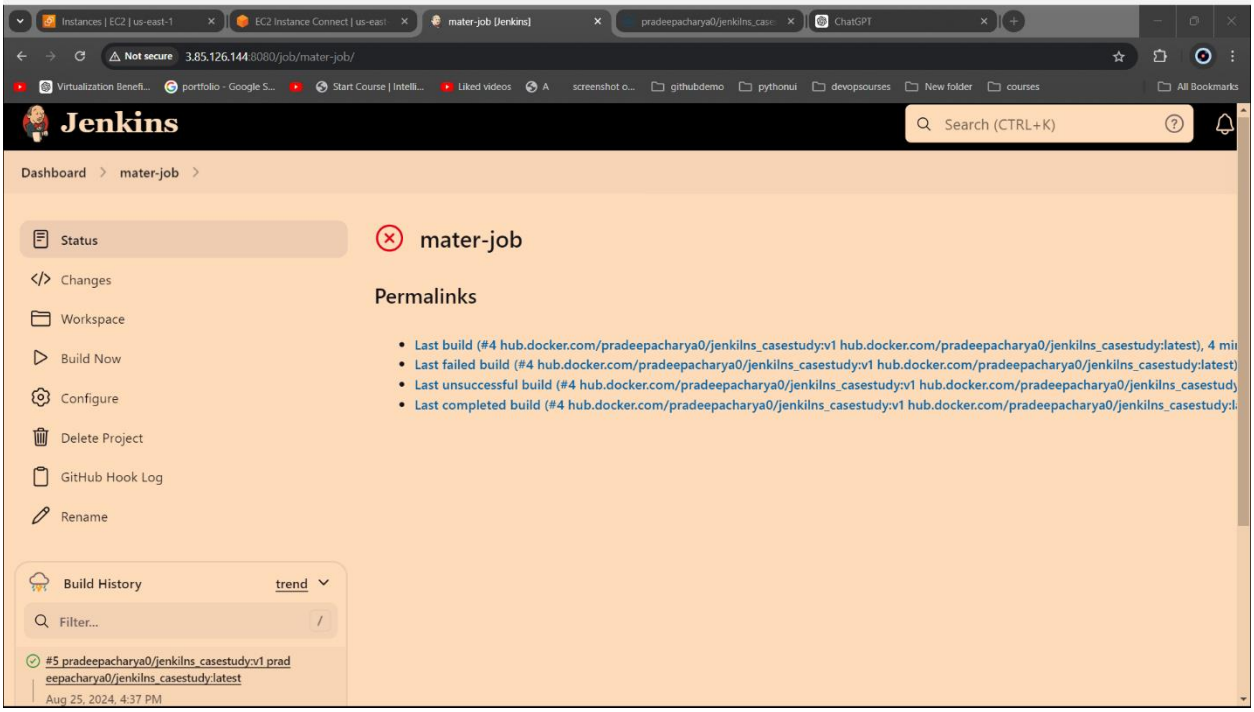
In order to check that we need to run a command.

➔ Id jenkins

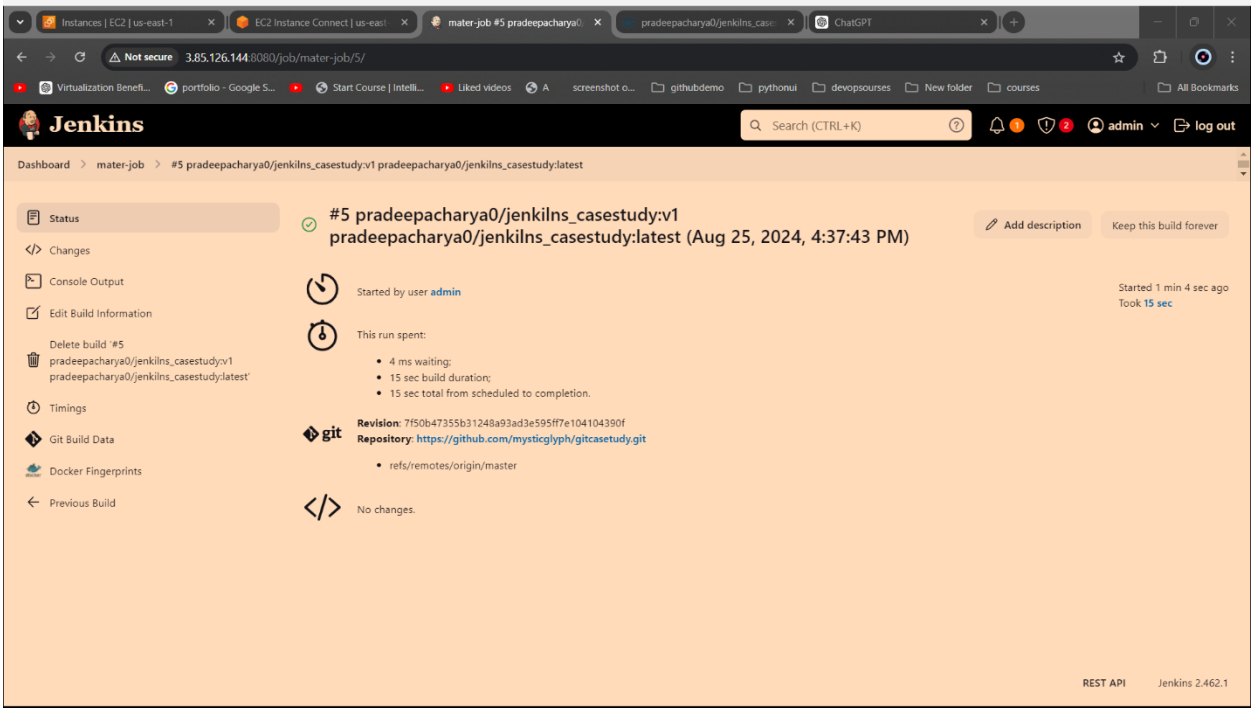


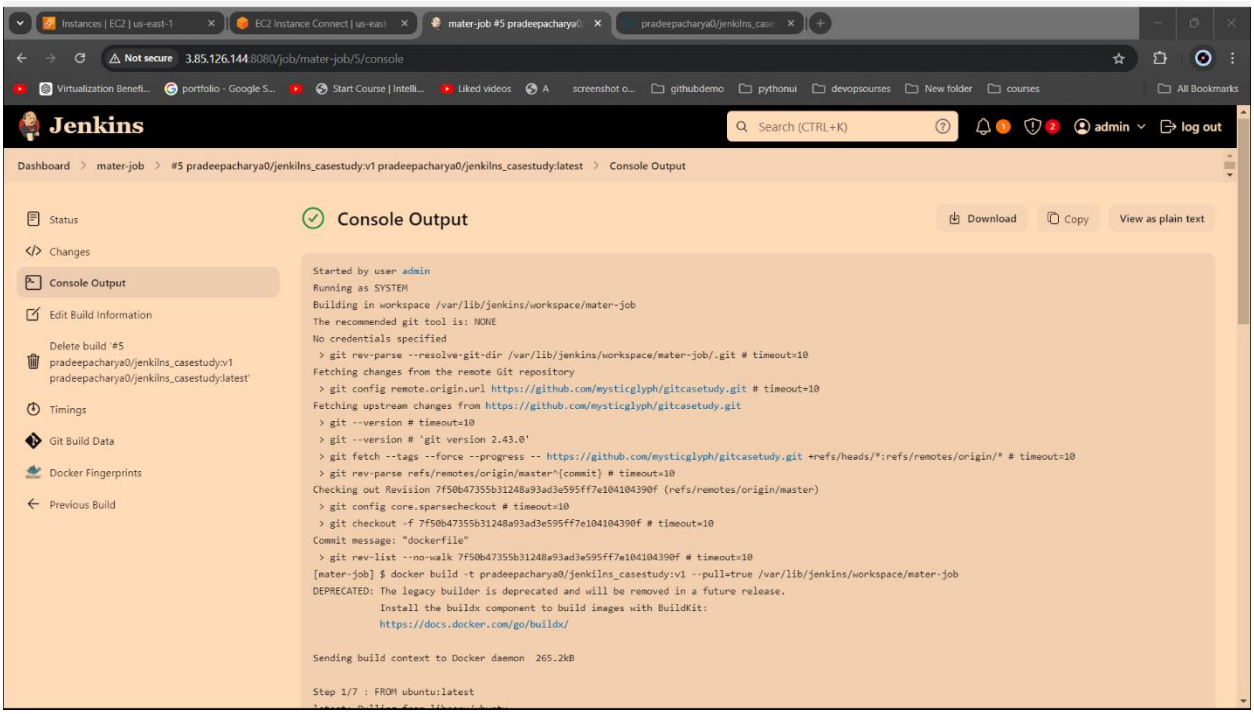
Now Jenkins can run Docker commands.

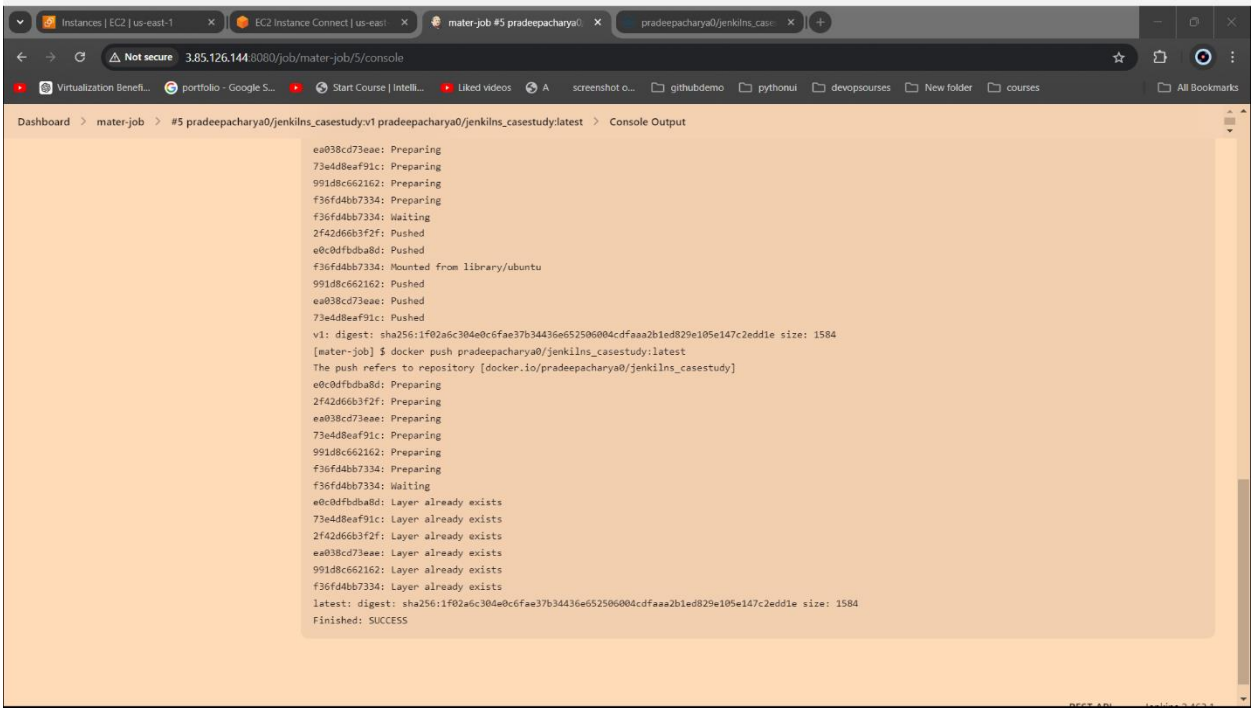
Now build the job and test if it executes according to the requirements.



Job is build successfully.




Instances | EC2 | us-east-1
EC2 Instance Connect | us-east
mater-job #5 pradeepacharya0
pradeepacharya0/jenkilns_case
Not secure 3.85.126.144:8080/job/mater-job/5/console
Jenkins
Search (CTRL+K)
admin
log out
Dashboard > mater-job > #5 pradeepacharya0/jenkilns_casestudy:v1 pradeepacharya0/jenkilns_casestudy:latest > Console Output
Status
Changes
Console Output
Edit Build Information
Delete build '#5 pradeepacharya0/jenkilns_casestudy:v1 pradeepacharya0/jenkilns_casestudy:latest'
Timings
Git Build Data
Docker Fingerprints
Previous Build
Console Output
Download
Copy
View as plain text
Started by user admin
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/mater-job
The recommended git tool is: NONE
No credentials specified
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/mater-job/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/mysticglyph/gitcasetudy.git # timeout=10
Fetching upstream changes from https://github.com/mysticglyph/gitcasetudy.git
> git --version # timeout=10
> git --version # 'git version 2.43.0'
> git fetch --tags --force --progress -- https://github.com/mysticglyph/gitcasetudy.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision 7f50b47355b31248a93ad3e595ff7e104104390f (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 7f50b47355b31248a93ad3e595ff7e104104390f # timeout=10
Commit message: "dockerfile"
> git rev-list --no-walk 7f50b47355b31248a93ad3e595ff7e104104390f # timeout=10
[mater-job] $ docker build -t pradeepacharya0/jenkilns_casestudy:v1 --pull=true /var/lib/jenkins/workspace/mater-job
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
Install the buildx component to build images with BuildKit:
https://docs.docker.com/go/buildx/
Sending build context to Docker daemon 265.2kB
Step 1/7 : FROM ubuntu:latest


Instances | EC2 | us-east-1
EC2 Instance Connect | us-east
mater-job #5 pradeepacharya0
pradeepacharya0/jenkilns_case
Not secure 3.85.126.144:8080/job/mater-job/5/console
Dashboard > mater-job > #5 pradeepacharya0/jenkilns_casestudy:v1 pradeepacharya0/jenkilns_casestudy:latest > Console Output
ea038cd73eae: Preparing
73e4d8eaf91c: Preparing
991d8c662162: Preparing
f36fd4bb7334: Preparing
f36fd4bb7334: Waiting
2f42d66b3f2f: Pushed
e0c0dfbdba8d: Pushed
f36fd4bb7334: Mounted from library/ubuntu
991d8c662162: Pushed
ea038cd73eae: Pushed
73e4d8eaf91c: Pushed
v1: digest: sha256:1f02a6c304e0c6fae37b34436e652506004cdfaaa2b1ed829e105e147c2edd1e size: 1584
[mater-job] $ docker push pradeepacharya0/jenkilns_casestudy:latest
The push refers to repository [docker.io/pradeepacharya0/jenkilns_casestudy]
e0c0dfbdba8d: Preparing
2f42d66b3f2f: Preparing
ea038cd73eae: Preparing
73e4d8eaf91c: Preparing
991d8c662162: Preparing
f36fd4bb7334: Preparing
f36fd4bb7334: Waiting
e0c0dfbdba8d: Layer already exists
73e4d8eaf91c: Layer already exists
2f42d66b3f2f: Layer already exists
ea038cd73eae: Layer already exists
991d8c662162: Layer already exists
f36fd4bb7334: Layer already exists
latest: digest: sha256:1f02a6c304e0c6fae37b34436e652506004cdfaaa2b1ed829e105e147c2edd1e size: 1584
Finished: SUCCESS

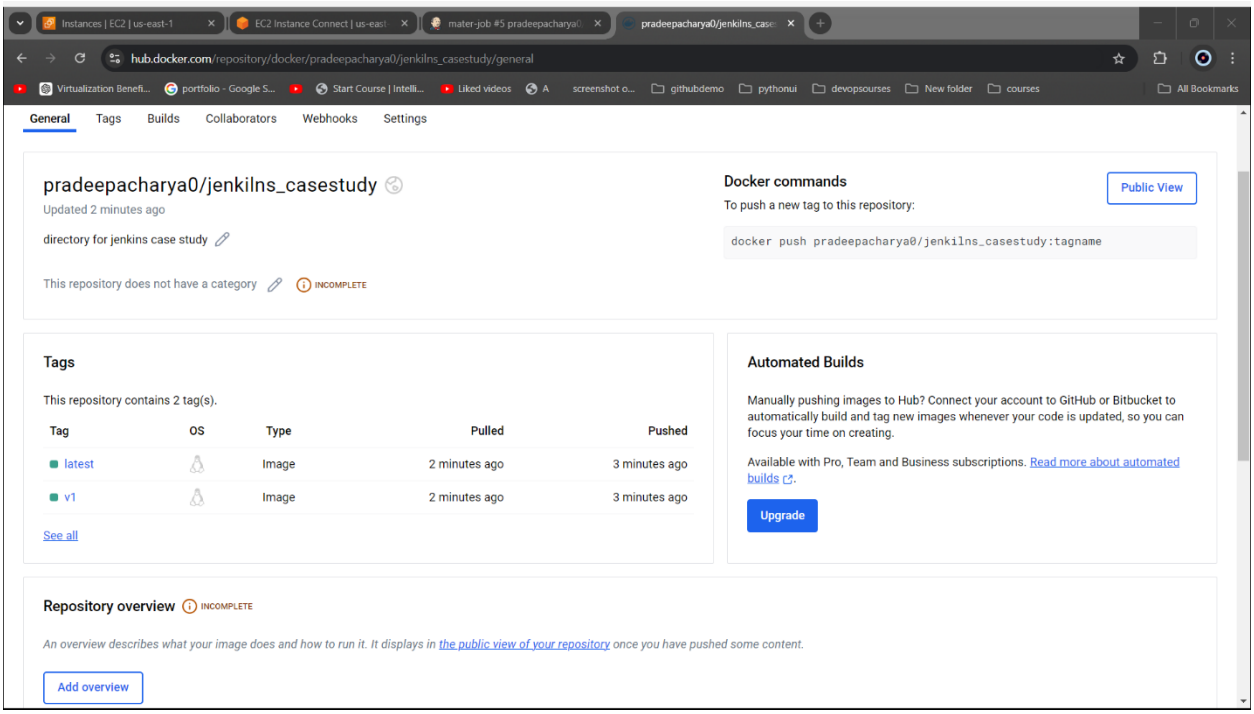
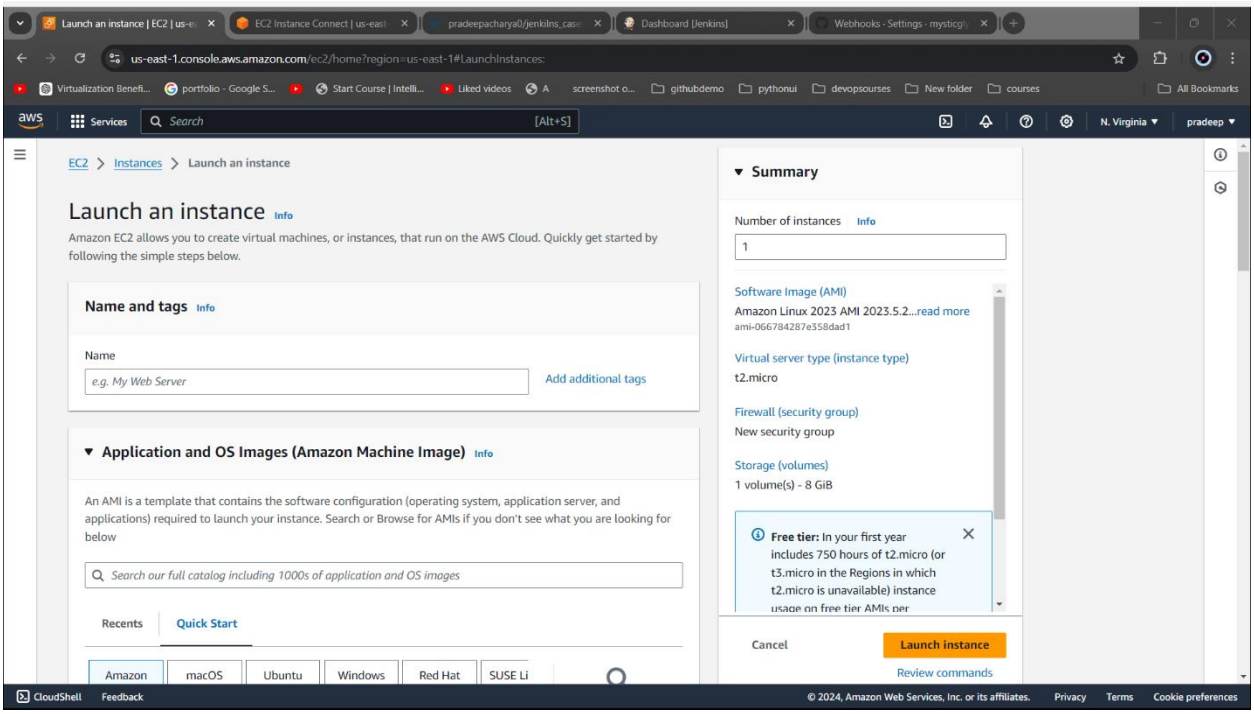


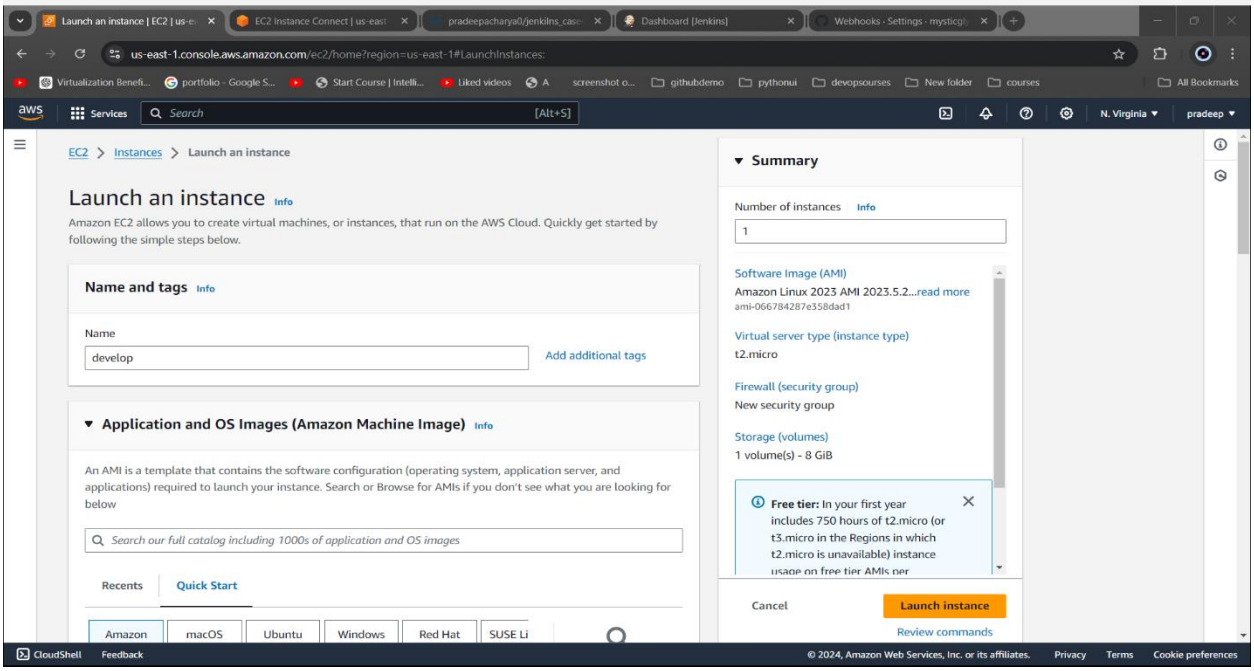
Image is published in dockerhub.

In order to build the develop job, create a new Jenkins node, connect the node to the Jenkins server, and create the develop job. This job should be executed on the new node.

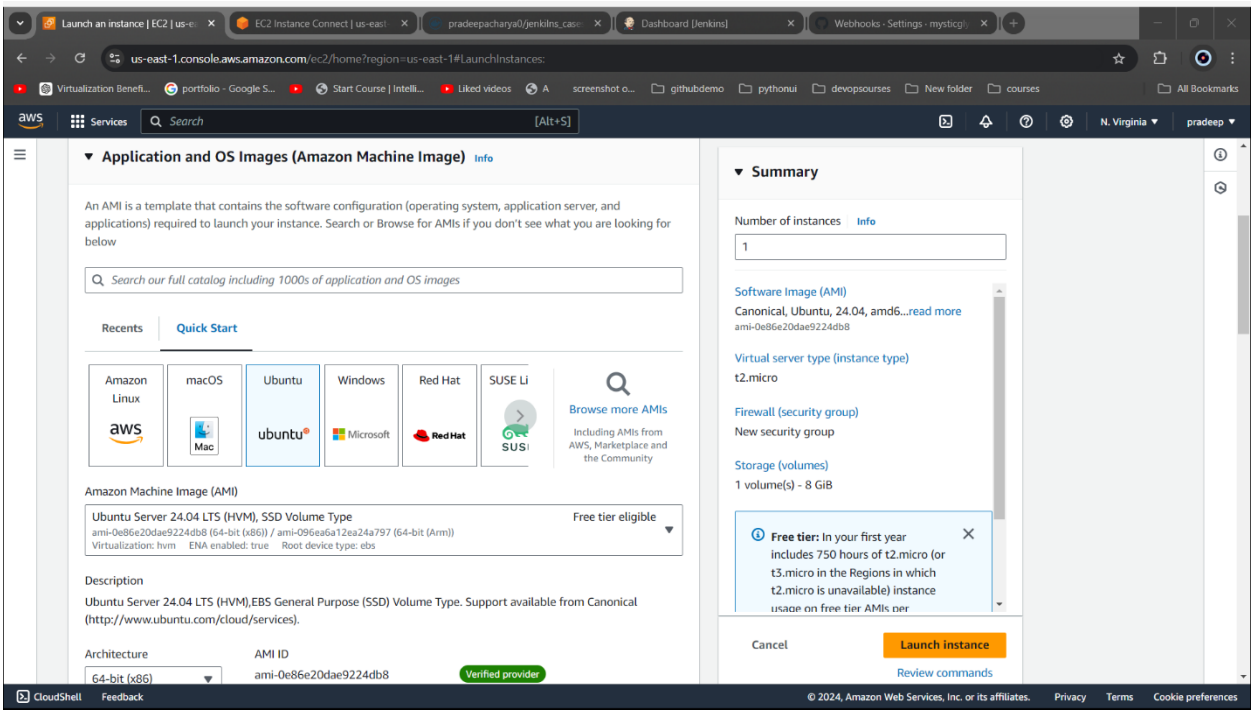
Creating a node called “develop”:



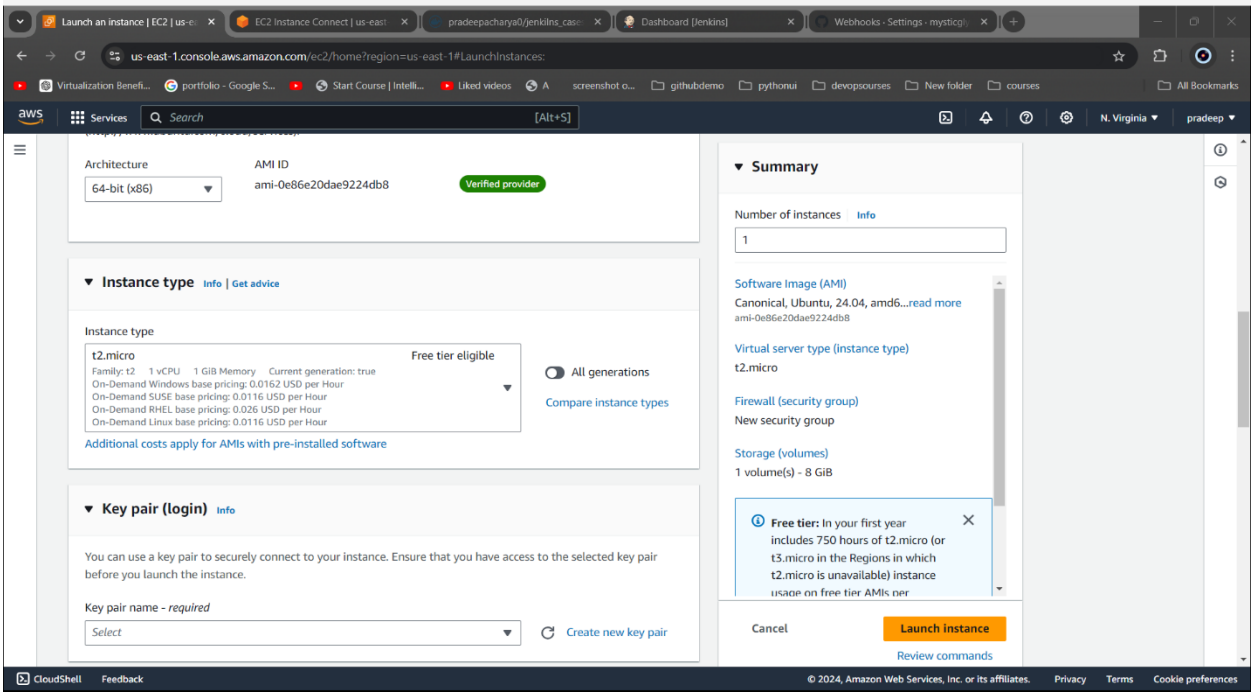
Instance name:



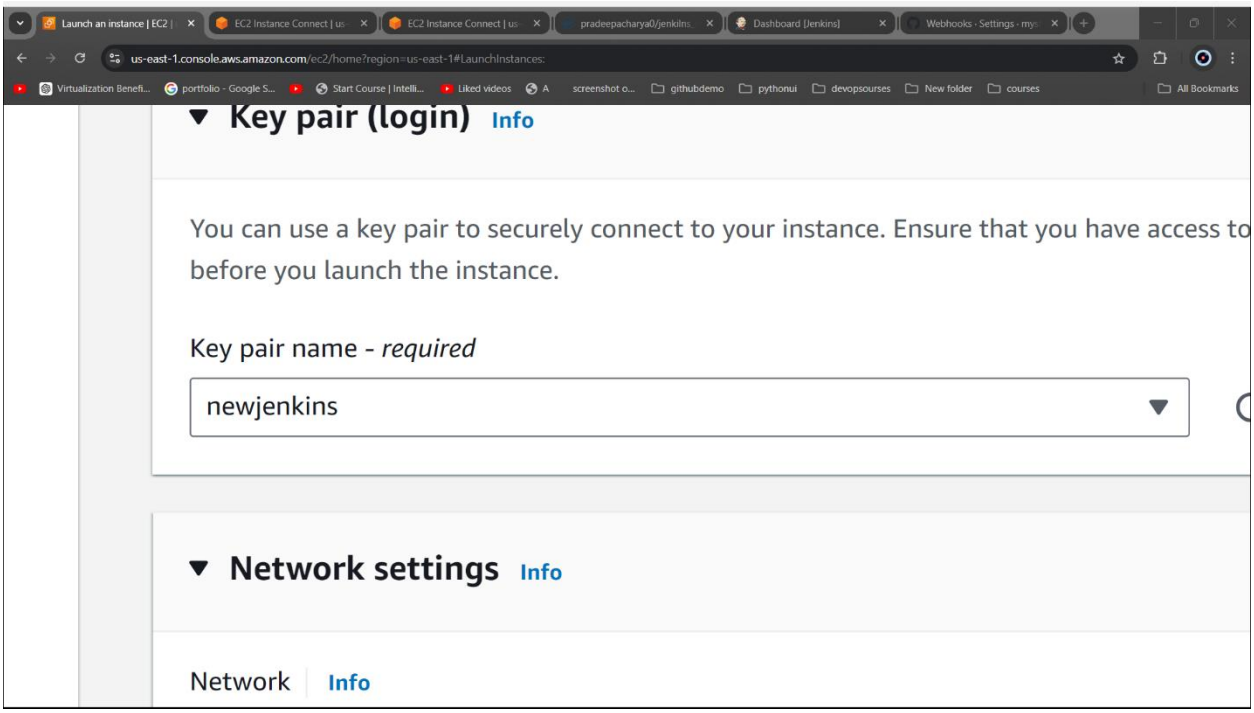
AMI:



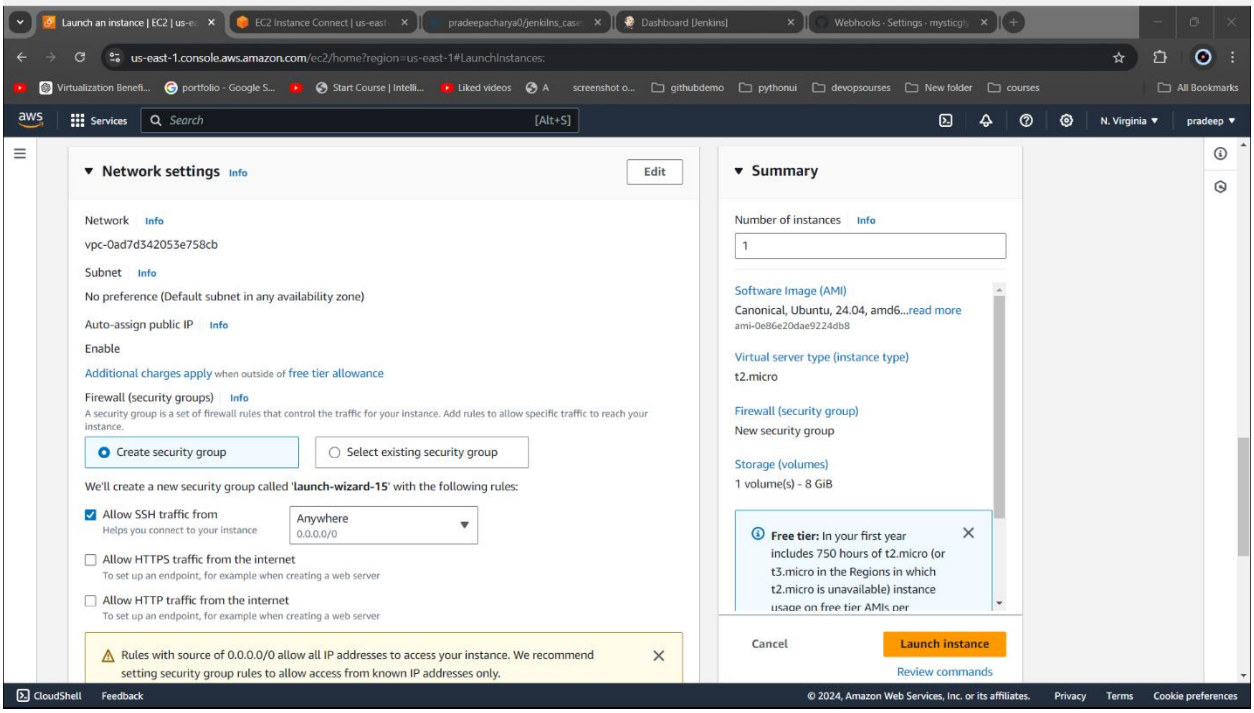
Instance type:



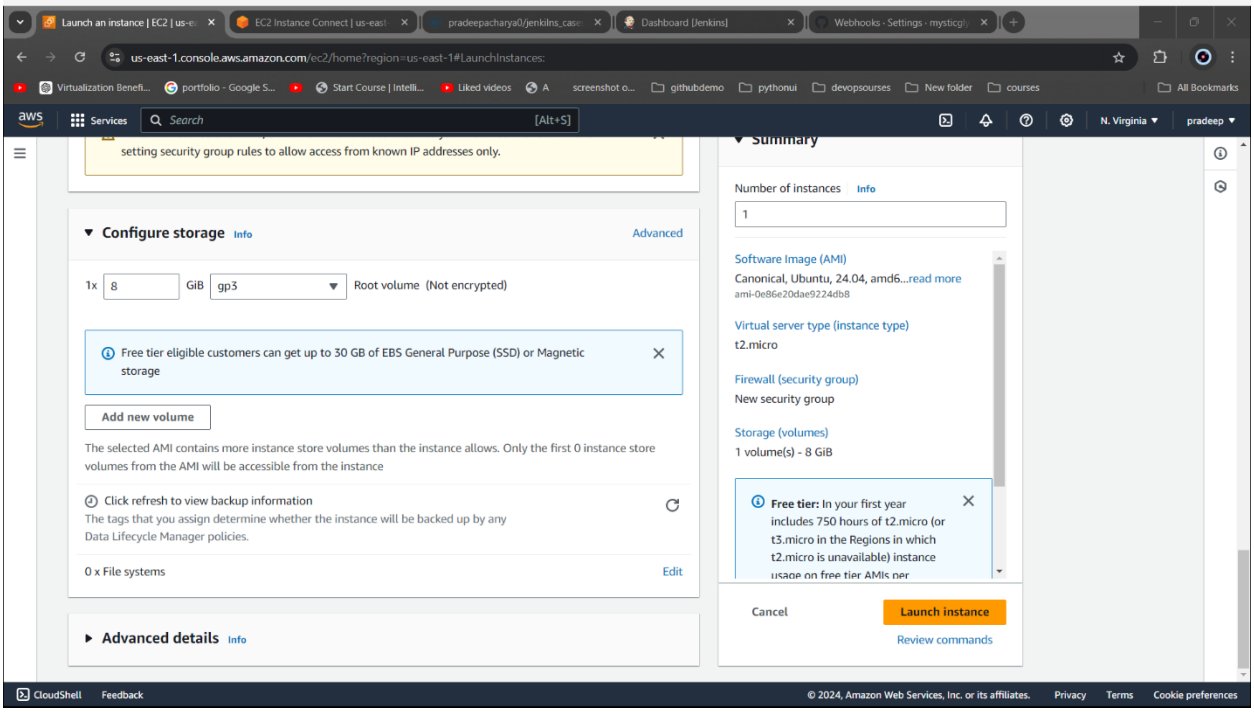
key pair:

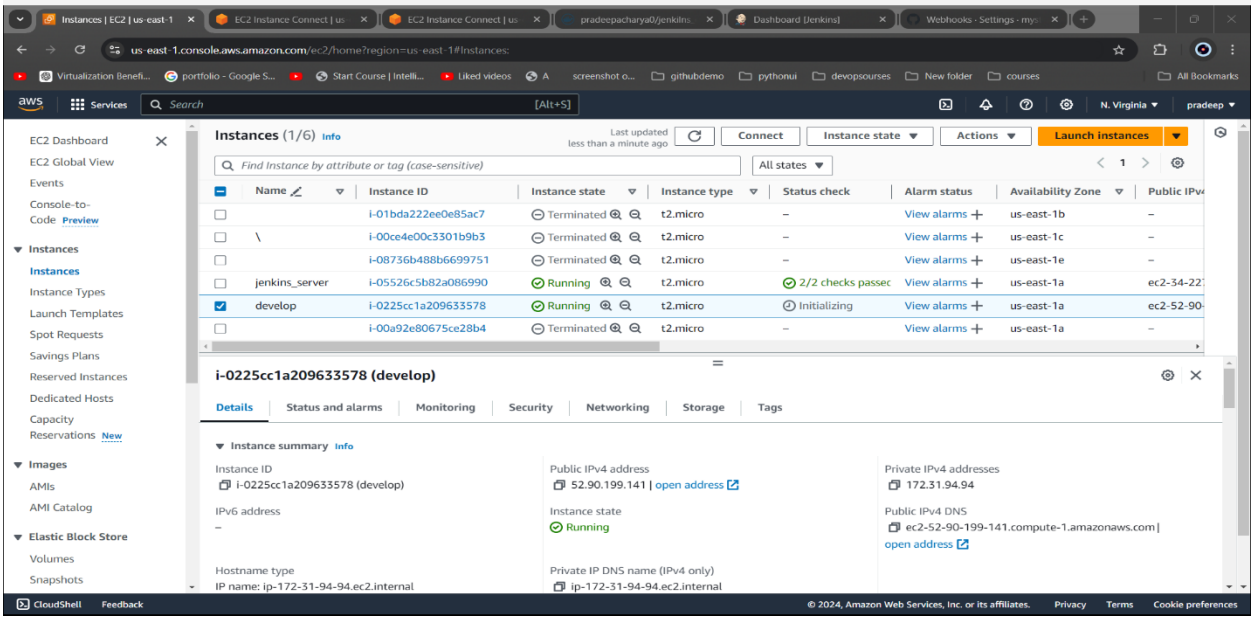


Default **Network settings:**

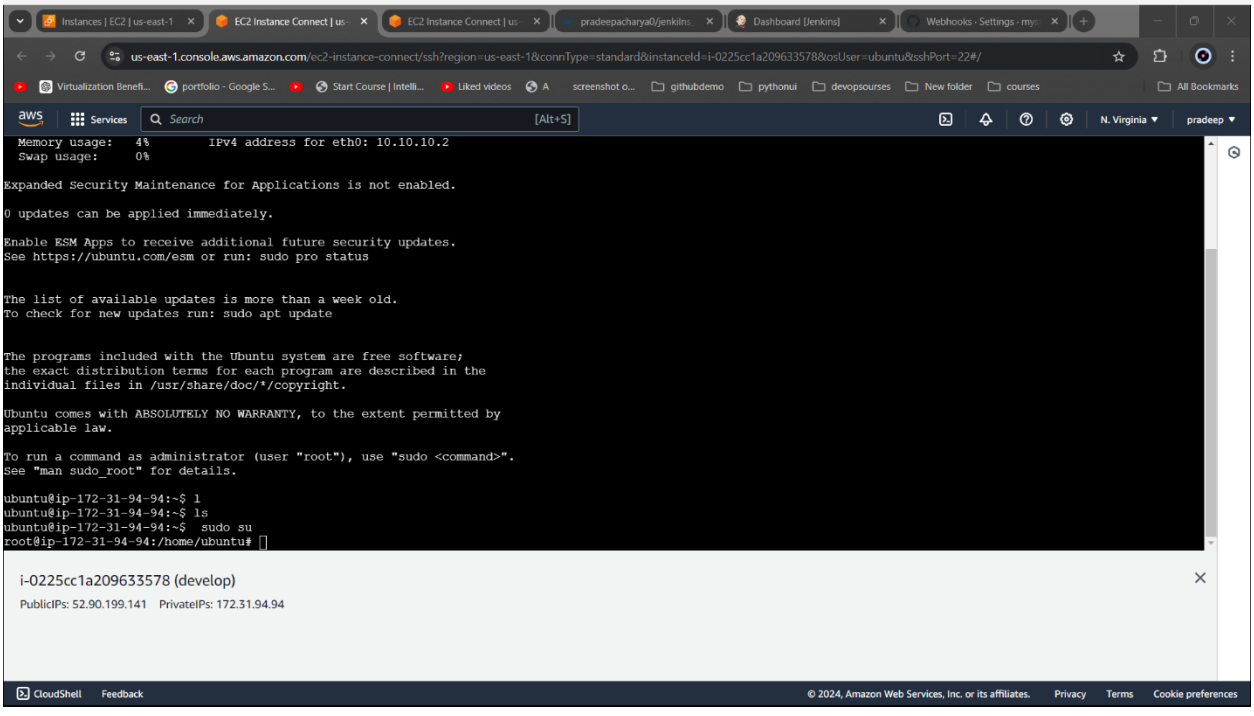


Default **Configure storage:**





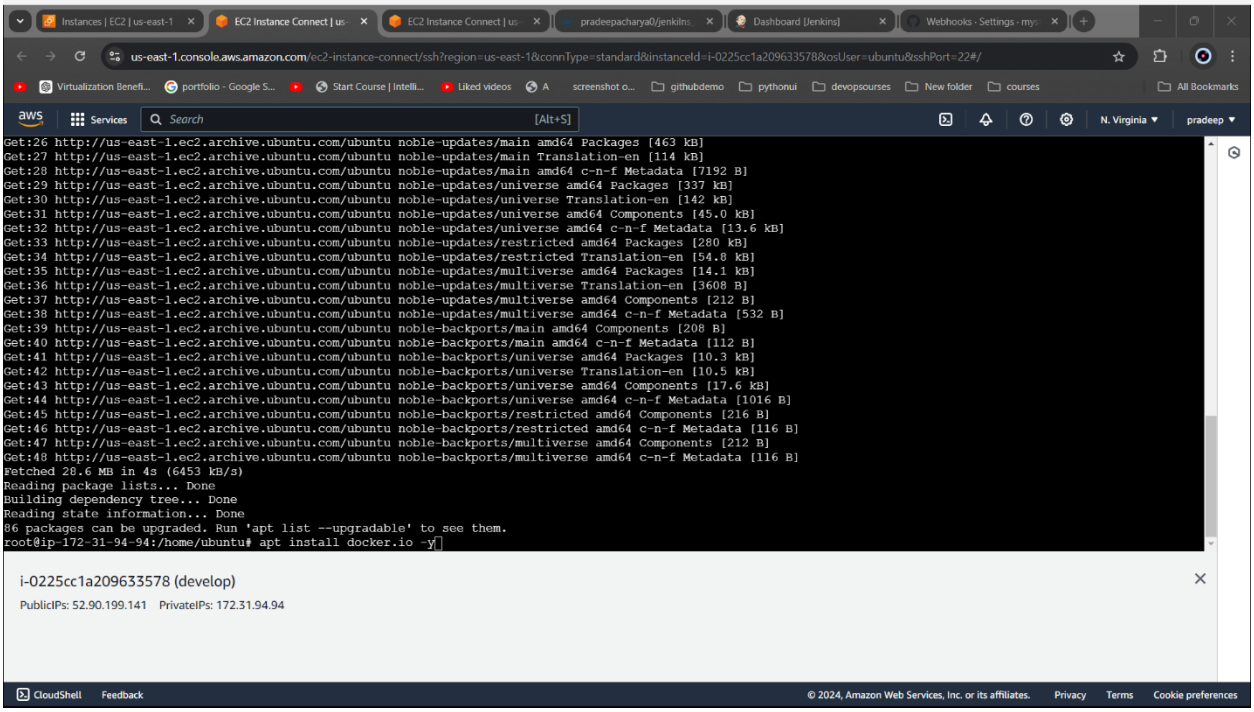
Connect to develop instance and install java and docker:



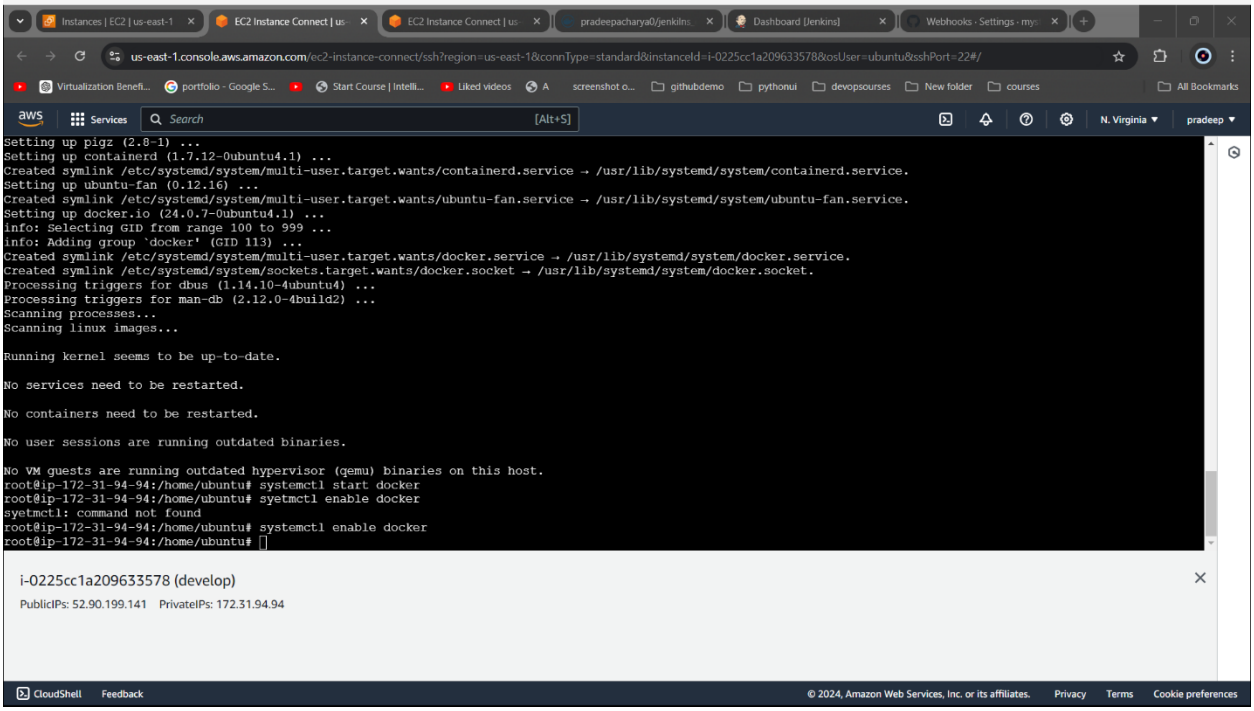
Connected.

Installing docker:

➔ **Apt install docker.io -y**

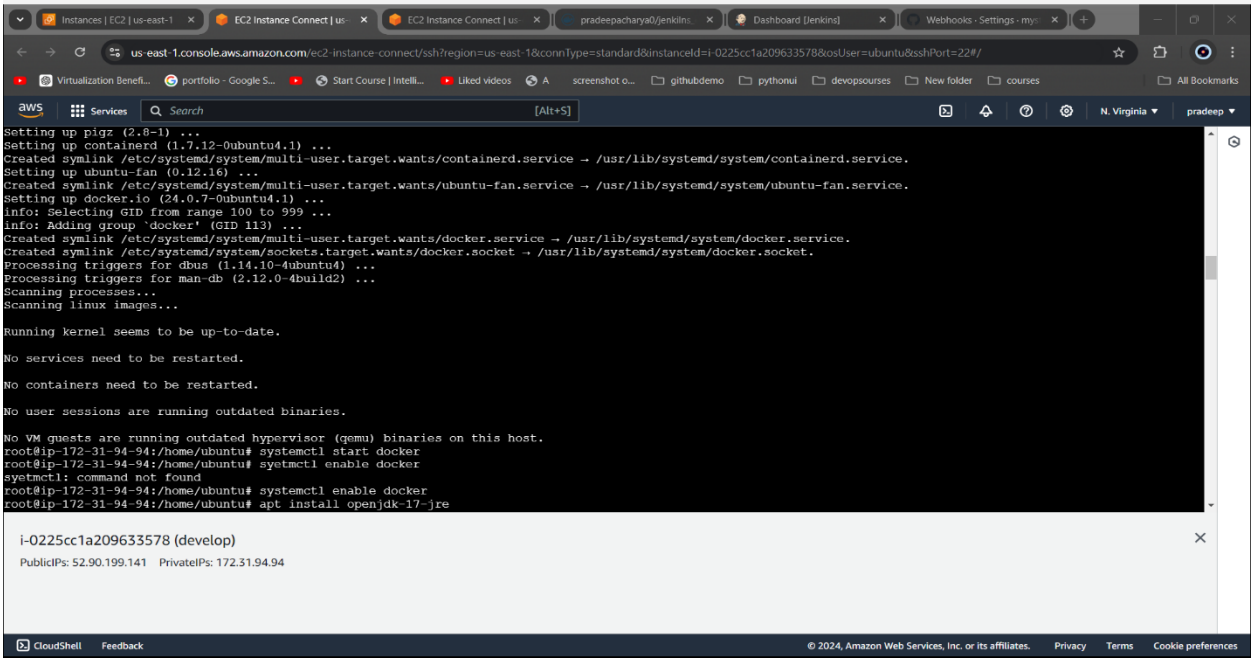


Docker started :

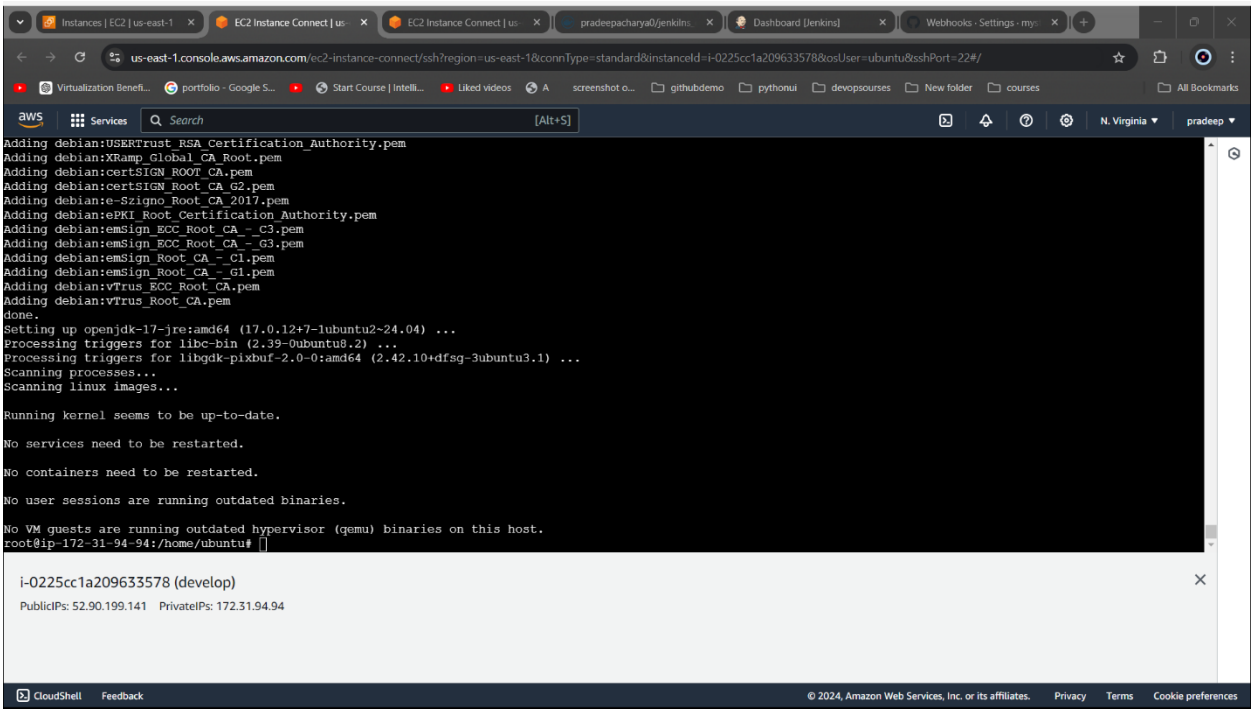


Installing java-17:

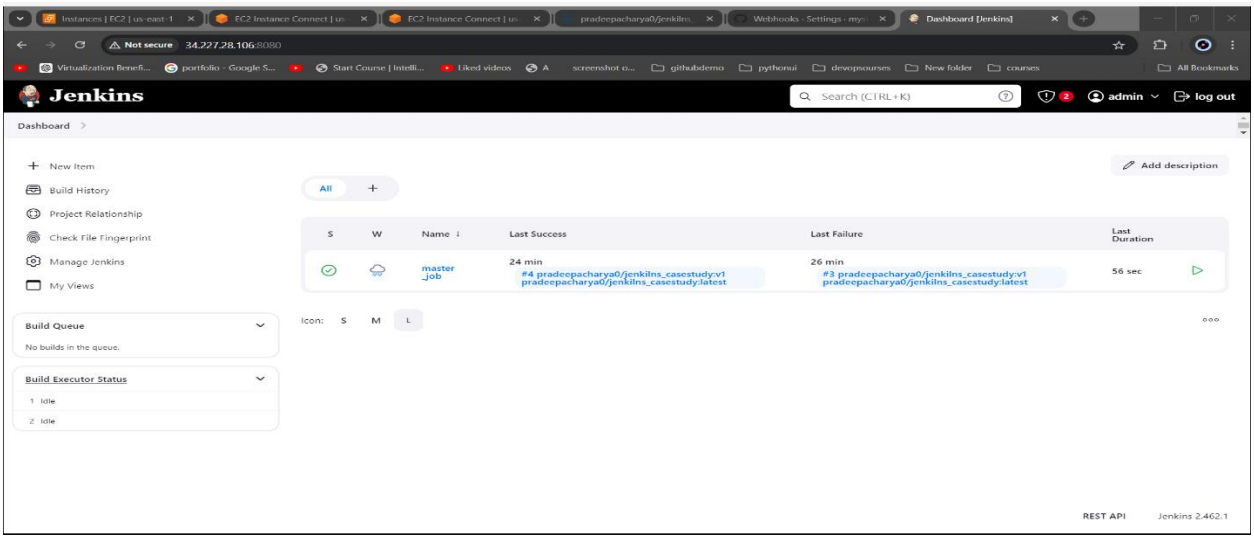
➔ **Apt install openjdk-17-jre**



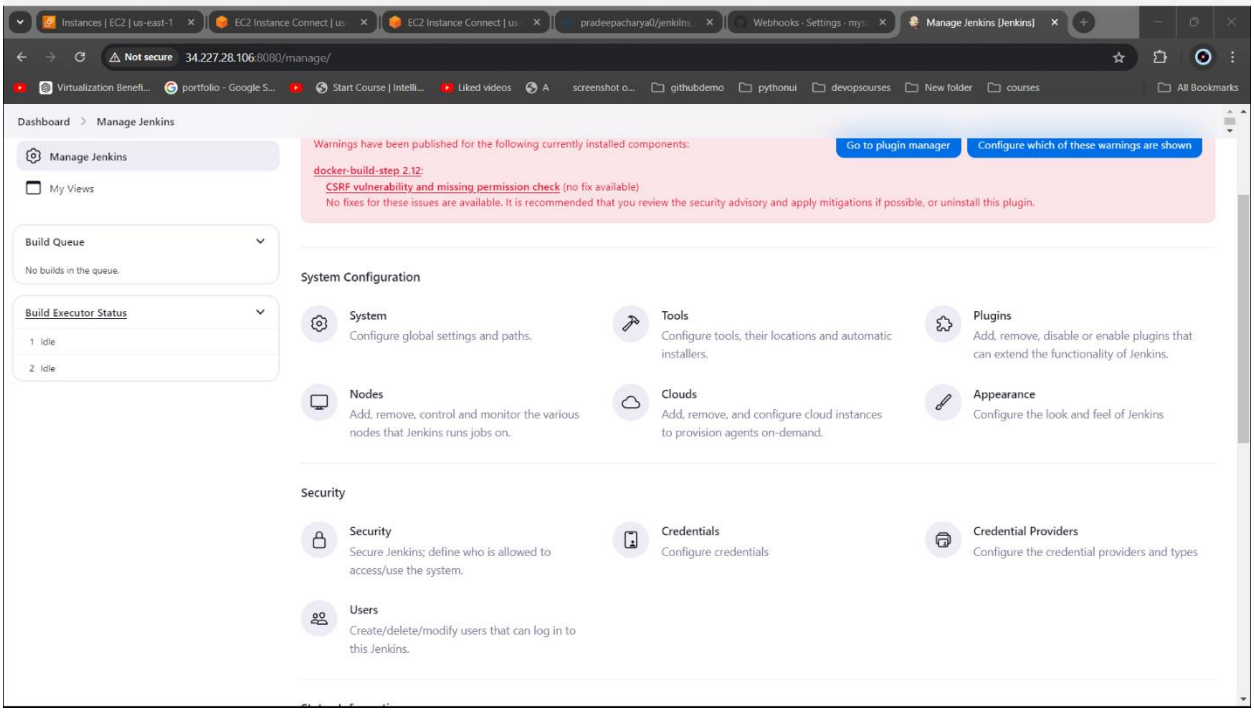
Installed:

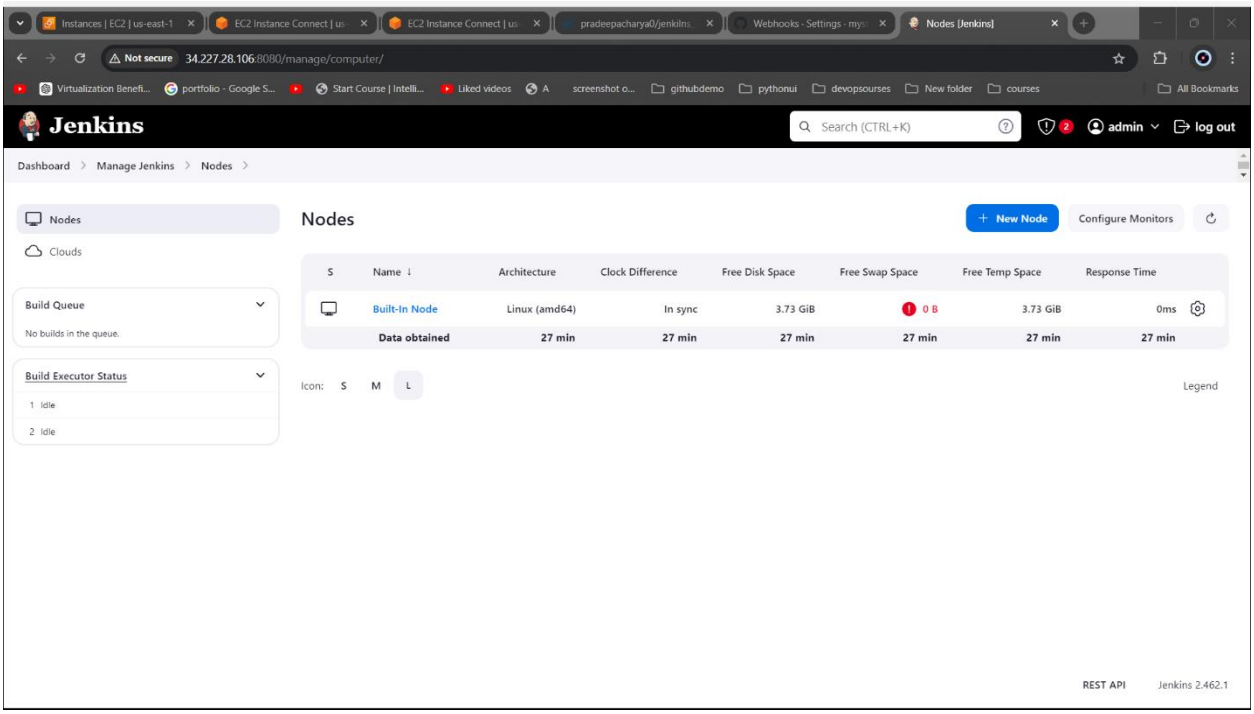


Connect the node to jenkins:



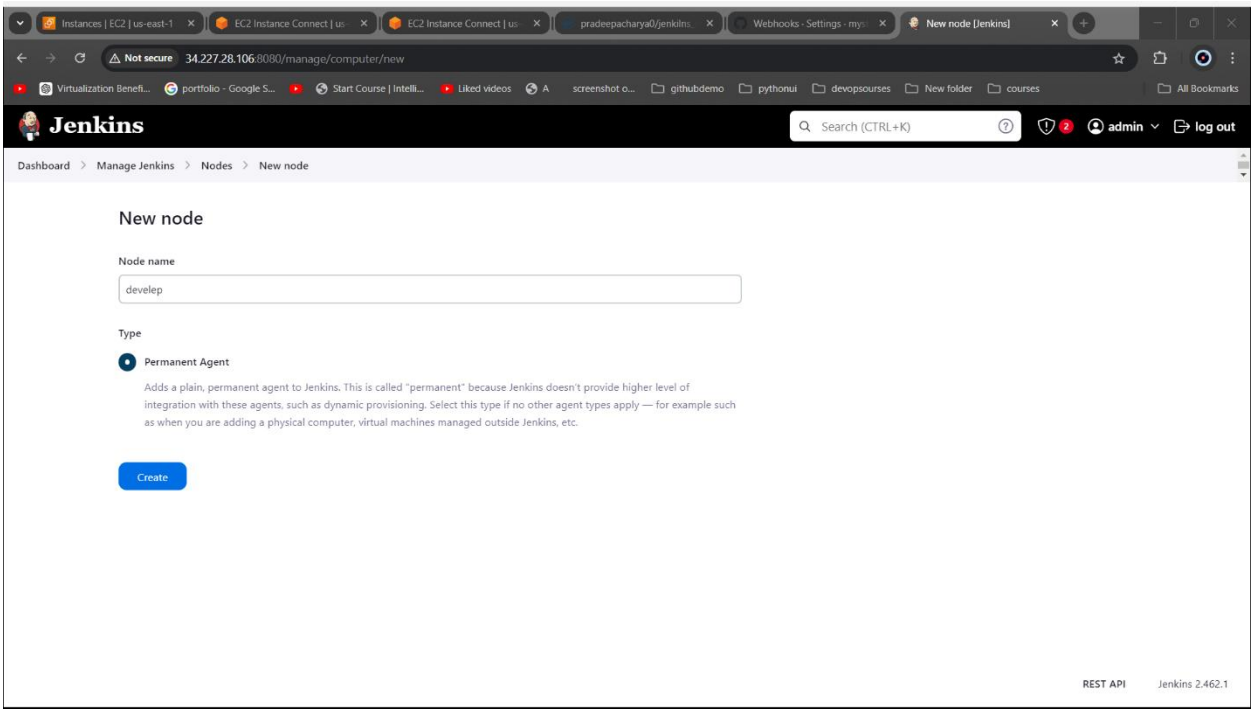
➔ **Manage jenkins-> node -> add new node**





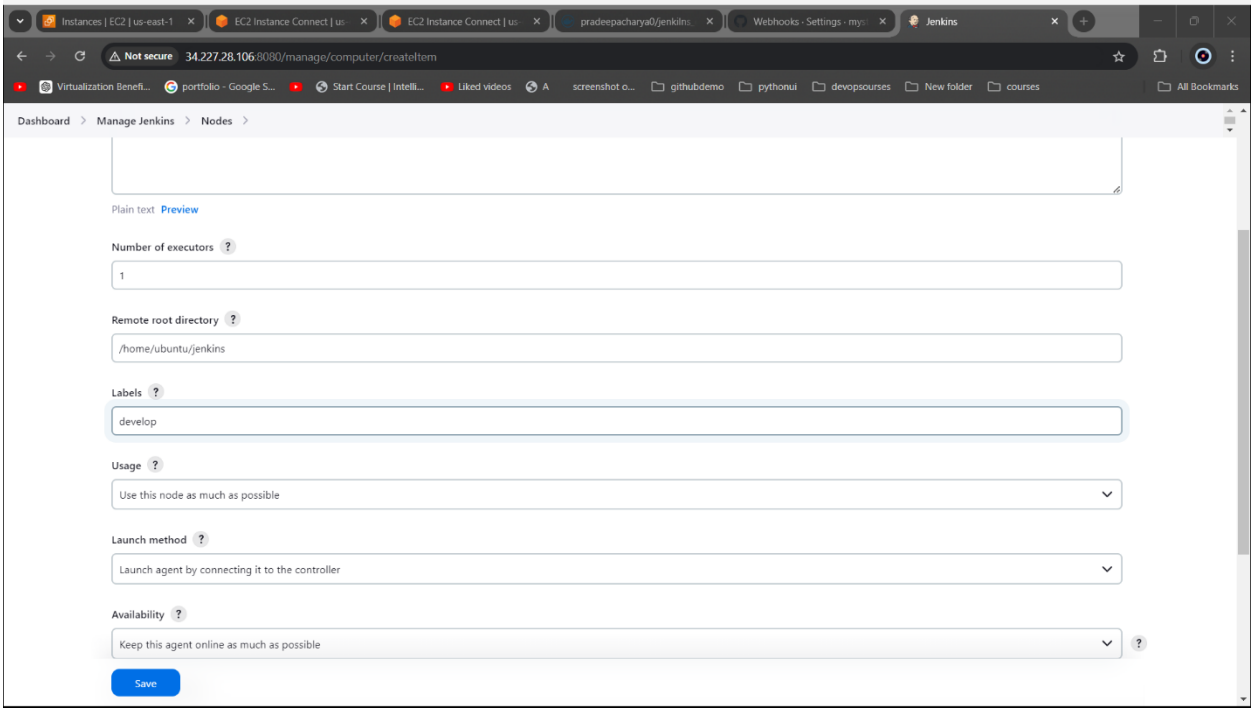
In order to add new node click on new node:

Node name and type:

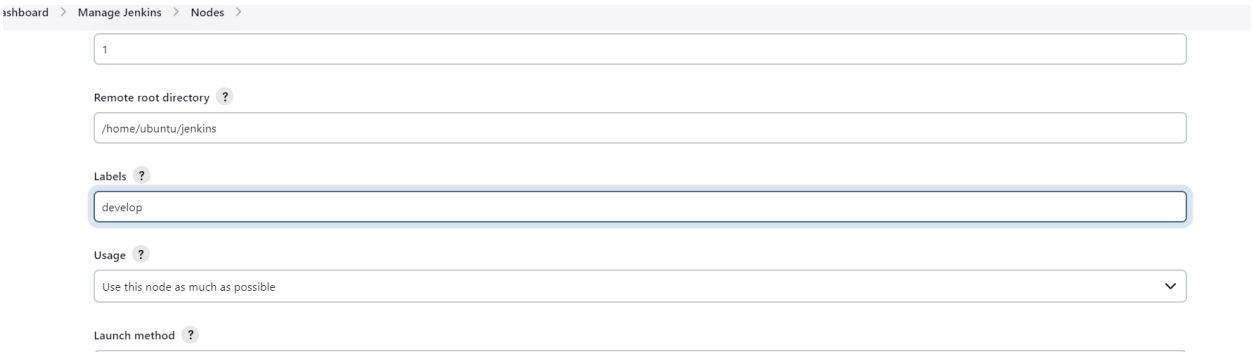


Here in remote root directory we need to provide the jenkins home directory in develop_node.

/home/ubuntu/jenkins



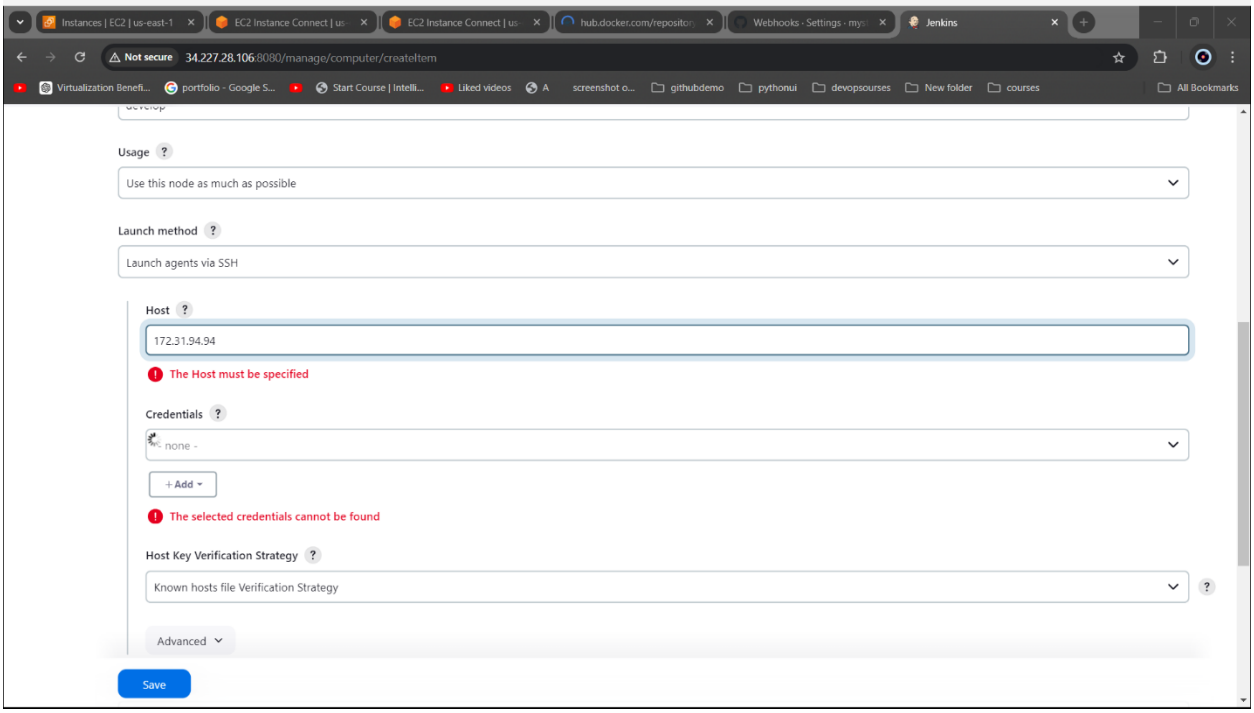
Labels:



Launch method:

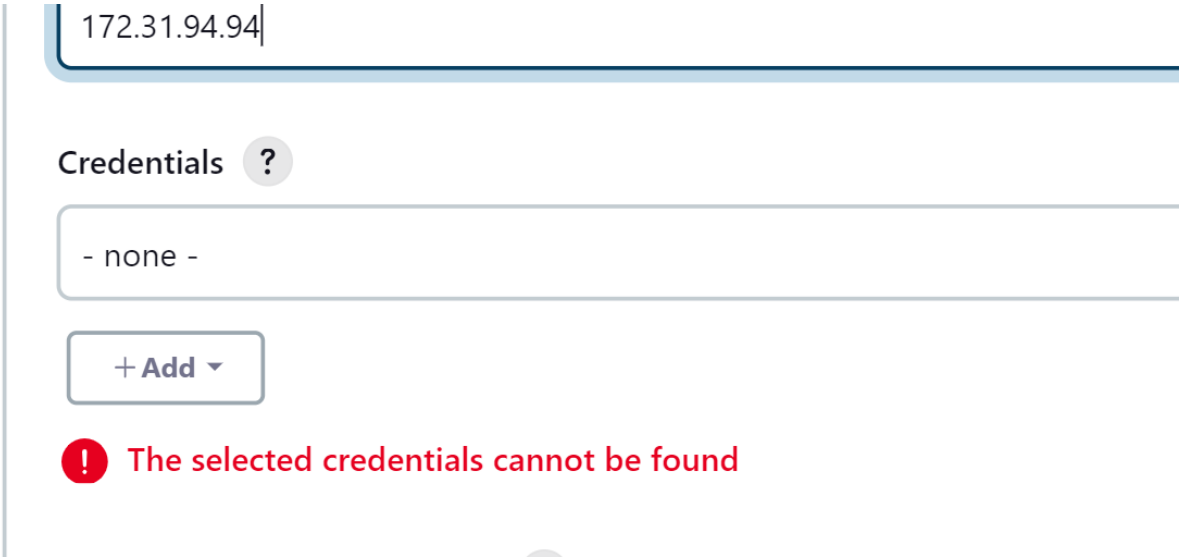
Via ssh :

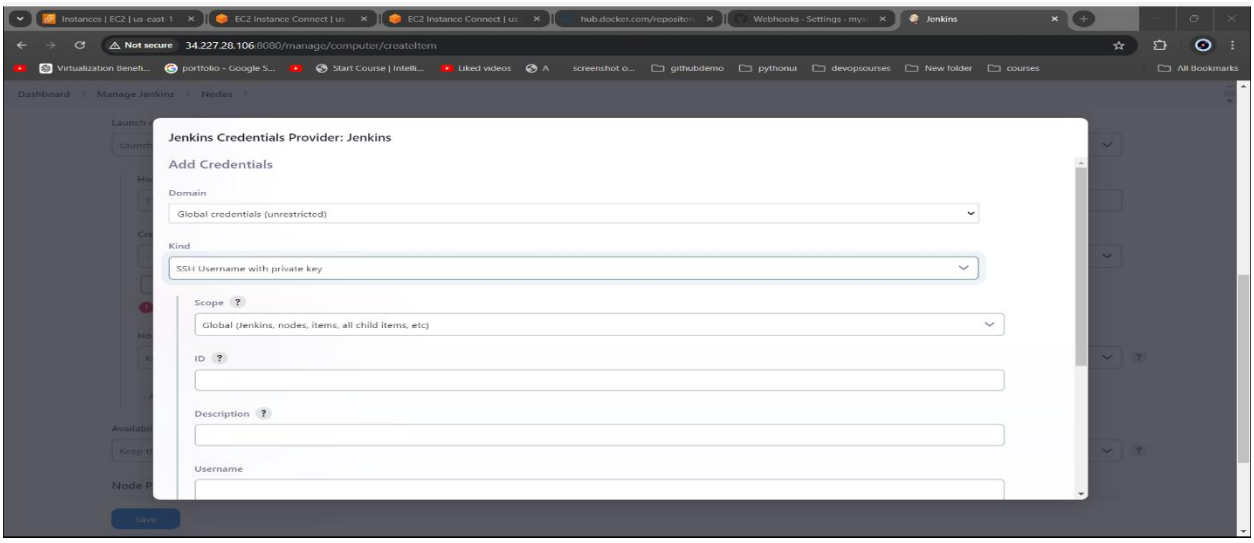
In host we need to provide the private IP address:



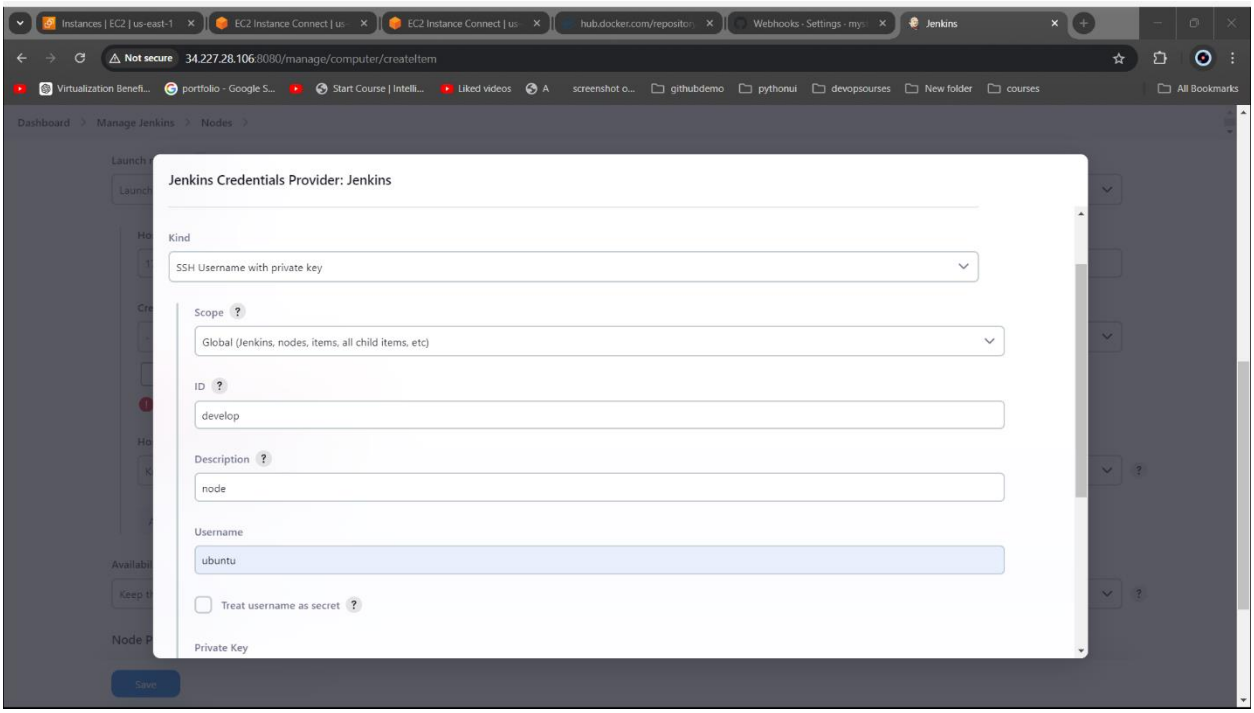
Now we need to provide credentials so we need to create credentials:

Adding credentials:

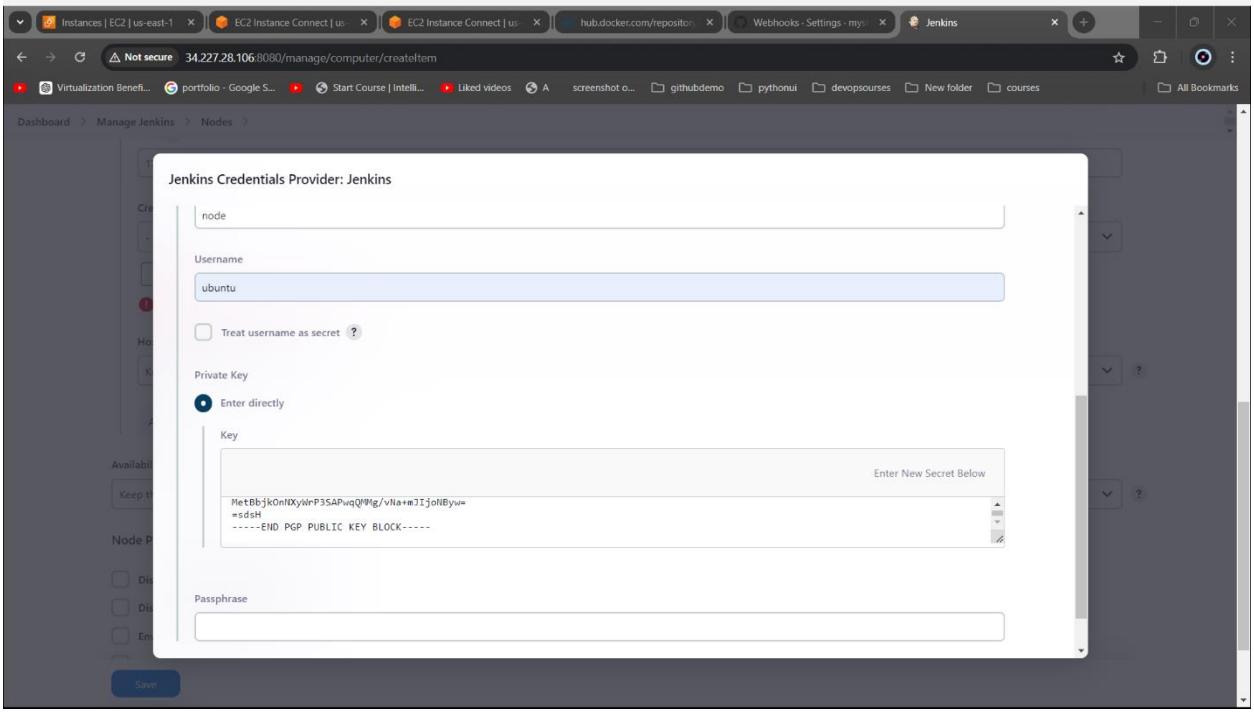




Providing id,description,etc,username.

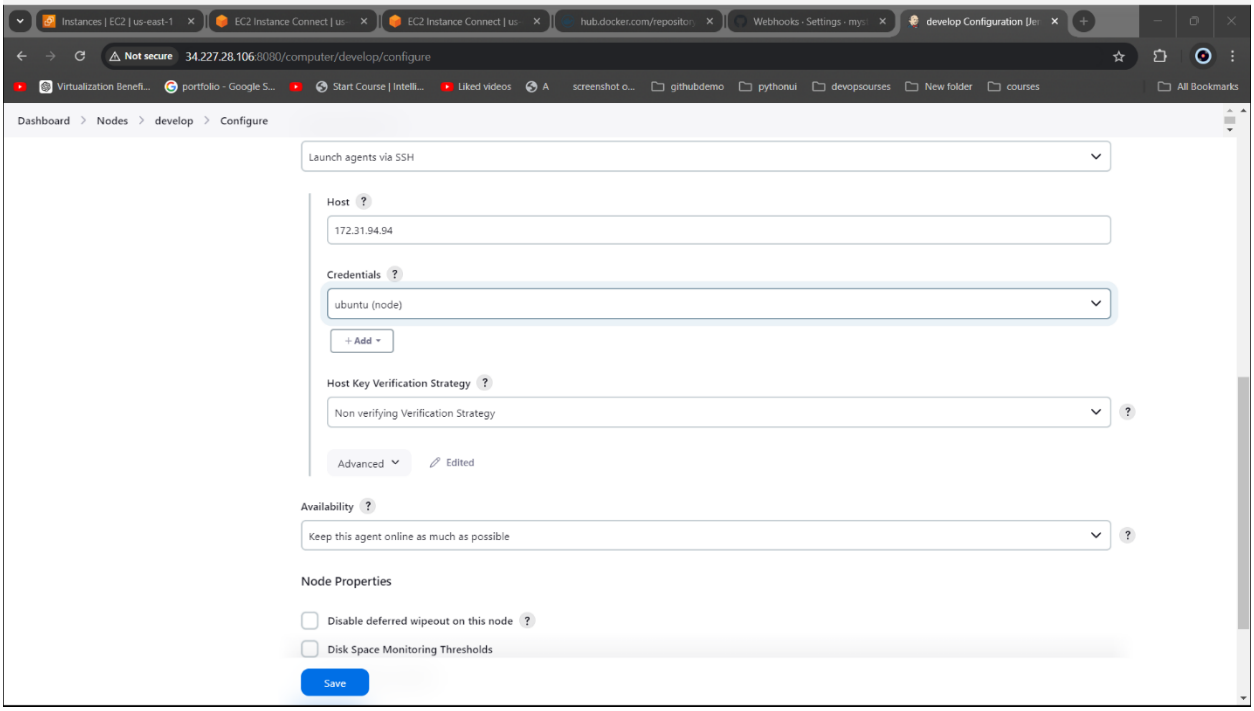


Private Key:

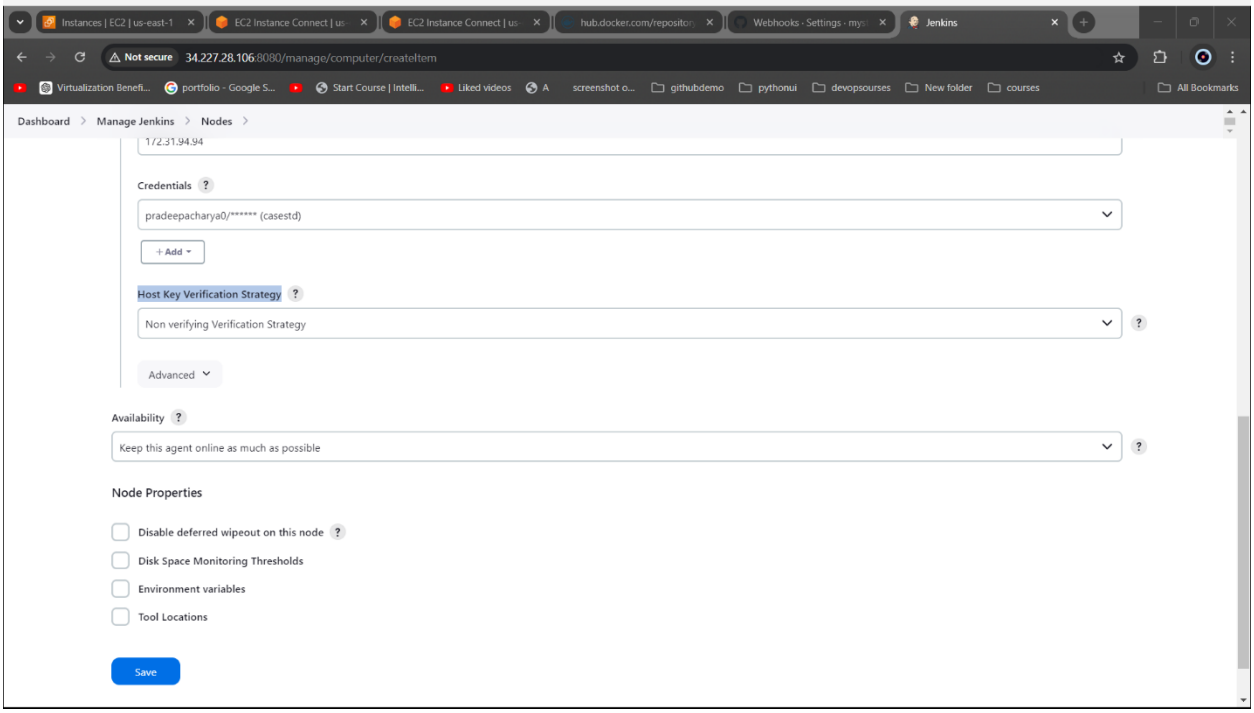


Private key is added.

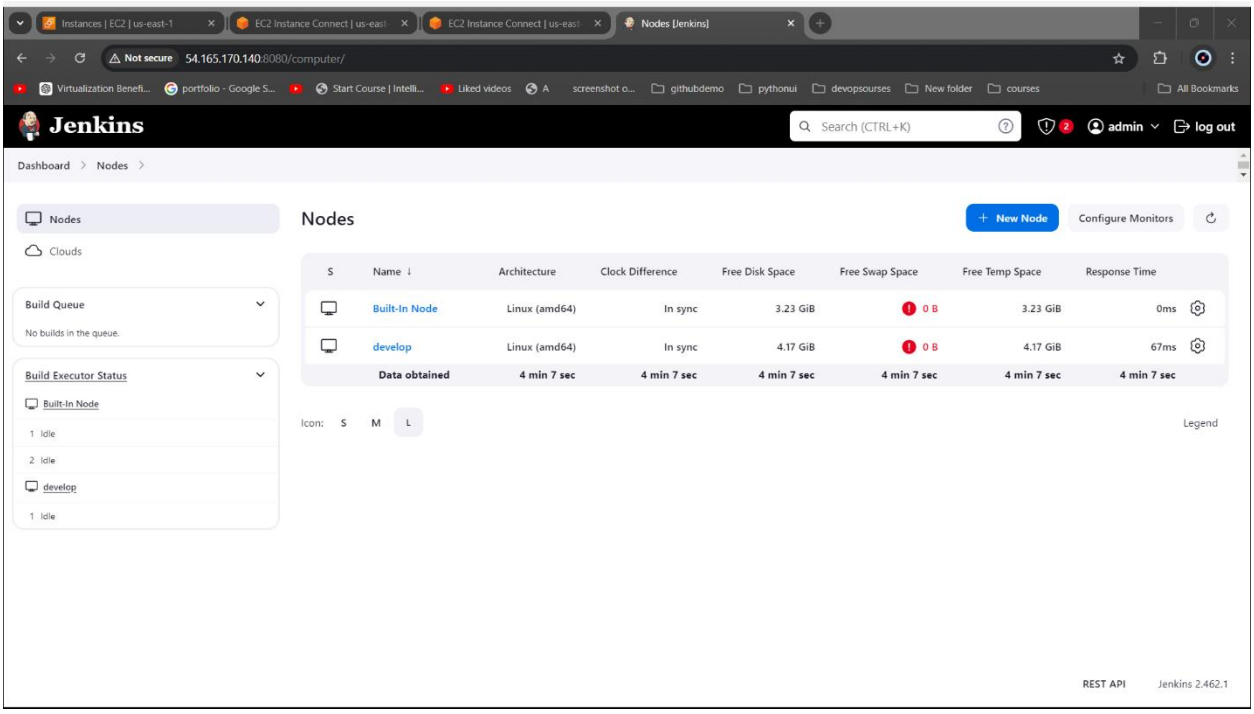
Selecting the credentials:



Host Key Verification Strategy:



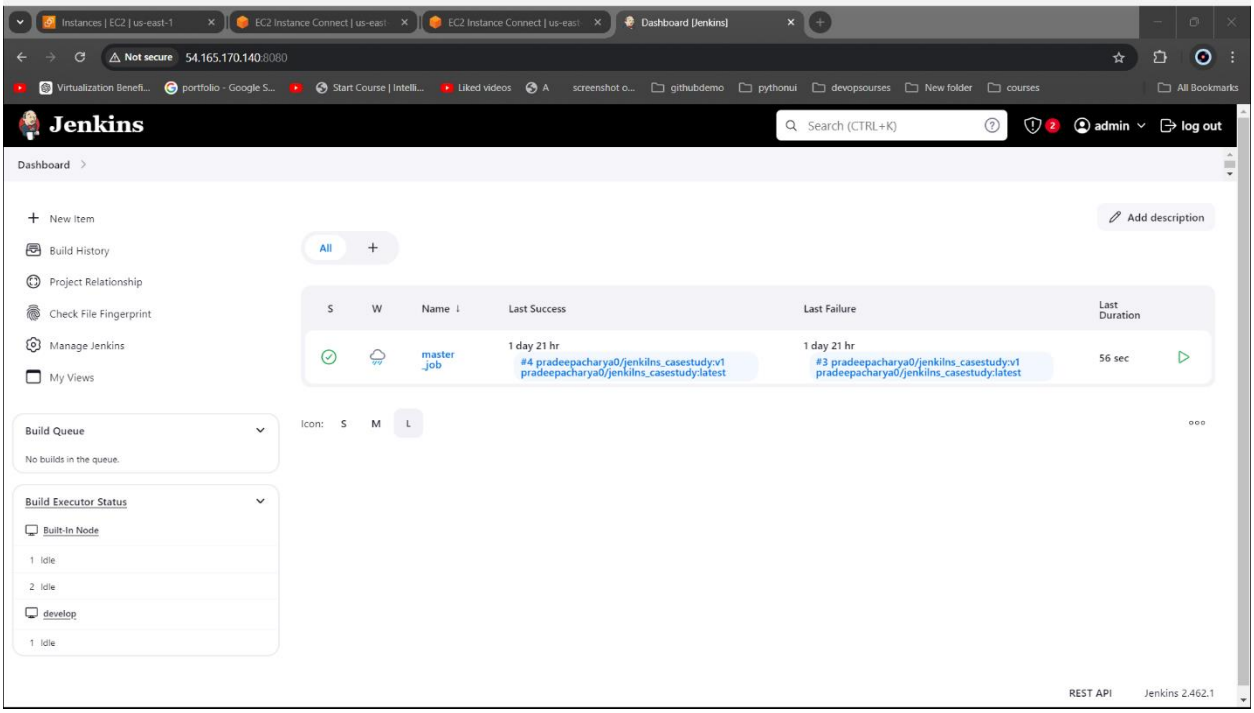
And save.



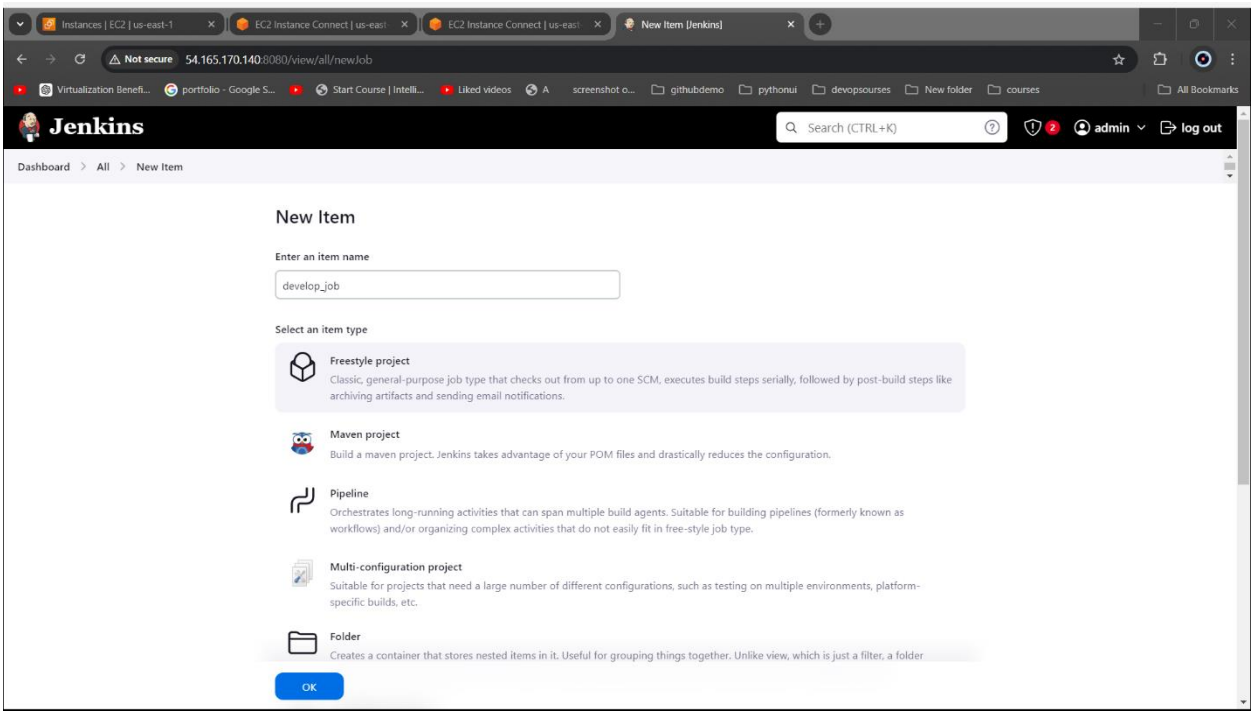
Node is connected.

In order to do this, if a commit is made to the develop branch, just build the product without publishing it. Let's create a job for this.

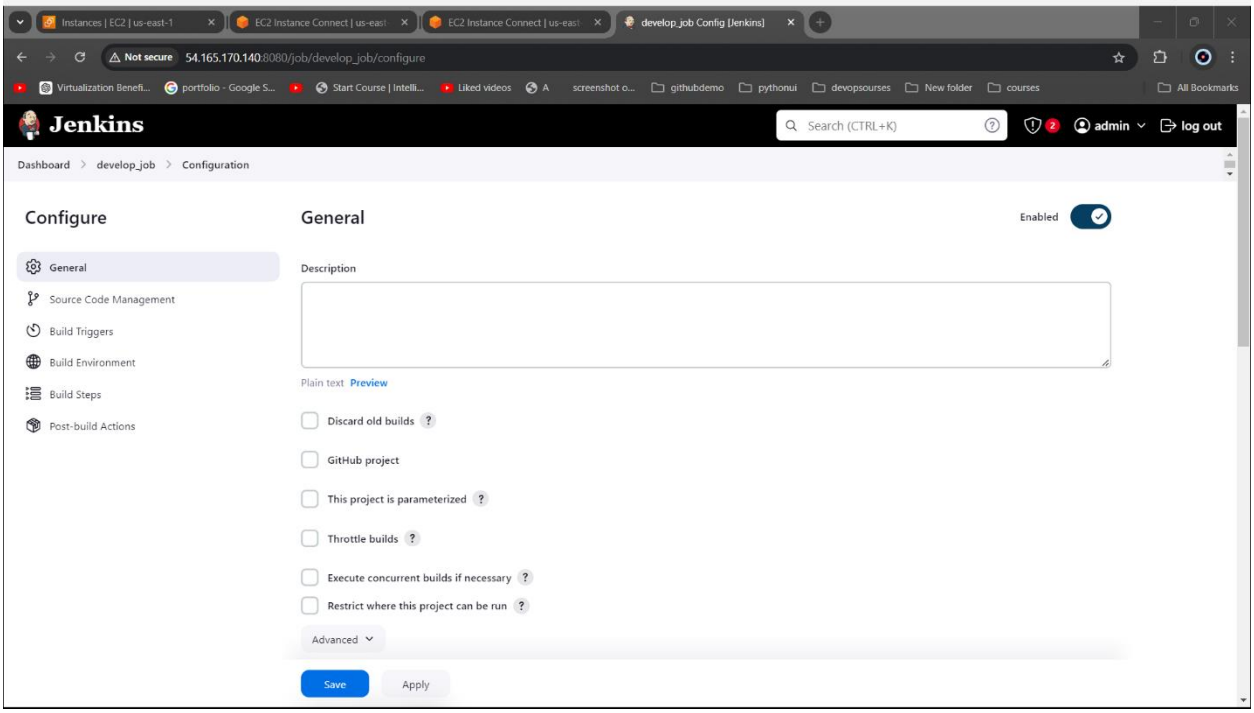
NAME : develop



Name:develop _job and the type will be free style:

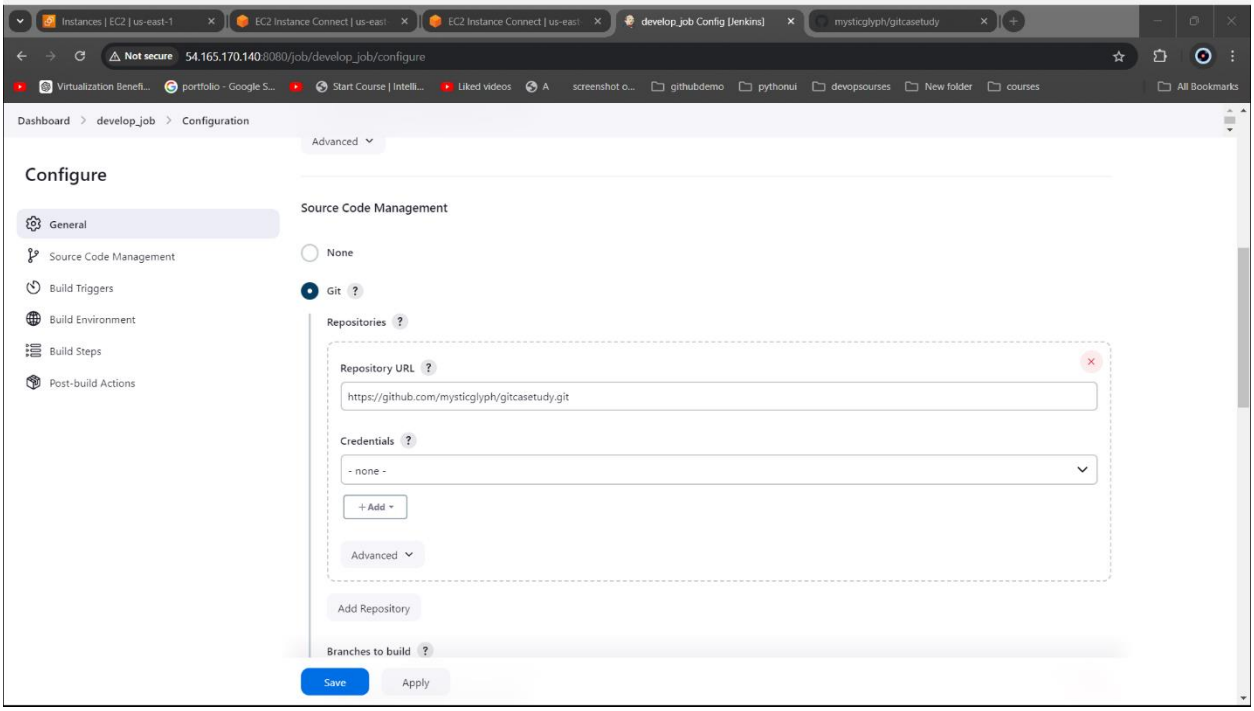


Job is created:

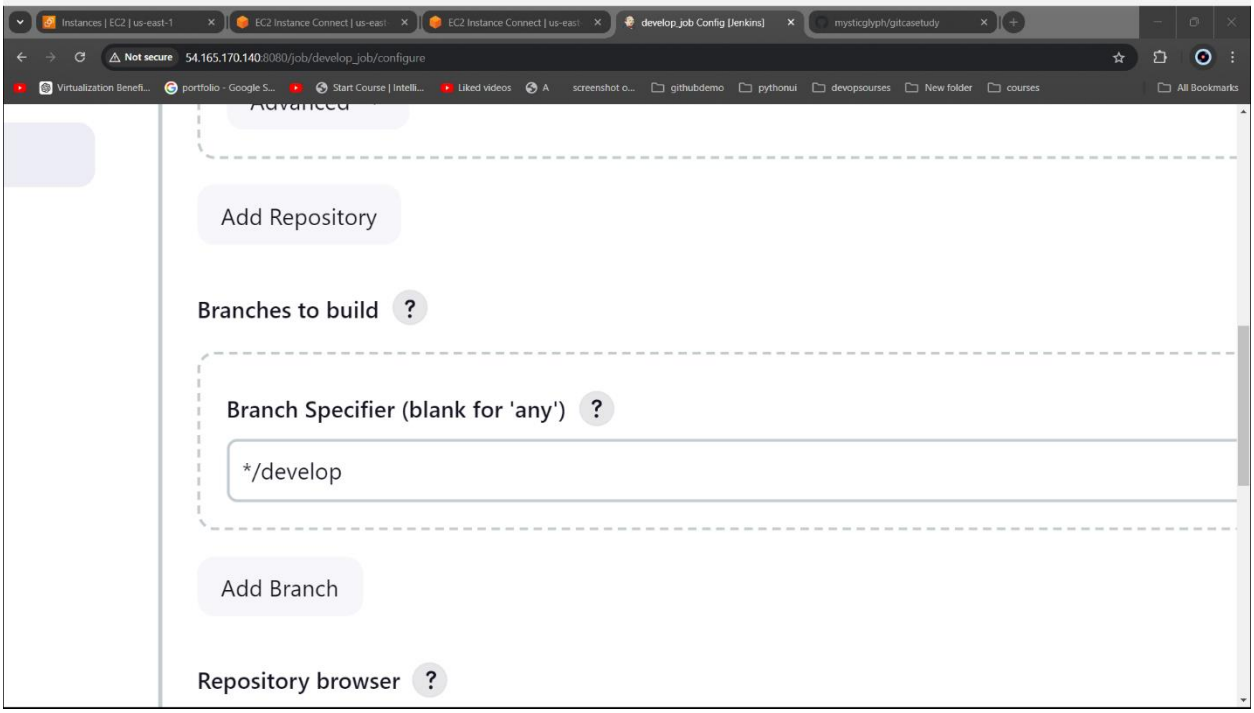


Source Code Management:

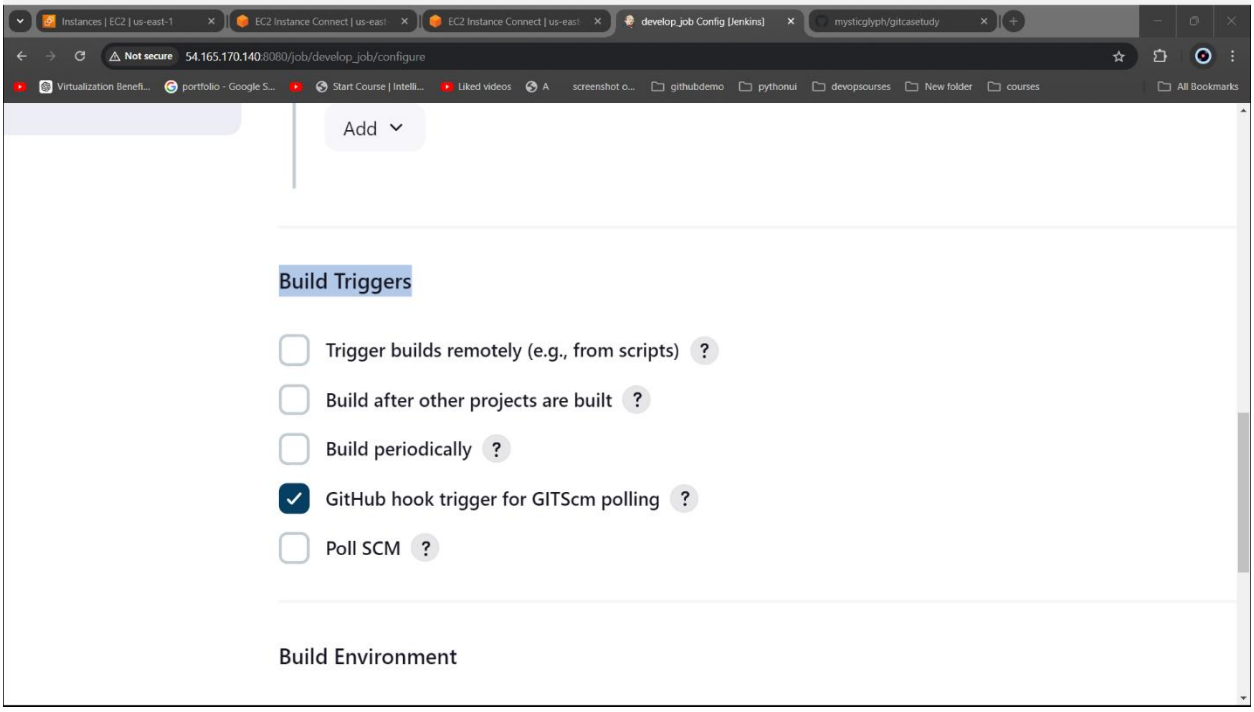
Git



Branch Specifier:

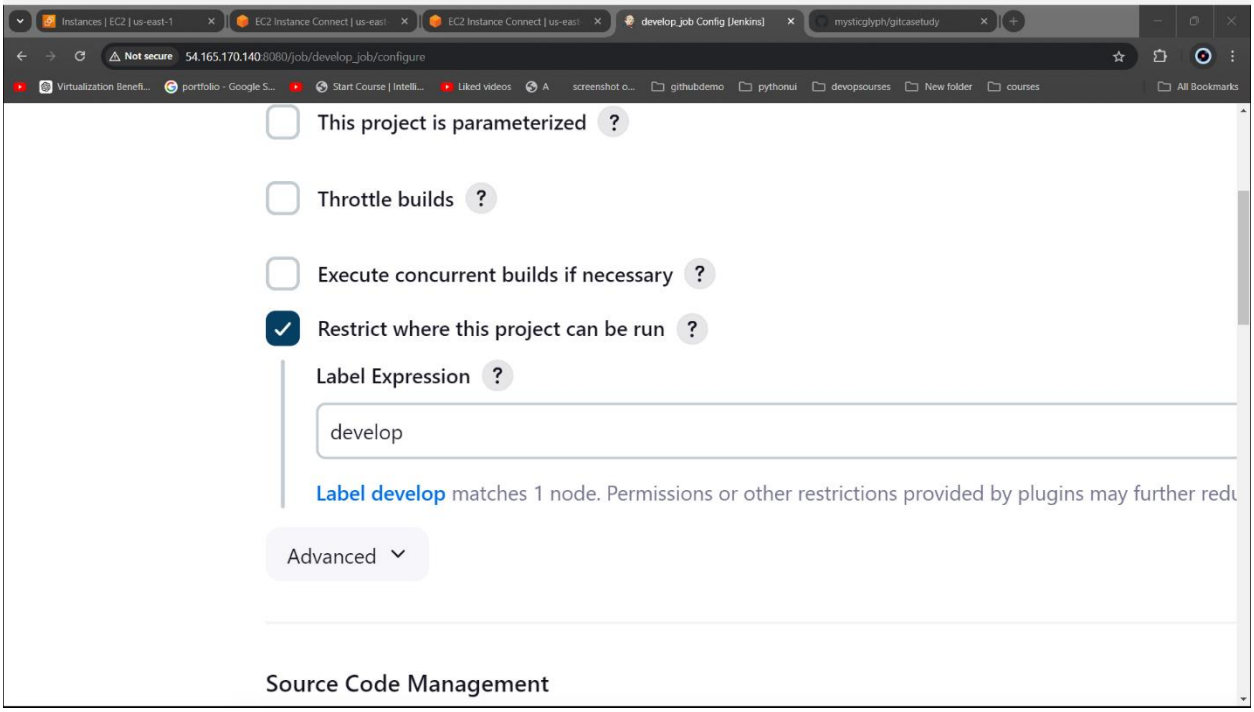


Build Triggers:



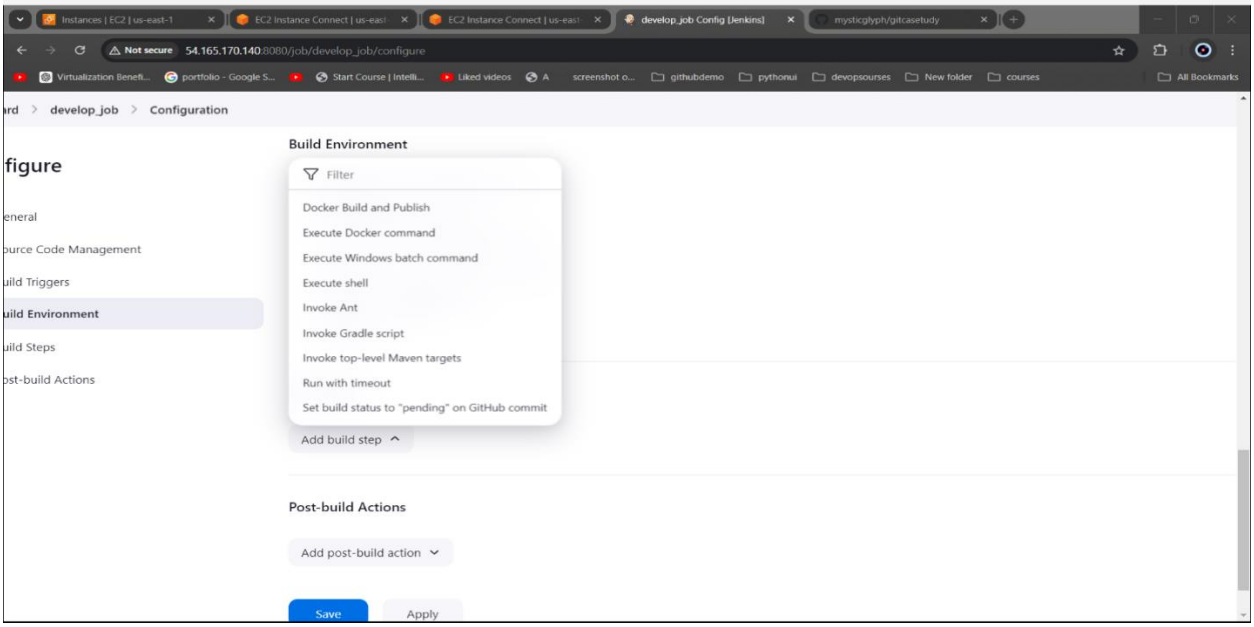
Select execute docker command:

Restrict where this project can be run:

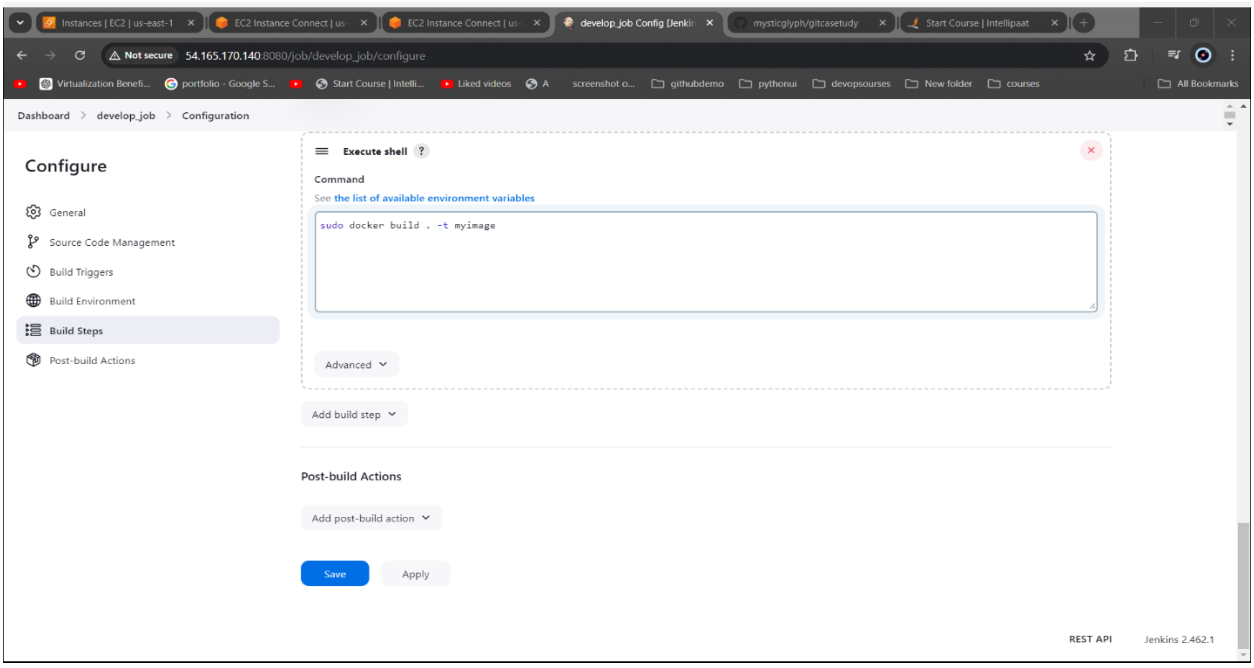


Build Steps:

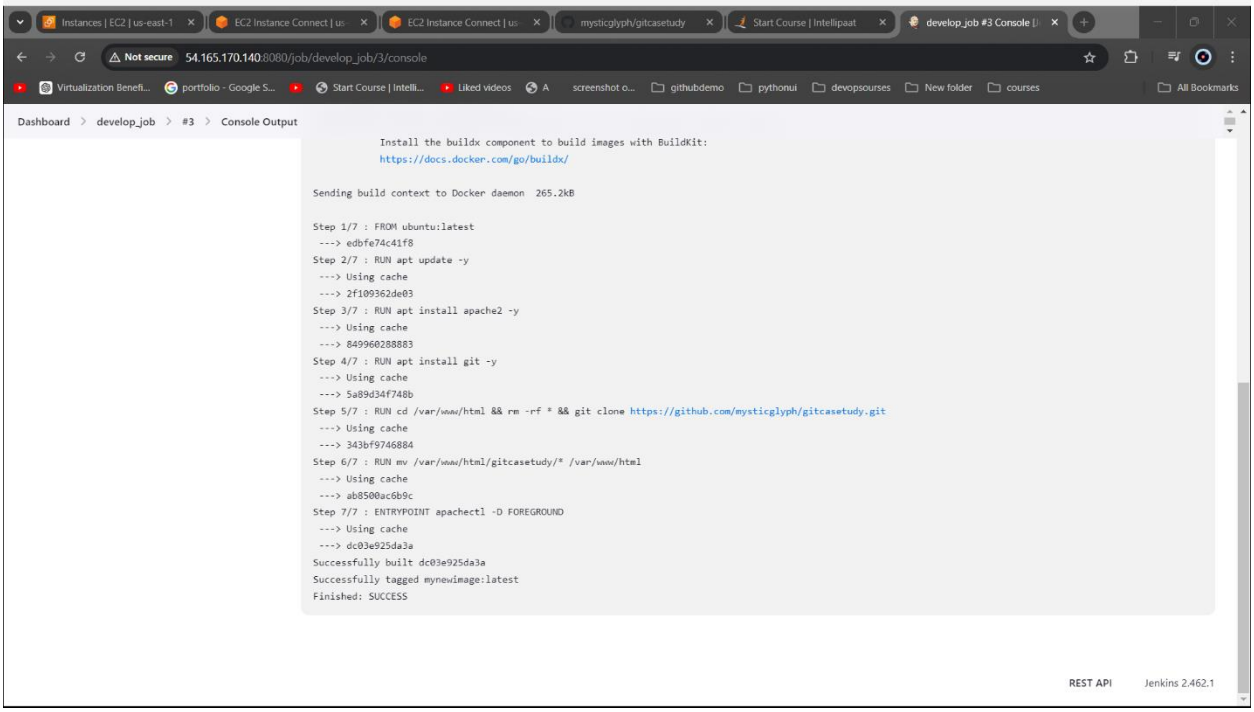
In order to build we will use exixute shell:

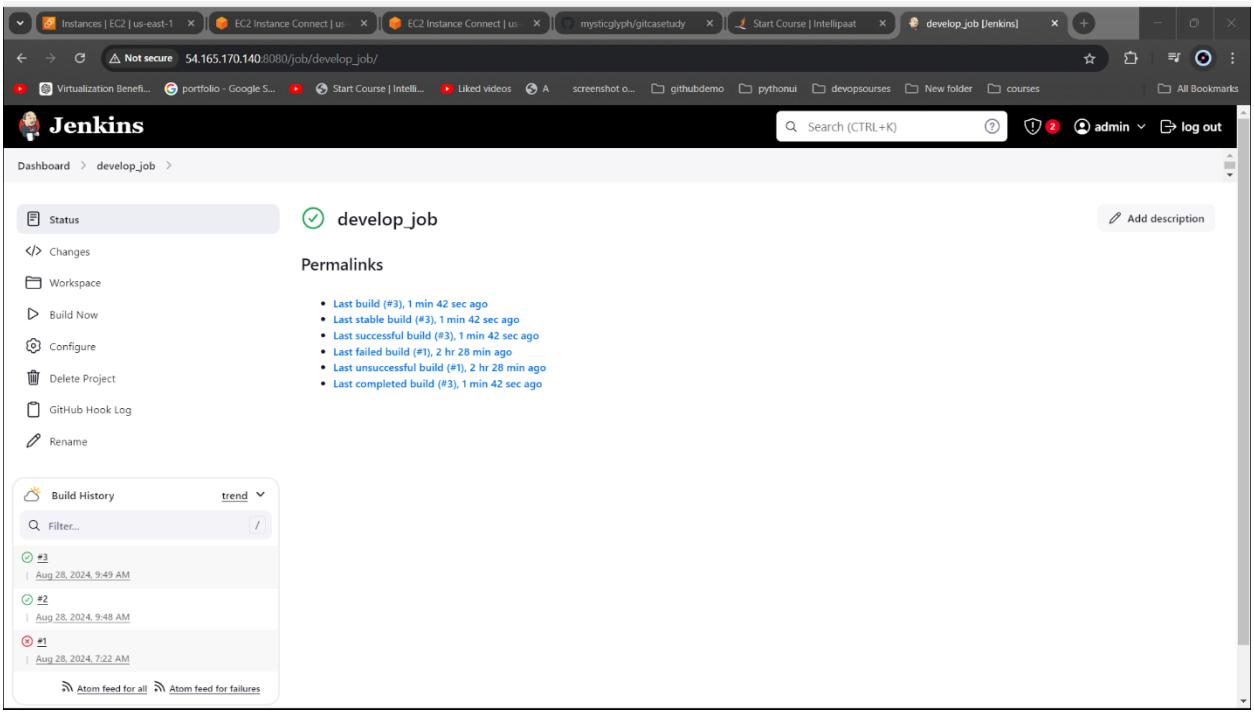


Docker command for build a image:

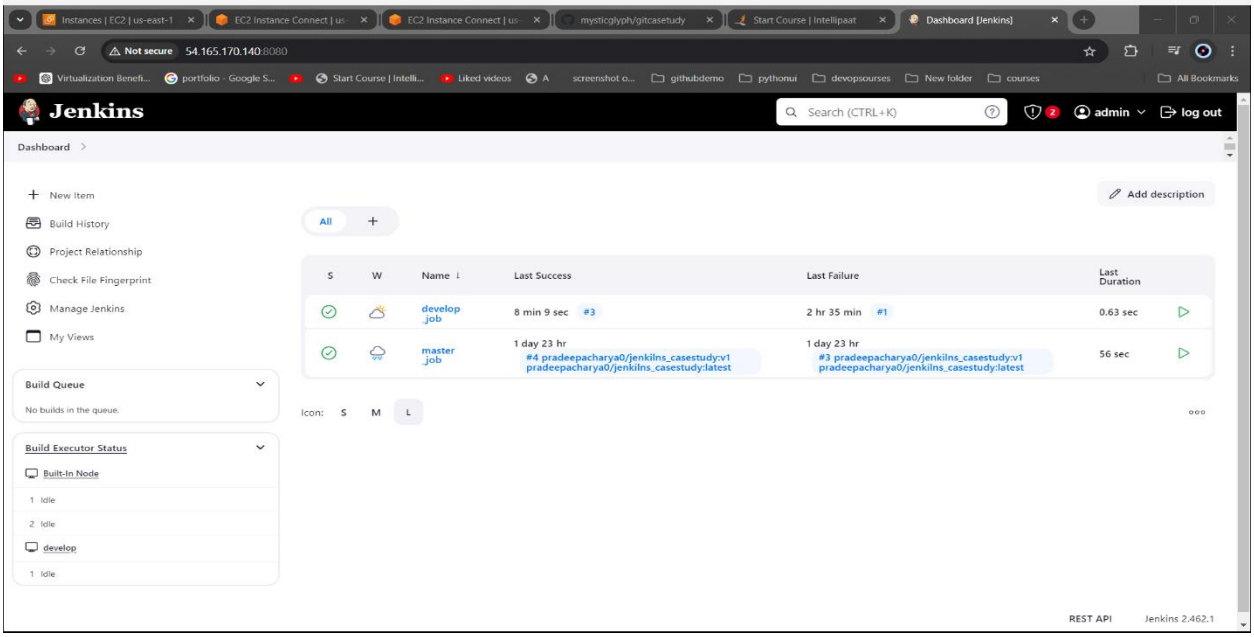


Job is executed successfully:



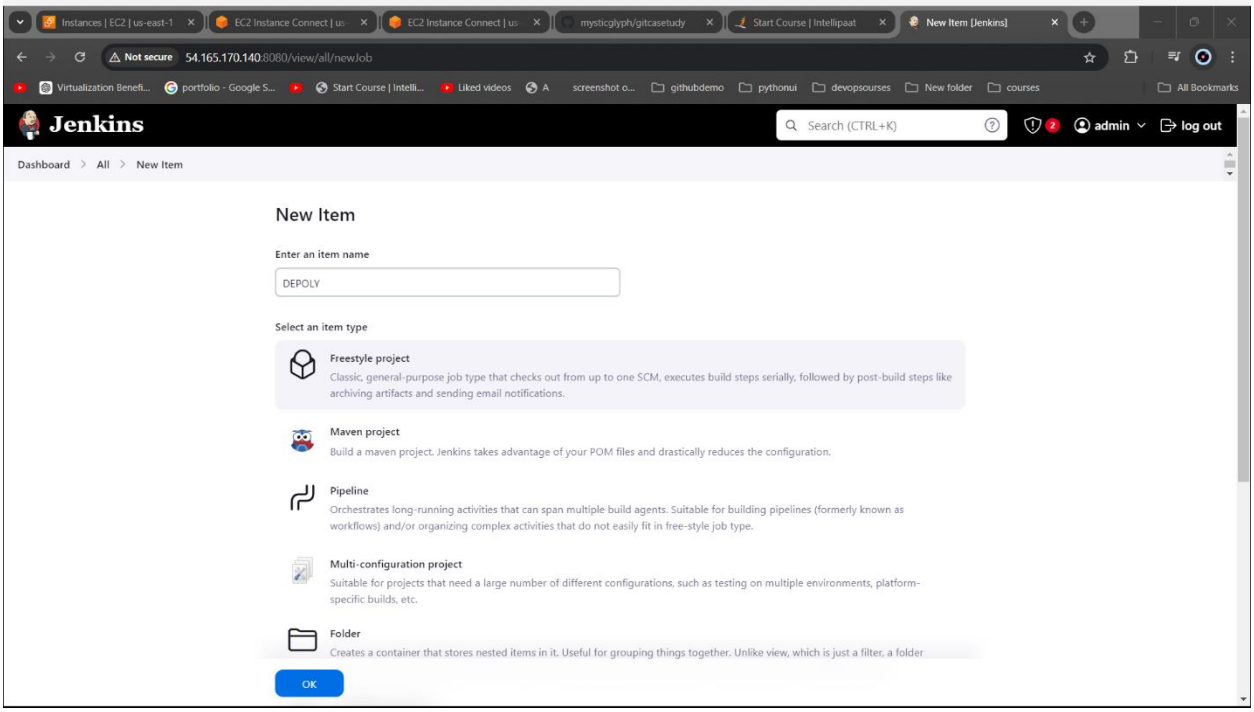

Not secure 54.165.170.140:8080/job/develop_job/
Jenkins
Search (CTRL+K)
admin
log out
Dashboard
develop_job
Status
Changes
Workspace
Build Now
Configure
Delete Project
GitHub Hook Log
Rename
develop_job
Add description
Permalinks
Last build (#3), 1 min 42 sec ago
Last stable build (#3), 1 min 42 sec ago
Last successful build (#3), 1 min 42 sec ago
Last failed build (#1), 2 hr 28 min ago
Last unsuccessful build (#1), 2 hr 28 min ago
Last completed build (#3), 1 min 42 sec ago
Build History
trend
Filter...
#3
Aug 28, 2024, 9:49 AM
#2
Aug 28, 2024, 9:48 AM
#1
Aug 28, 2024, 7:22 AM
Atom feed for all
Atom feed for failures

Now we need to create a container with Ubuntu and Apache installed in it. Then, create a new job.

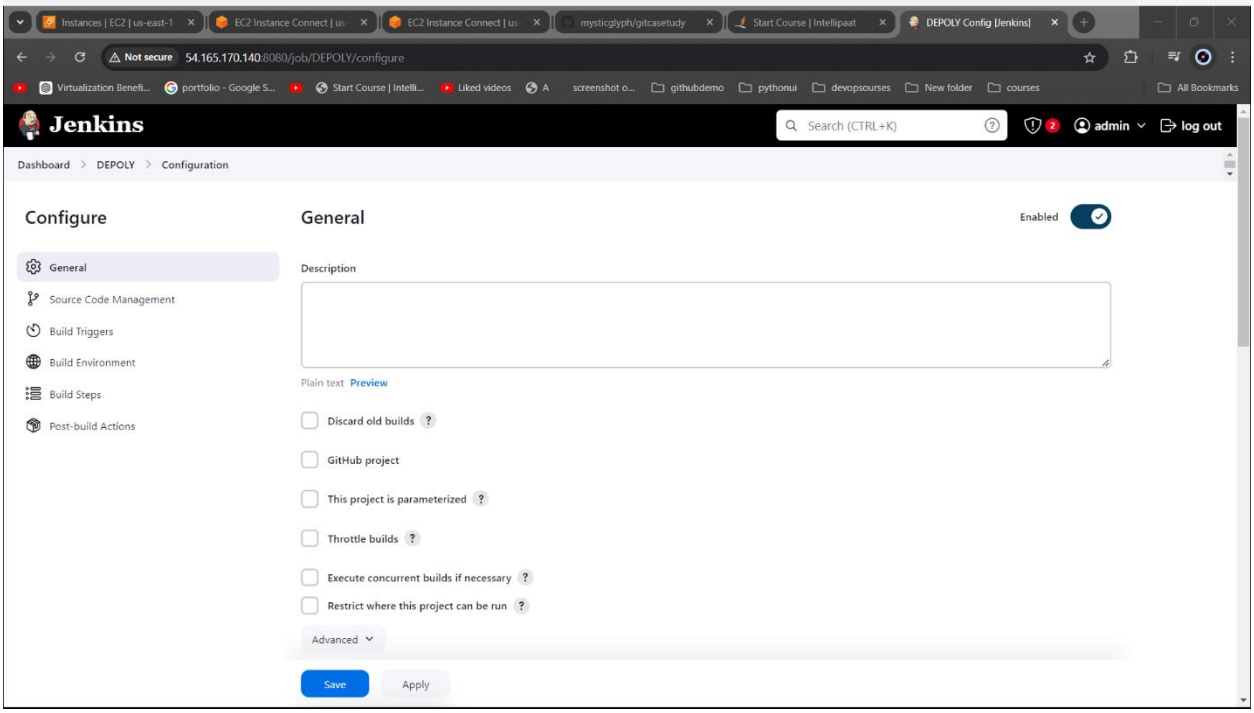


Creating container:

Name of the container and type of the container:



Job is created.

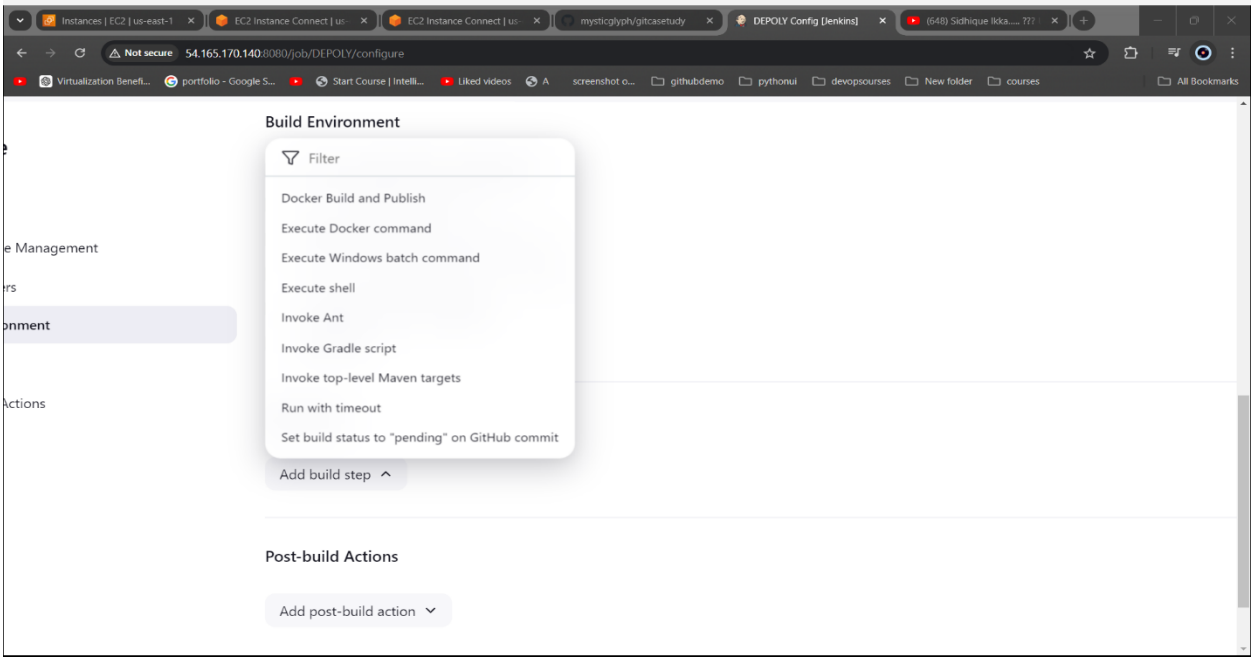


Configuring the job:

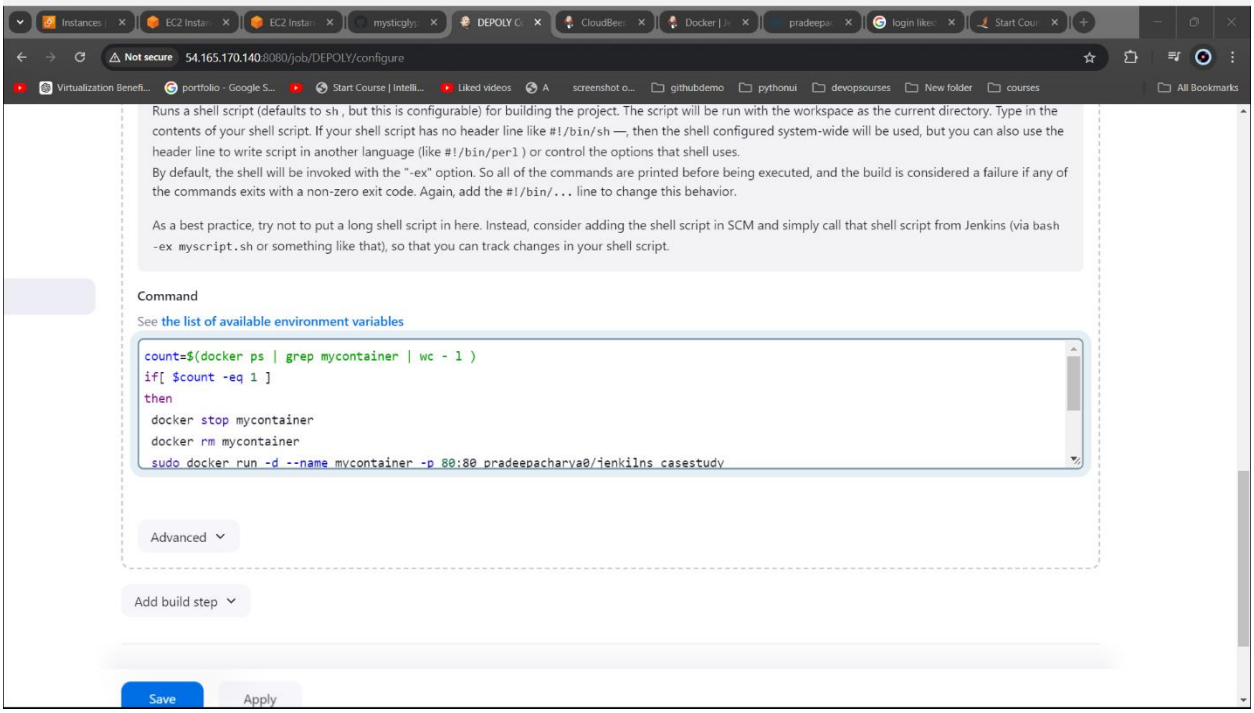
We are using a pulg in for creating a docker whis is previously downloaded.

➔ **Pug in: docker compose build step**

So . in build step:



We will use execute shell:



count=$(docker ps | grep mycontainer | wc - l)

if[$count -eq 1]

then

docker stop mycontainer

docker rm mycontainer

sudo docker run -d --name mycontainer -p 80:80 pradeepacharya0/jenkilns_casestudy

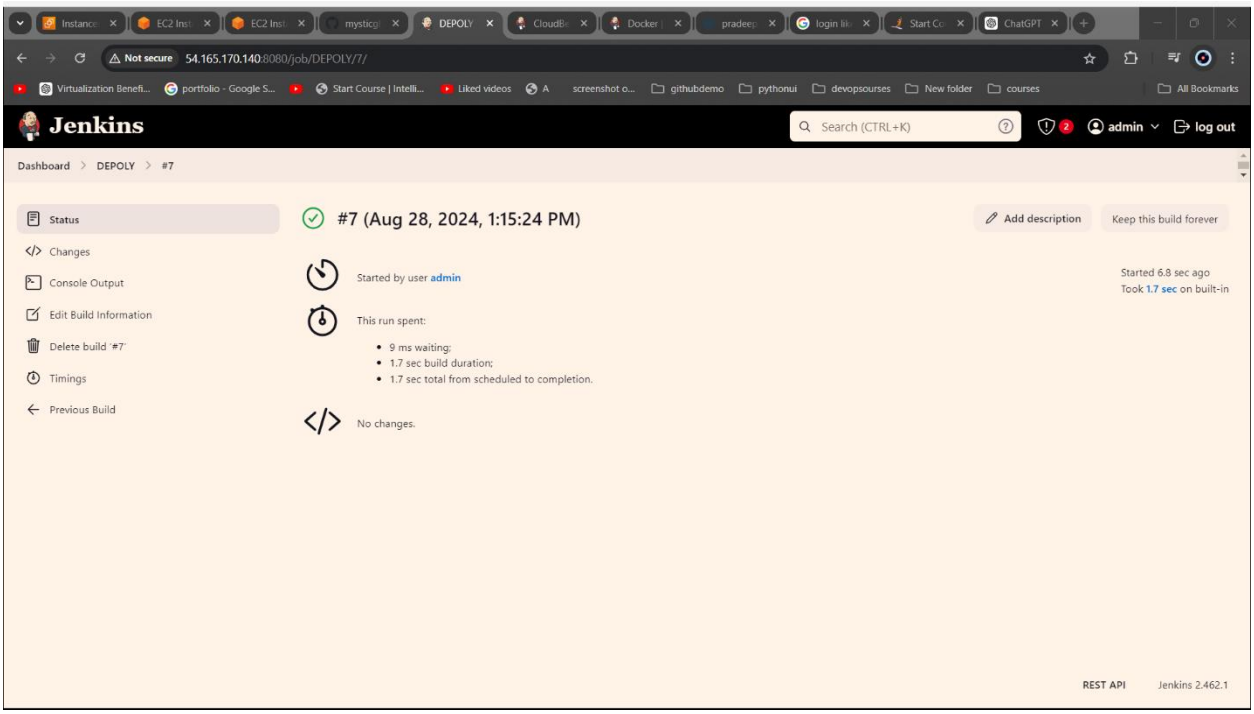
else

sudo docker run -d --name mycontainer -p 80:80 pradeepacharya0/jenkilns_casestudy

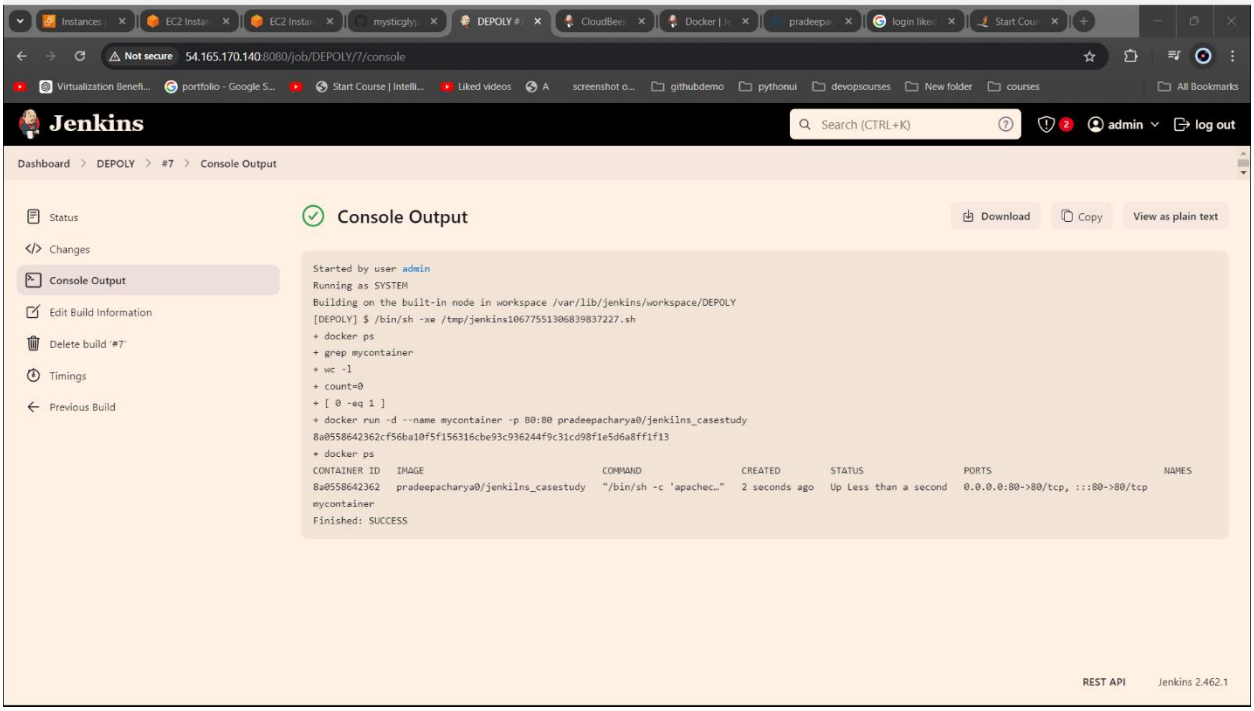
fi

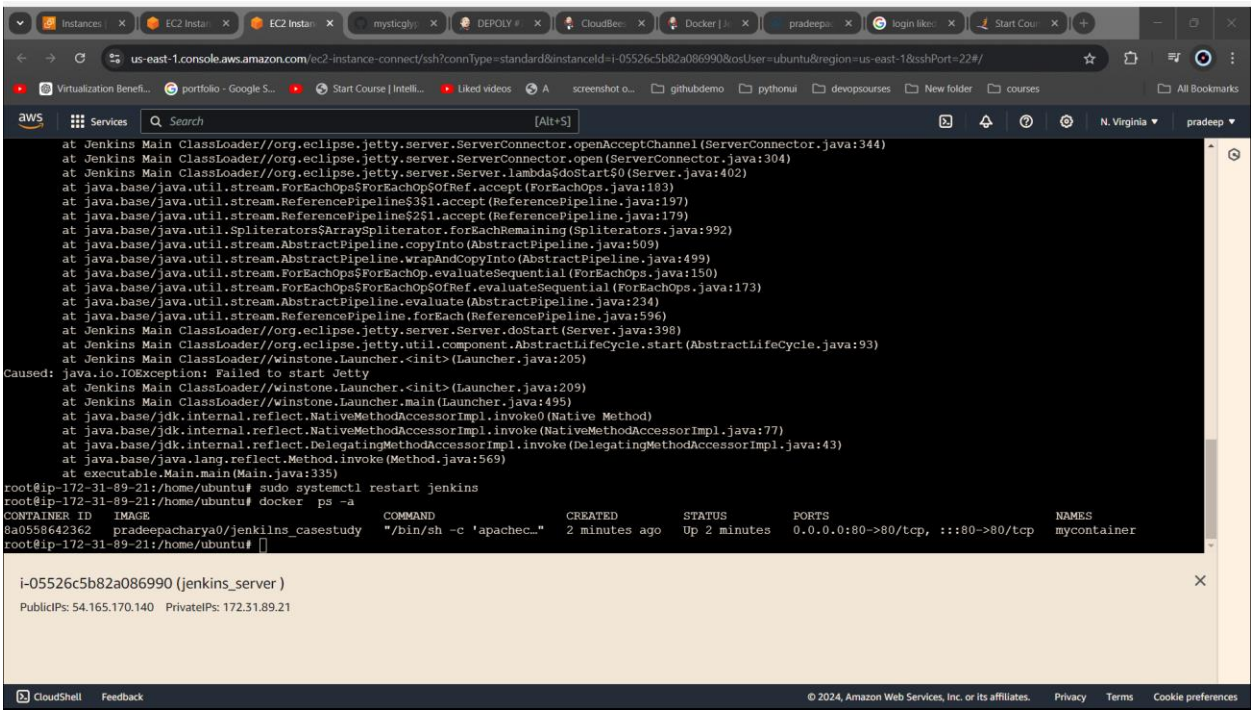
docker ps

this will be the cammnda for creating a container.



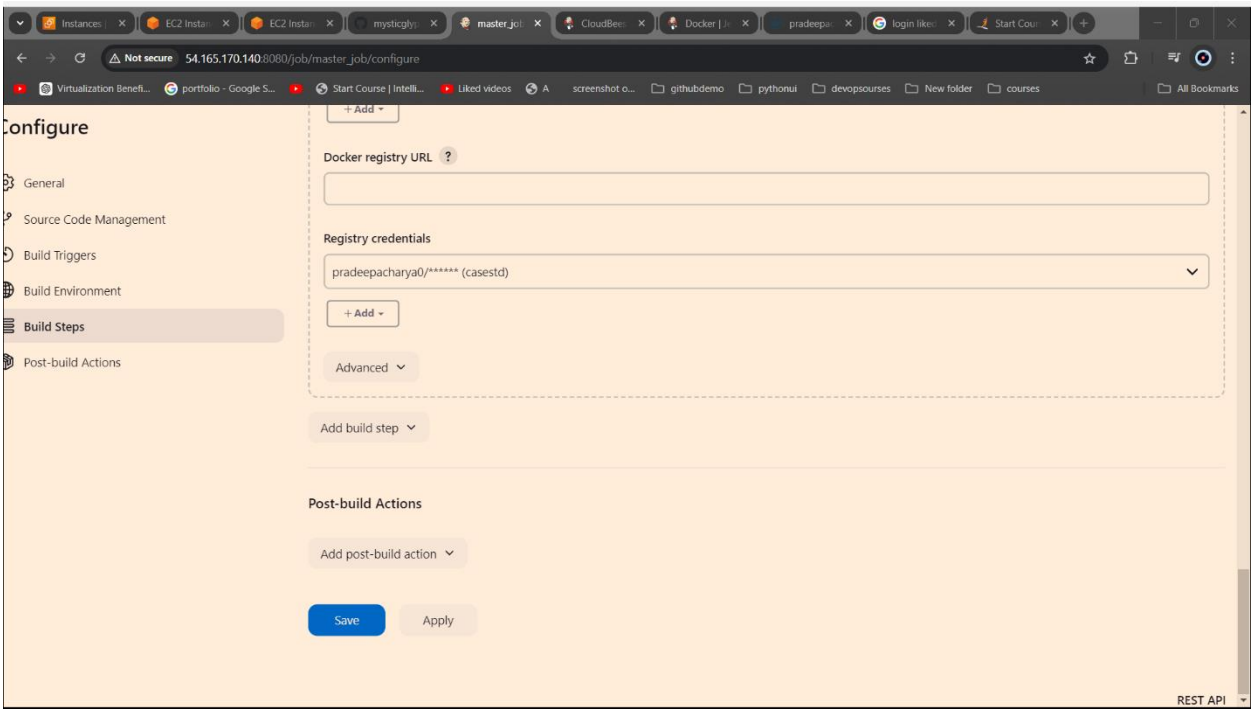
Job executed successfully.



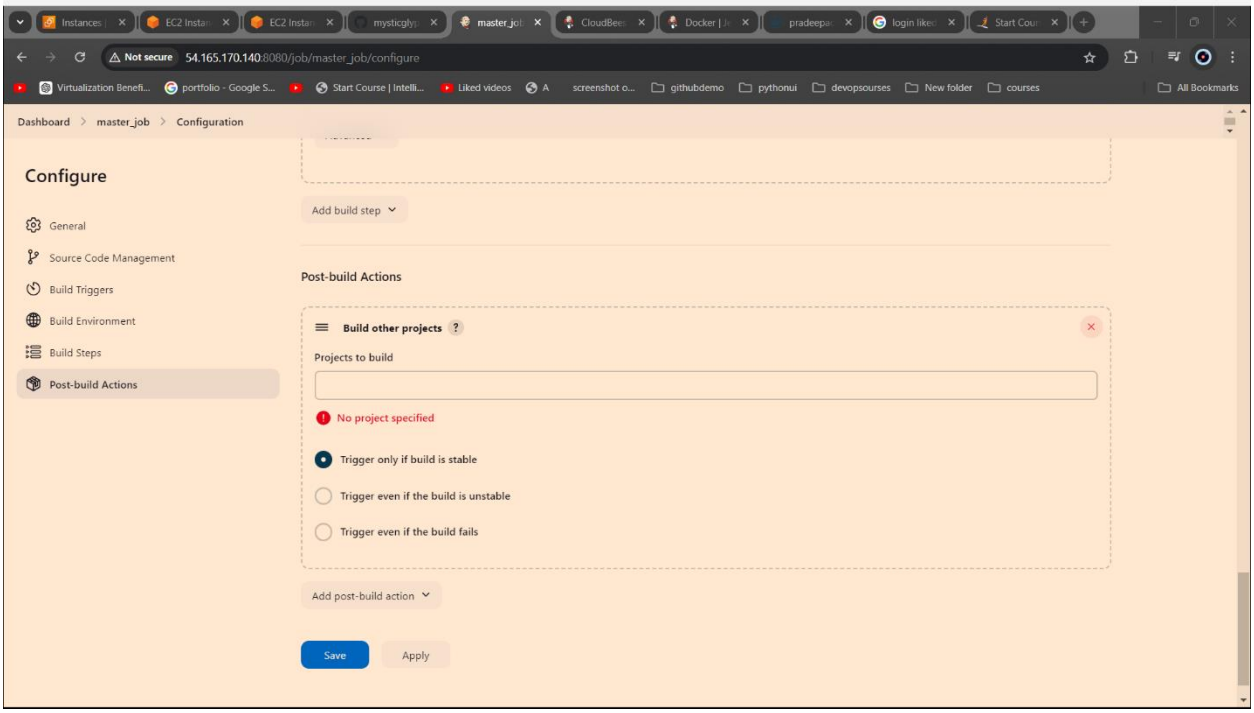


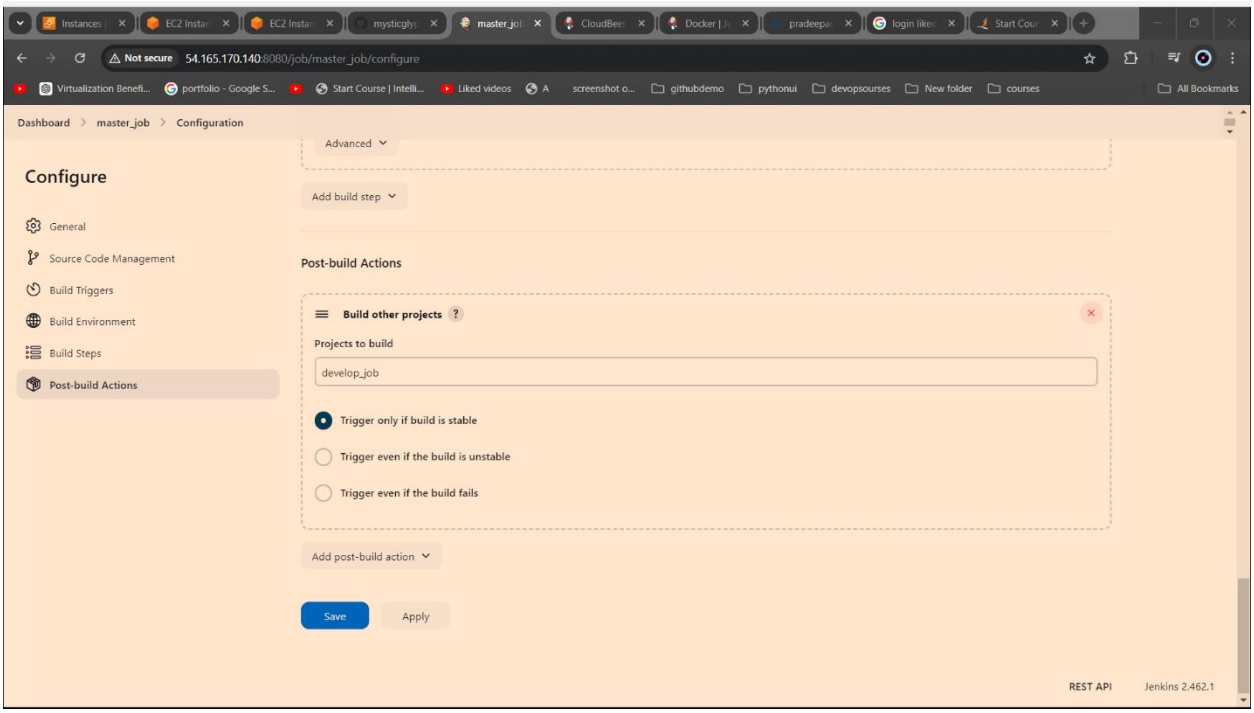
We can see that in server container is created.

Post-build Actions:



Build other project:





Save.

case study is completed

Conclusion

The Jenkins CI/CD pipeline implementation for the given case study successfully integrates essential DevOps tools and practices to streamline software development and deployment. By setting up automated Git workflows, the pipeline ensures code quality and consistency through triggered builds on every commit to the master or develop branches. Additionally, the containerization approach, leveraging an Ubuntu environment with Apache, facilitates efficient code building and deployment. This solution not only enhances operational efficiency but also supports the organization's goal of delivering reliable and scalable software products.