Deploying Java Application using Jenkins CI/CD Pipeline

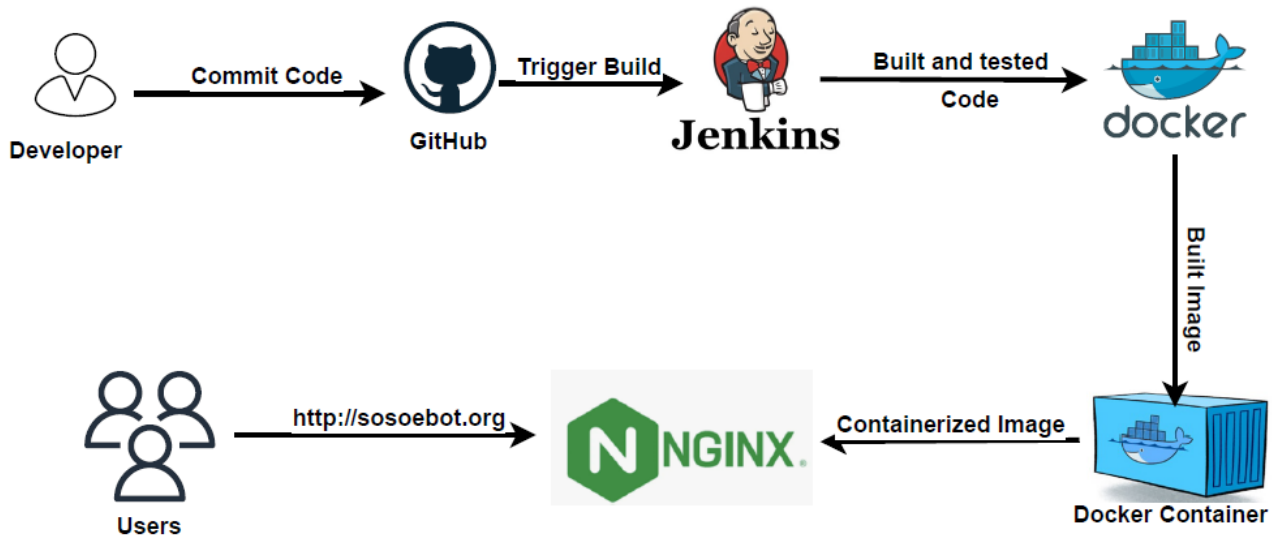
In this comprehensive guide, we dive into the world of building, testing, and deploying Java applications using CI/CD pipelines. Whether you are a beginner or an experienced developer looking to streamline your workflow, this series has something for you.

In these tutorials, we will cover:

- Setting up a Java development environment
- Integrating version control with Git
- Automating builds using Maven
- Writing effective unit tests with JUnit
- Configuring CI/CD pipelines with tools like Jenkins
- Pipeline Scripting
- Deploying your Java application to various environments (development)

By the end of this tutorial, you'll have a solid understanding of how to leverage CI/CD to automate repetitive tasks, ensure code quality, and deliver software more reliably. Join us on this journey to mastering CI/CD for Java applications.

Architecture



The developer writes the code and pushes it to **GitHub**. When the code is push, it will trigger **Jenkins**, the CI/CD tool that will pick up the code from GitHub and build it using **Maven**. Maven will build the application and create a file in the format war or jar file.

Maven is a build automation tool primarily used for Java projects. It simplifies the process of building and managing Java projects by providing a uniform build system, project management, and dependency management.

After building the code, it will be sent to Docker. Docker will build a docker image and containerize it. Then containerized image will be deploy to the EC2 instance, that is our server. From this server the users can access the application through the internet.

TOOLS

This project will be executed using these tools:

- Maven: Build tool
- GitHub: SCM
- Jenkins: CI/CD
- Docker: Container
- Nginx: Webserver
- SSL: Cerbot

IMPLEMENTATION

We will now implement the above architecture following these steps:

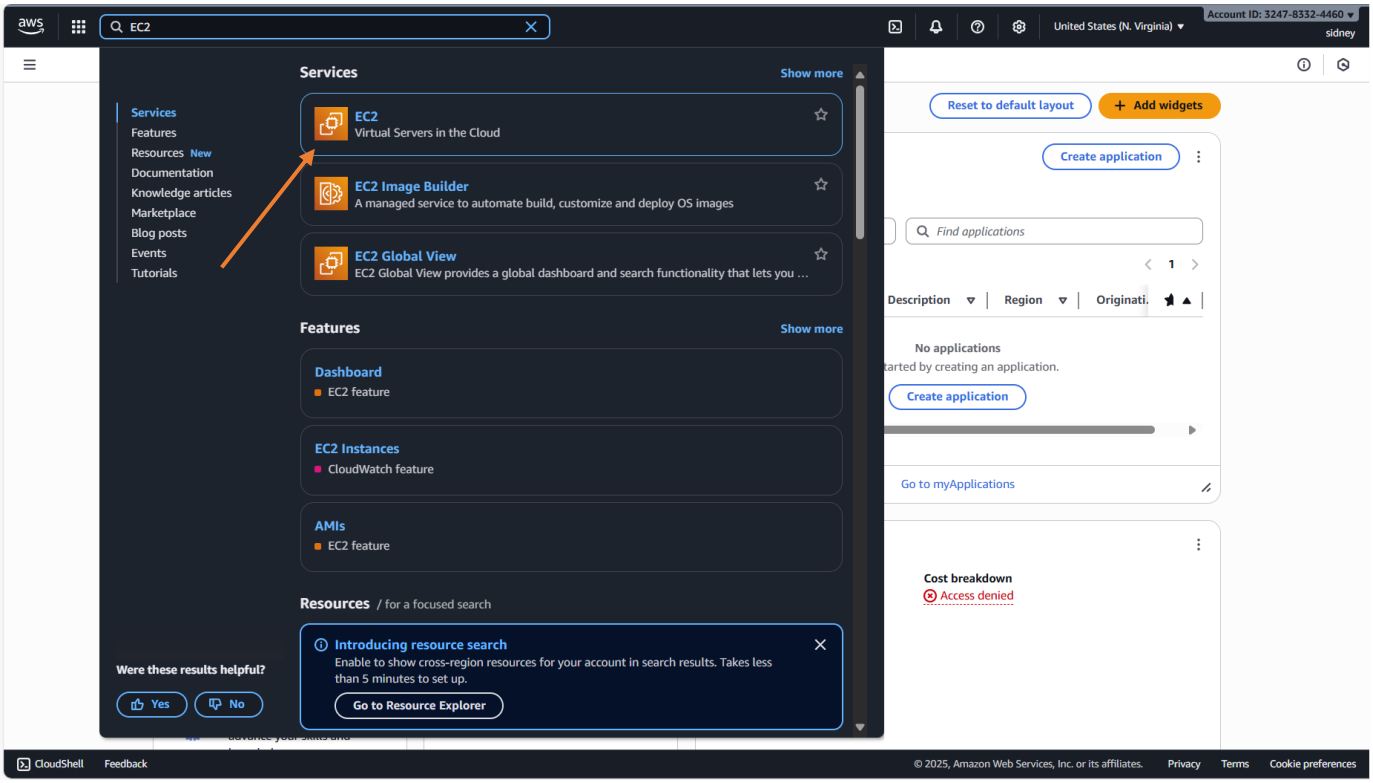
1. **Launch an ubuntu EC2 instance that will serve as our server**
    - Name: Jenkins-Server
    - Enable Port 22 for SSH
    - Enable Port 43 for HTTPS
    - Enable Port 80 for HTTP
    - Add Port 8080 for Jenkins
    - Add Port 9000 for Reverse Proxy
    - SSH Connect to the server
2. **Install JDK and Maven on EC2 server**
3. **Install Docker on EC2 server**
4. **Install Jenkins on EC2 server**
5. **Setup Docker in Jenkins**
6. **Access Jenkins on Browser**
7. **Create Pipeline on Jenkins**
    - Test “Hello World” pipe line
8. **Install and Configure JDK on Jenkins**
    - Install Java JDK plugins on Jenkins
    - Install Java JDK as a tool on Jenkins
9. **Install and Configure Maven on Jenkins**
    - Install Maven plugins on Jenkins
    - Install Maven as a tool on Jenkins
10. **Install and Configure Docker on Jenkins**
    - Install Docker plugins on Jenkins
    - Install Docker as a tool on Jenkins
11. **Add Stages to the pipeline and build:**
    - Git Checkout Stage
    - Code Compile Stage
    - Maven Build Application
    - Docker Build & Push
    - Deploy
12. **Install and configure Nginx and Certbot on EC2 server for Reverse Proxy**

STEP 1: Launch an Ubuntu EC2 instance that will serve as a server

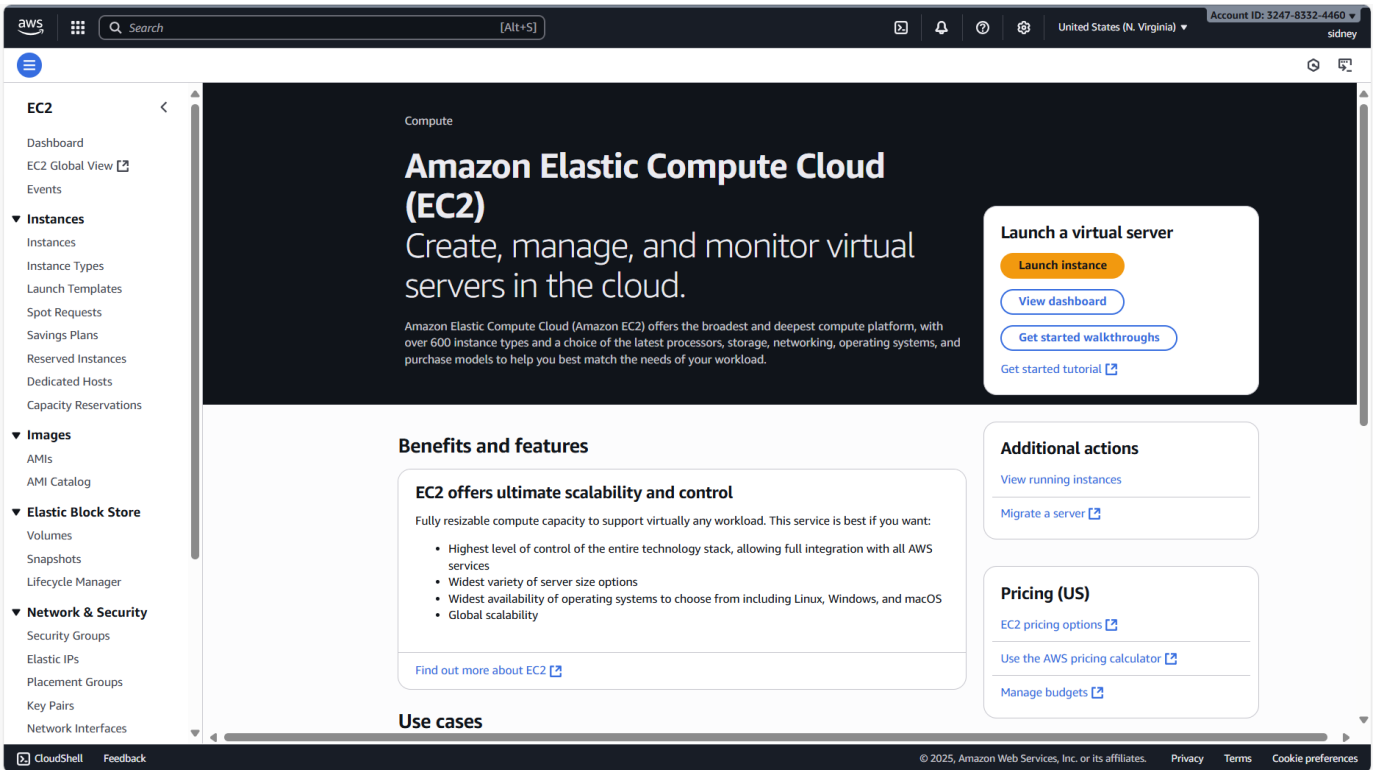
We will launch an EC2 instance and later on SSH connect to it to install our dependencies.

Part 1: Launch an Ubuntu EC2 instance

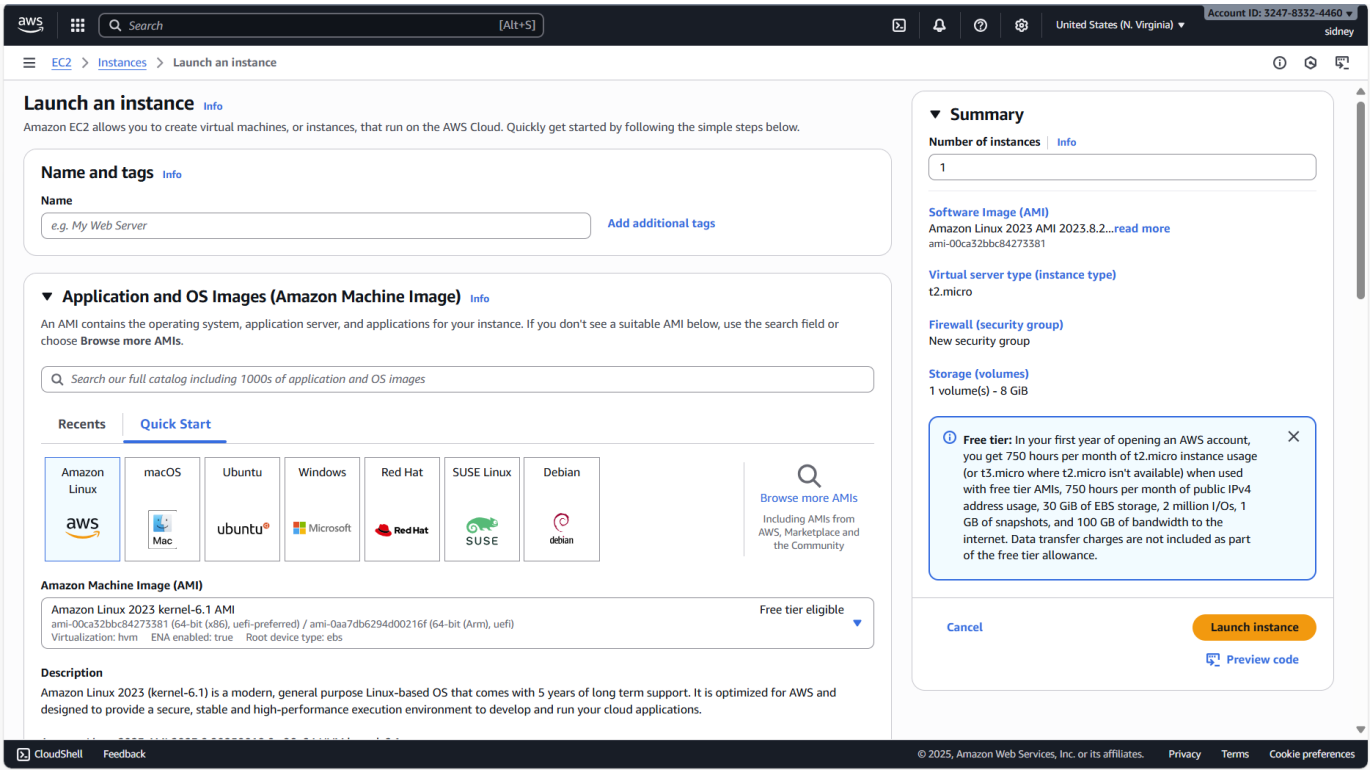
Go to AWS Management console and search for "**EC2**" Instance.



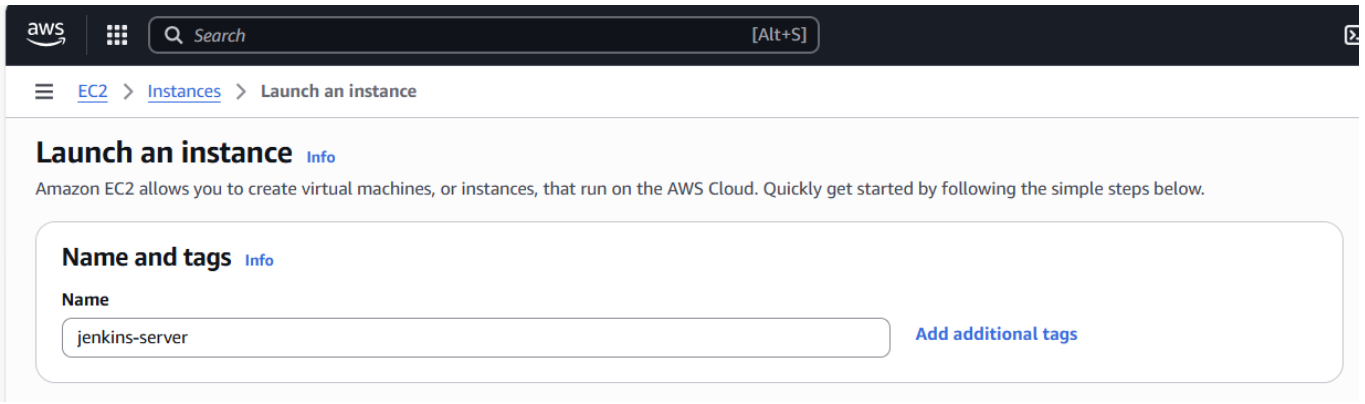
Click on "**EC2**"



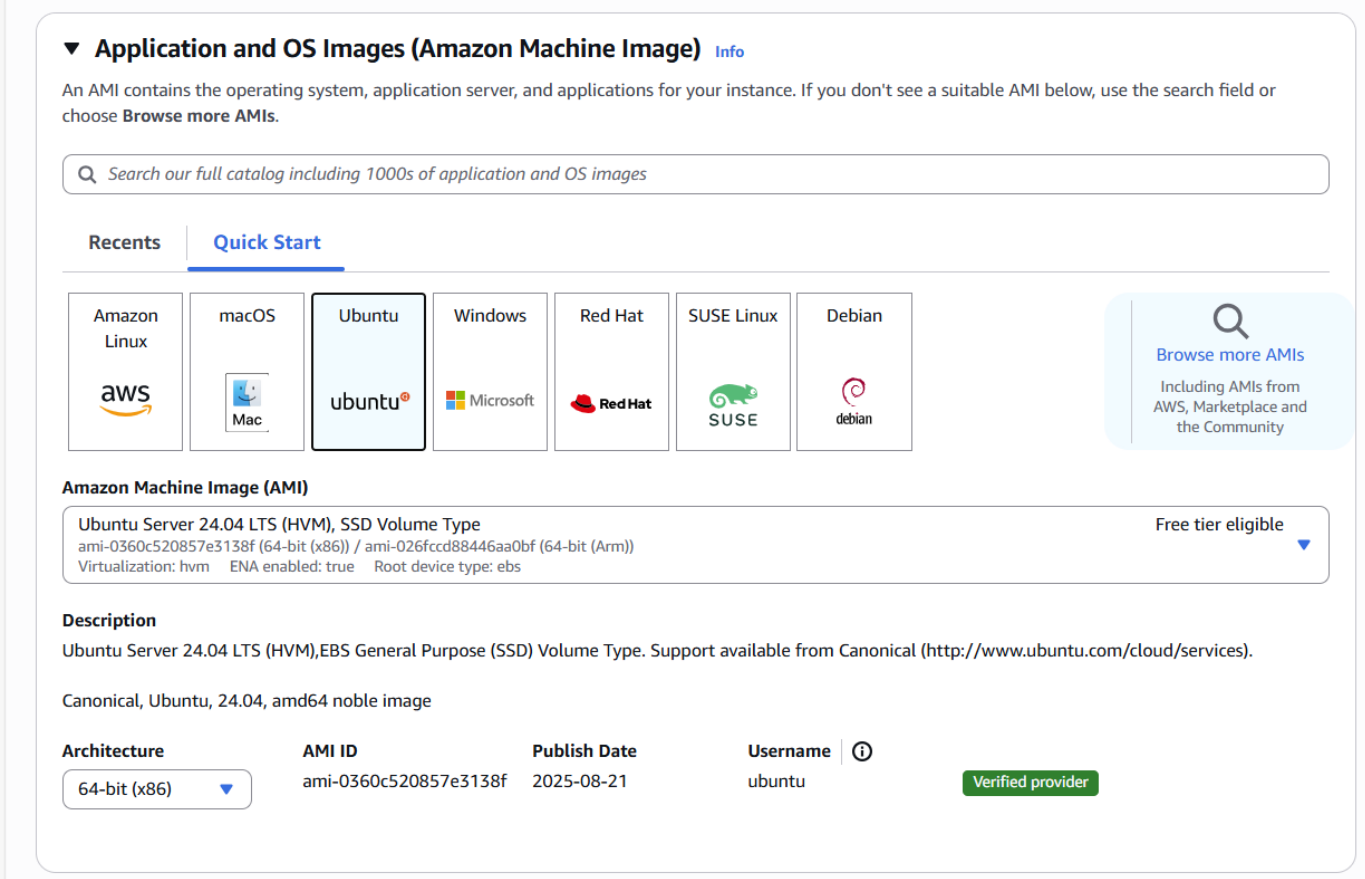
Click on “**Launch Instance**”



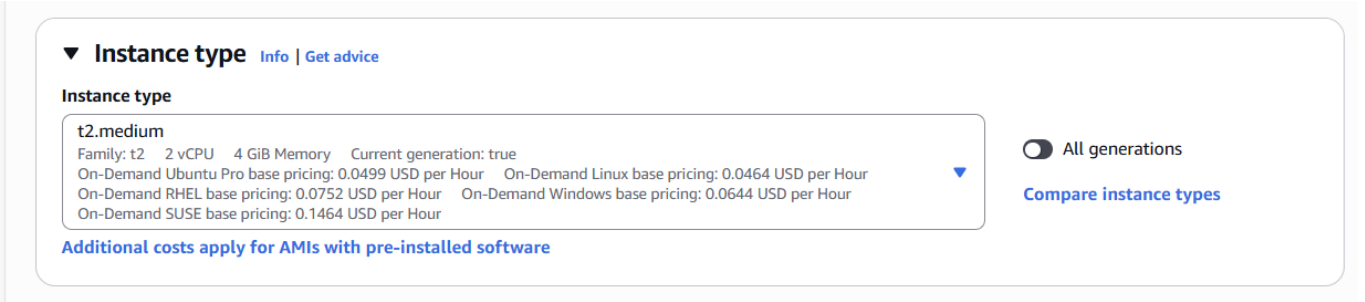
We will call the instance “**jenkins-server**”



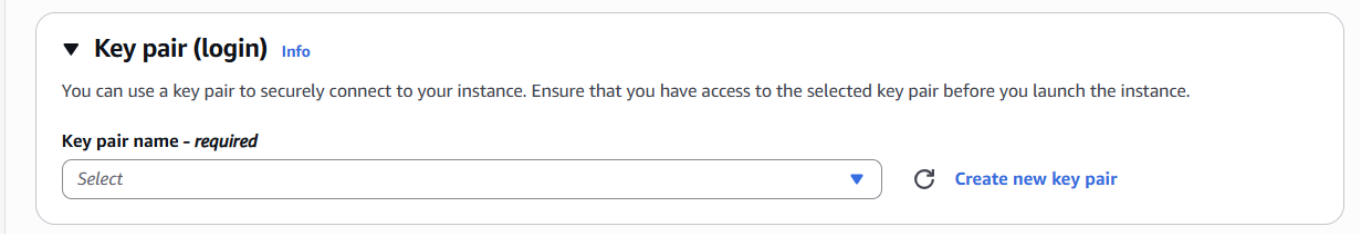
Scroll down to "**AMI**", select "**ubuntu**"



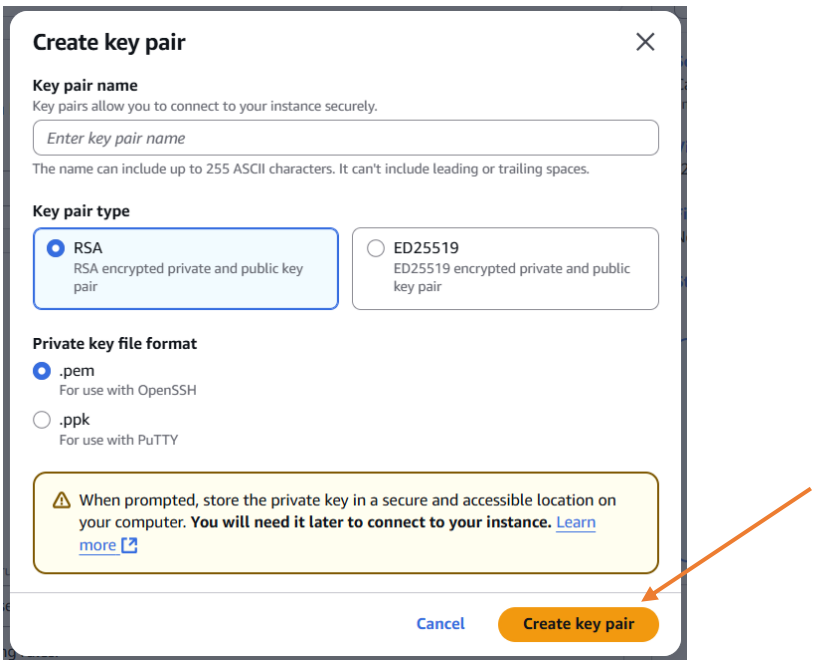
Scroll down to "**Instance Type**" and select "**t2.medium**"



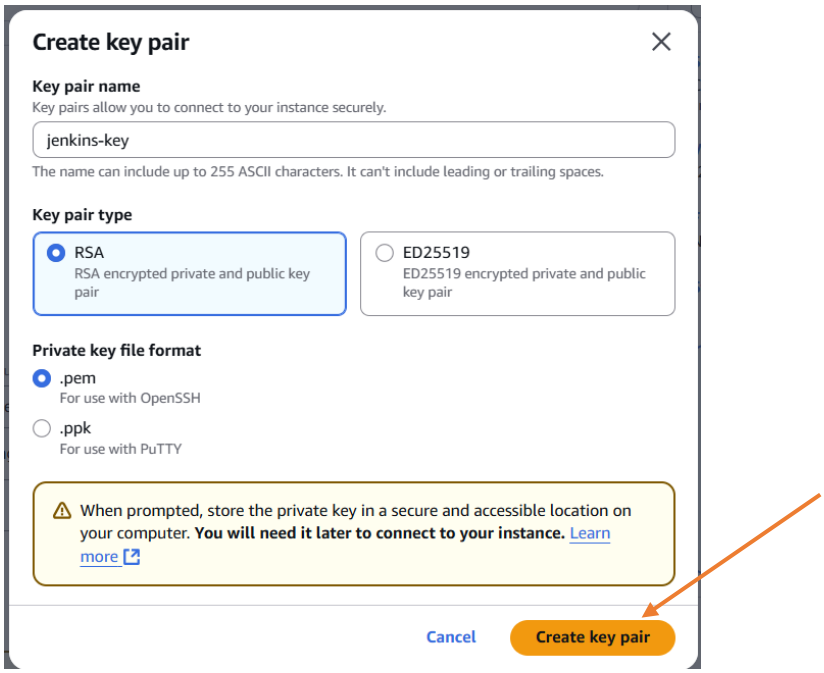
Scroll down to "**Key Pair**"



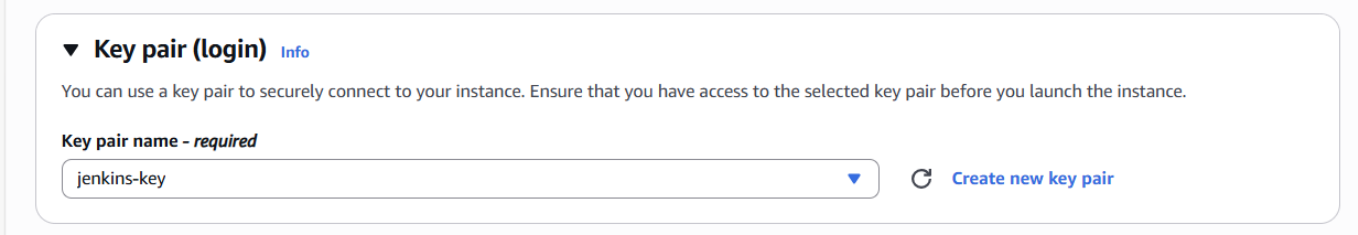
Click on "**Create new key pair**"



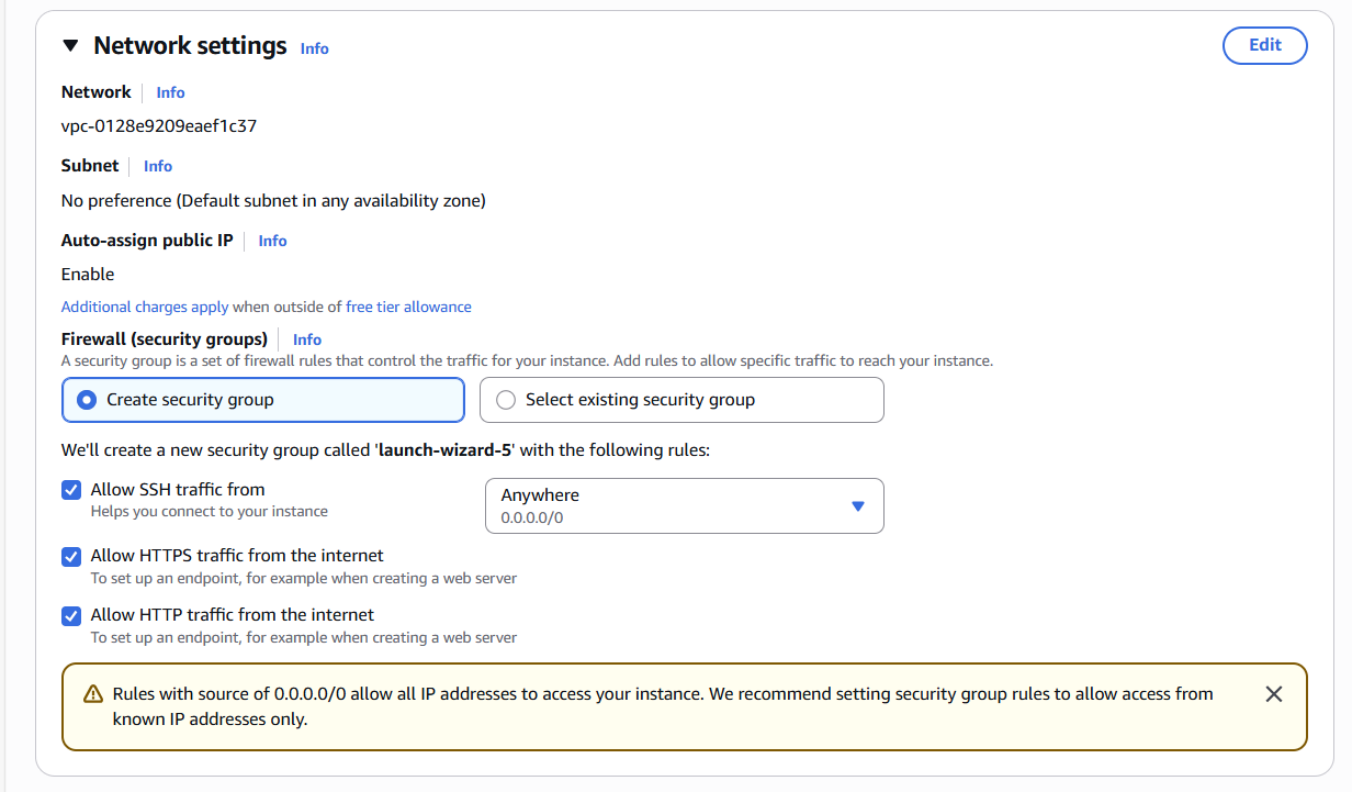
Give the key pair a name, I will call it "**jenkins-key**"



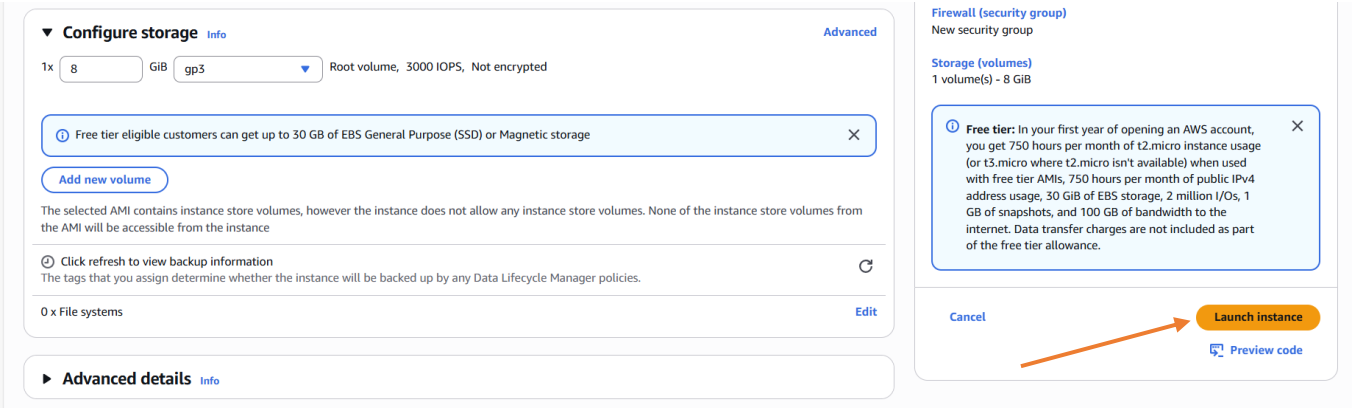
Click on "**Create key pair**"



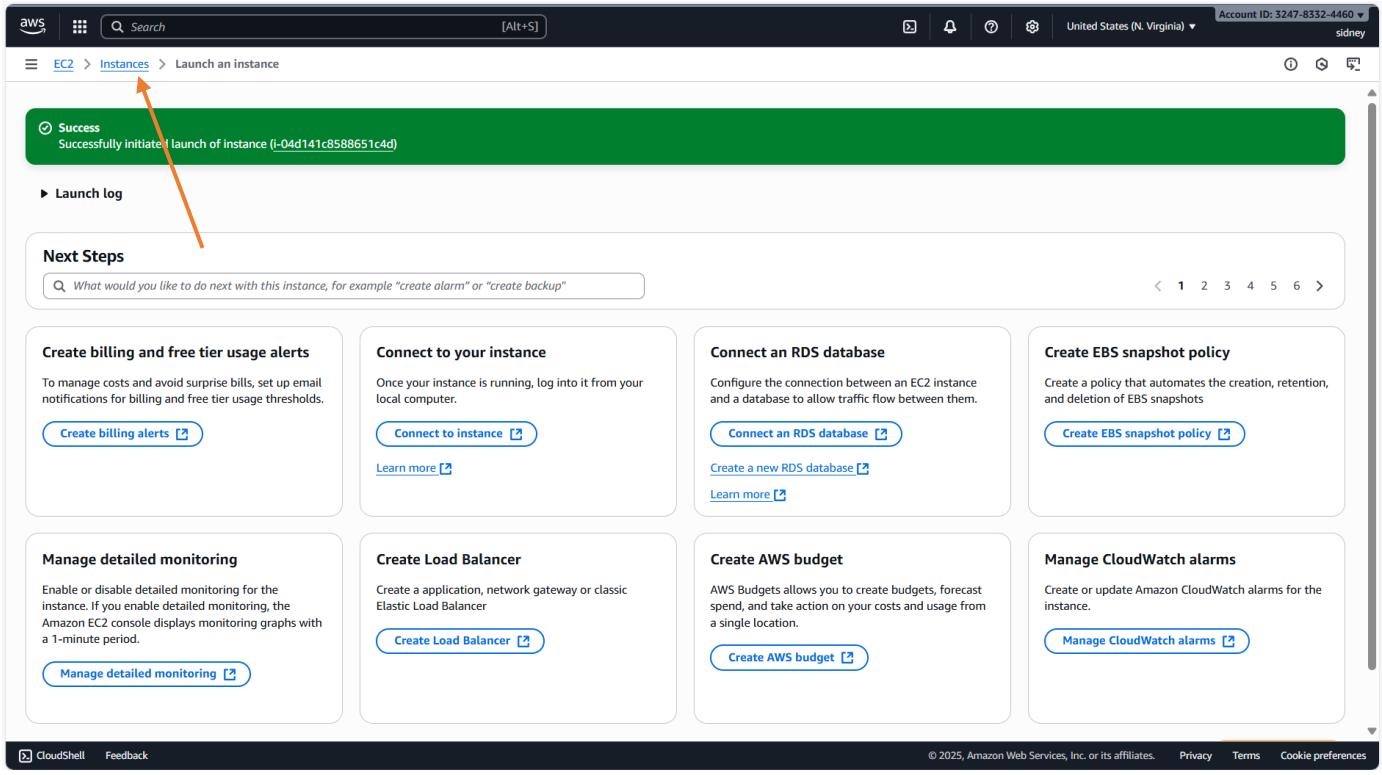
Part 2: Scroll down to "**Network Settings**". Select "**Allow HTTP traffic from the internet**", "**Allow HTTP traffic from the internet** and "**Allow SSH traffic from**".



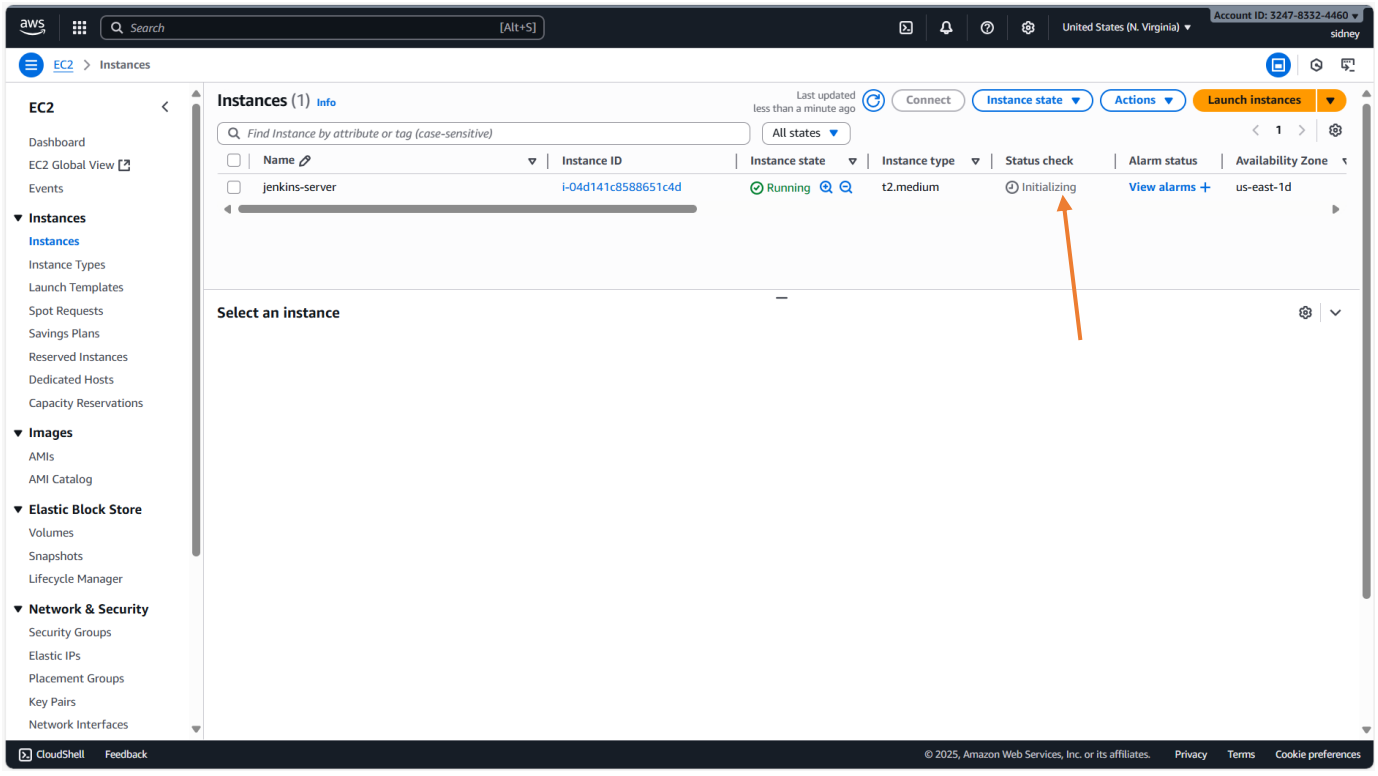
Scroll down to the end



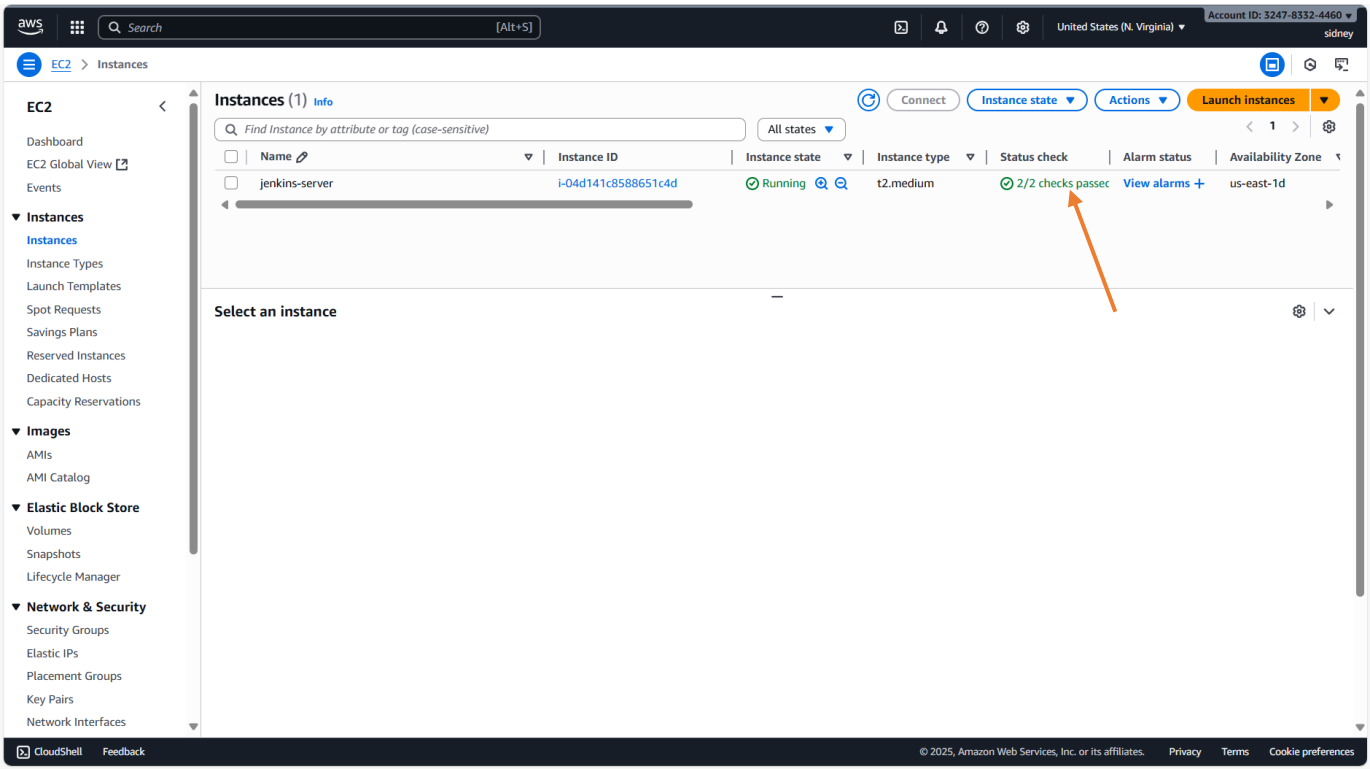
Click on "**Launch Instance**"



Click on "**Instances**"



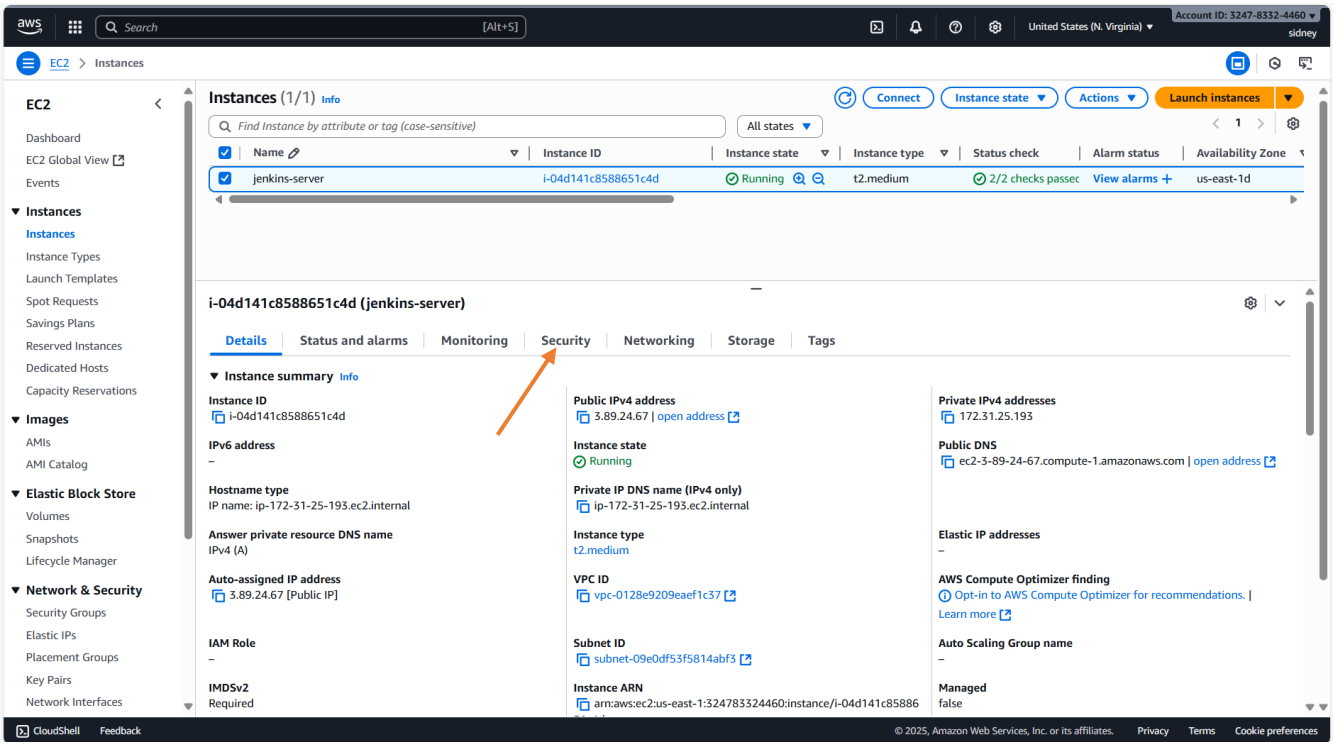
You can see that our just created instance is initializing. Wait for it to pass the "**2/2 check**"



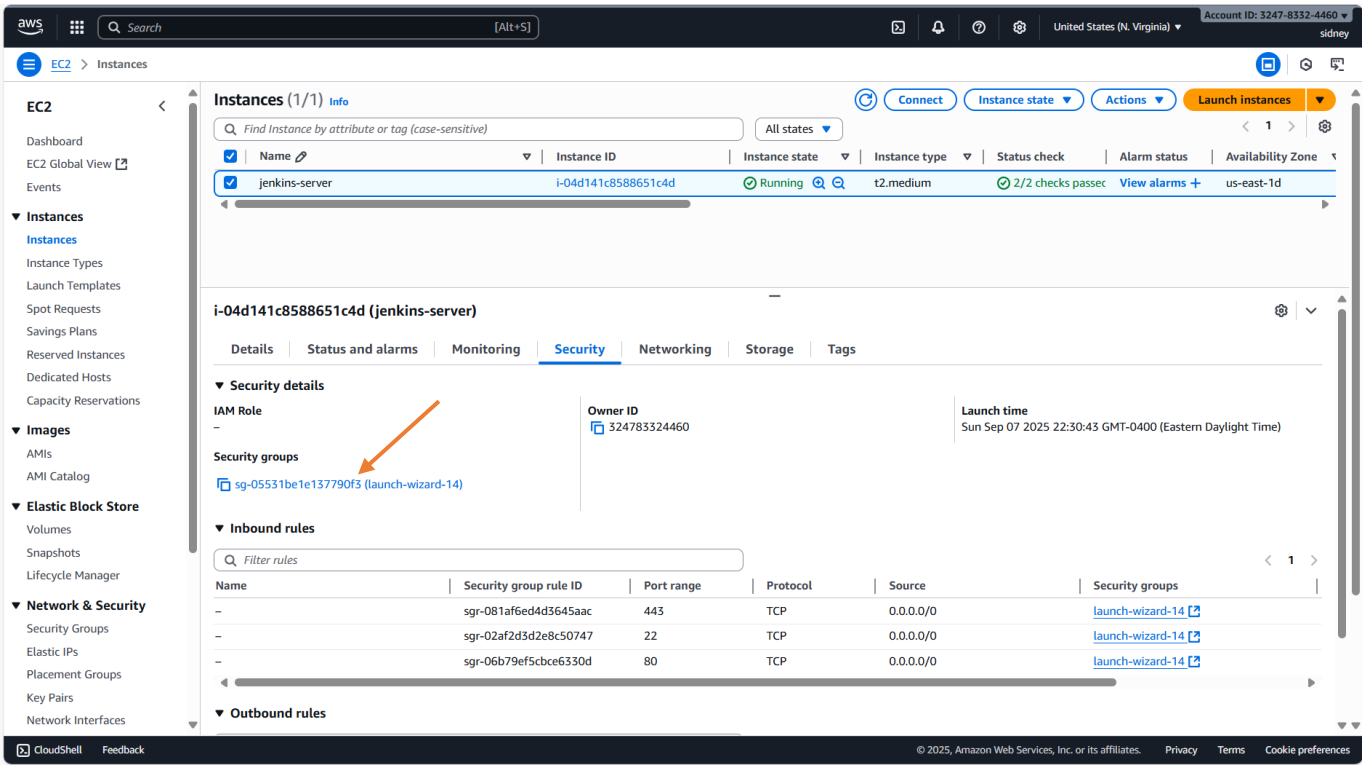
The instance has passed the "**2/2 check**"

Part 3: Add Port 8080 for Jenkins

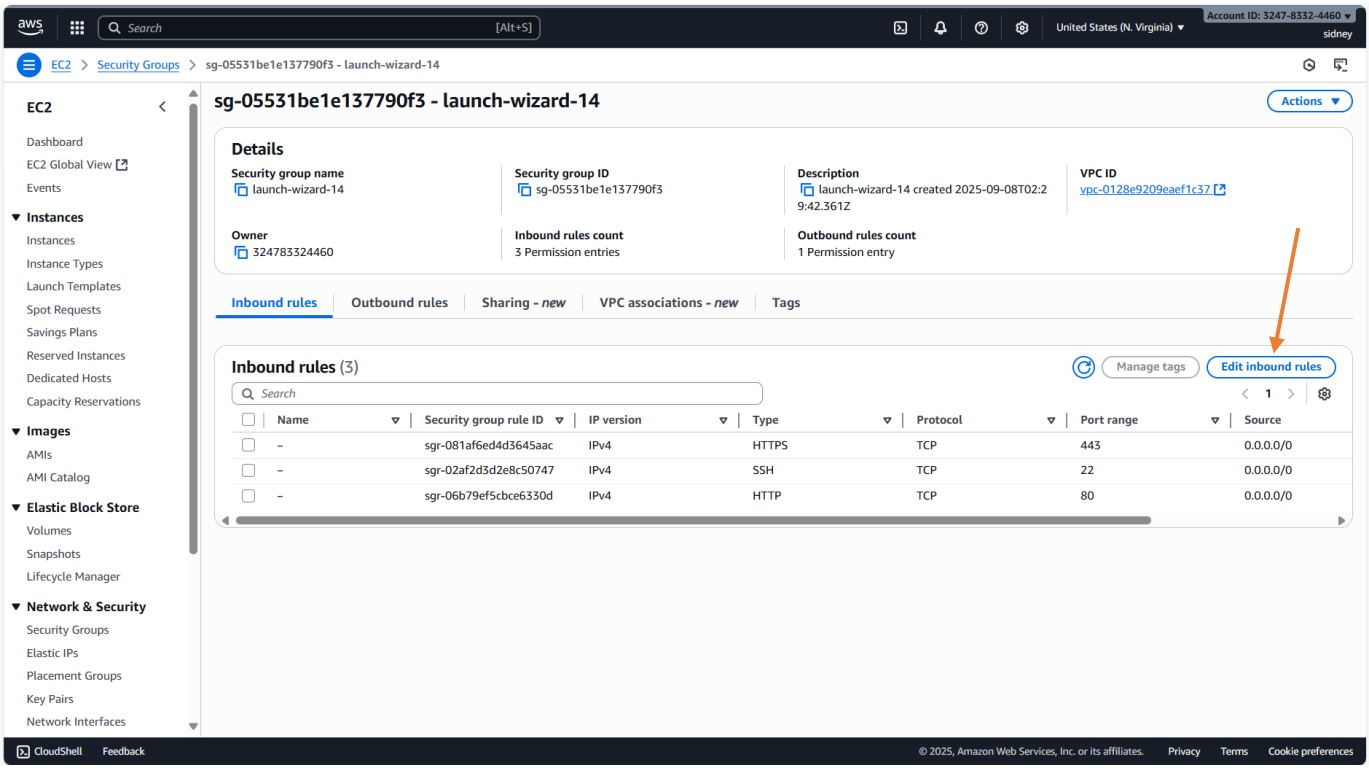
We will add ports 8080 for Jenkins as follows



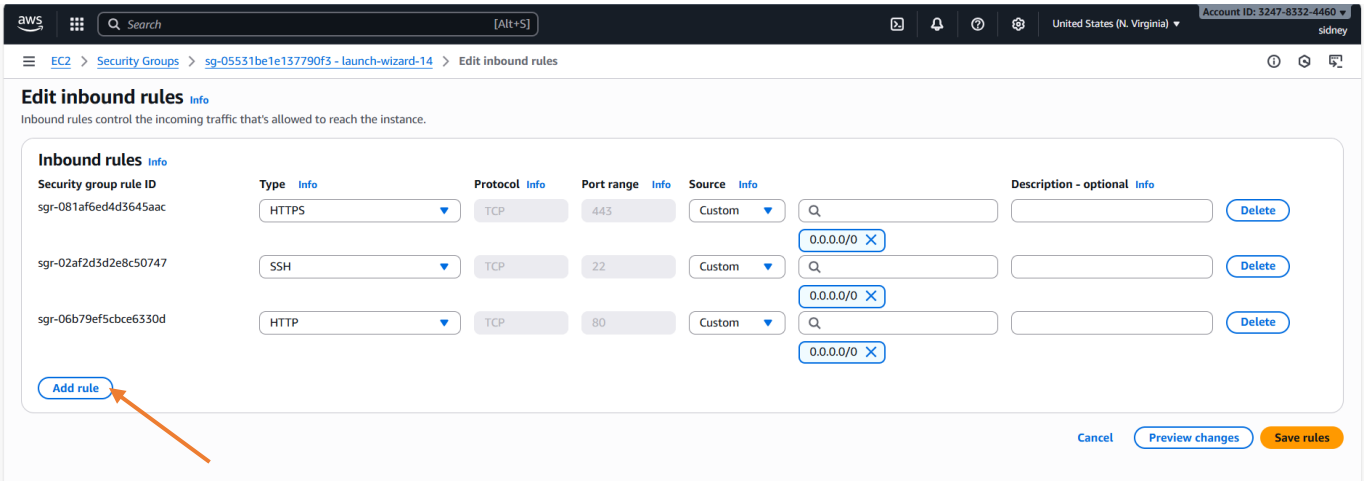
Click on the "**Security**" tab



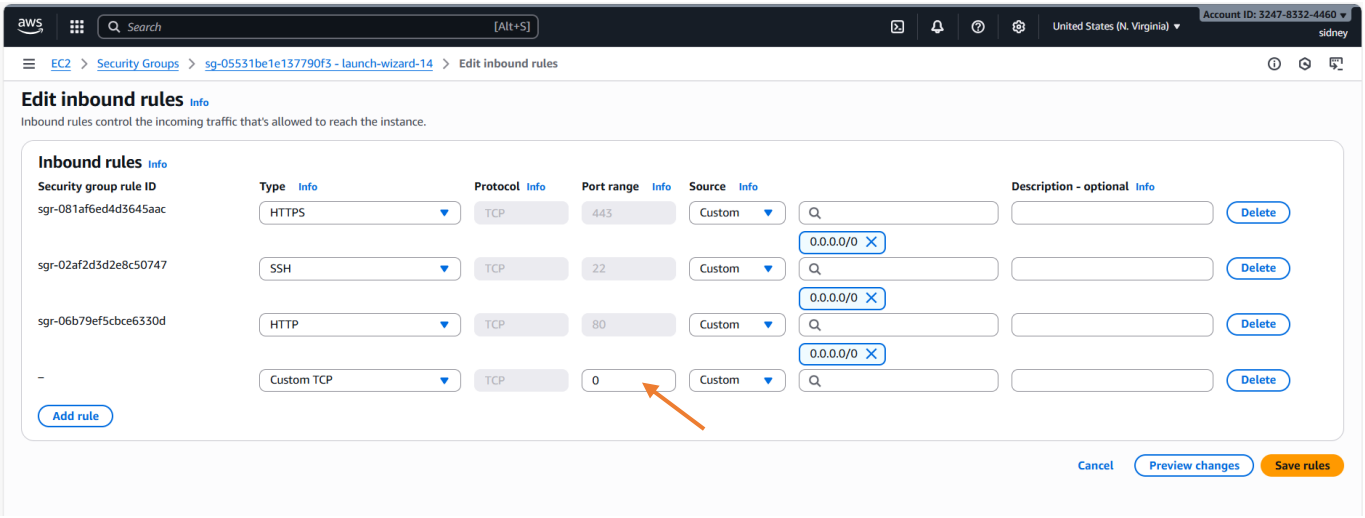
Click on “**Security Groups**” url



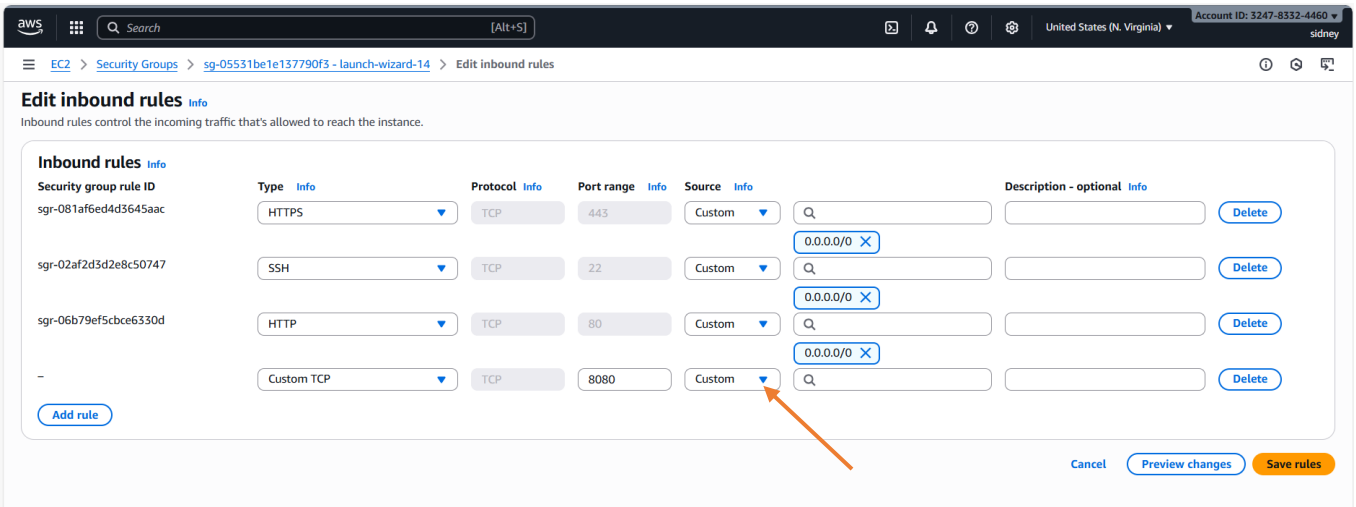
Click on “**Edit Inbound Rules**”



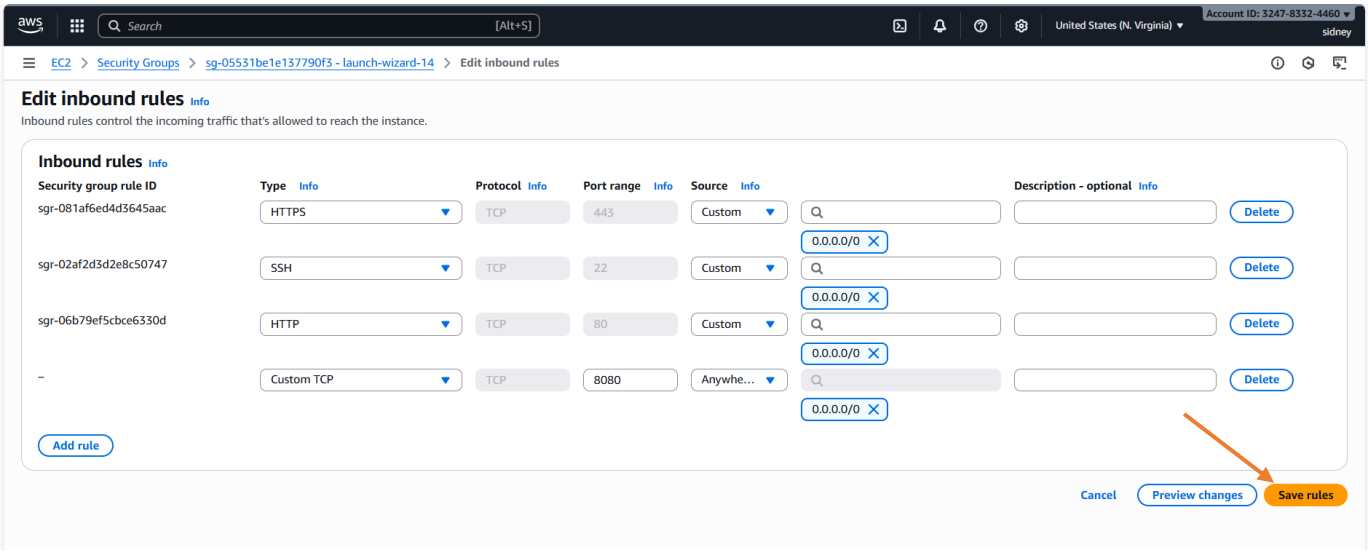
Click on “**Add Rule**”



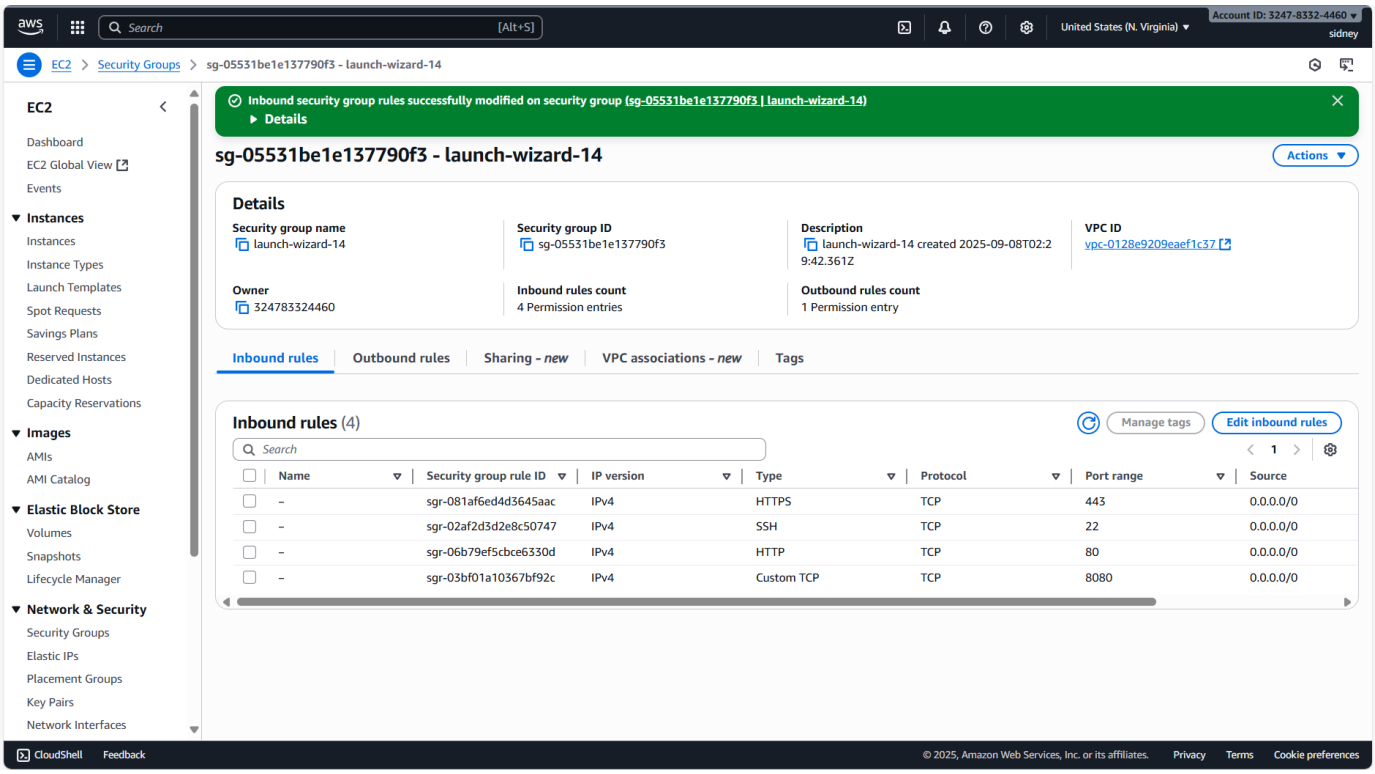
Add Port 8080



For source, click on the drop down and select “**Anywhere-IPv4**”

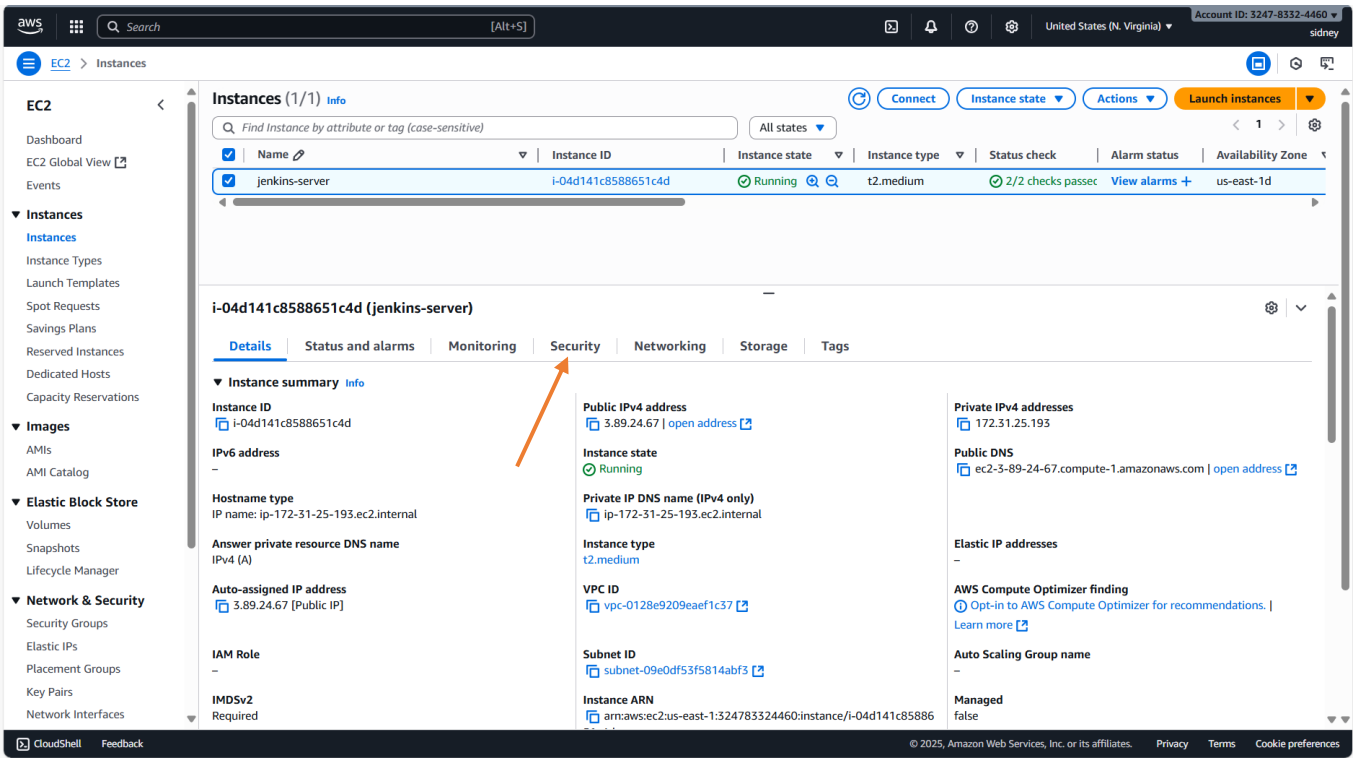


Then click on "**Save Rules**"

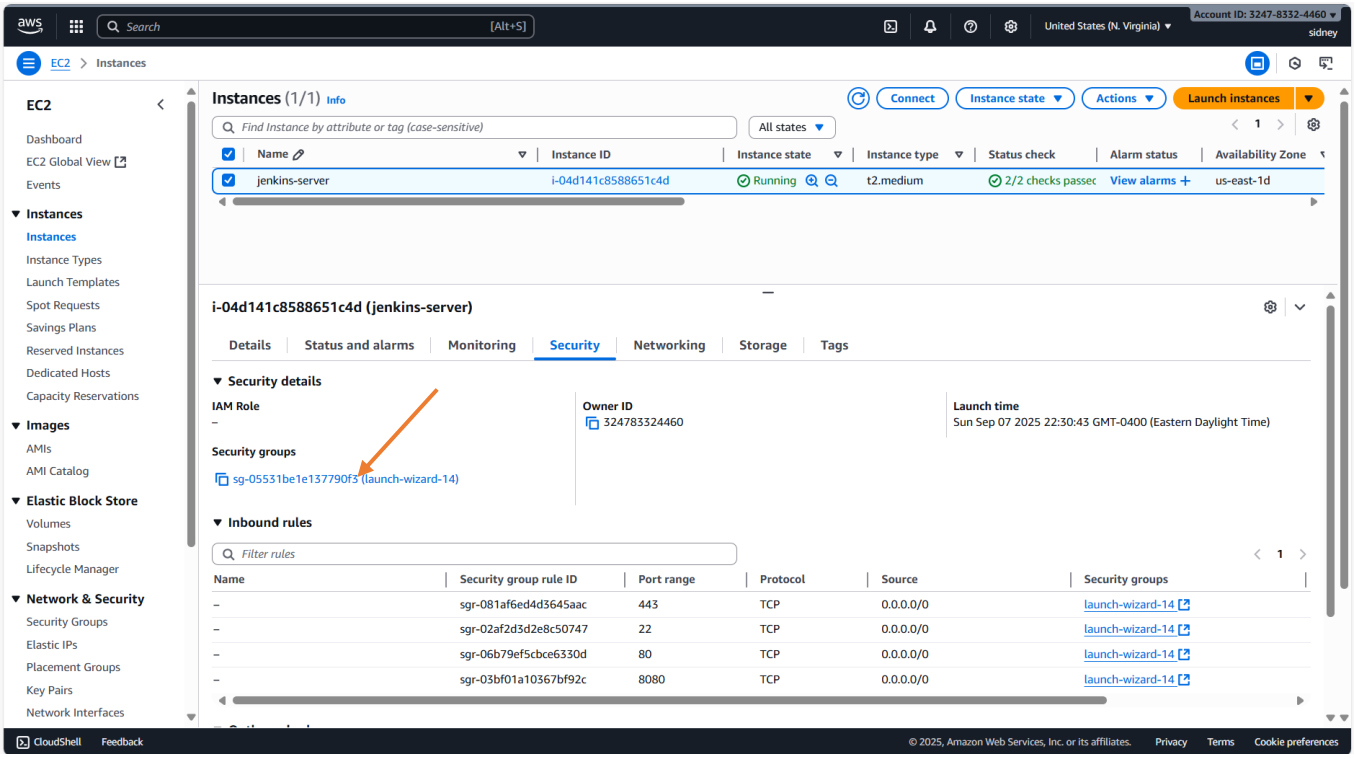


Part 4: Add Port 9000 for Reverse Proxy

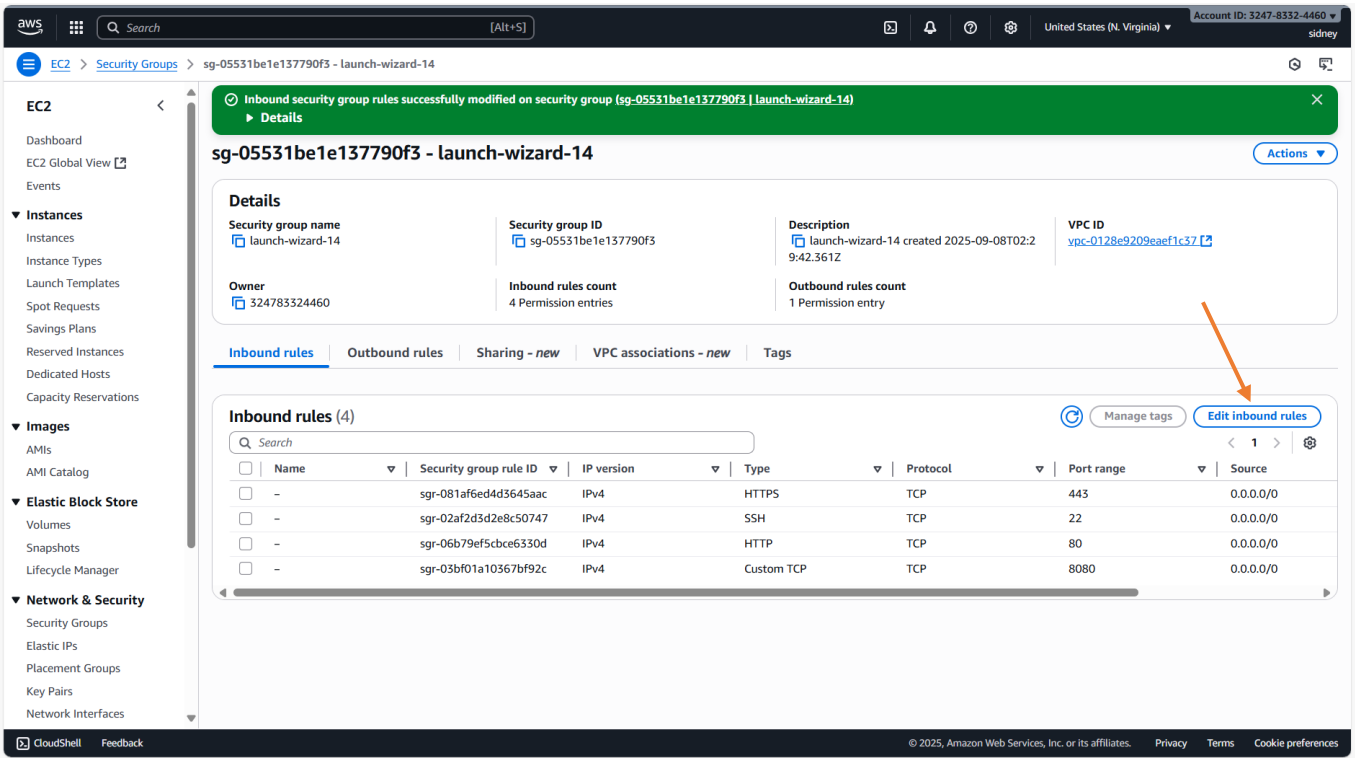
Now let us go to our EC2 server and add Port 9000



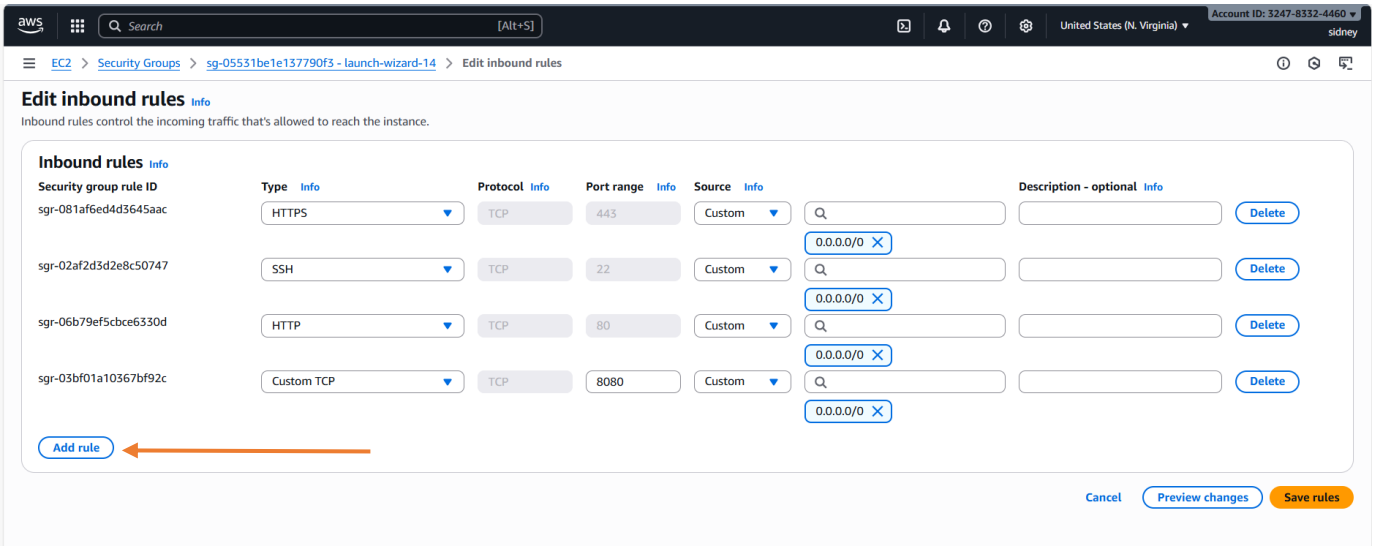
Click on “**Security Tab**”



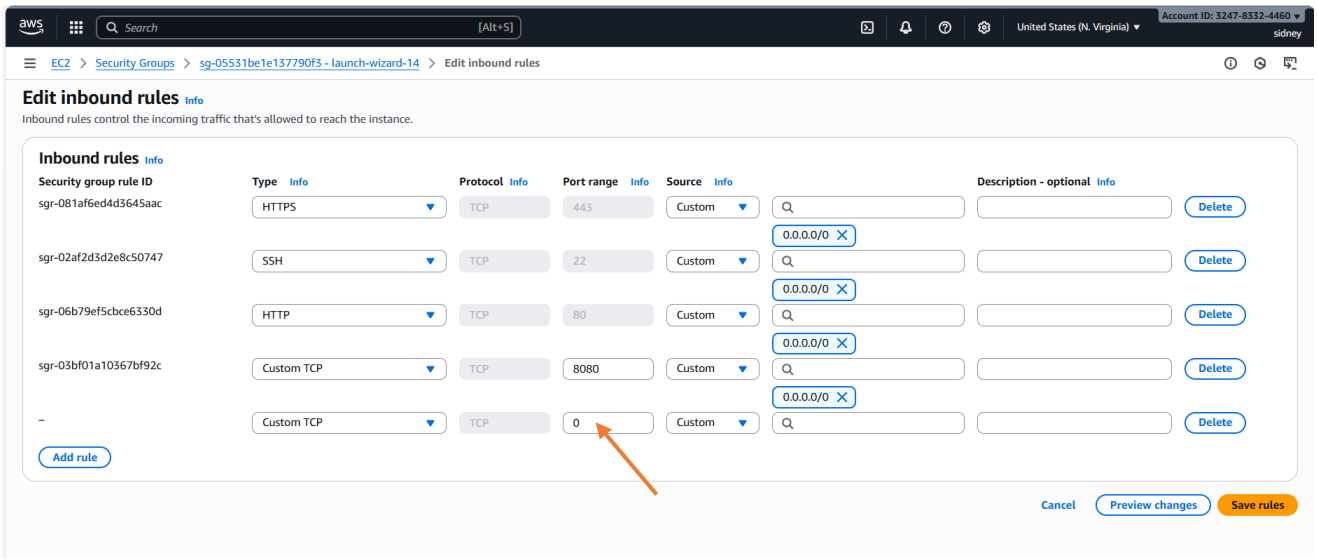
Click on the Security Group



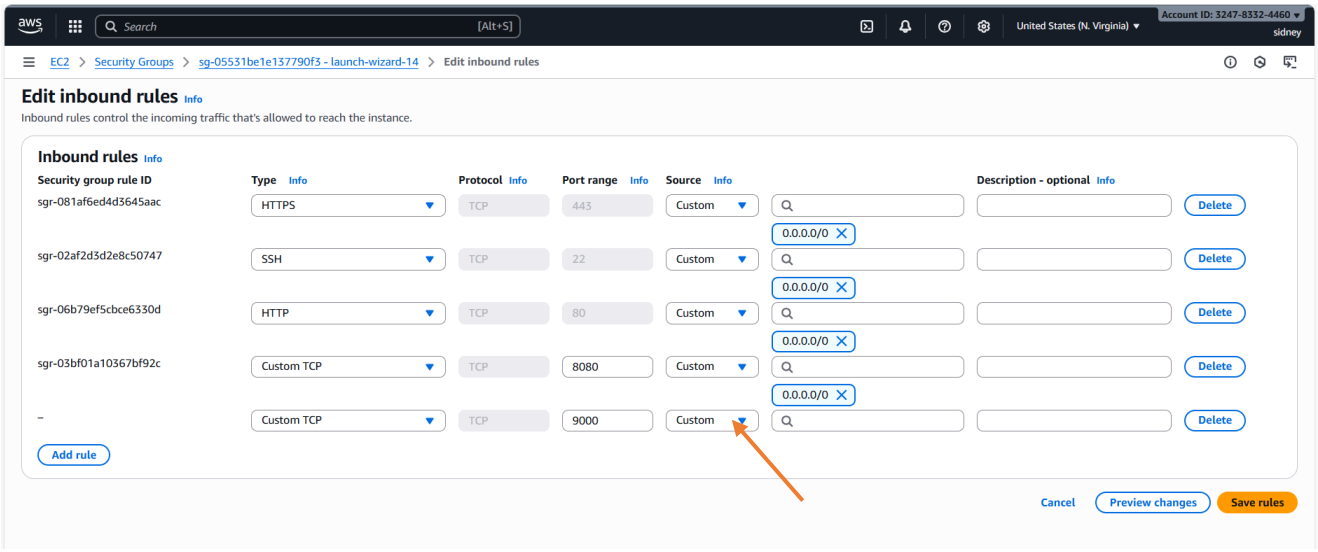
Click on “**Edit Inbound Rules**”



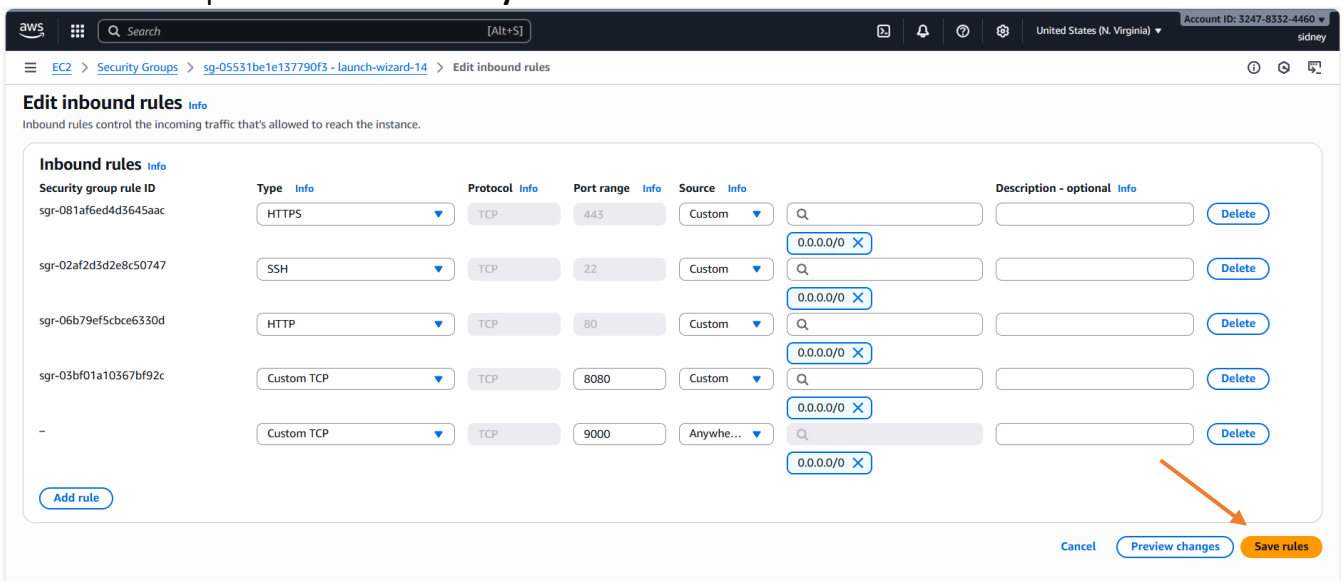
Click on “**Add Rule**”



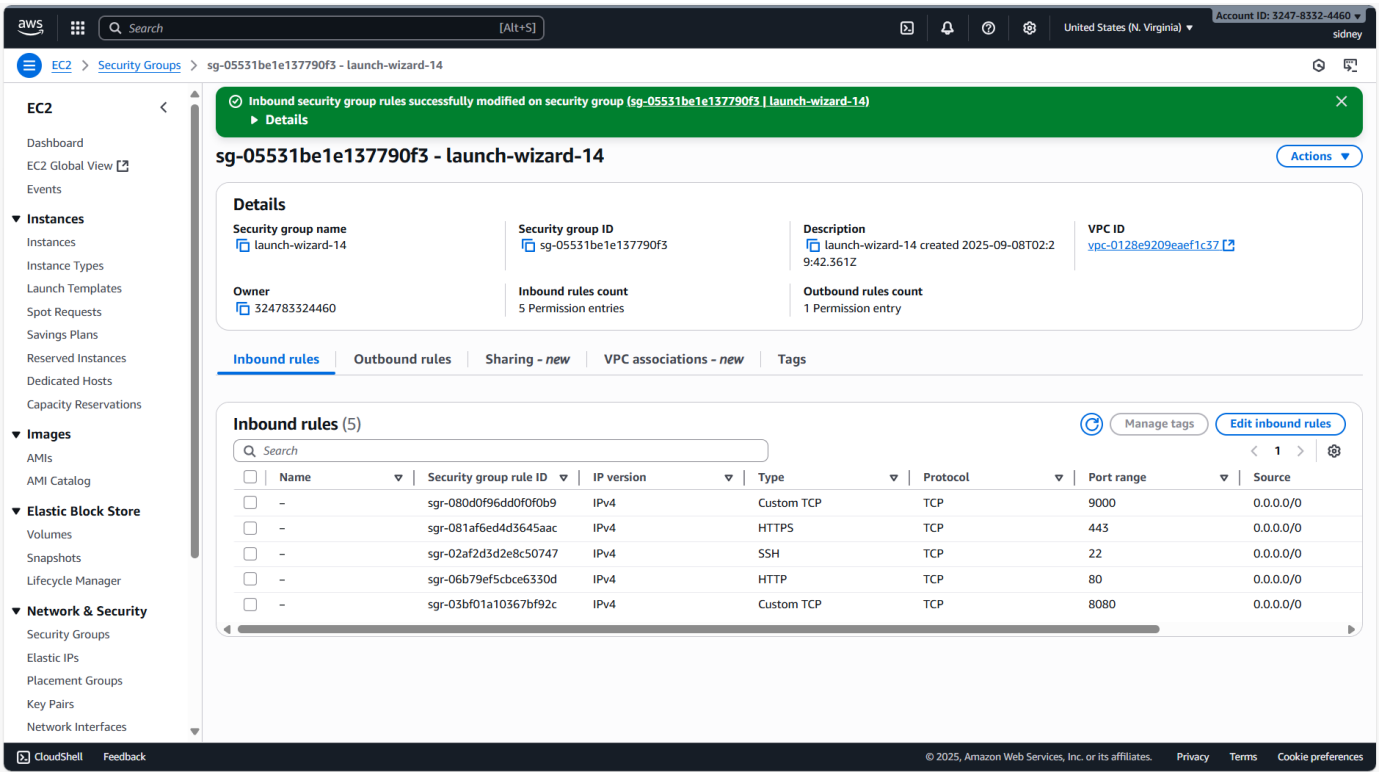
Add Port 9000



Click on the drop down and select “**Anywhere-IPv4**”

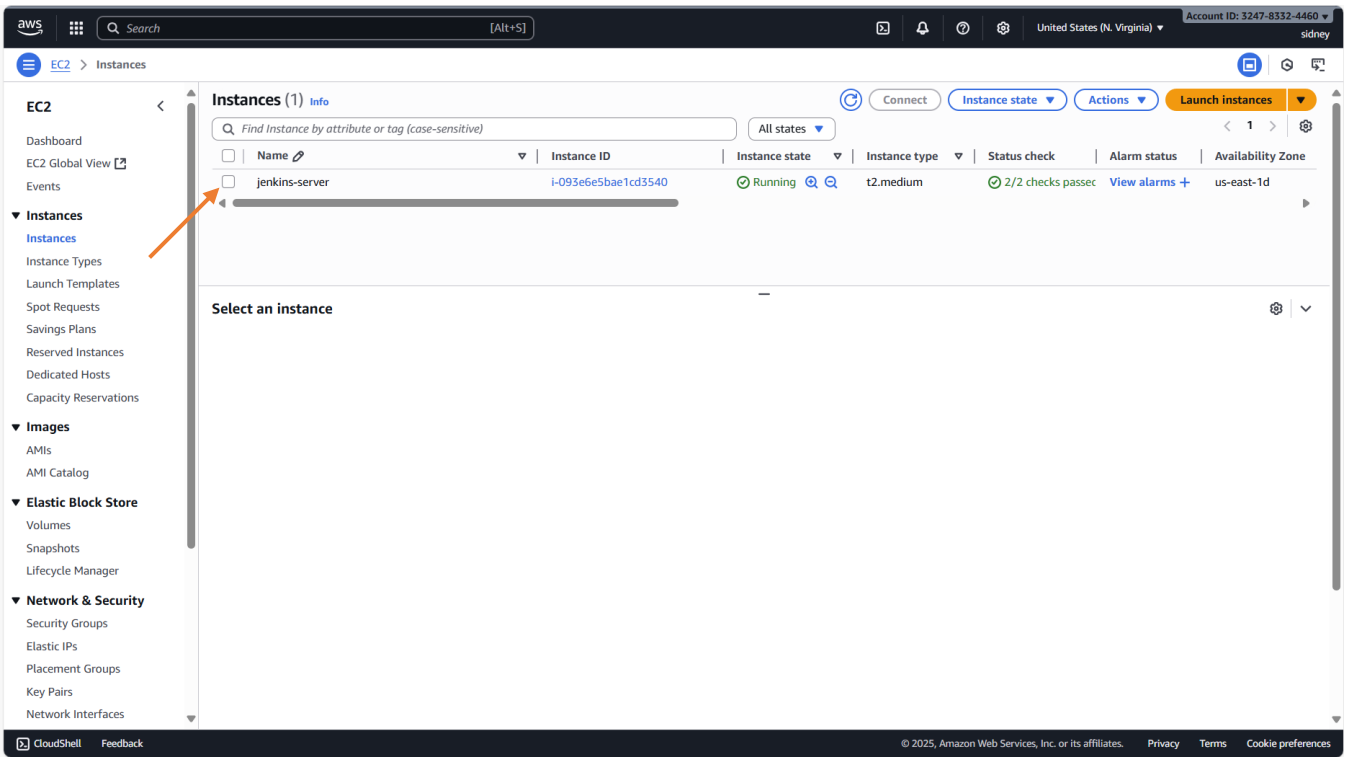


Click on “**Save Rules**”

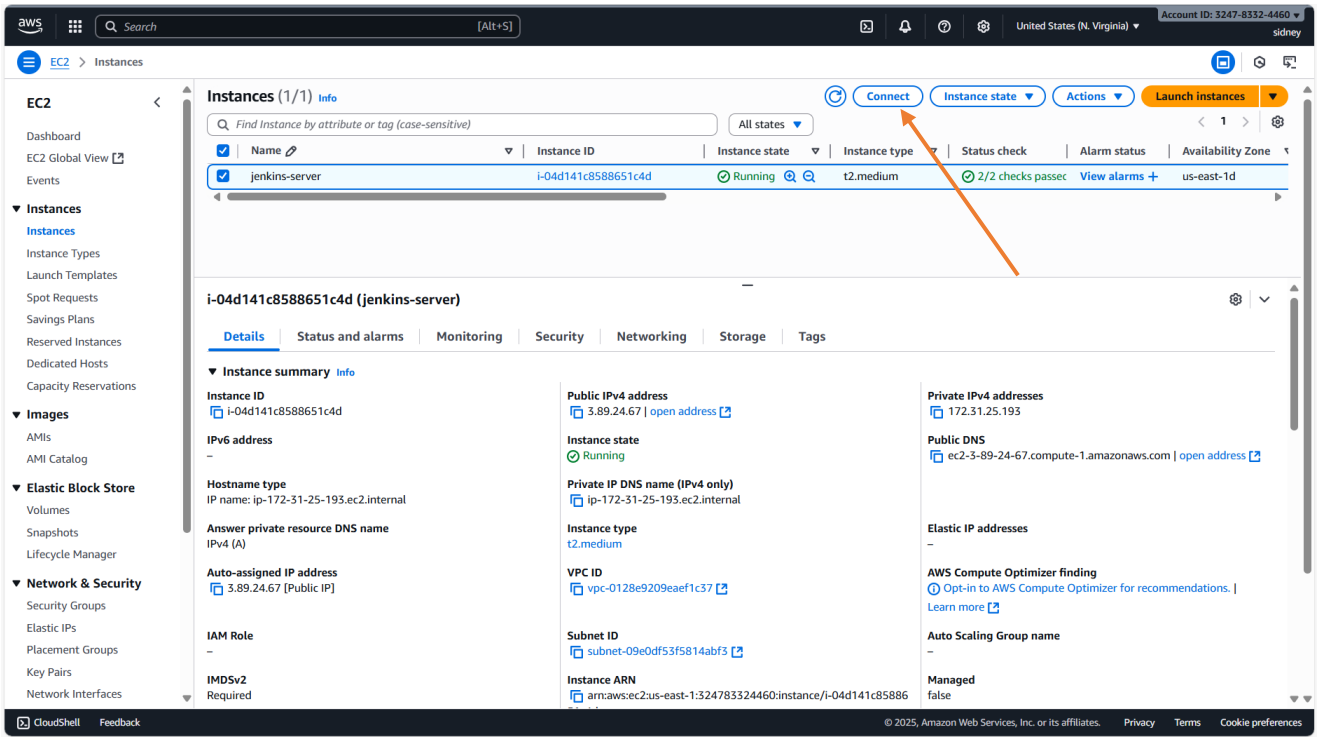


PART 4: SSH Connect to the EC2 instance

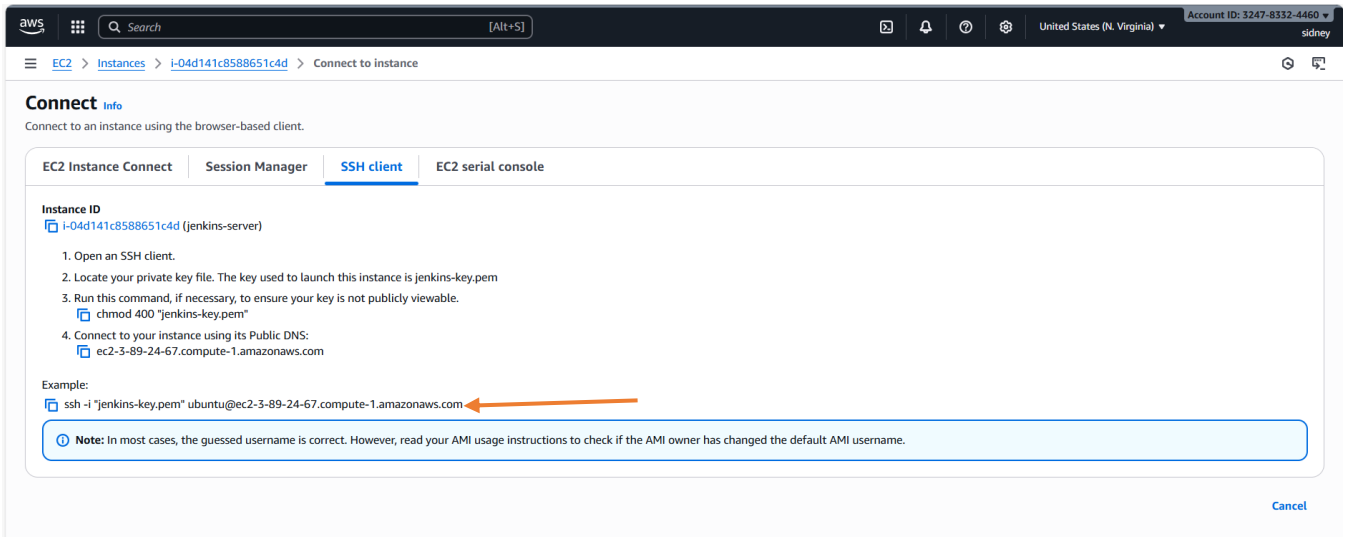
Now, let us connect to our EC2 instance through SSH



Select the instance

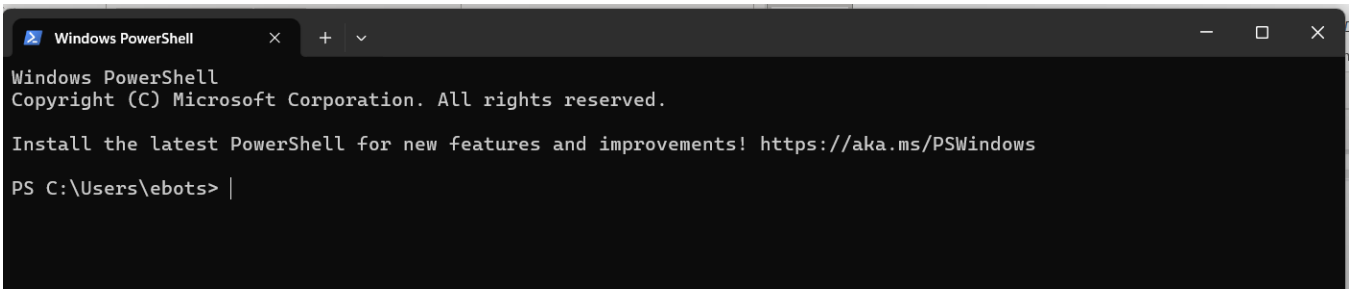


Click on “**Connect**”



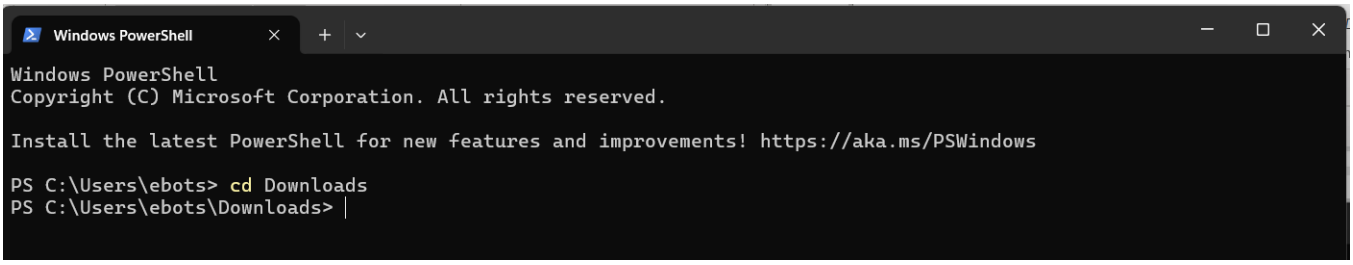
Copy this command and open PowerShell

```
ssh -i "jenkins-key.pem" ubuntu@ec2-54-82-118-6.compute-1.amazonaws.com
```



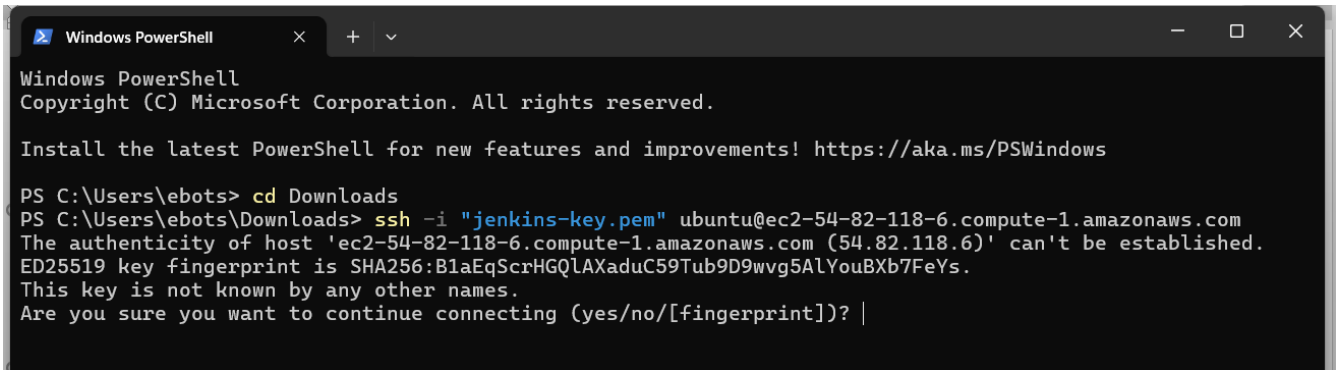
Navigate to the “**Downloads**” folder where our “**.pem**” file is stored by using the command:

```
cd Downloads
```

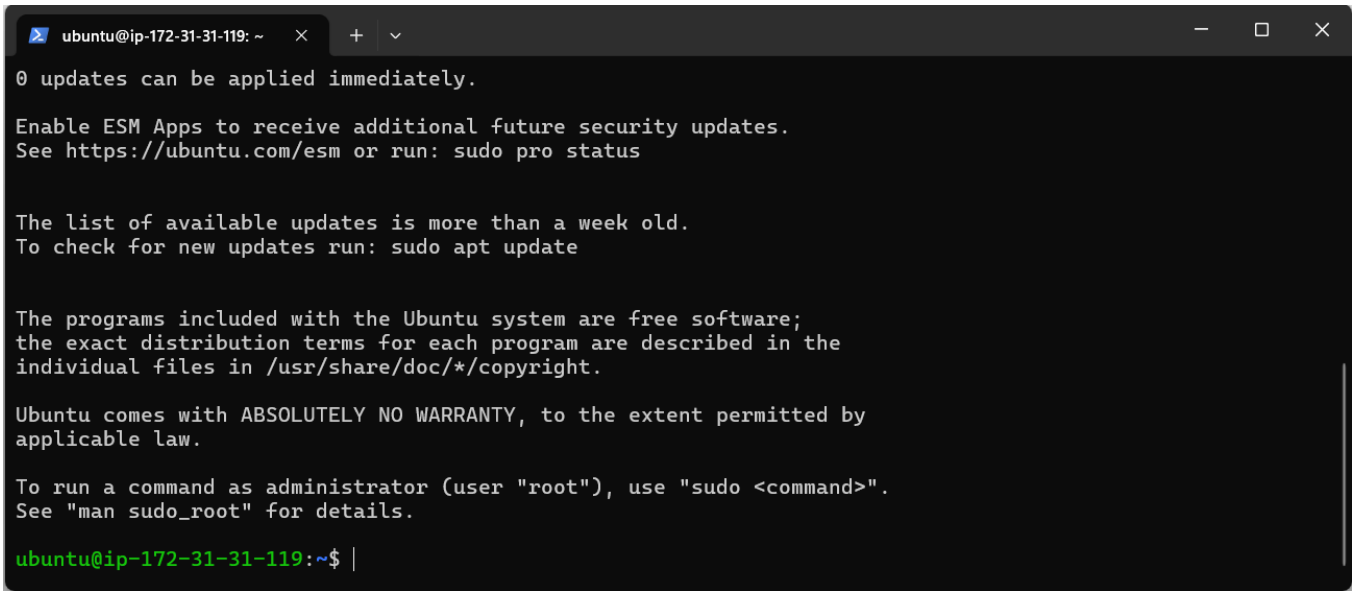


Then run the copied command:

```
ssh -i "jenkins-key.pem" ubuntu@ec2-54-82-118-6.compute-1.amazonaws.com
```



Type "**yes**" and press ENTER



We have successfully SSH connected to our EC2 instance

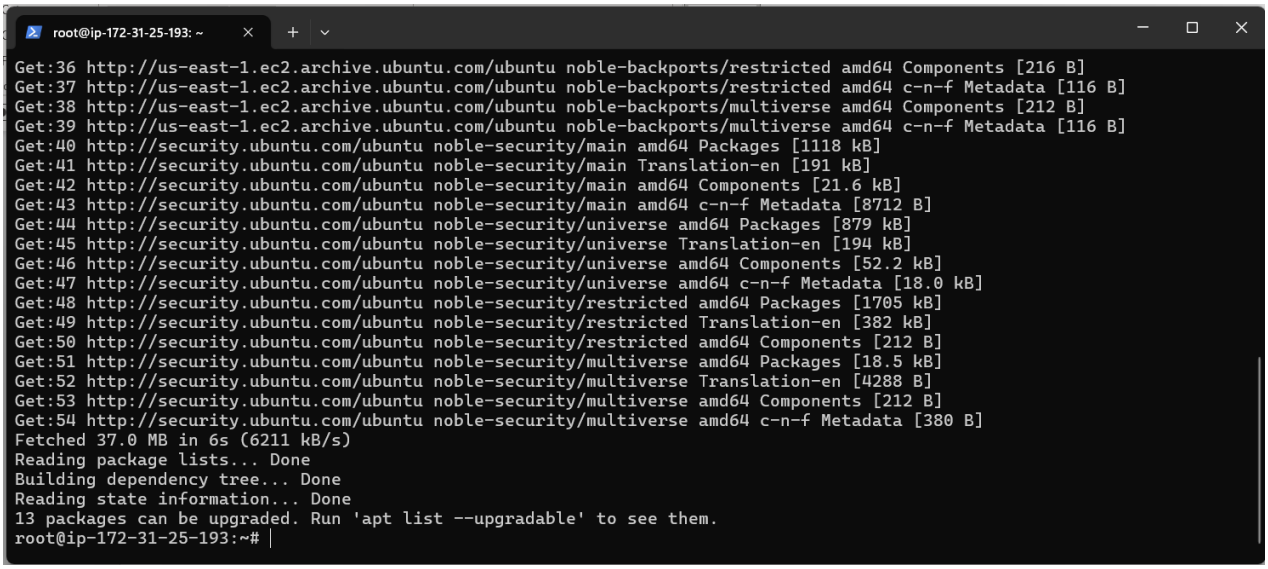
STEP 2: Install Java on Ubuntu EC2 instance

The next thing is to install Java on our server, since Jenkins is written in Java. To do this we first of all switch to the root user using the command:

```
sudo su -
```

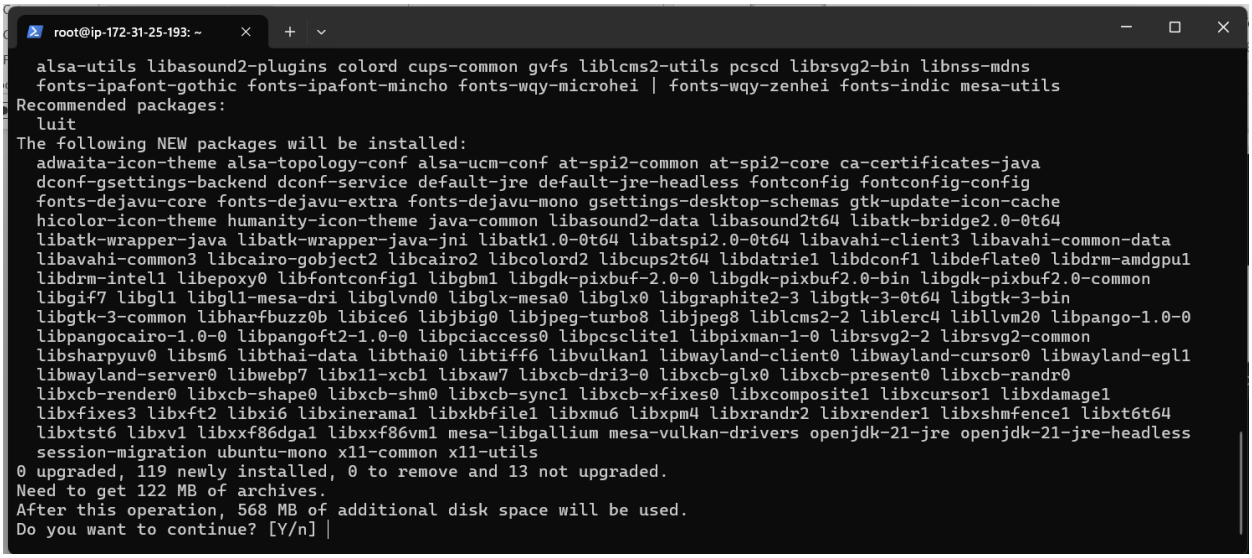
Update the packages by running the command:

```
apt update -y
```

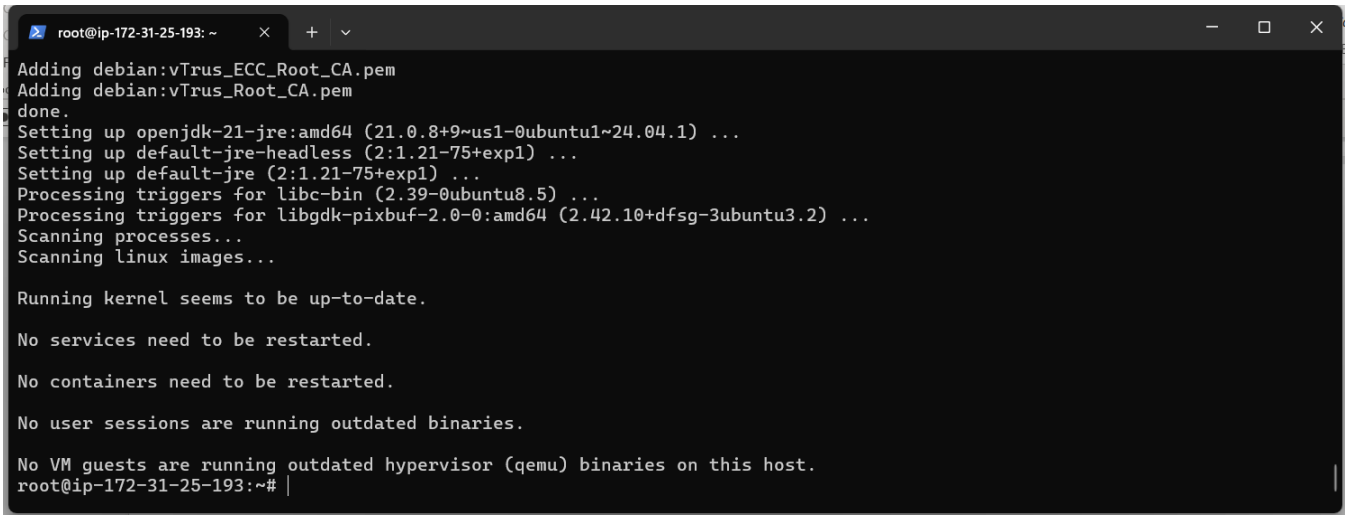


Install Java using the command:

```
sudo apt install default-jre
```

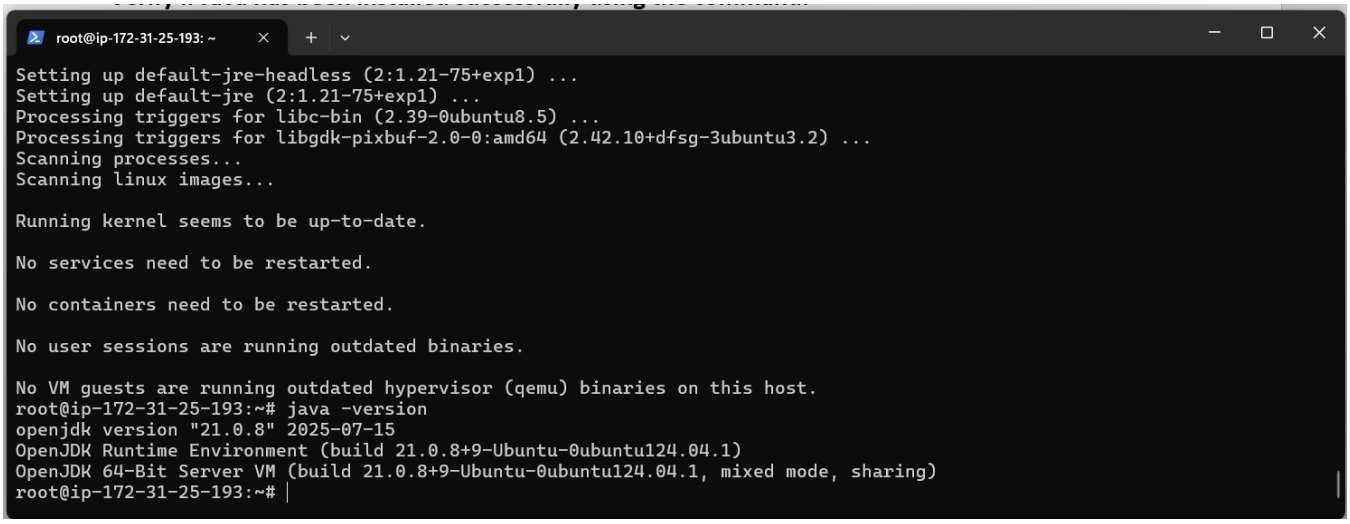


Type "**Y**" and press Enter



Verify if Java has been installed successfully using the command:

```
java -version
```



You can see that Java has been installed successfully

STEP 3: Install Docker on EC2 server

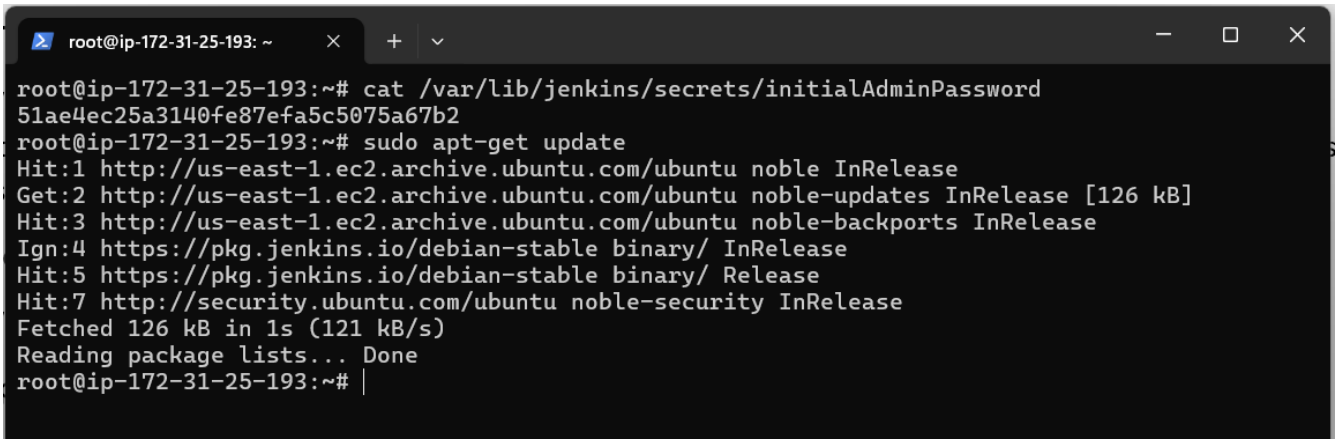
We have to install Docker on our EC2 server. Installing Docker on Ubuntu typically involves setting up Docker's official APT repository and then installing the Docker Engine. The recommended approach is to use the repository to ensure you receive the latest updates and security patches.

Steps to Install Docker Engine on Ubuntu:

We will install Docker engine on our server

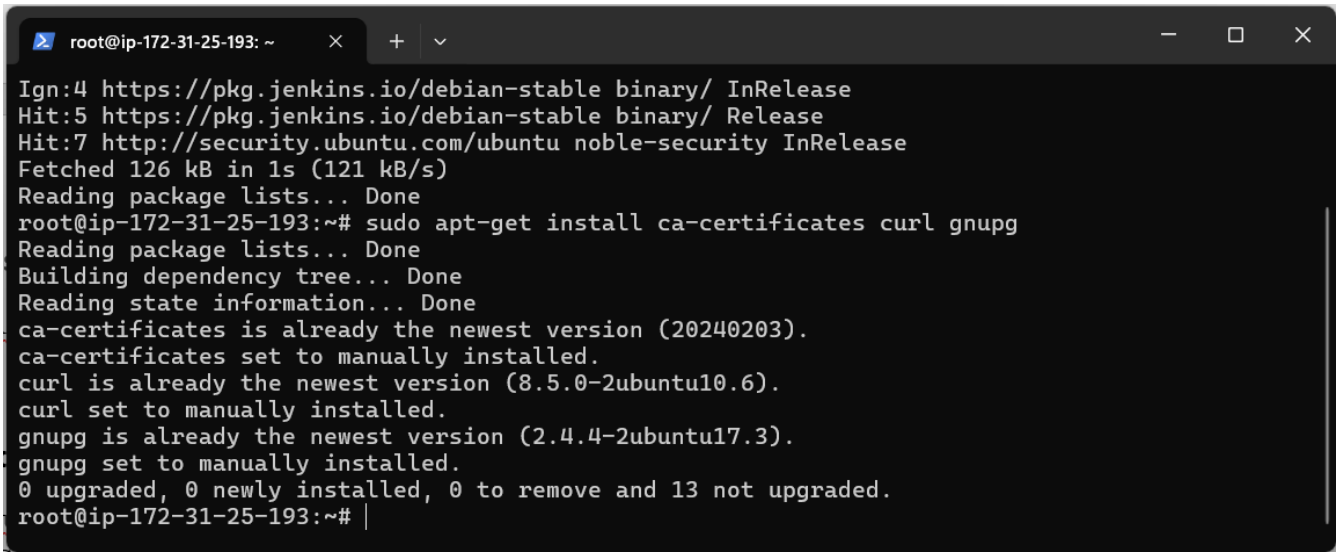
Update your package list using the command:

```
sudo apt-get update
```



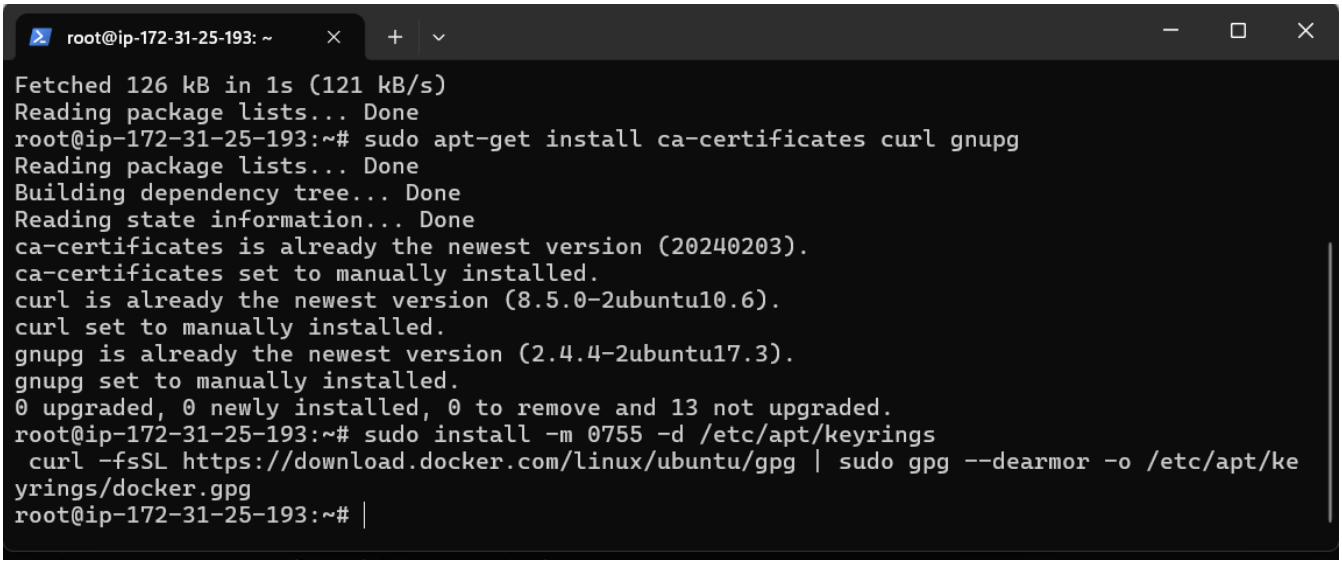
Install necessary packages for adding a repository over HTTPS:

```
sudo apt-get install ca-certificates curl gnupg
```



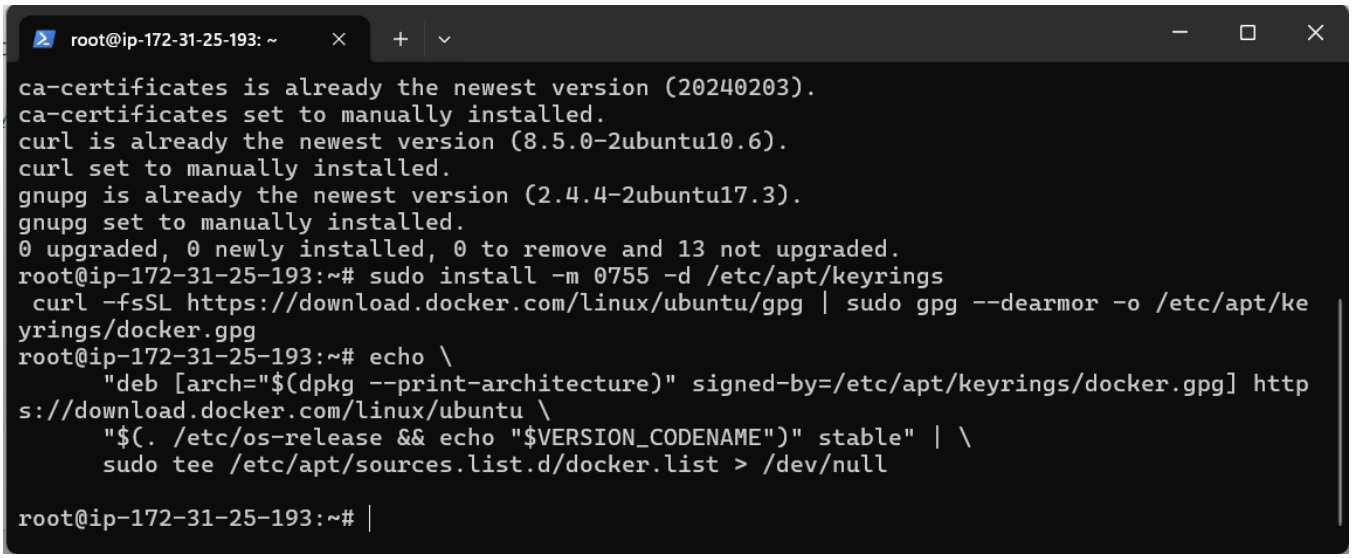
Add Docker's official GPG key

```
sudo install -m 0755 -d /etc/apt/keyrings
 curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
dearmor -o /etc/apt/keyrings/docker.gpg
```



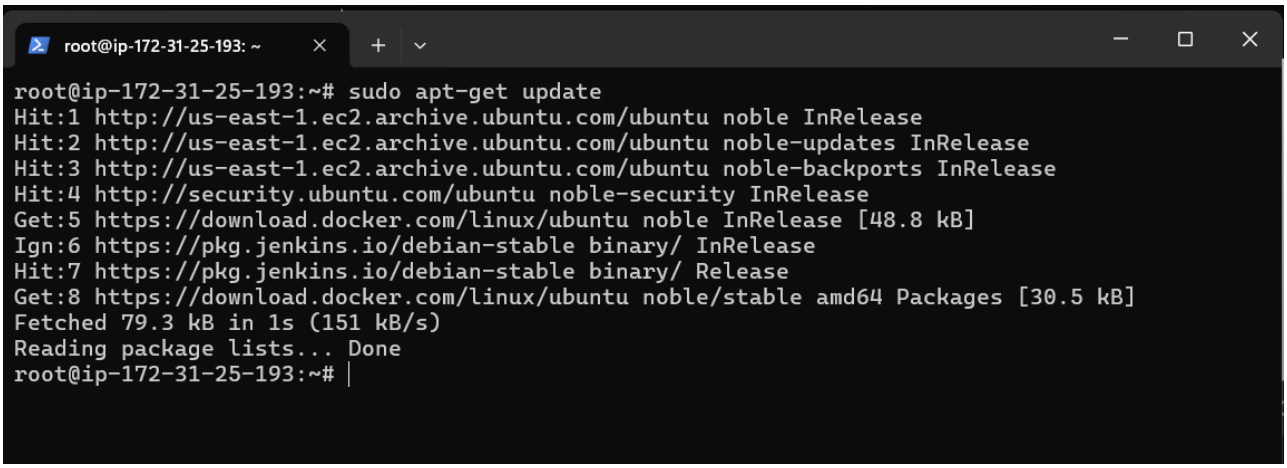
Set up the Docker APT repository.

```
echo \
      "deb [arch="$(dpkg --print-architecture)" signed-
by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
      "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
      sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```



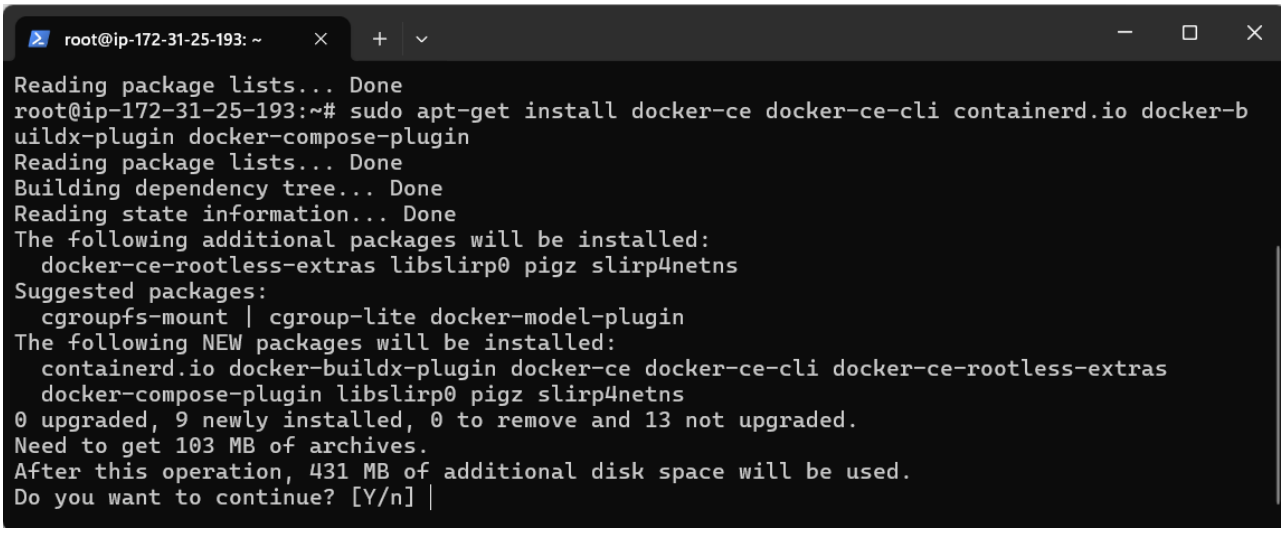
Update your package list again to include the new Docker repository:

```
sudo apt-get update
```

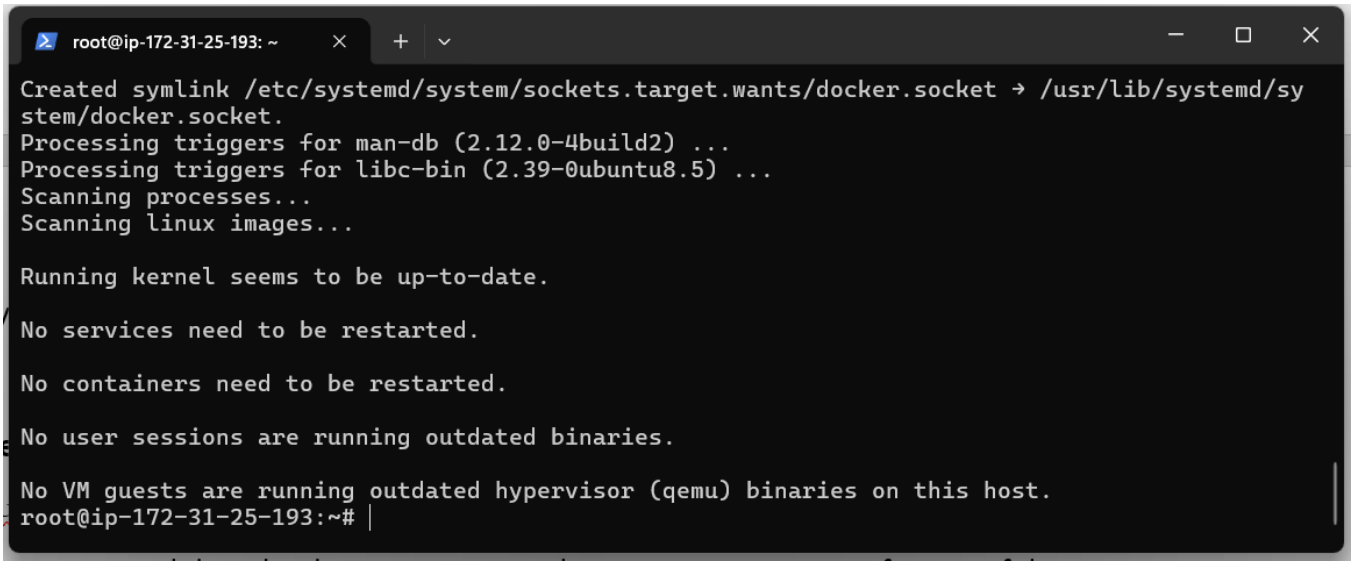


Install Docker Engine, containerd, and Docker Compose:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```



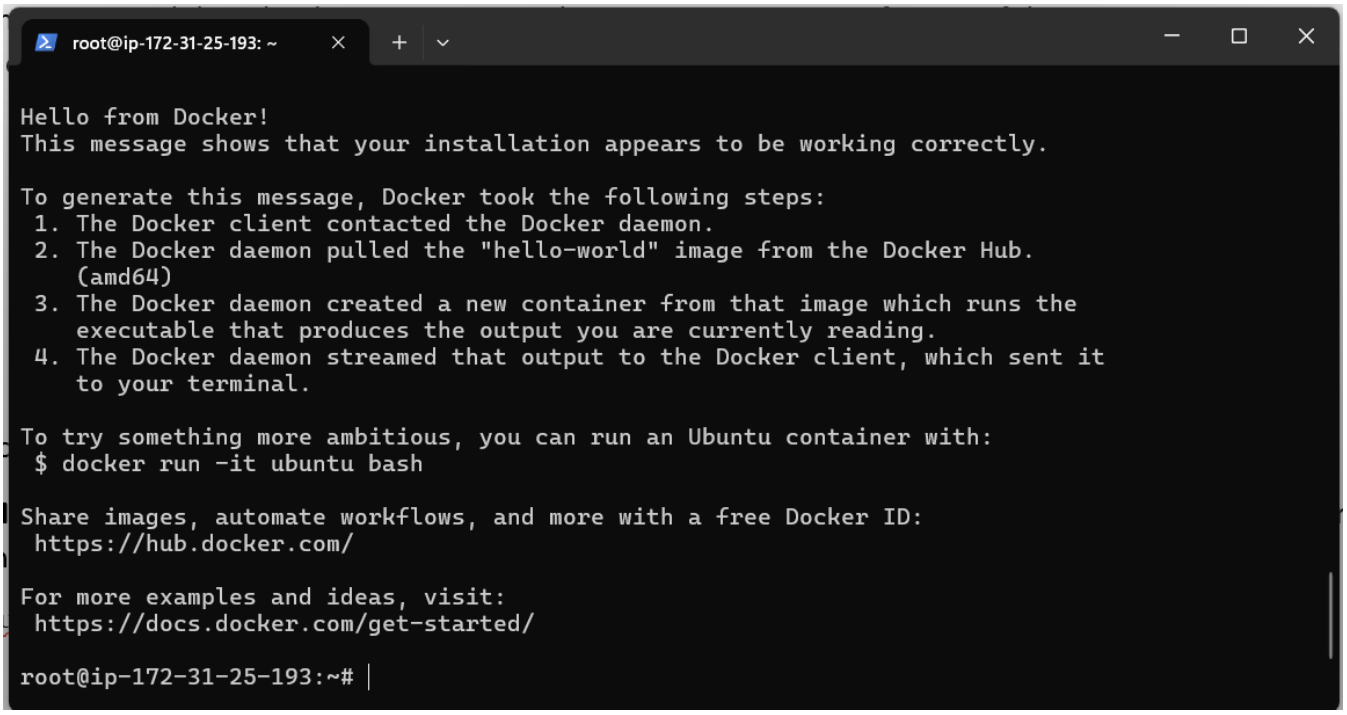
Type "**Y**" and press ENTER



Verify the installation by running the hello-world image:

```
sudo docker run hello-world
```

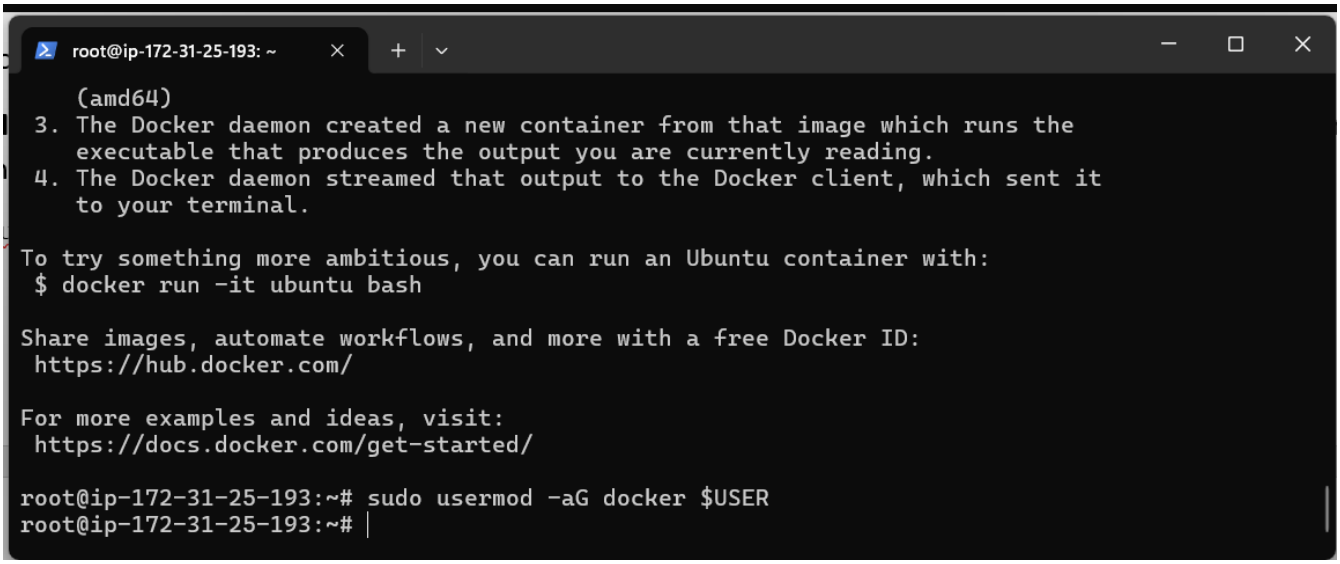
This command downloads a test image and runs it in a container. If successful, it prints a message indicating Docker is working.



You can see that Docker has been installed successfully.

Manage Docker as a non-root user: To avoid using sudo with every Docker command, add your user to the docker group:

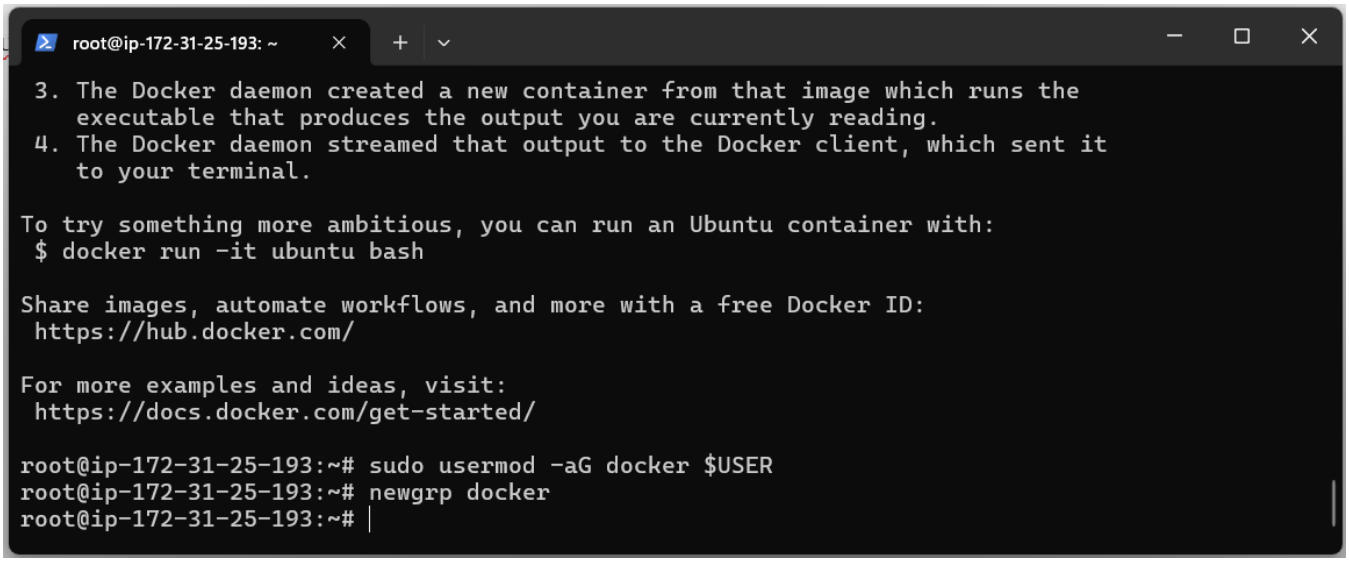
```
sudo usermod -aG docker $USER
```



You will need to log out and log back in (or restart your system) for this change to take effect. After logging back in, you can run docker run hello-world without sudo.

Activate a user's membership in the docker group using the command:

```
newgrp docker
```

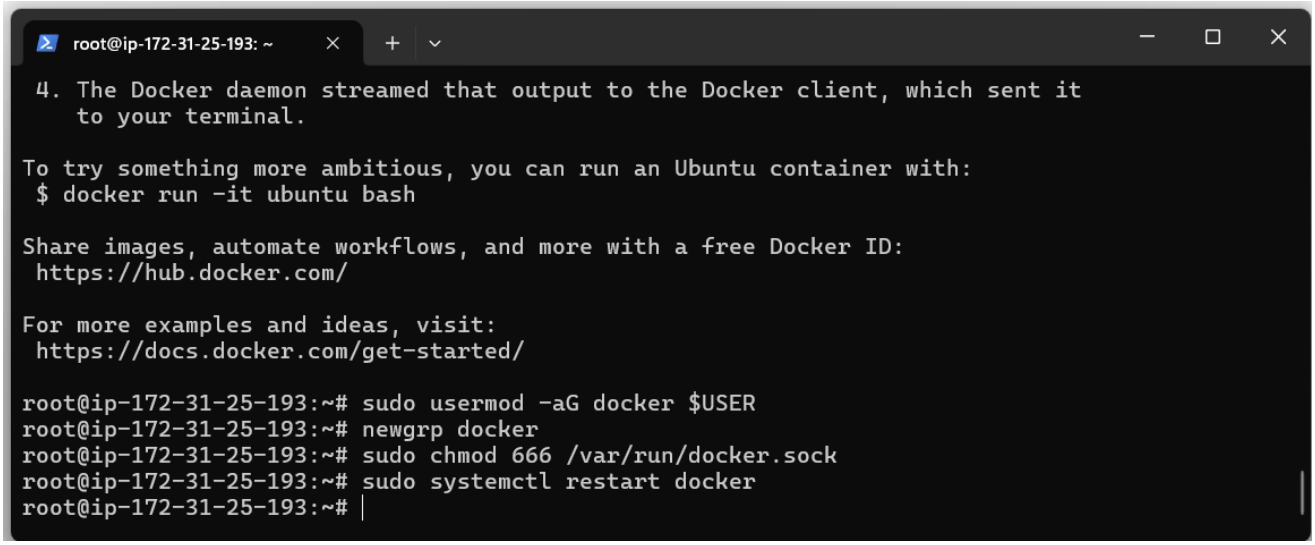


Run the command sudo chmod 666 /var/run/docker.sock modifies the permissions of the Docker Unix socket file located at /var/run/docker.sock

```
sudo chmod 666 /var/run/docker.sock
```

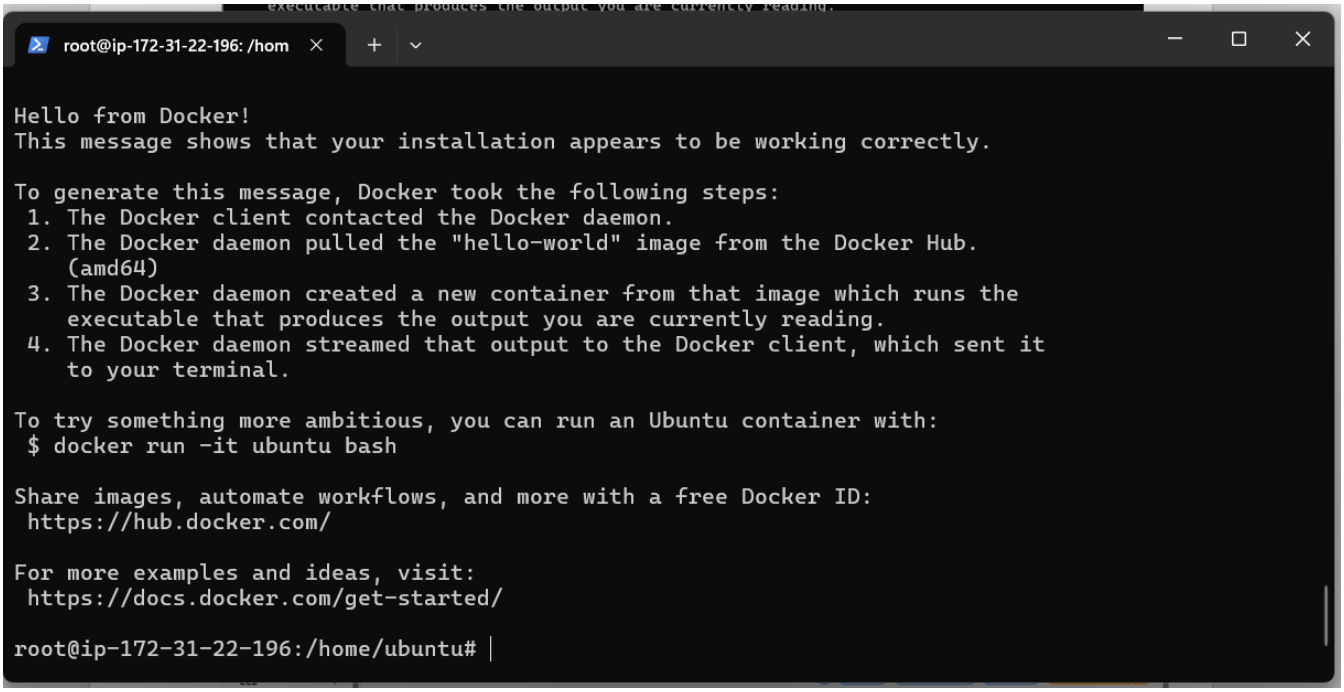
Then restart Docker using the command:

```
sudo systemctl restart docker
```



Then run the command to verify if docker is working again, but this time we will remove the “sudo”

```
docker run hello-world
```



You can see that Docker is still working

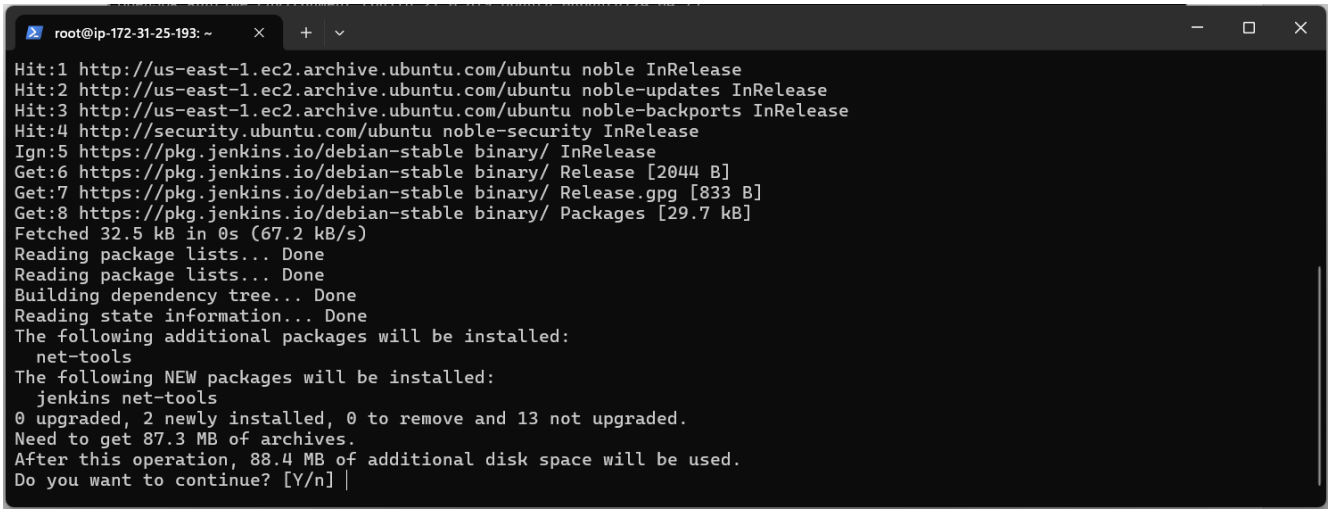
STEP 4: Install Jenkins and Access Jenkins on your browser

In this step, we will install Jenkins and access it on our browser

Install Jenkins on EC2 instance

Now, let us install Jenkins. This will be done by running the command:

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



Type "**Y**" and press ENTER

Enable Jenkins using the command:

```
sudo systemctl enable jenkins
```

```
root@ip-172-31-25-193: ~
Unpacking jenkins (2.516.2) ...
Setting up net-tools (2.10-0.1ubuntu4.4) ...
Setting up jenkins (2.516.2) ...
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-172-31-25-193:~# sudo 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-172-31-25-193:~#
```

Start Jenkins by using the command:

`sudo systemctl start jenkins`

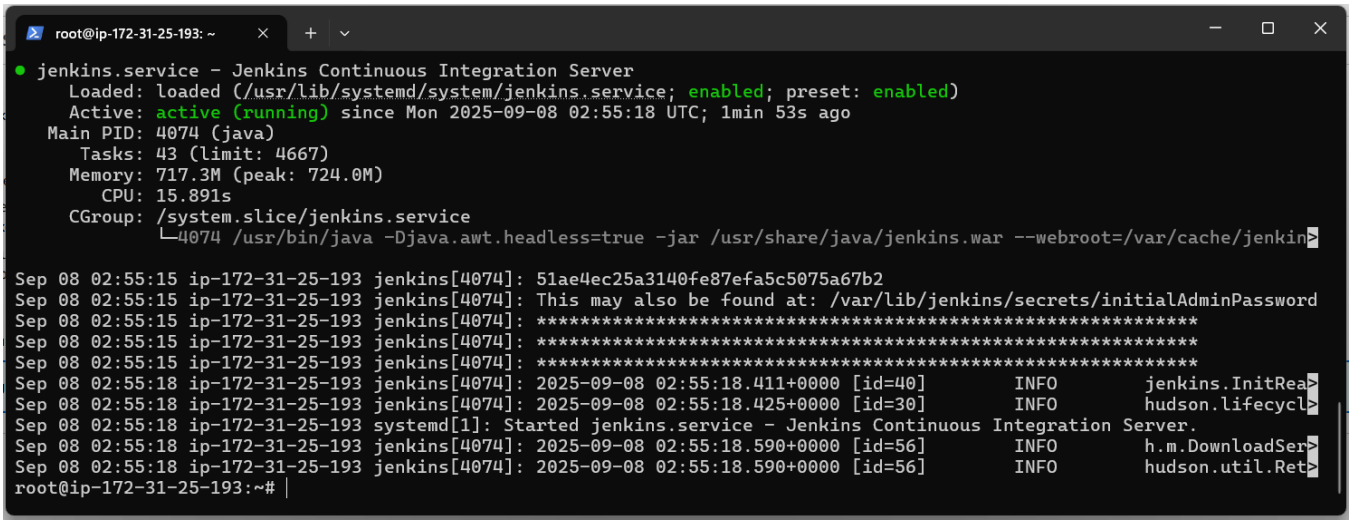
```
root@ip-172-31-25-193: ~
Setting up net-tools (2.10-0.1ubuntu4.4) ...
Setting up jenkins (2.516.2) ...
Created symlink /etc/systemd/system/multi-user.target.wants/jenkins.service → /usr/lib/systemd/system/jenkins.service.
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning 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-172-31-25-193:~# sudo 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-172-31-25-193:~# sudo systemctl start jenkins
root@ip-172-31-25-193:~#
```

Verify if Jenkins is running by using the command:

`sudo systemctl status jenkins`

```
root@ip-172-31-25-193: ~
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
     Active: active (running) since Mon 2025-09-08 02:55:18 UTC; 1min 53s ago
   Main PID: 4074 (java)
      Tasks: 43 (limit: 4667)
     Memory: 717.3M (peak: 724.0M)
        CPU: 15.891s
     CGroup: /system.slice/jenkins.service
             └─4074 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkin>

Sep 08 02:55:15 ip-172-31-25-193 jenkins[4074]: 51ae4ec25a3140fe87efa5c5075a67b2
Sep 08 02:55:15 ip-172-31-25-193 jenkins[4074]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Sep 08 02:55:15 ip-172-31-25-193 jenkins[4074]: ************************************************************
Sep 08 02:55:15 ip-172-31-25-193 jenkins[4074]: ************************************************************
Sep 08 02:55:15 ip-172-31-25-193 jenkins[4074]: ************************************************************
Sep 08 02:55:18 ip-172-31-25-193 jenkins[4074]: 2025-09-08 02:55:18.411+0000 [id=40]        INFO        jenkins.InitRea>
Sep 08 02:55:18 ip-172-31-25-193 jenkins[4074]: 2025-09-08 02:55:18.425+0000 [id=30]        INFO        hudson.lifecycl>
Sep 08 02:55:18 ip-172-31-25-193 systemd[1]: Started jenkins.service - Jenkins Continuous Integration Server.
Sep 08 02:55:18 ip-172-31-25-193 jenkins[4074]: 2025-09-08 02:55:18.590+0000 [id=56]        INFO        h.m.DownloadSer>
Sep 08 02:55:18 ip-172-31-25-193 jenkins[4074]: 2025-09-08 02:55:18.590+0000 [id=56]        INFO        hudson.util.Ret>
lines 1-20
```

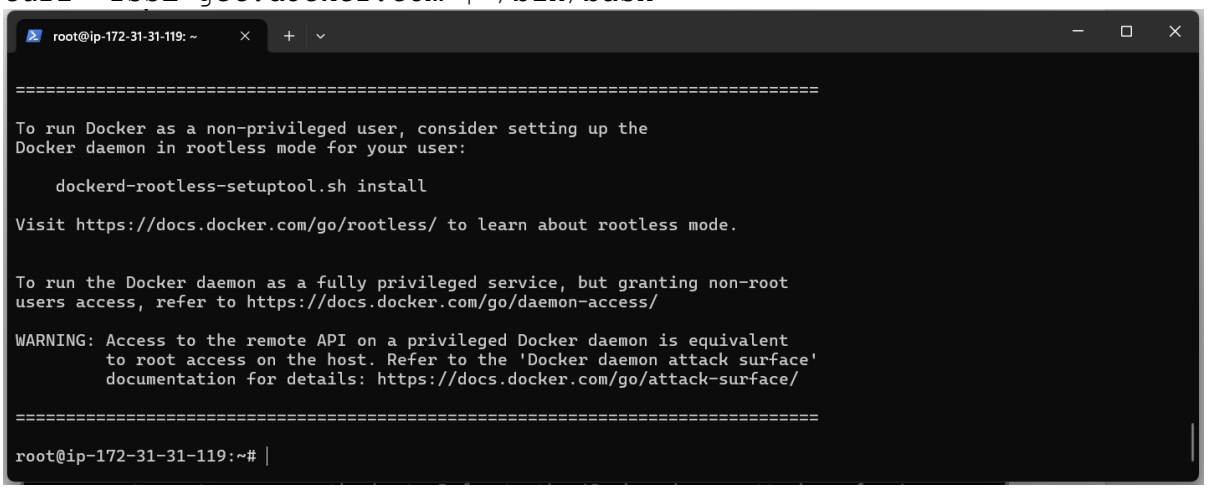
You can see that Jenkins is Active and running. To leave this mode on the terminal, type "**q**"



STEP 5: Setup Docker in Jenkins

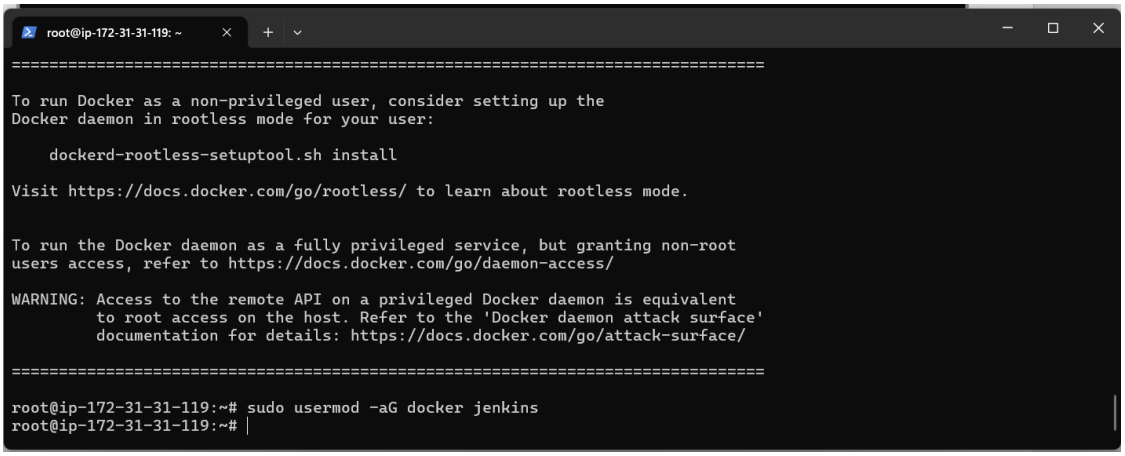
We will run this command to add Jenkins user to Docker group on the EC2 server's terminal:

```
curl -fsSL get.docker.com | /bin/bash
```



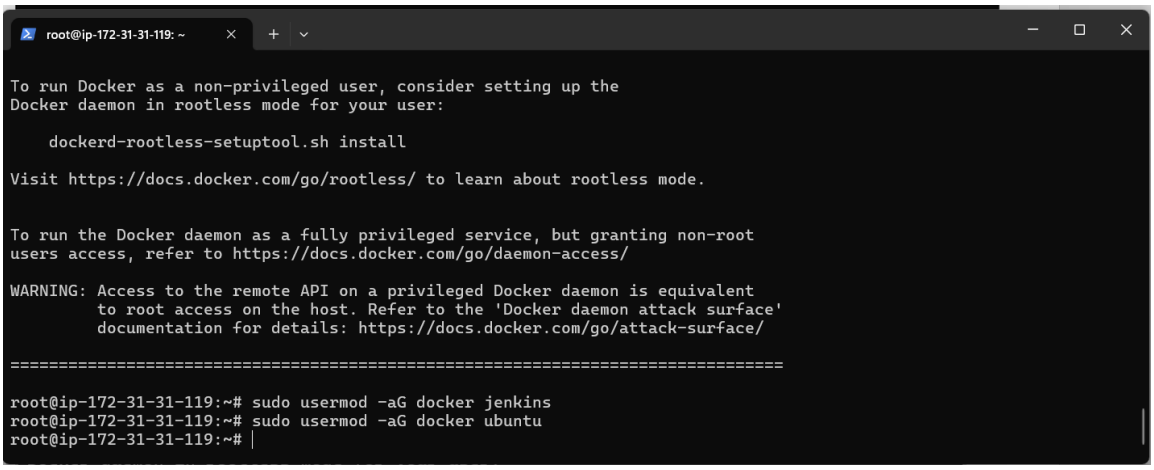
Add Jenkins user to Docker group

```
sudo usermod -aG docker Jenkins
```



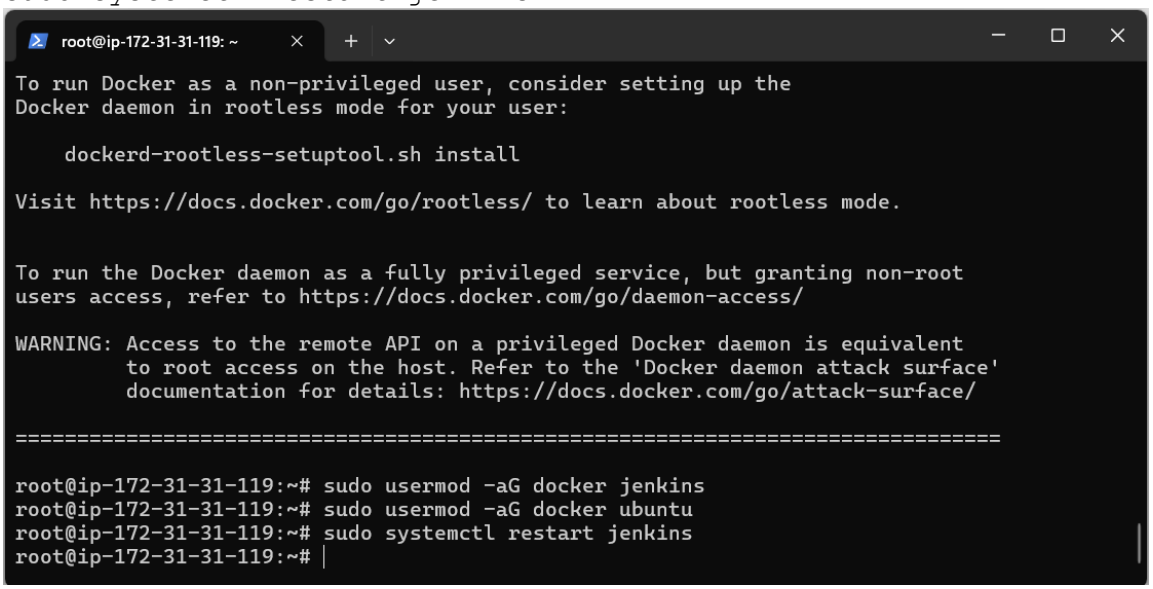
Add Ubuntu user to Docker group

```
sudo usermod -aG docker ubuntu
```



Restart Jenkins using the command:

```
sudo systemctl restart jenkins
```



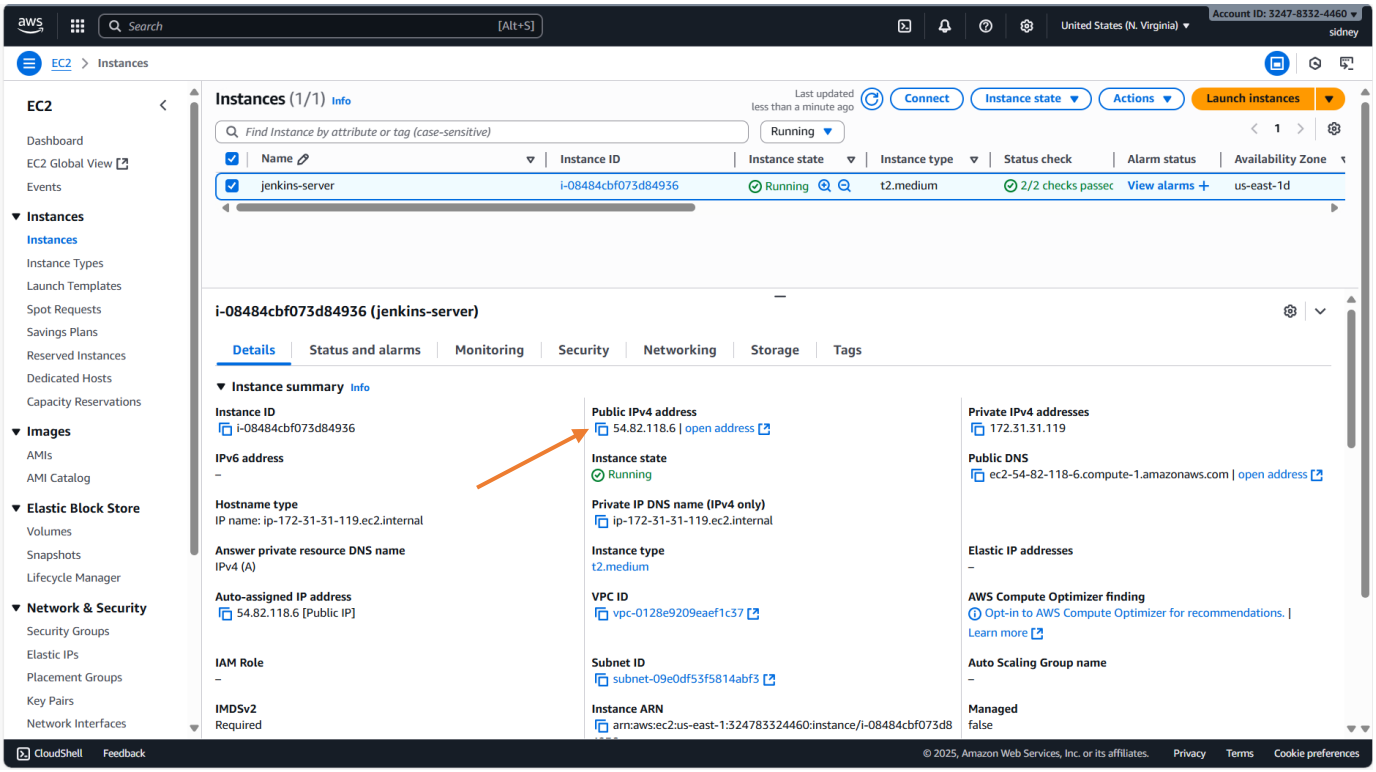
Docker has been setup in Jenkins

STEP 6: Access Jenkins server in browser using EC2 server's public IP

We have to access Jenkins through **Port 8080** on our browser. To do this we have to first grant access to Port 8080 in our EC2 instance Inbound rules.

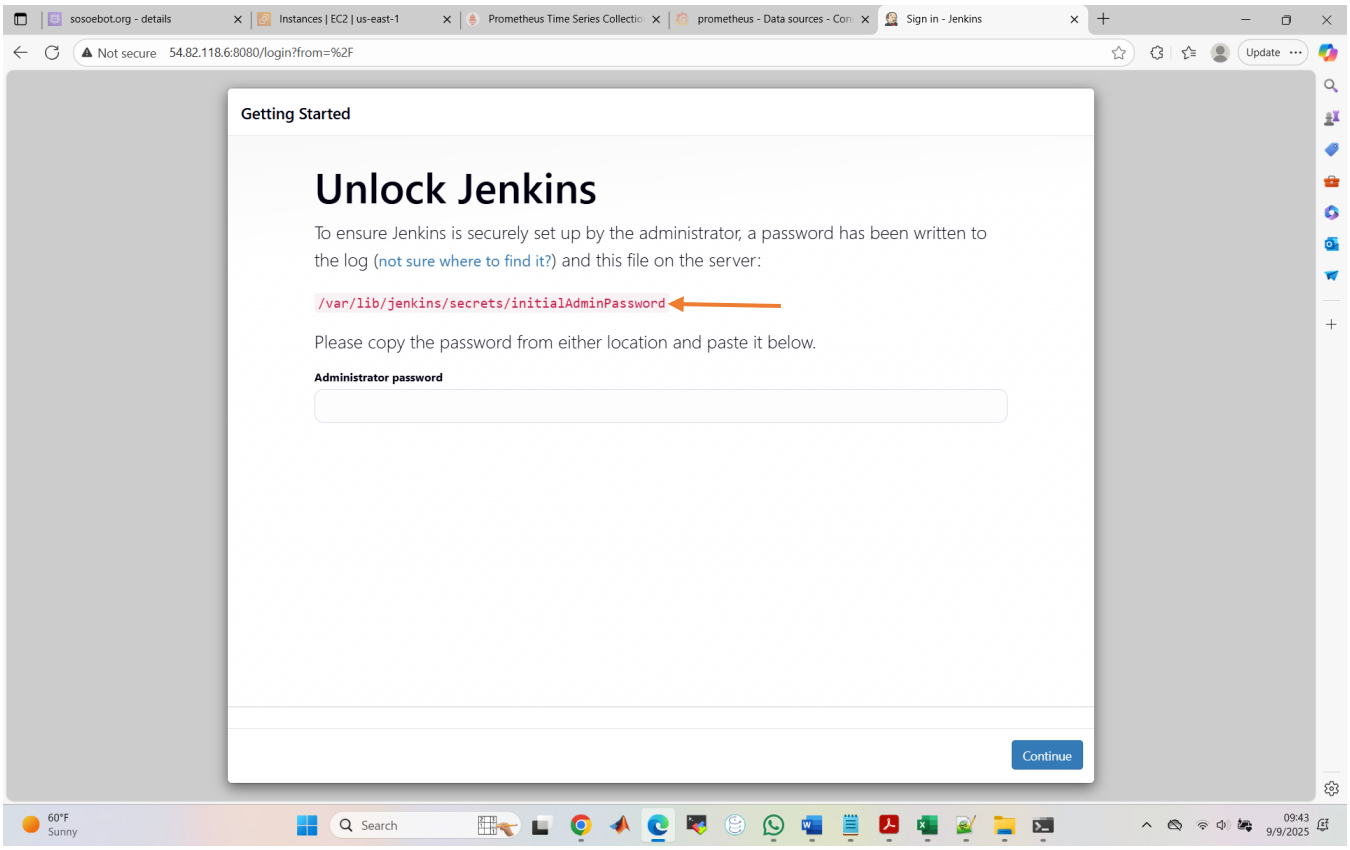
Now, let us access Jenkins on our browser by using
http://Public IPv4 Address:8080/

Go to our EC2 instance



The public IP address of our EC2 instance is 3.89.24.67
Paste **http://54.82.118.6:8080**

On our browser



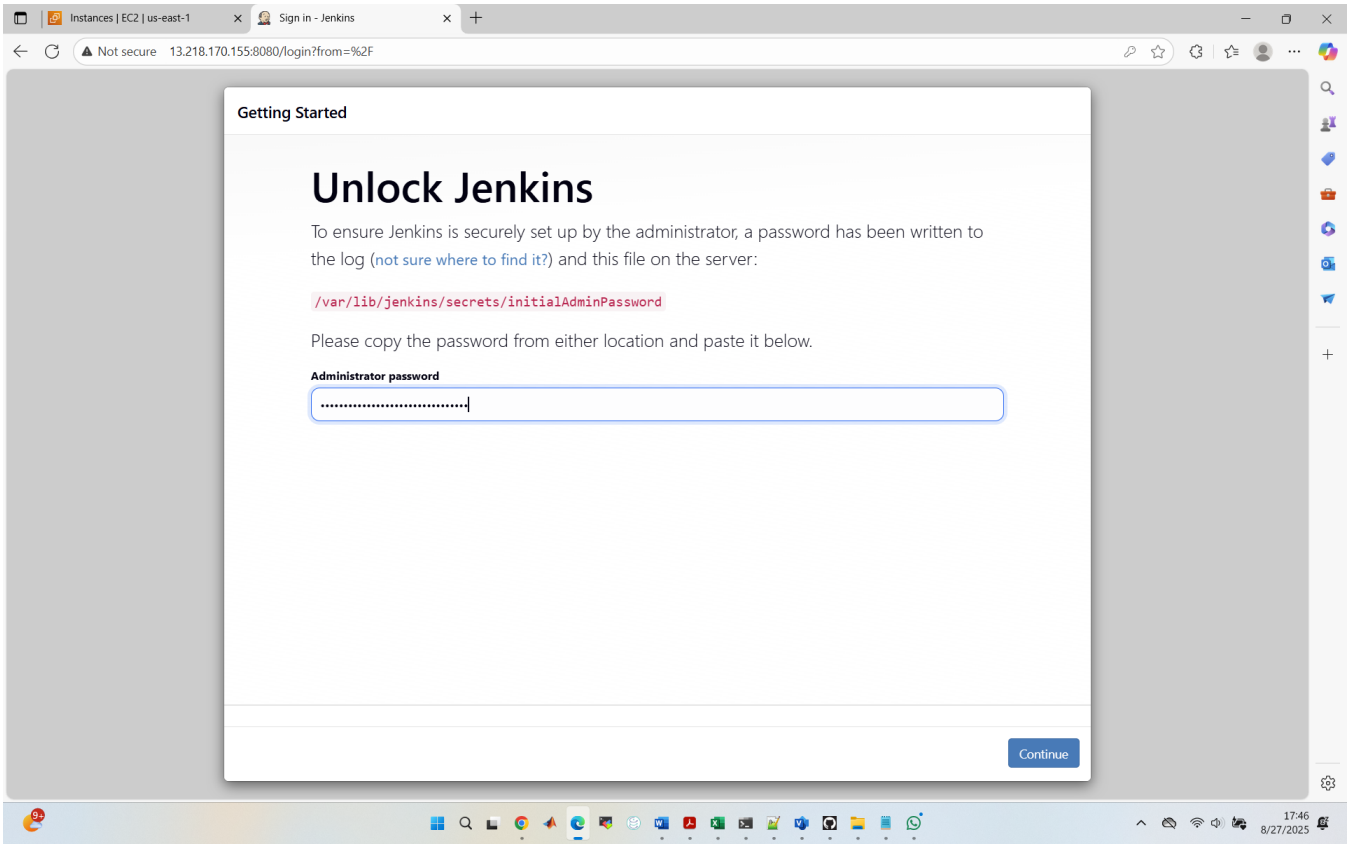
Getting the Jenkins Password

Now, let us get our password by copying the line above and run this command on our EC2 command prompt

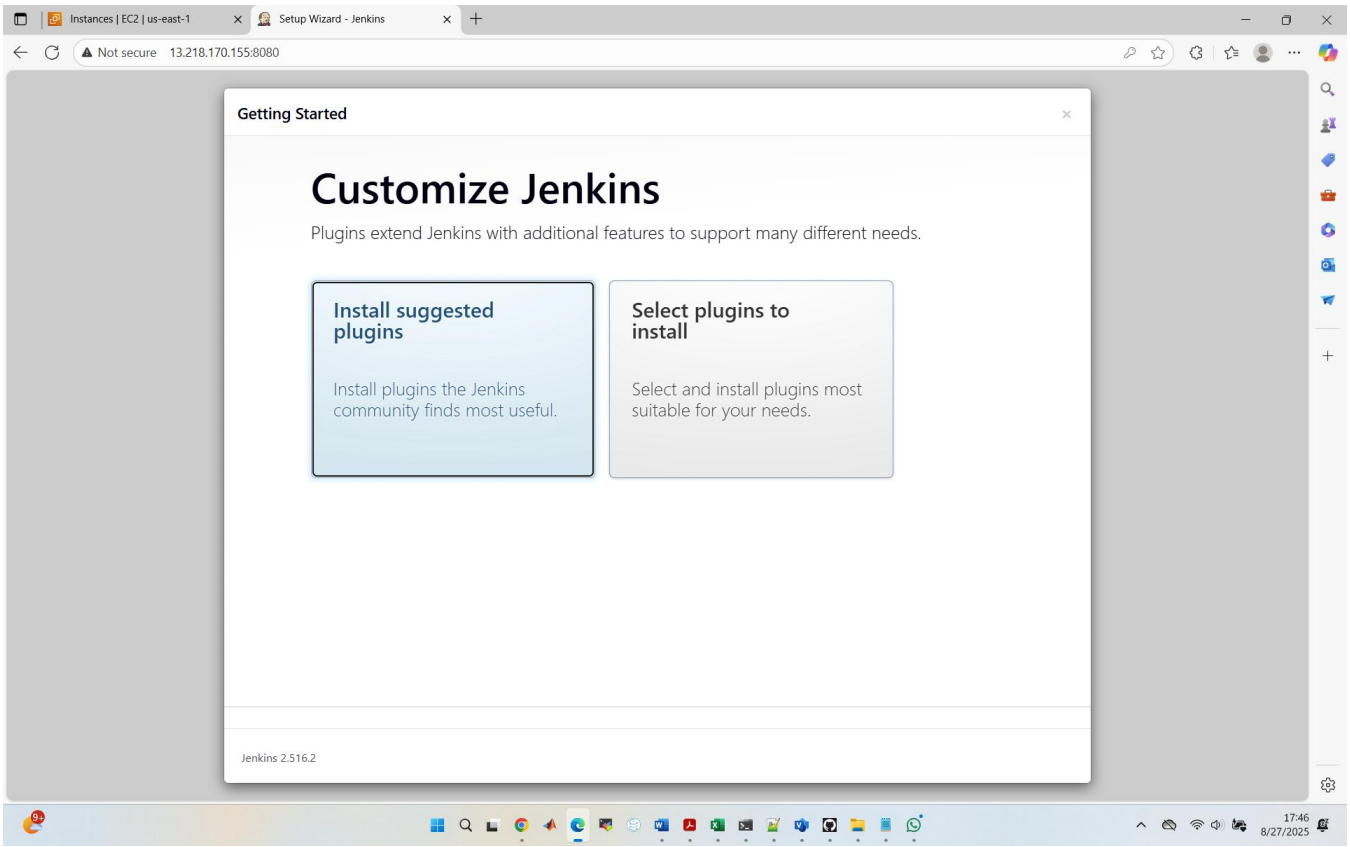
cat `/var/lib/jenkins/secrets/initialAdminPassword`

The password is 930e9e2e4a514caeaff048f6b9b38703

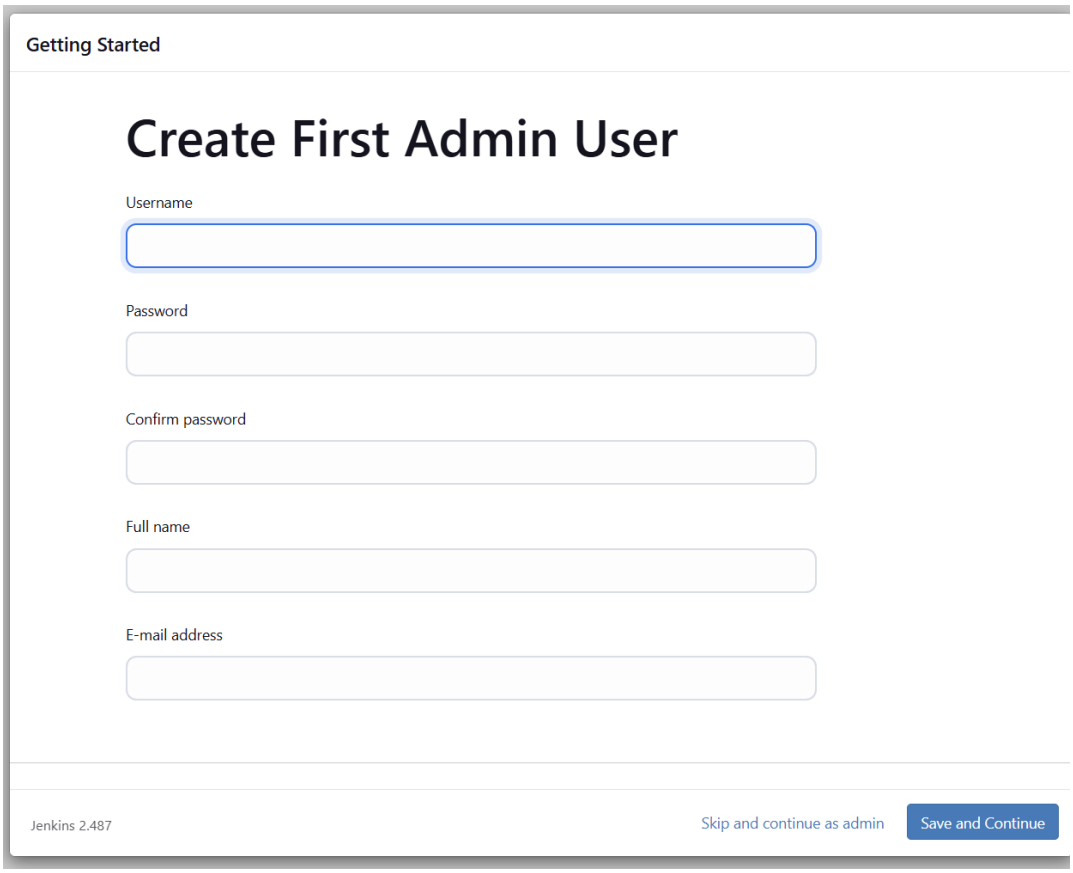
Paste it on the Jenkins browser



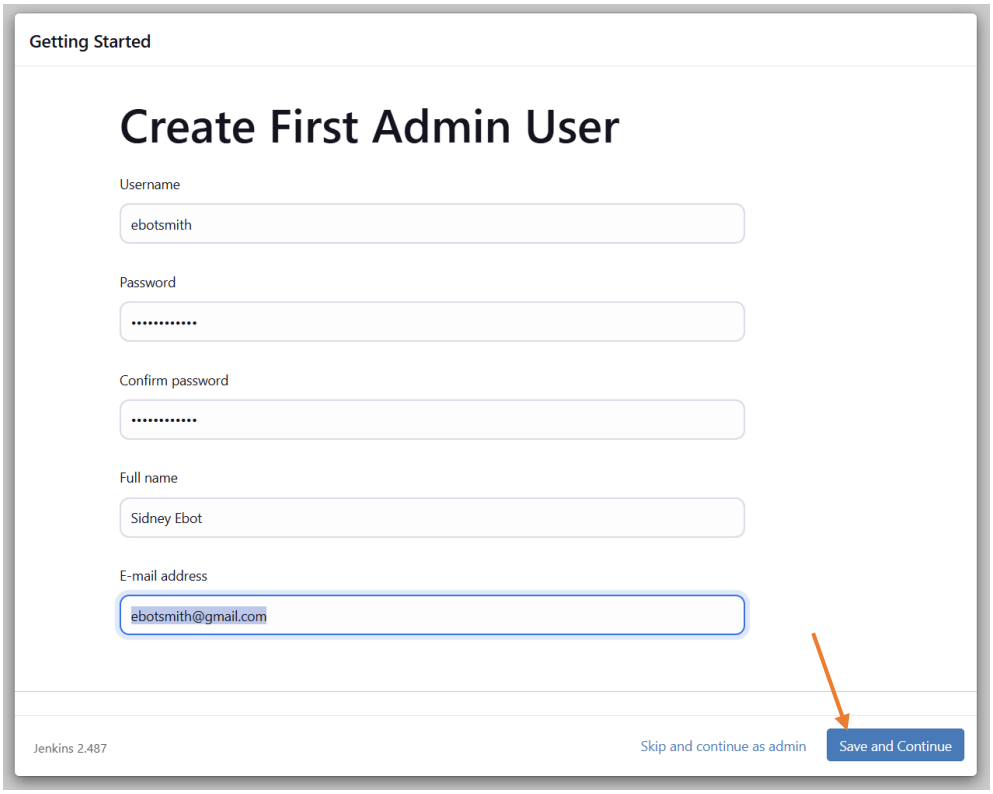
Click on “**Continue**”



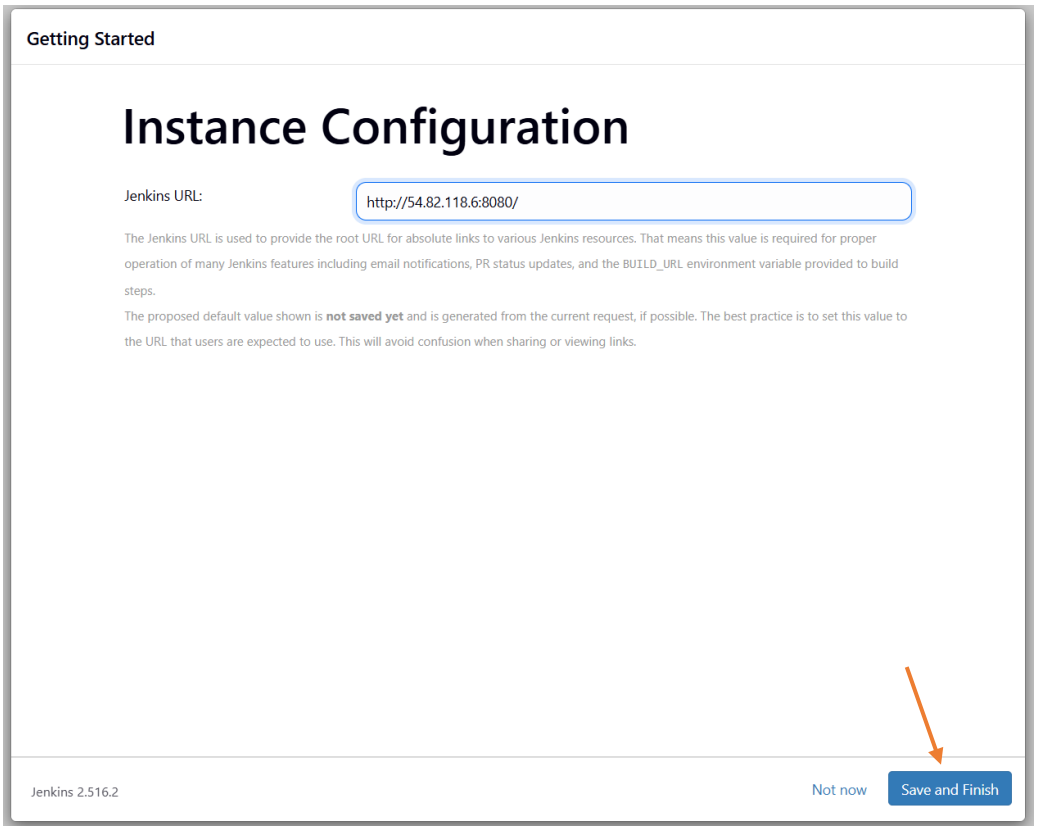
Click on “**Install Suggested Plugins**”



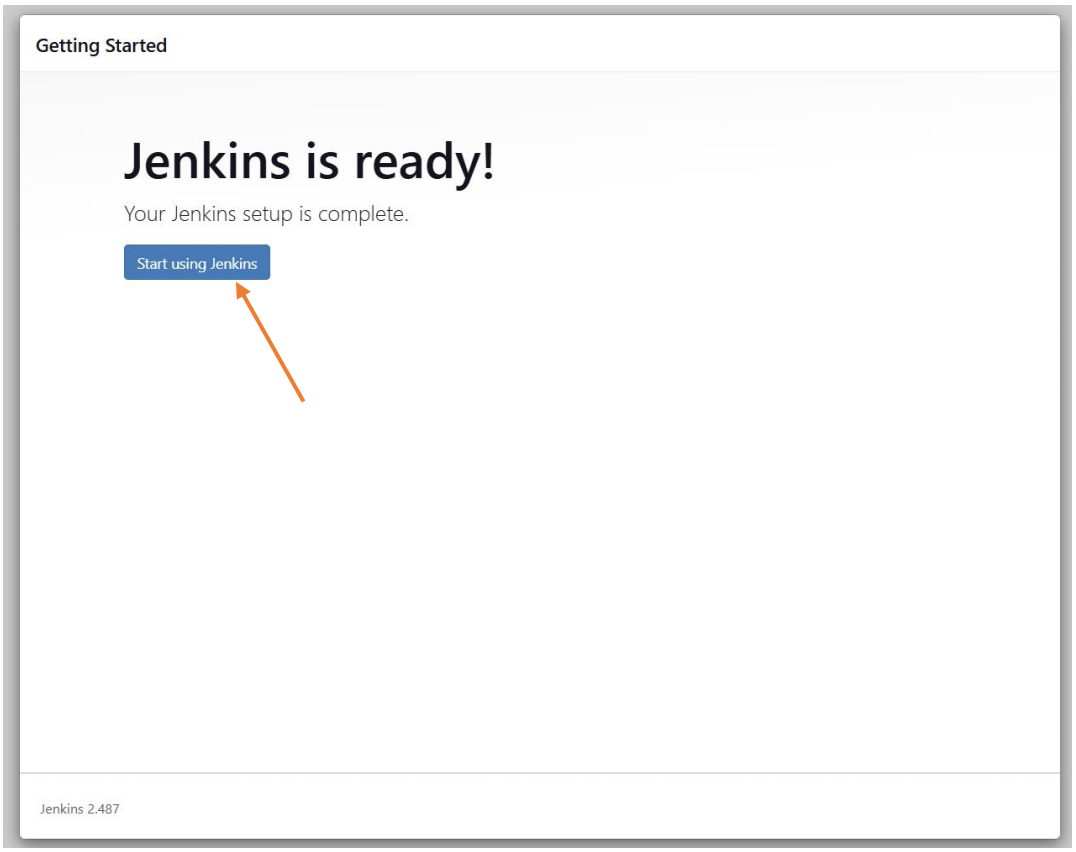
Enter the username, Password, Full name and email address.



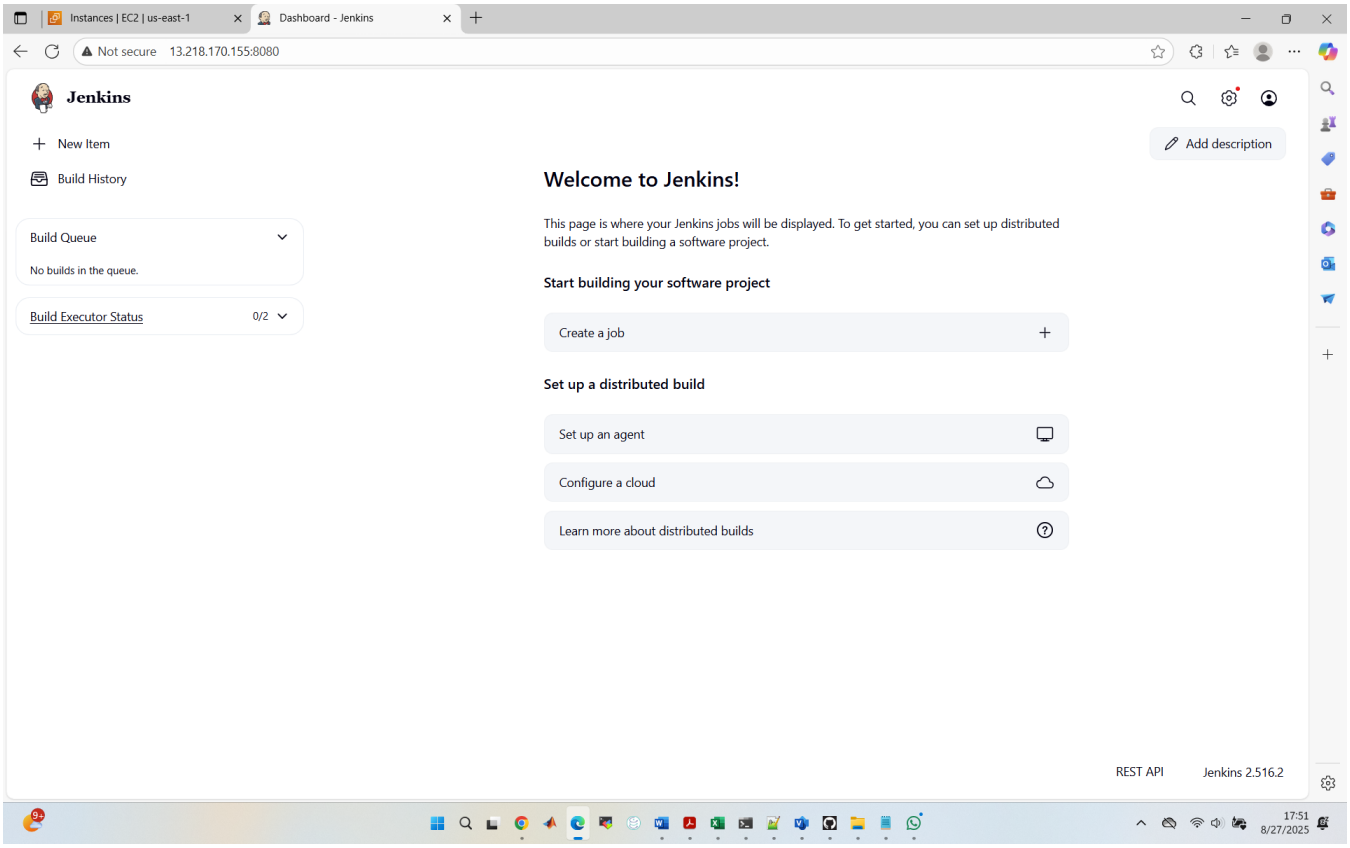
Click on "**Save and Continue**"



Click on "**Save and Finish**"



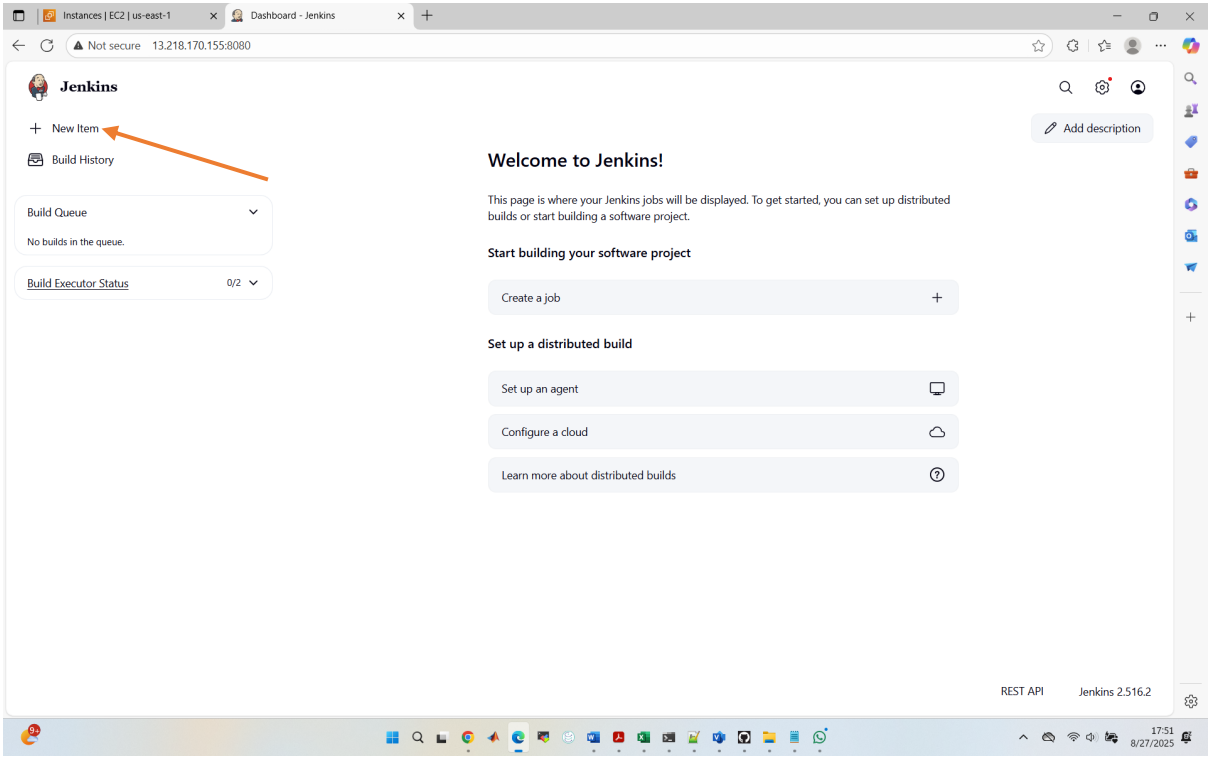
Click on "**Start using Jenkins**"



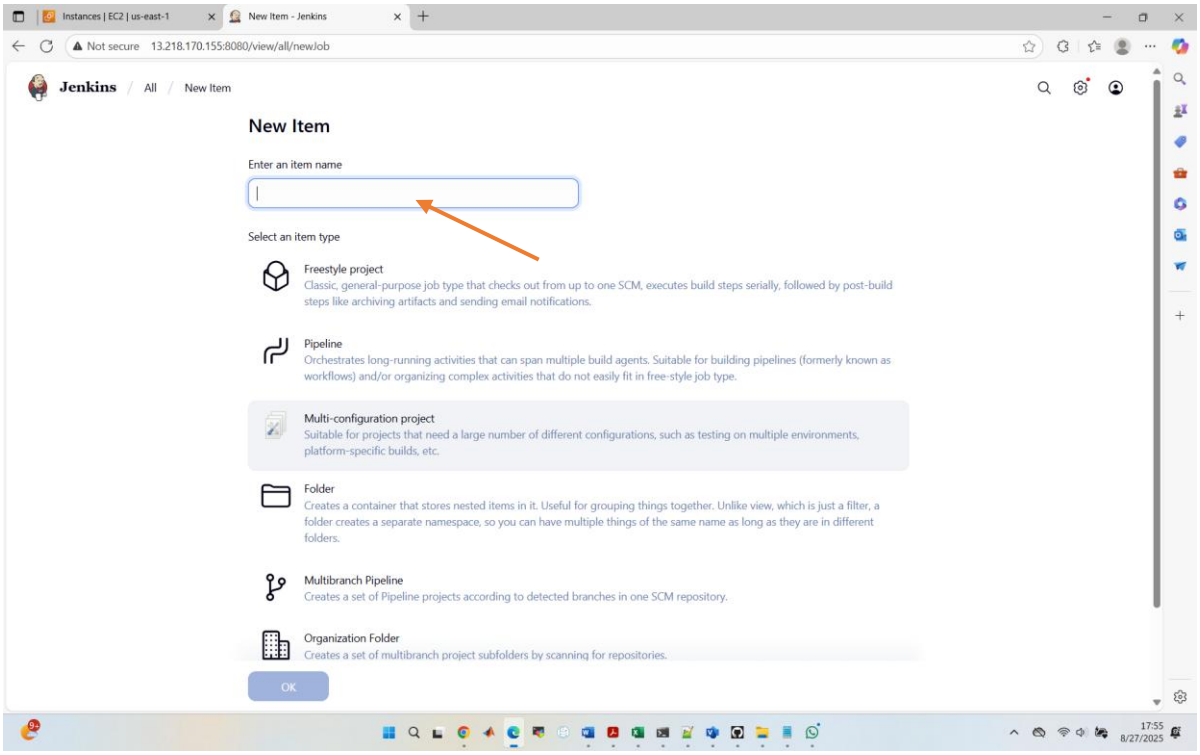
We are now on the Jenkins browser.

STEP 7: Create Pipeline on Jenkins

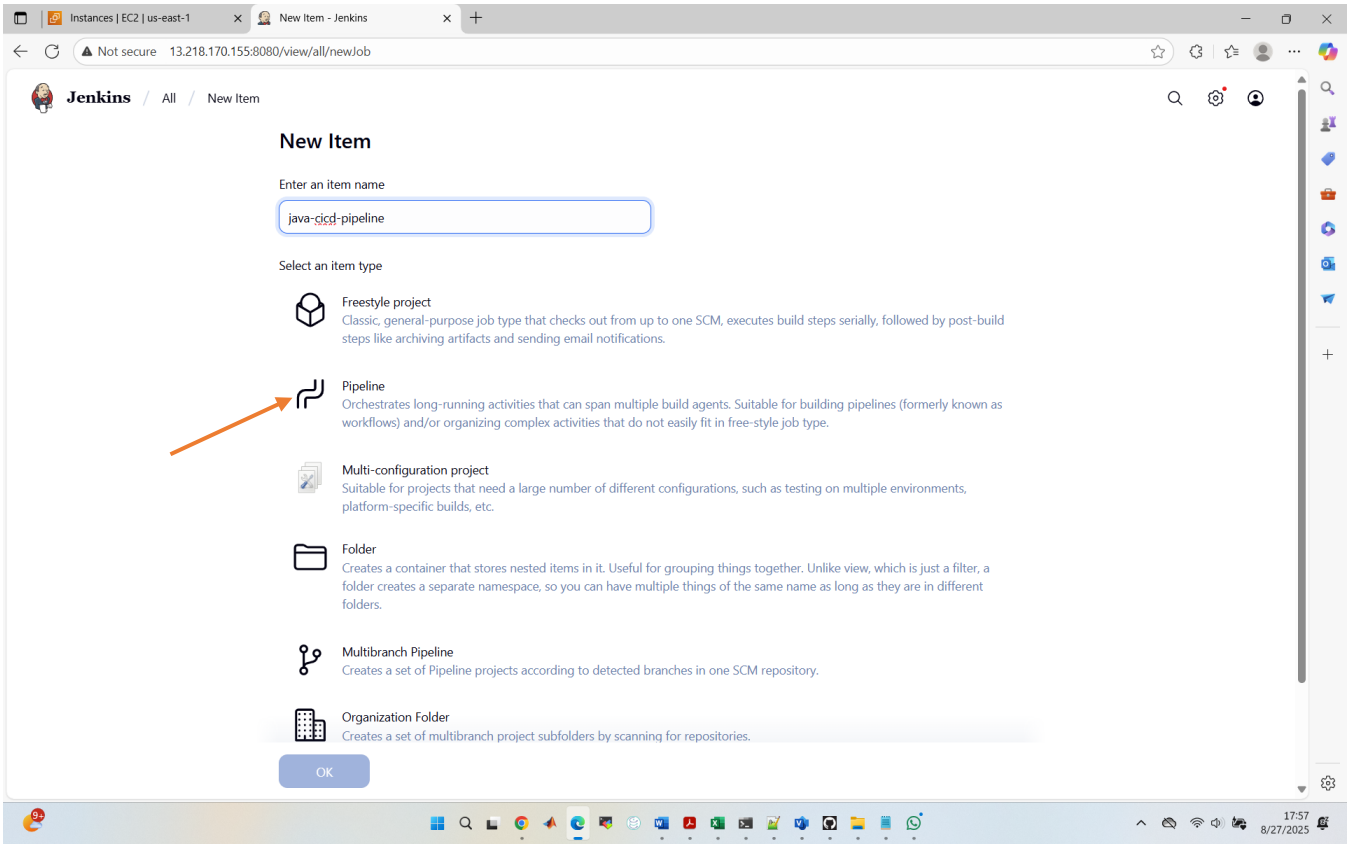
We will create a simple “Hello World” pipeline and test. Then we will later modify the pipeline to meet our project requirements.



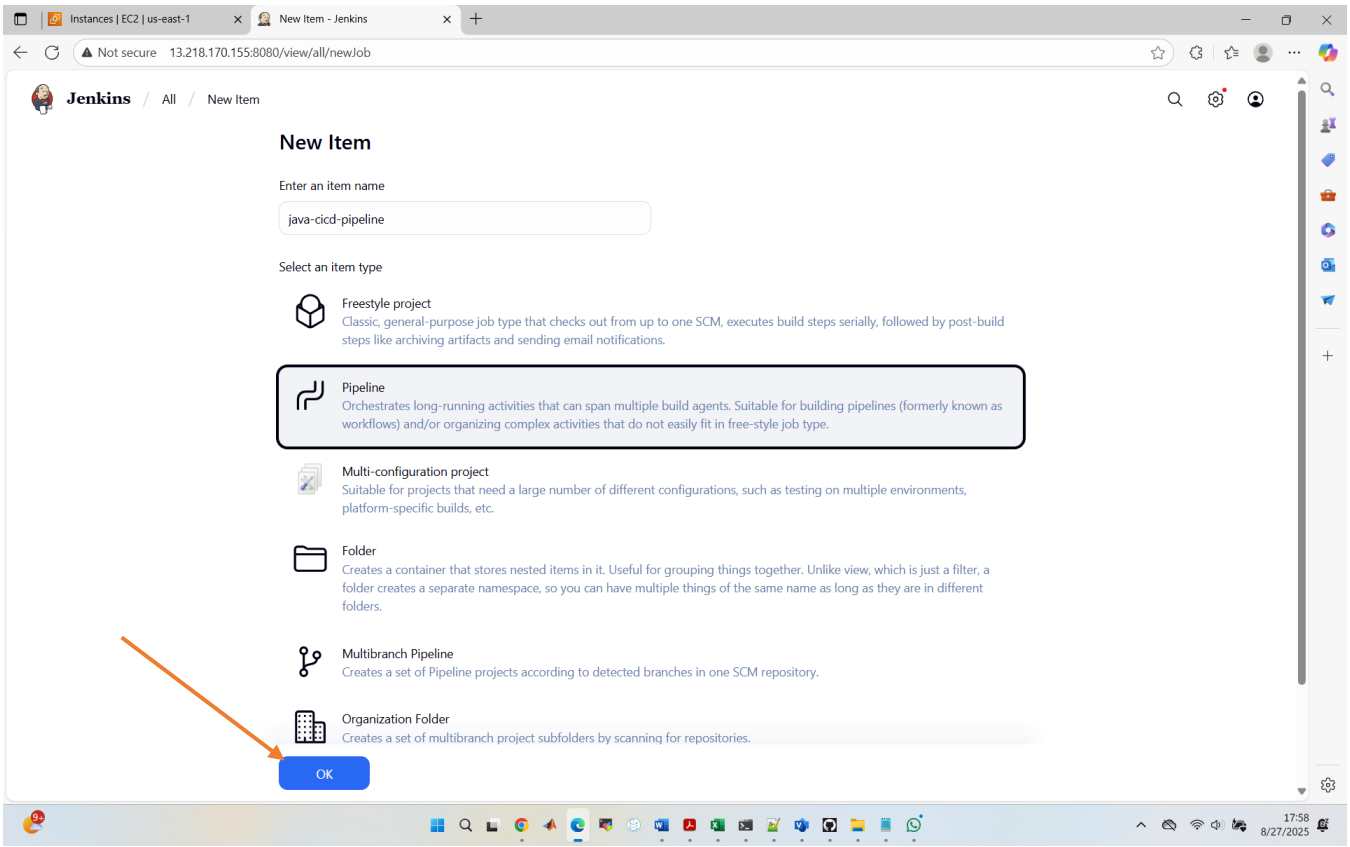
Now, let us create our pipeline. Click on “**New Item**”



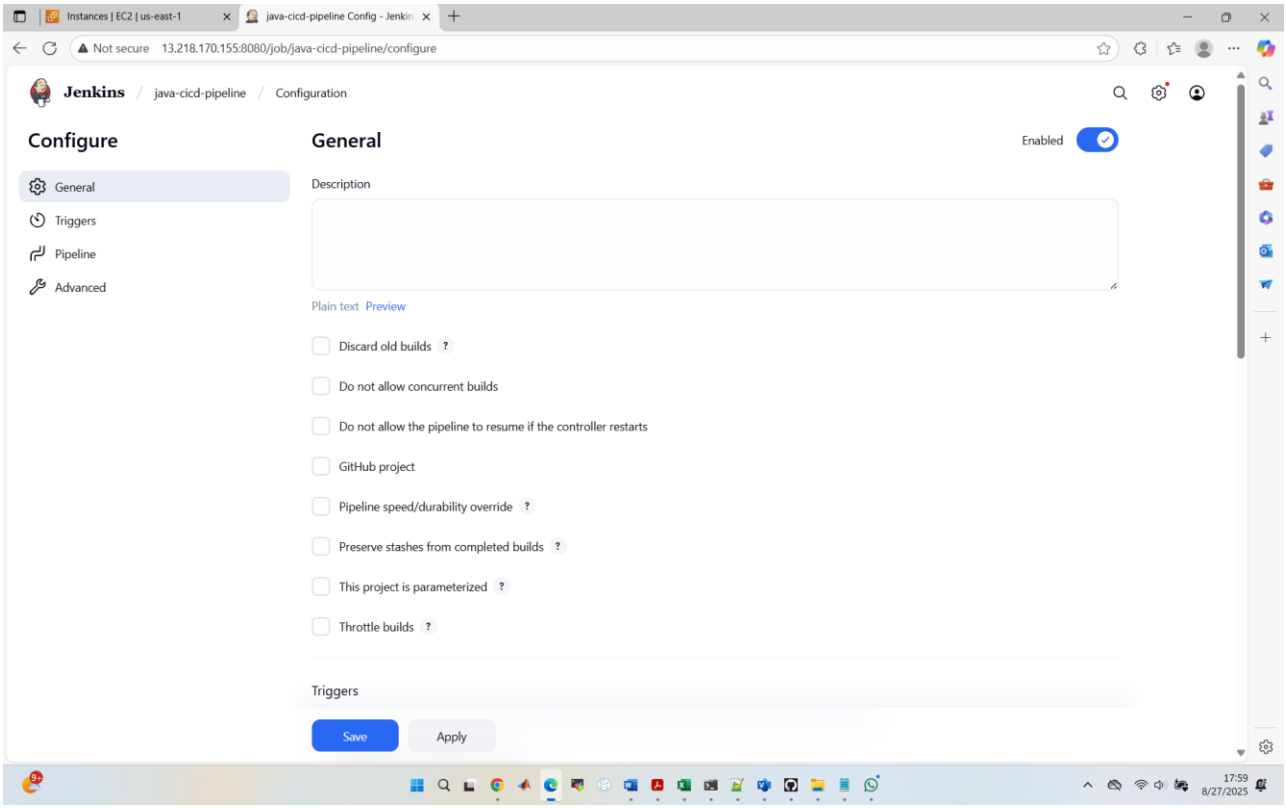
We have to give our pipeline a name, we will call it “**java-cicd-pipeline**”



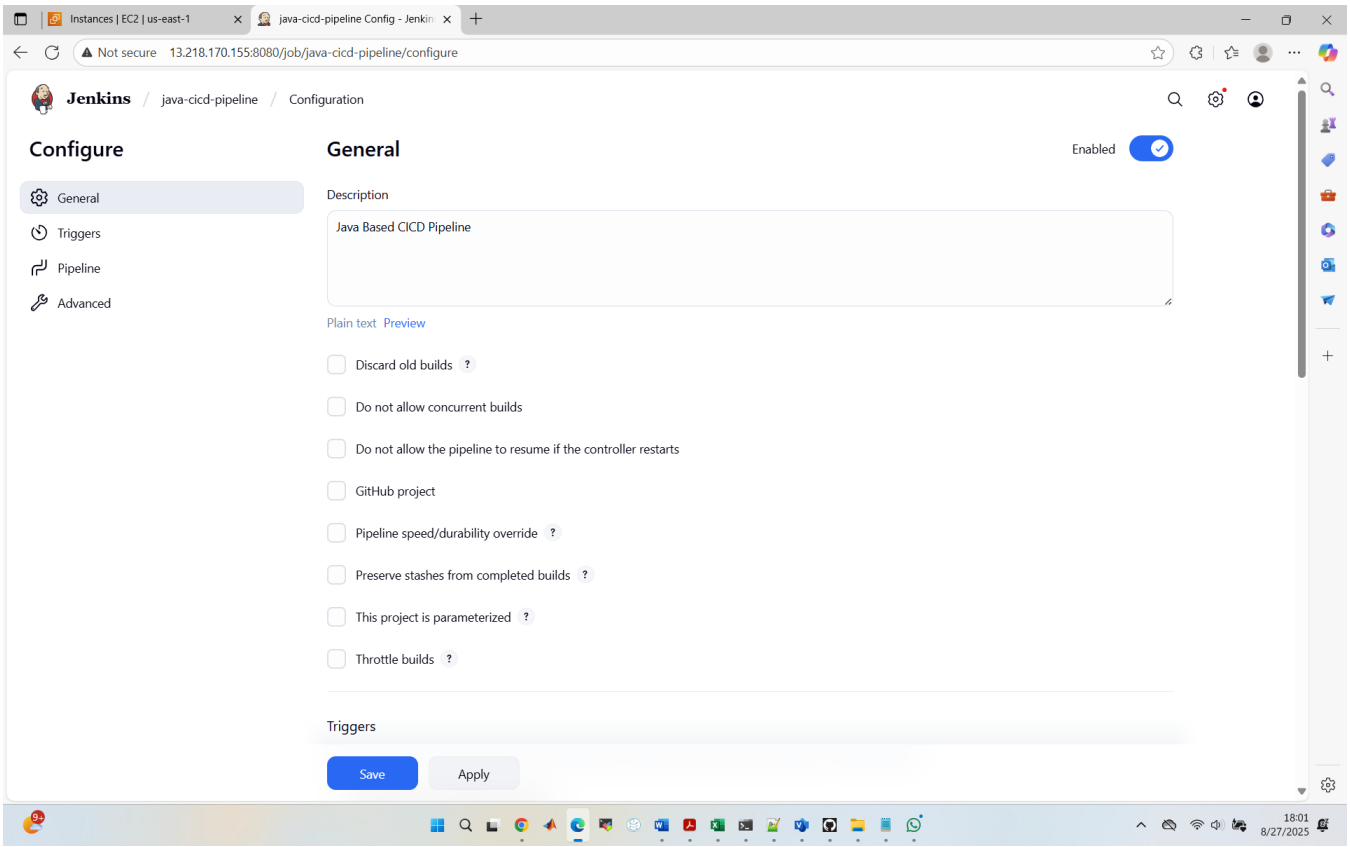
Select “**Pipeline**”



Click on “**OK**”



We can now start to configure our pipeline. For "**Description**", we will add "**Java Based CICD Pipeline**"



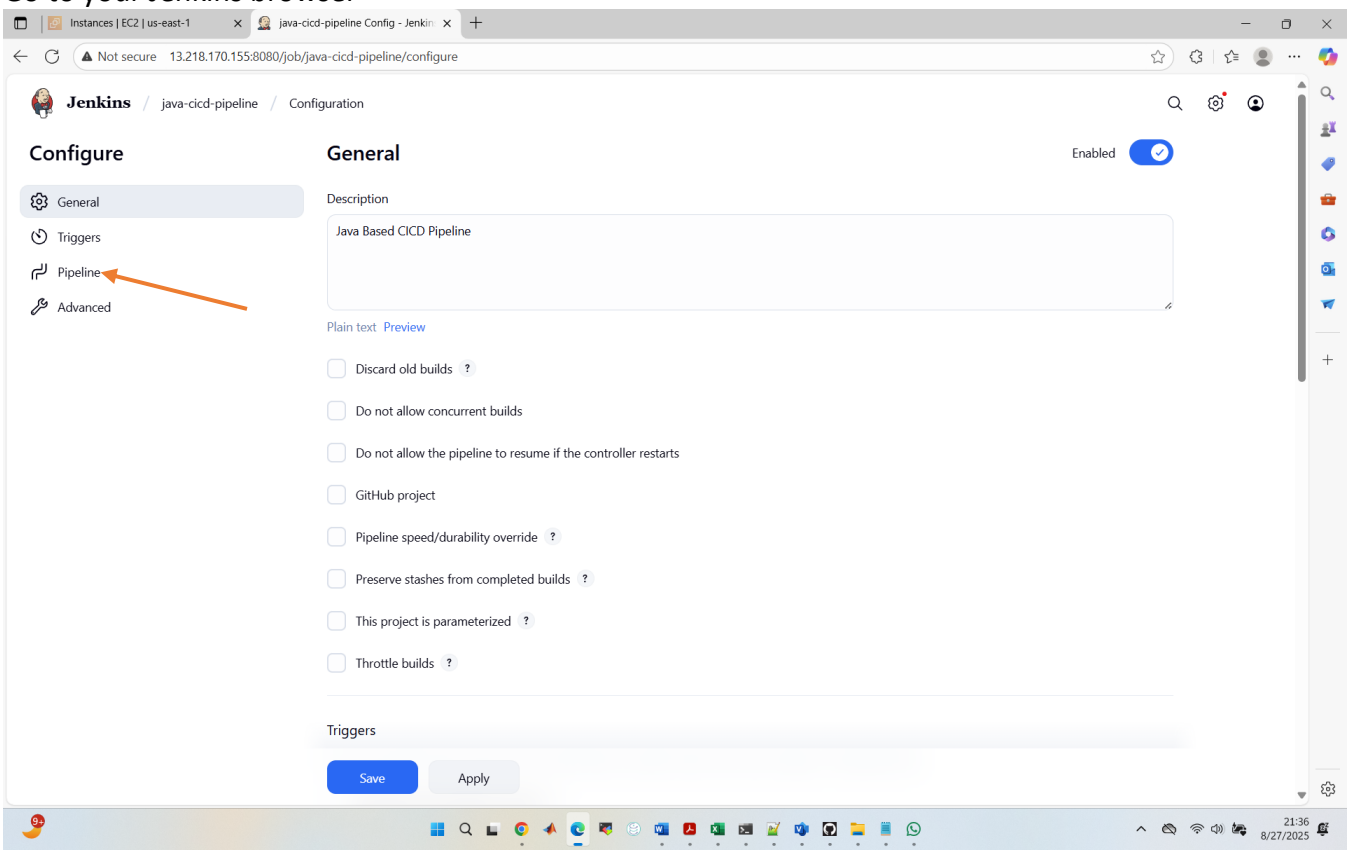
Now we can write the Pipeline. There are two ways to write the pipeline script: **Declarative pipeline**-simplified way Scripted Pipeline and the **Groovy Script**.

Declarative Pipeline: Structured and simplified way to define pipelines in Jenkins. Uses predefined syntax with specific keywords. Great for standardizing pipeline definitions and simpler requirements. Offers less flexibility compared to Scripted Pipeline.

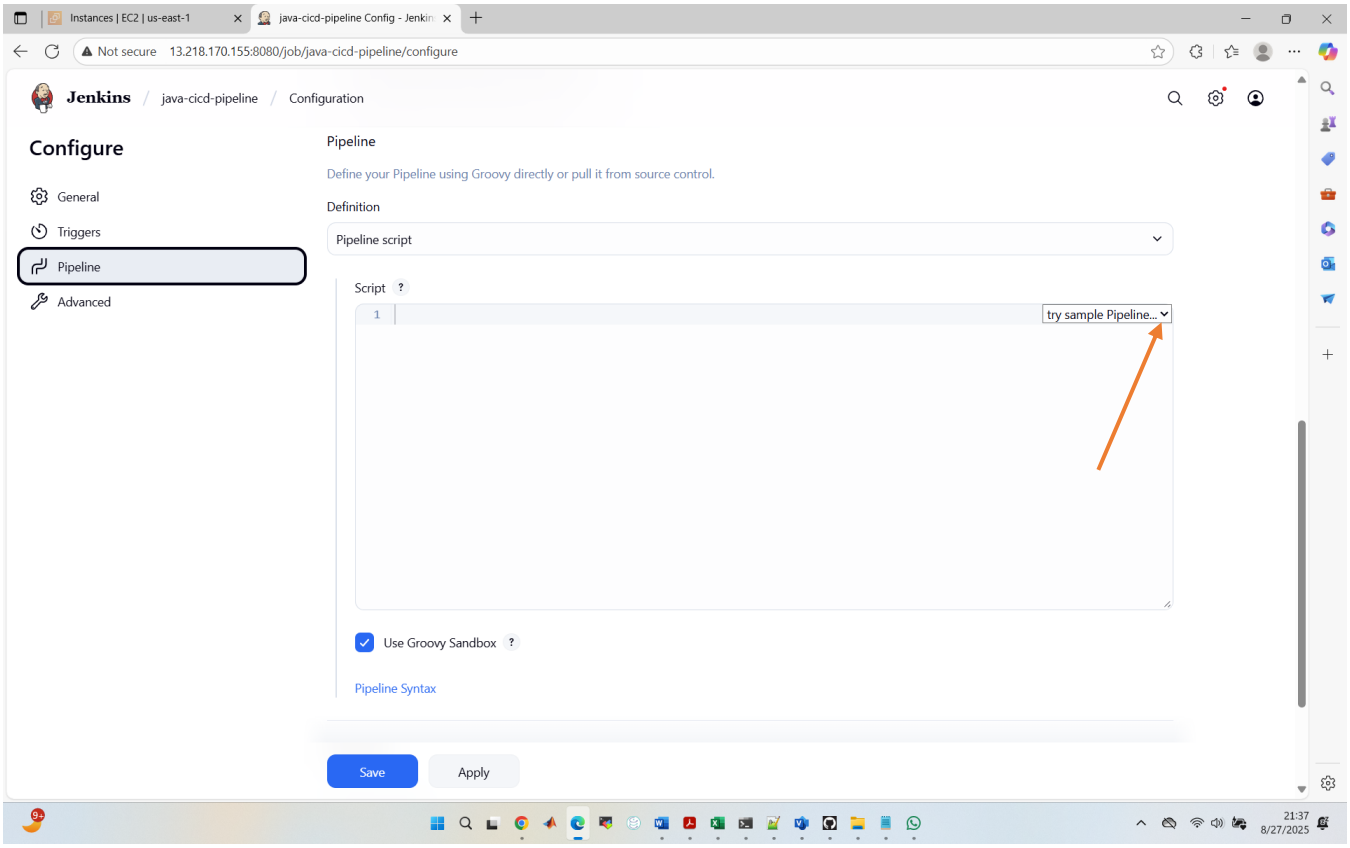
Scripted Pipeline: Allows writing pipelines as Groovy scripts. Provides more flexibility and control with Groovy scripting. Suitable for complex scenarios and advanced control flow. Can be verbose and complex compared to Declarative Pipeline.

In this project, we will be using declarative pipeline, where we are going to use simple English patterns. This will be demonstrated with a simple "**Hello World**" script.

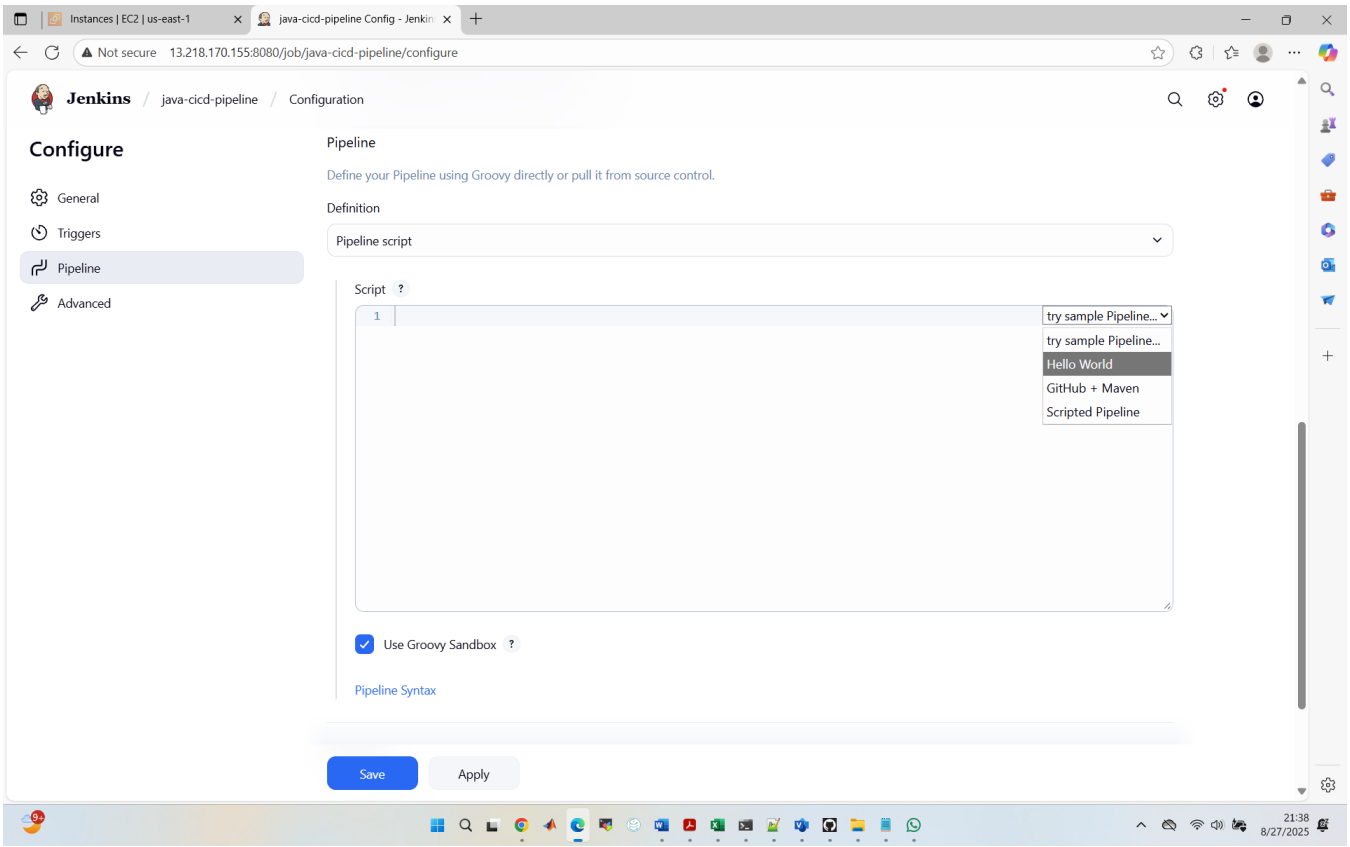
Go to your Jenkins browser



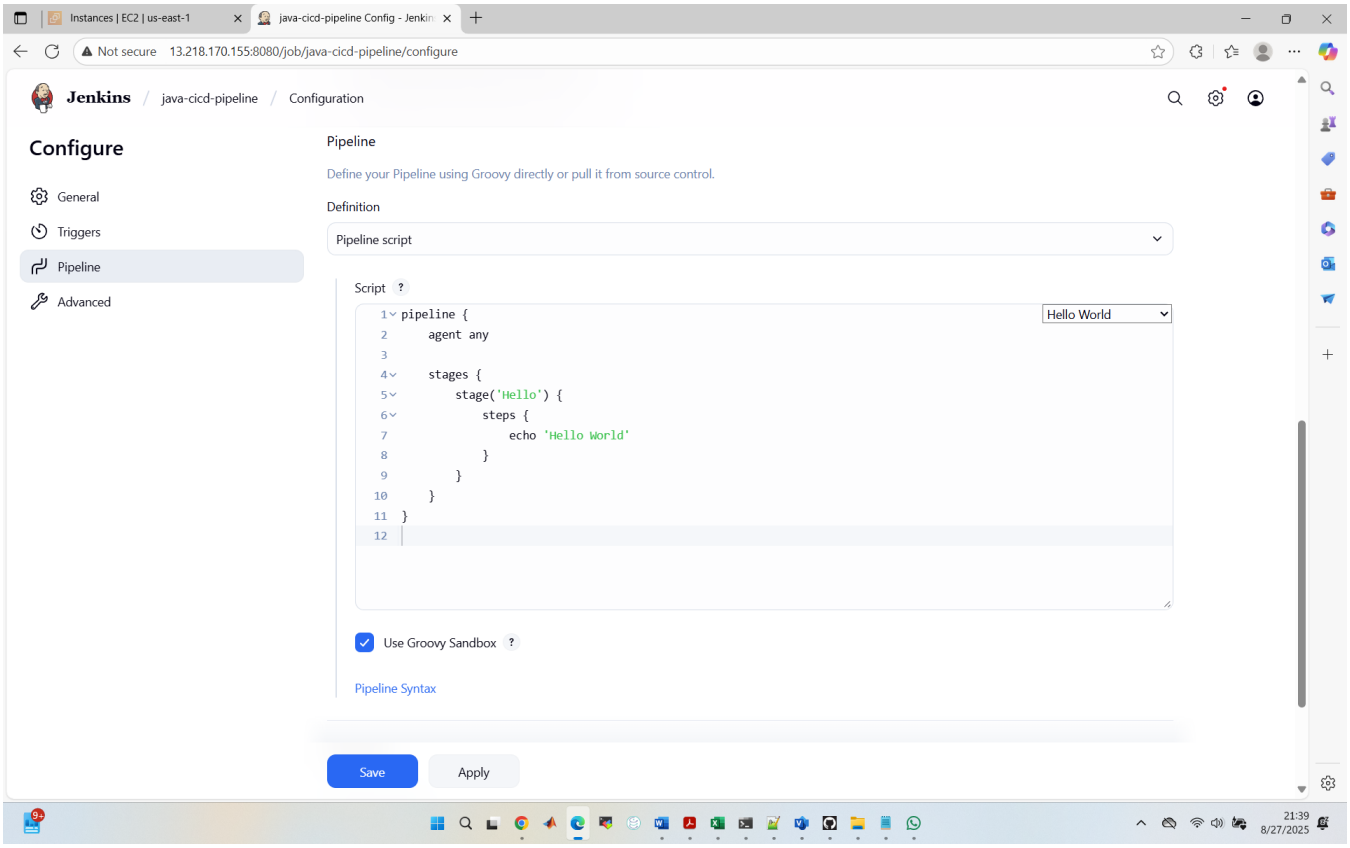
Click on "**Pipeline**" on the left-hand side



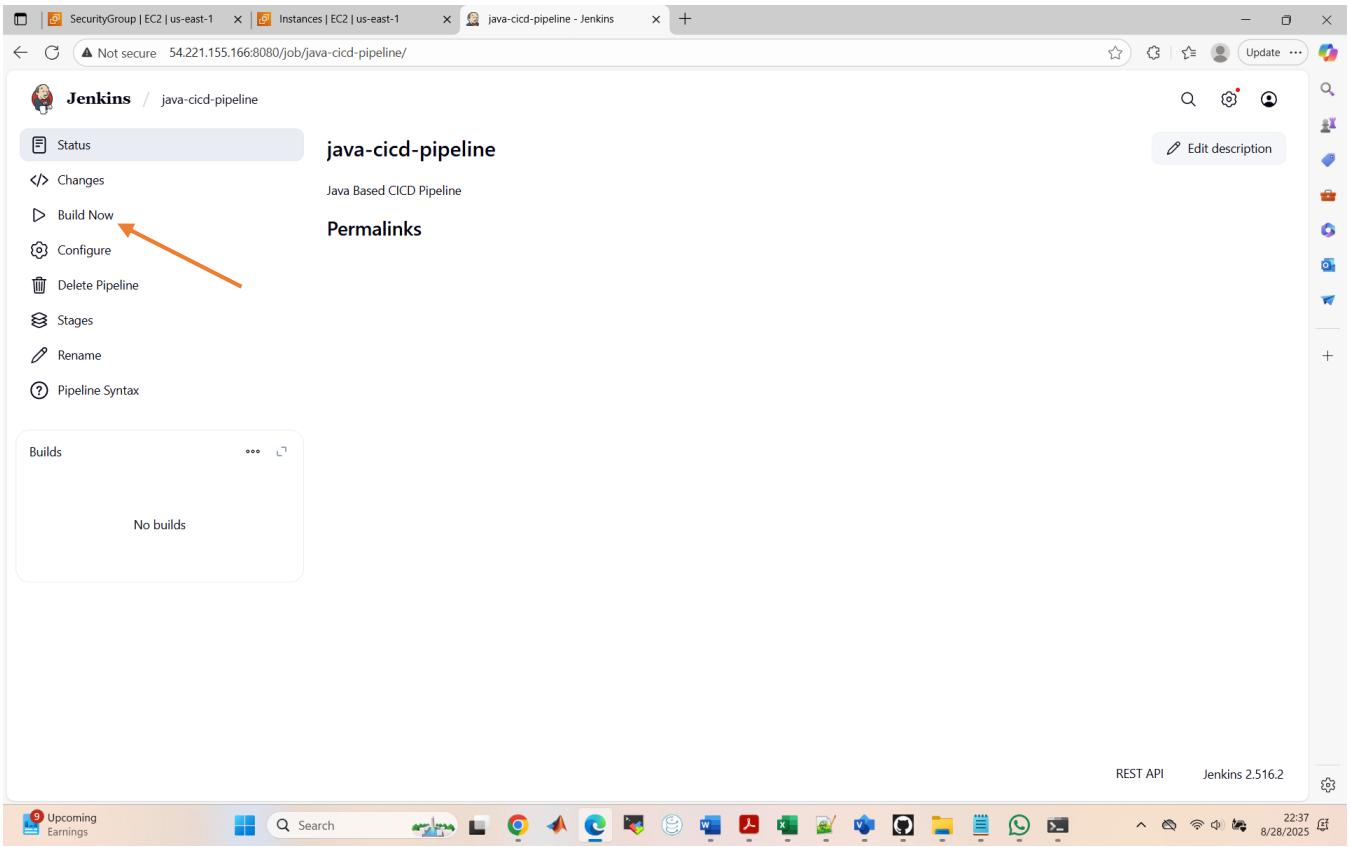
Click on the drop down on “**Try sample Pipeline**”



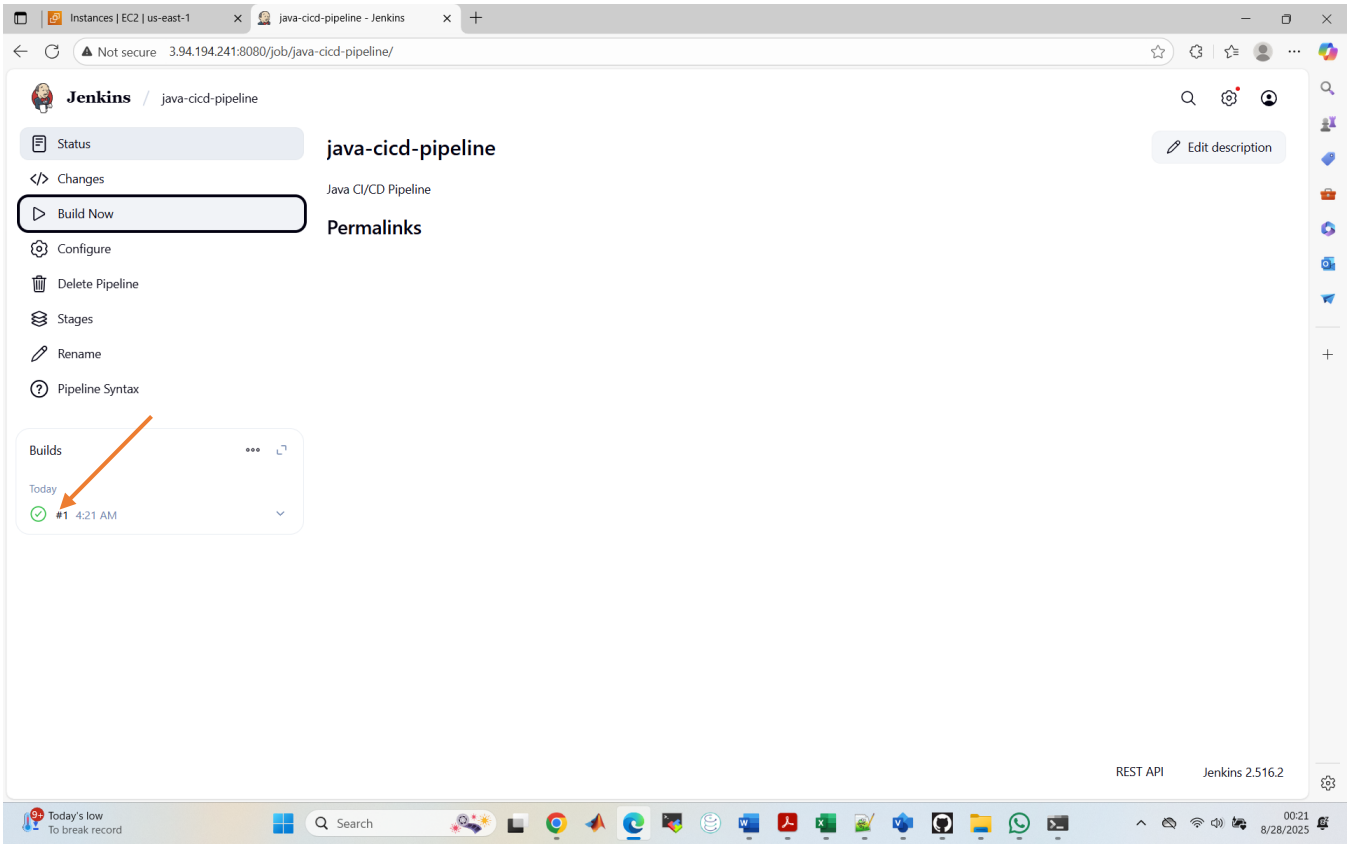
Select “**Hello World**”



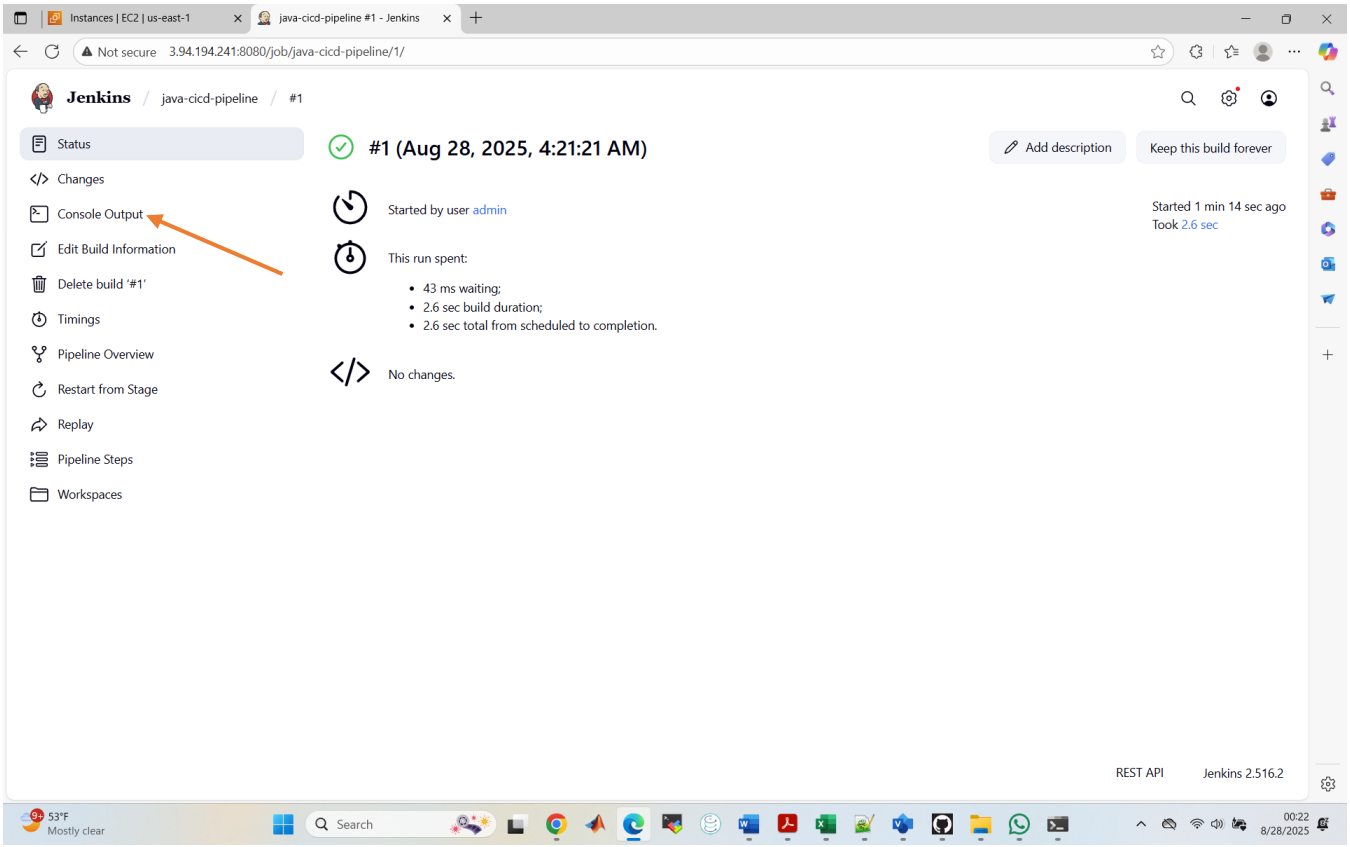
Click on “**Apply**”, followed by clicking on “**Save**”



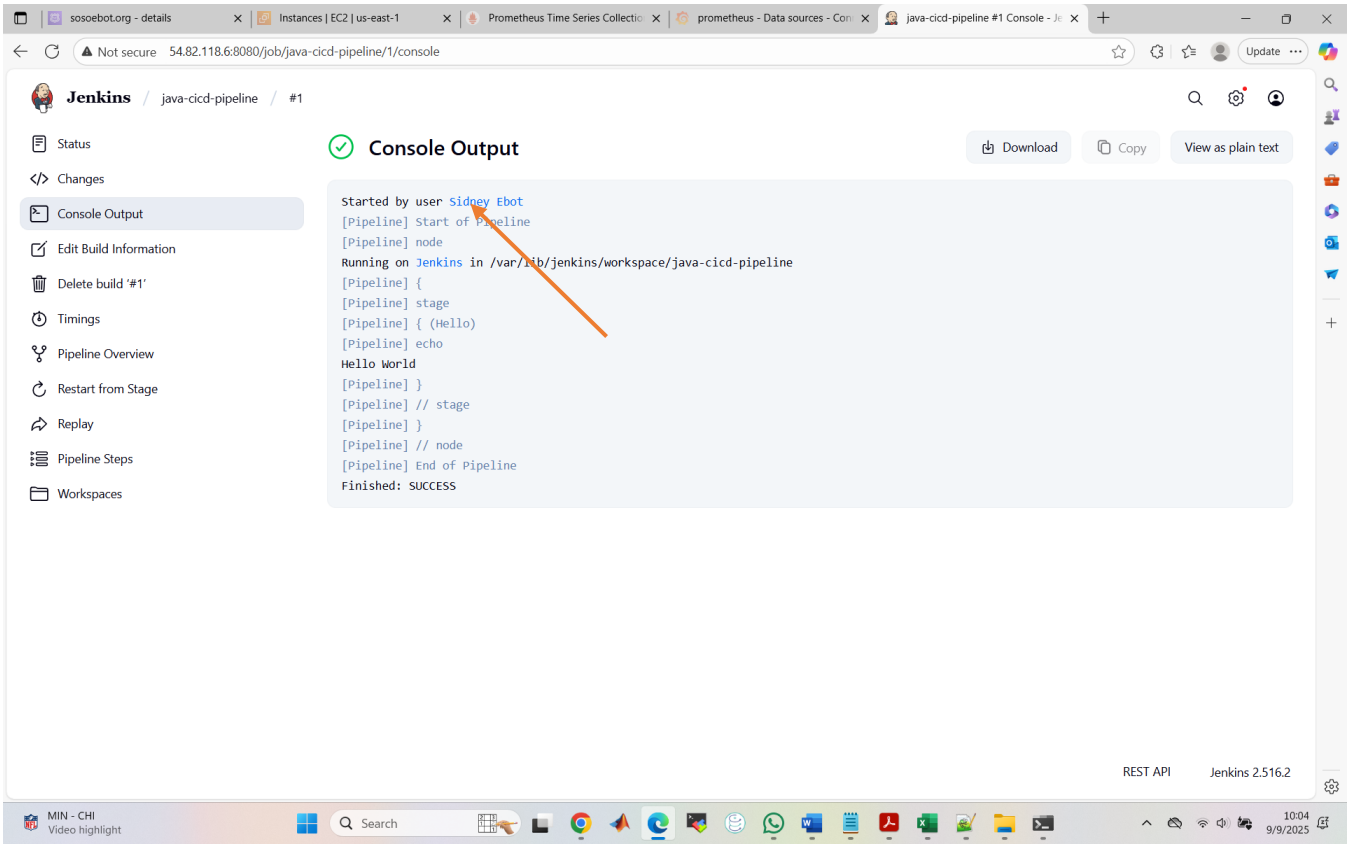
On the left-hand side of the dashboard, there are many options. We will build the pipeline by clicking on “**Build Now**” and refresh the page.



Click on "**#1**"



Then click on "**Console Output**"



Here you can see the user, that is me because I am the one. If the code is pushed from GitHub, the developer who pushed the code names will appear there as the user.

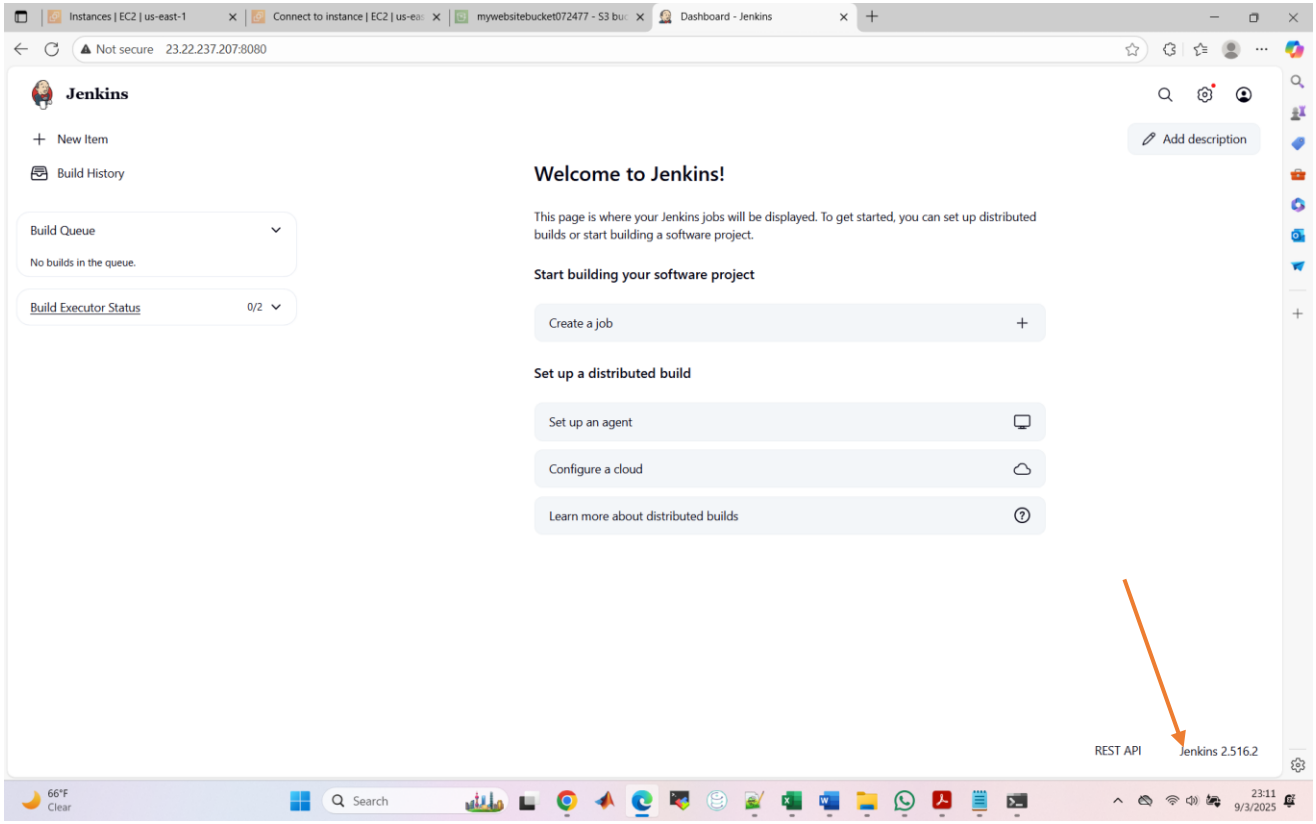
STEP 8: Install and Configure JDK on Jenkins

We will install JDK plugins and configure JDK on Jenkins

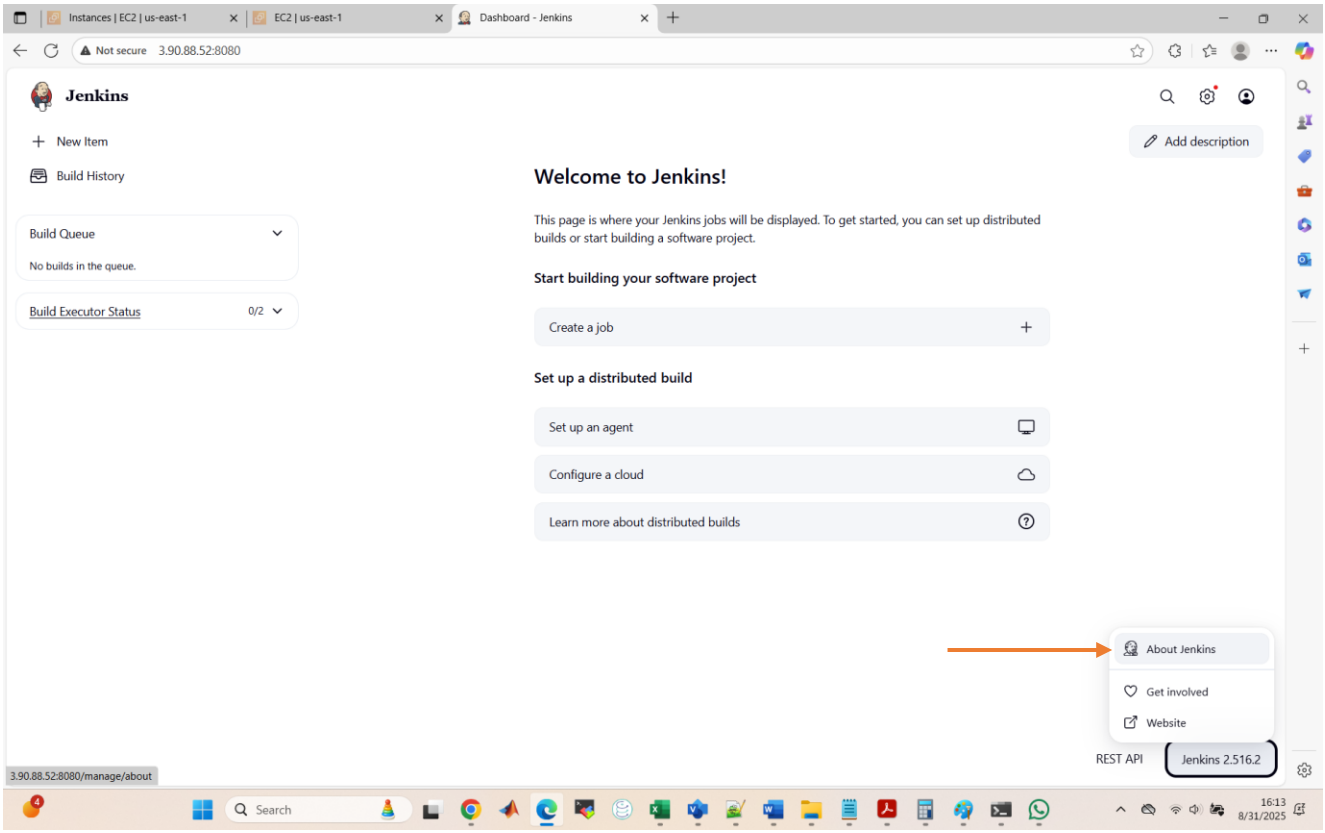
PART 1: Install JDK Plugin

We will install the plugin "**Eclipse Temurin Installer**"

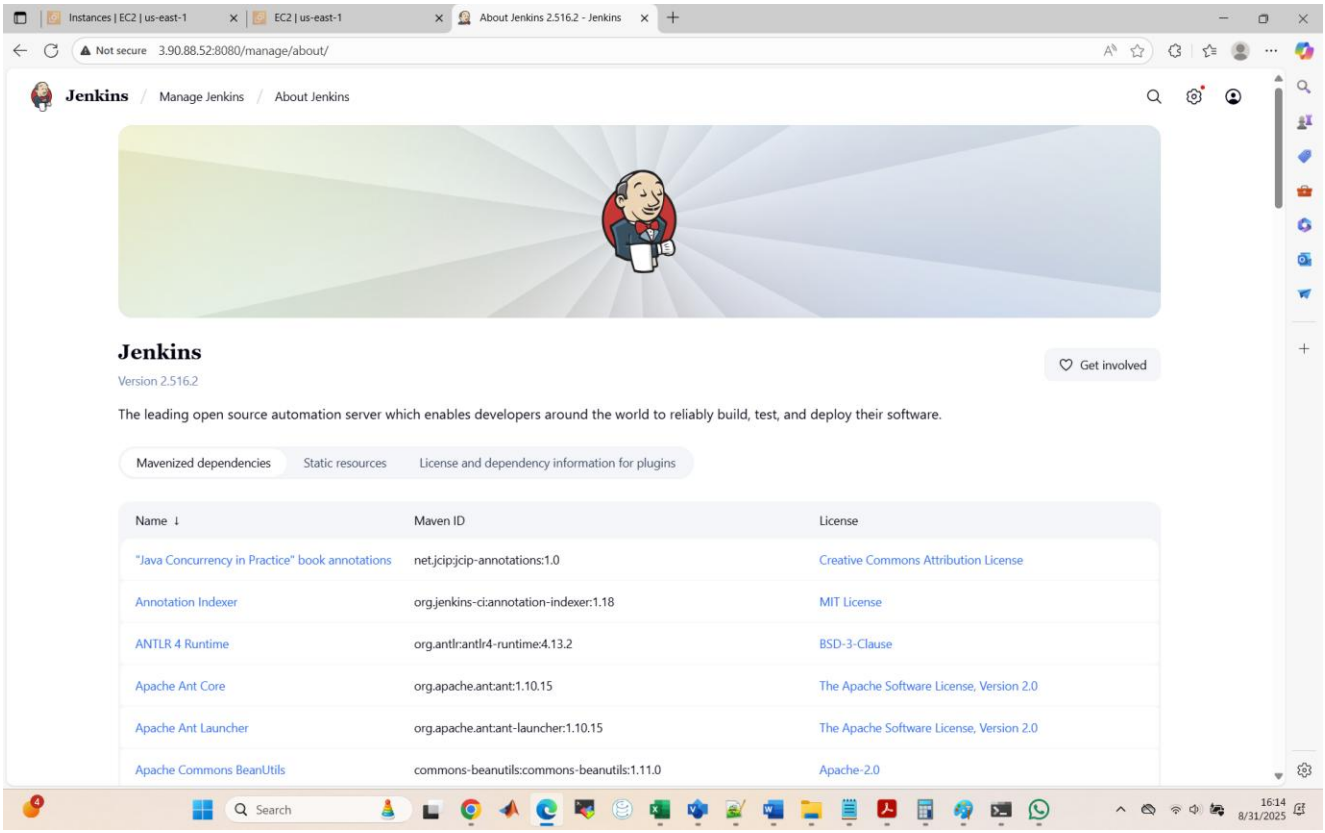
Go to "**Manage Jenkins**"



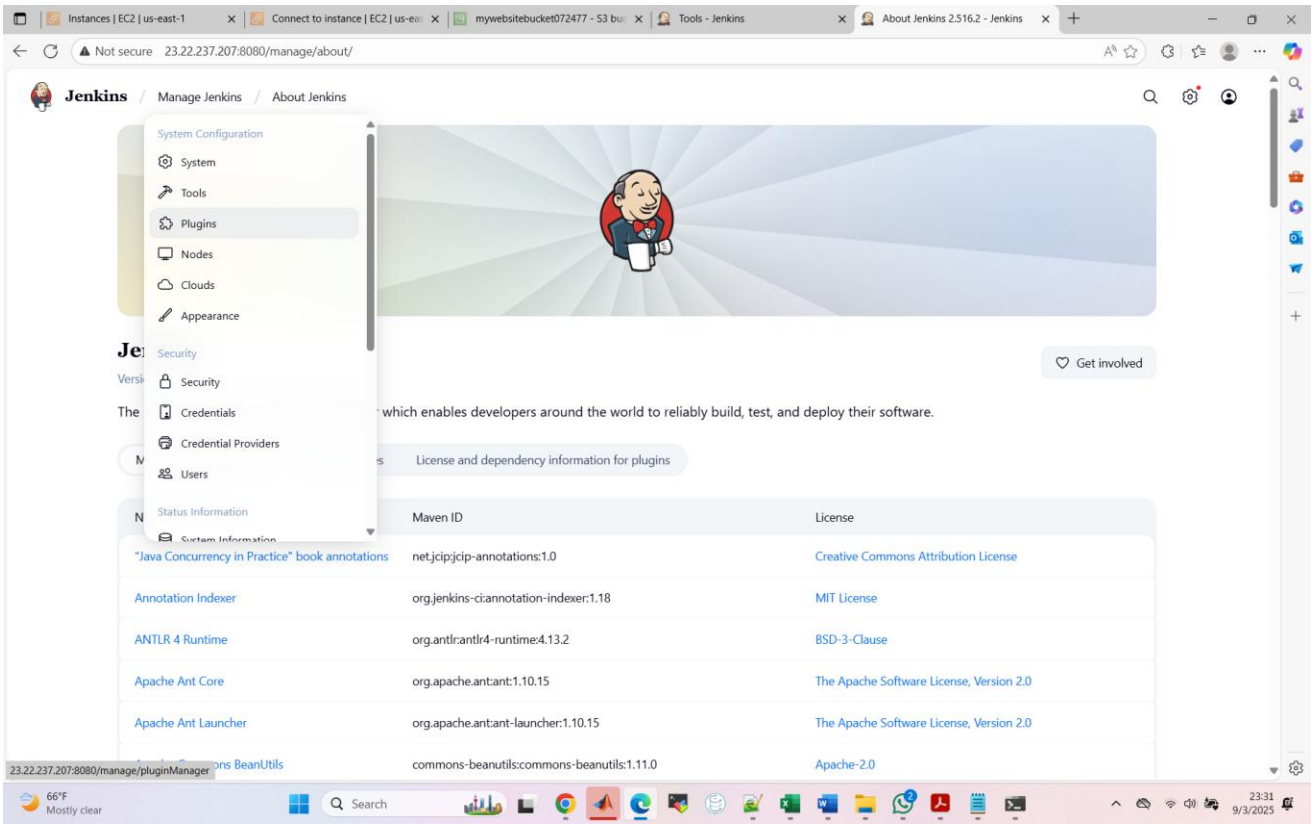
Click on "**Jenkins 2.516.2**"



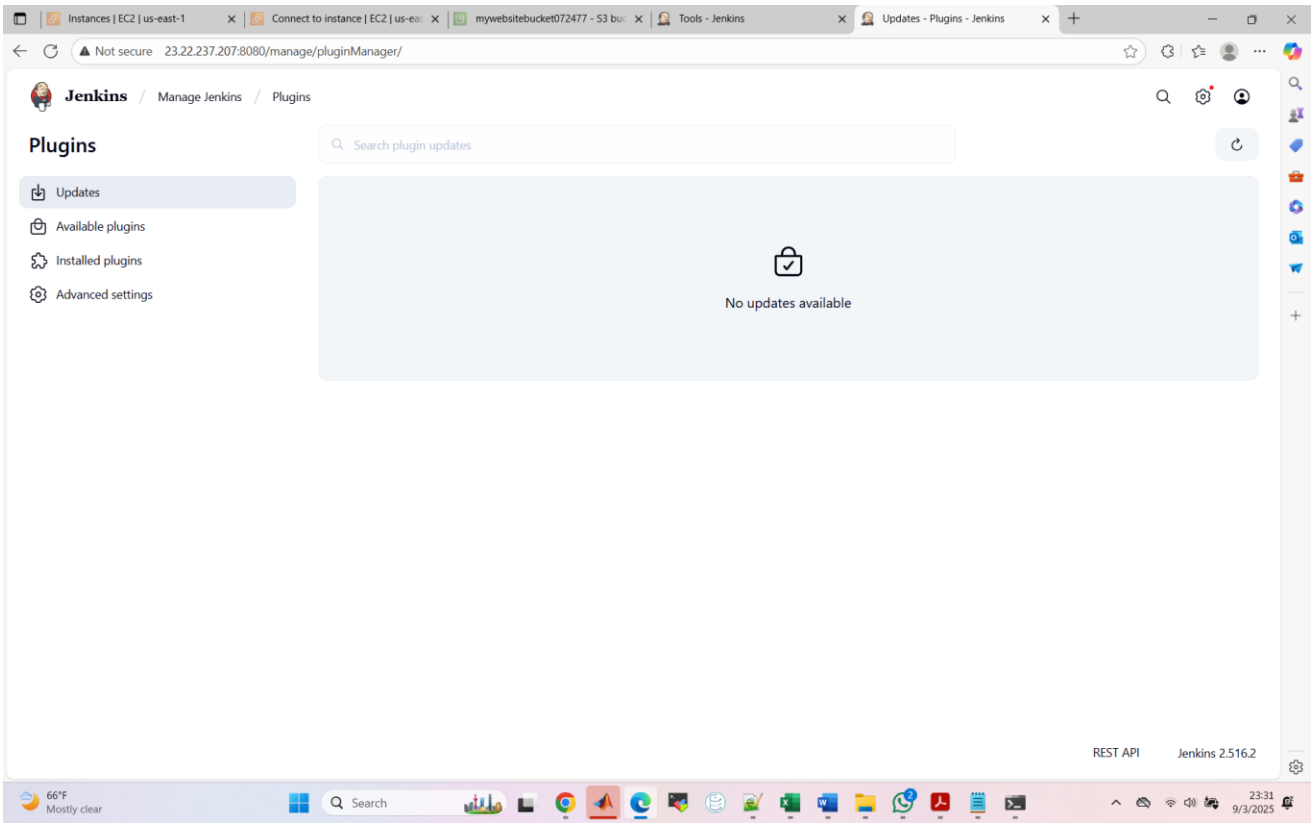
Select “**About Jenkins**”



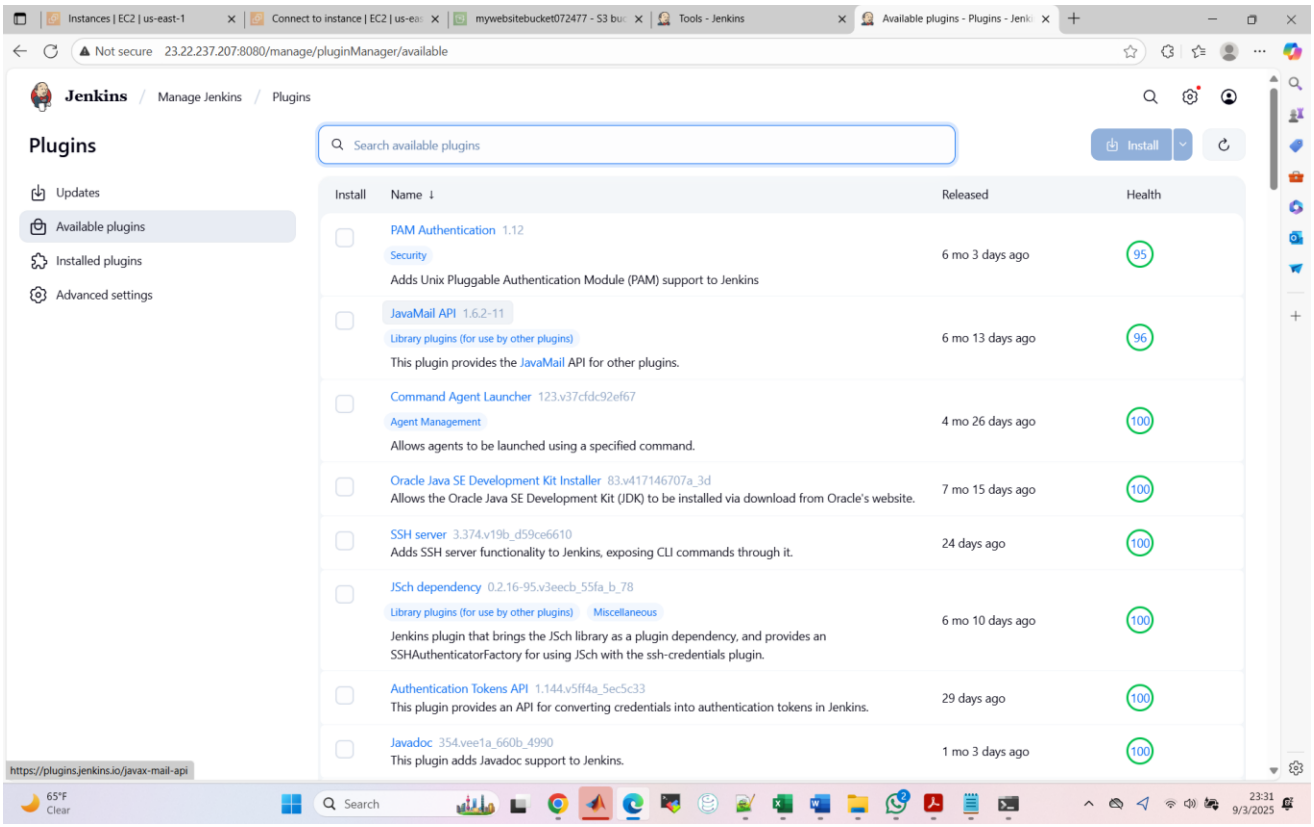
Click on “**Manage Jenkins**”



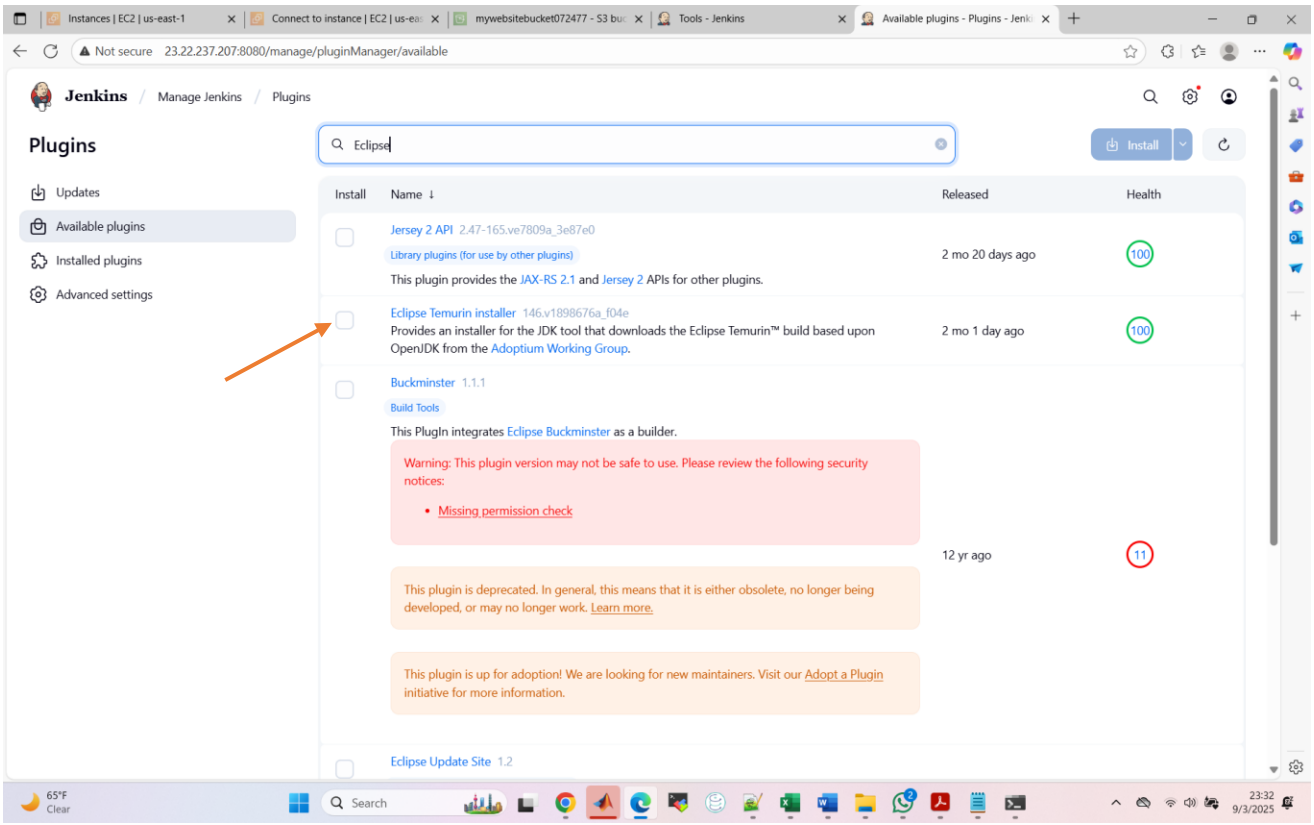
Select “**Plugins**”



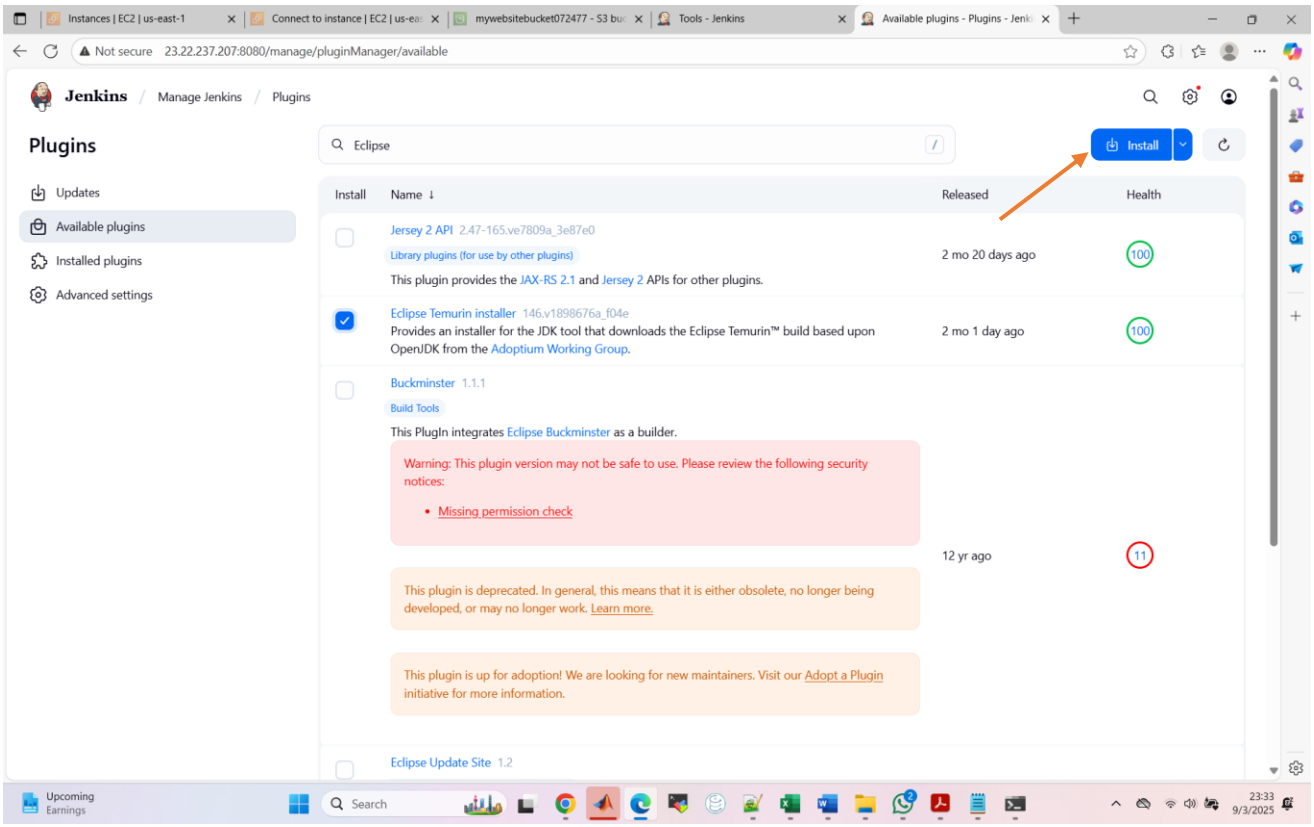
Click on “**Available Plugins**”



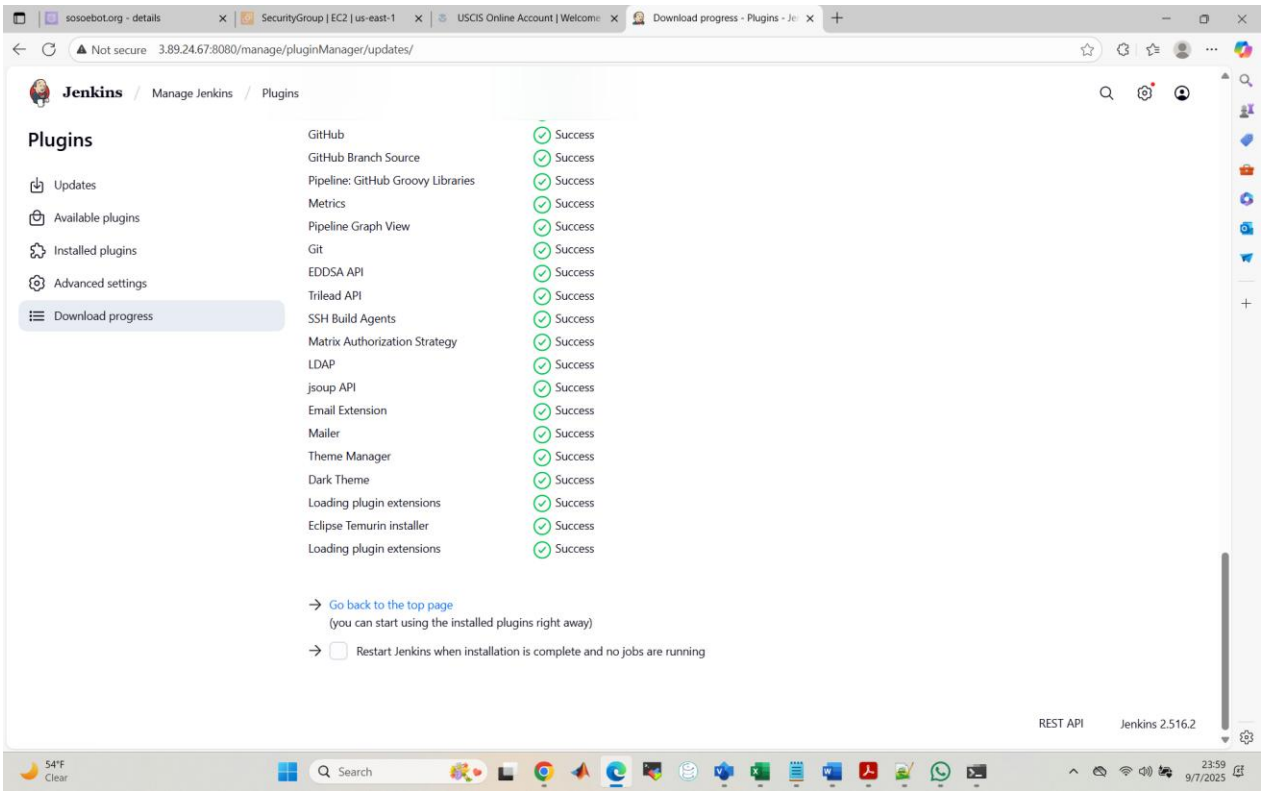
Search for “**Eclipse**”



Select “**Eclipse Temurin Installer**”



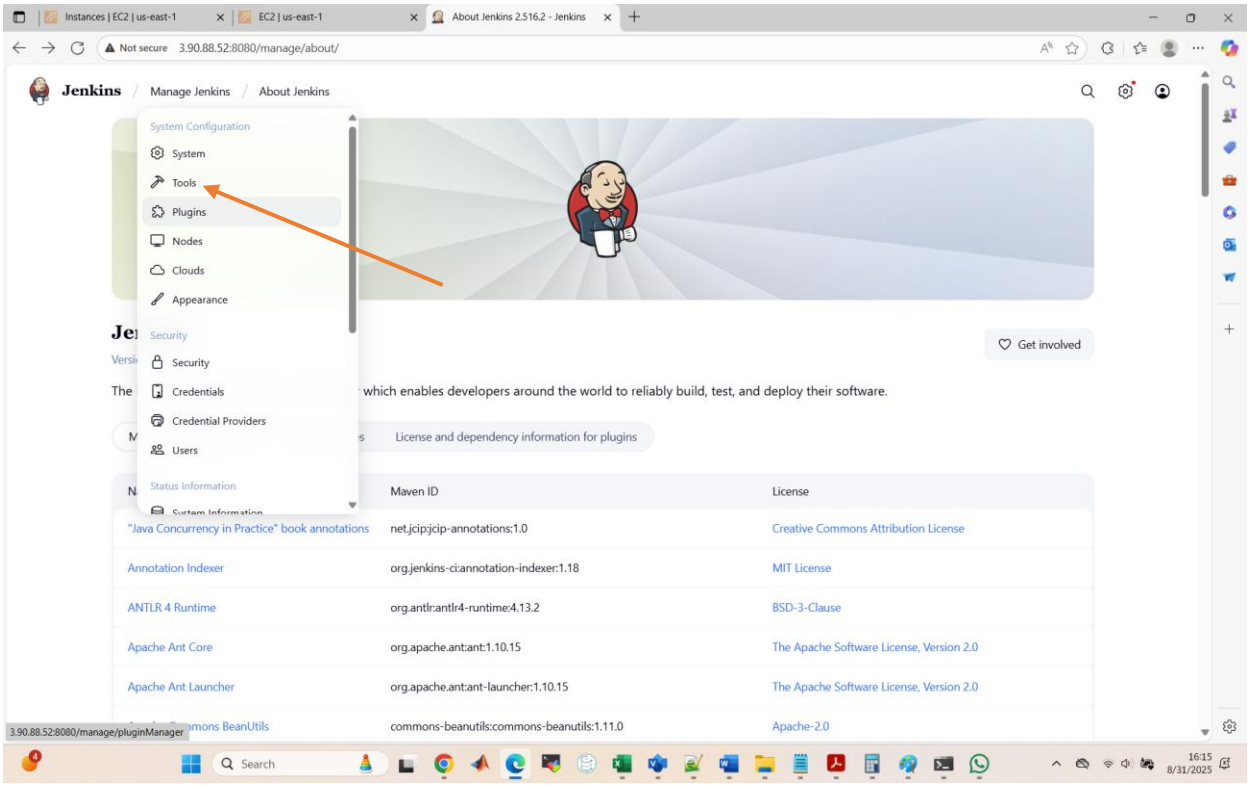
Click on "**Install**"



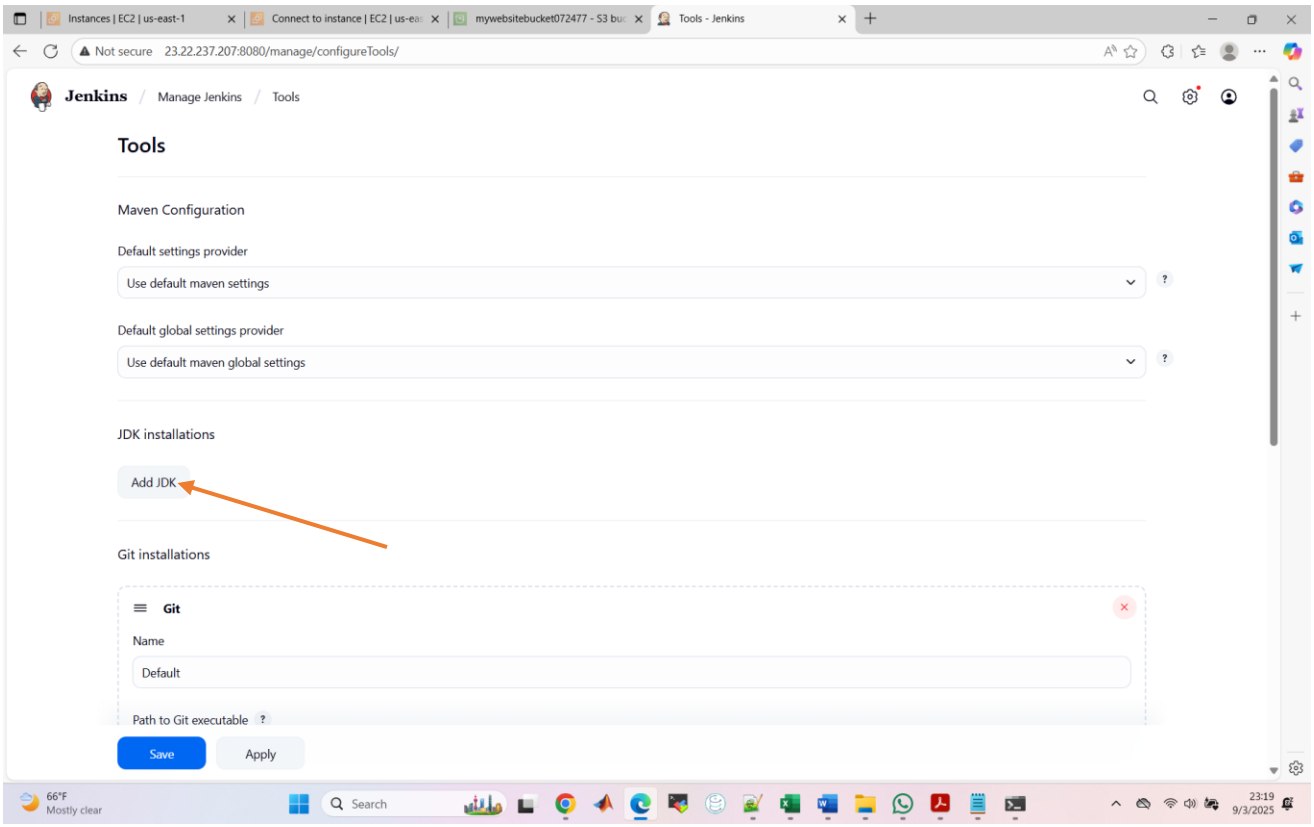
PART 2: Configure JDK as a tool on Jenkins

We will now configure JDK on Jenkins. We will use JDK-11 and we will install from adoptium.net.

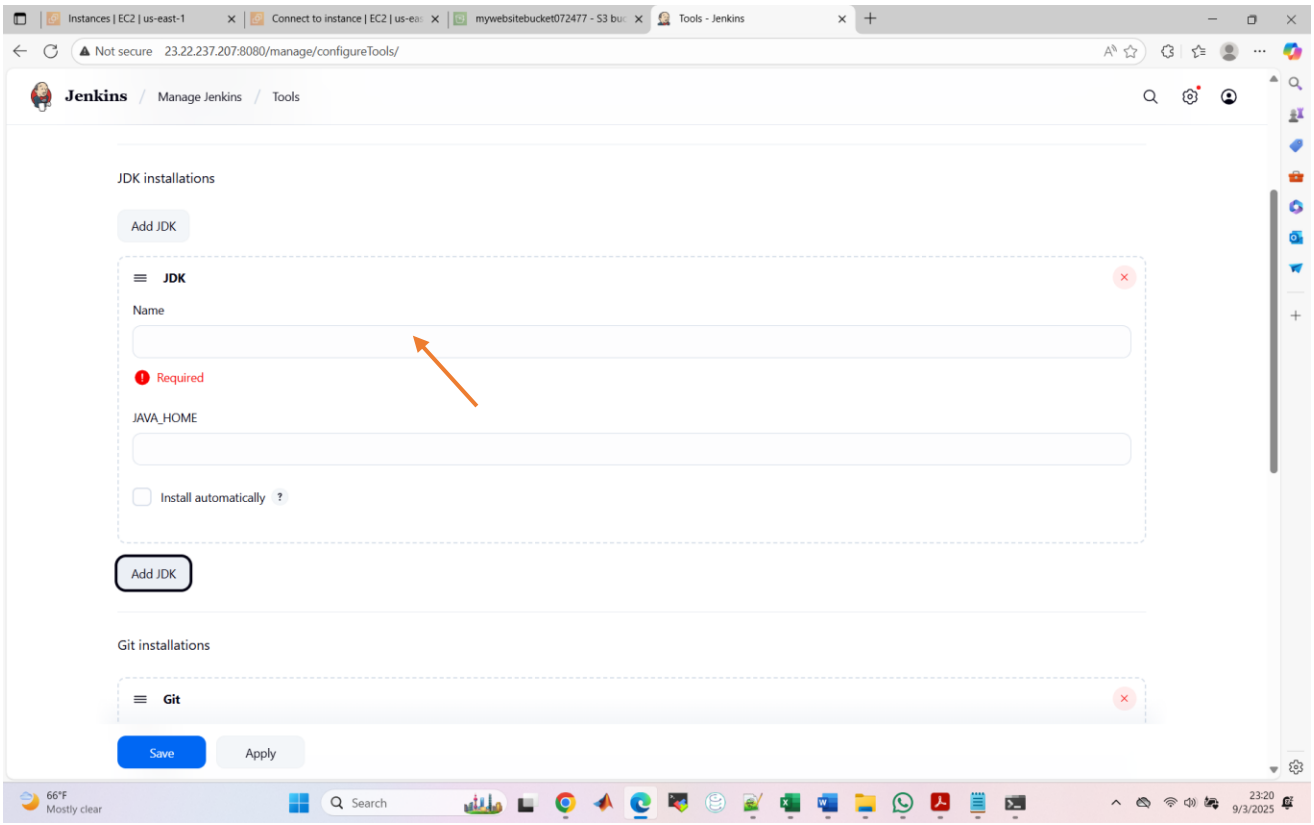
Click on "**Manage Jenkins**"



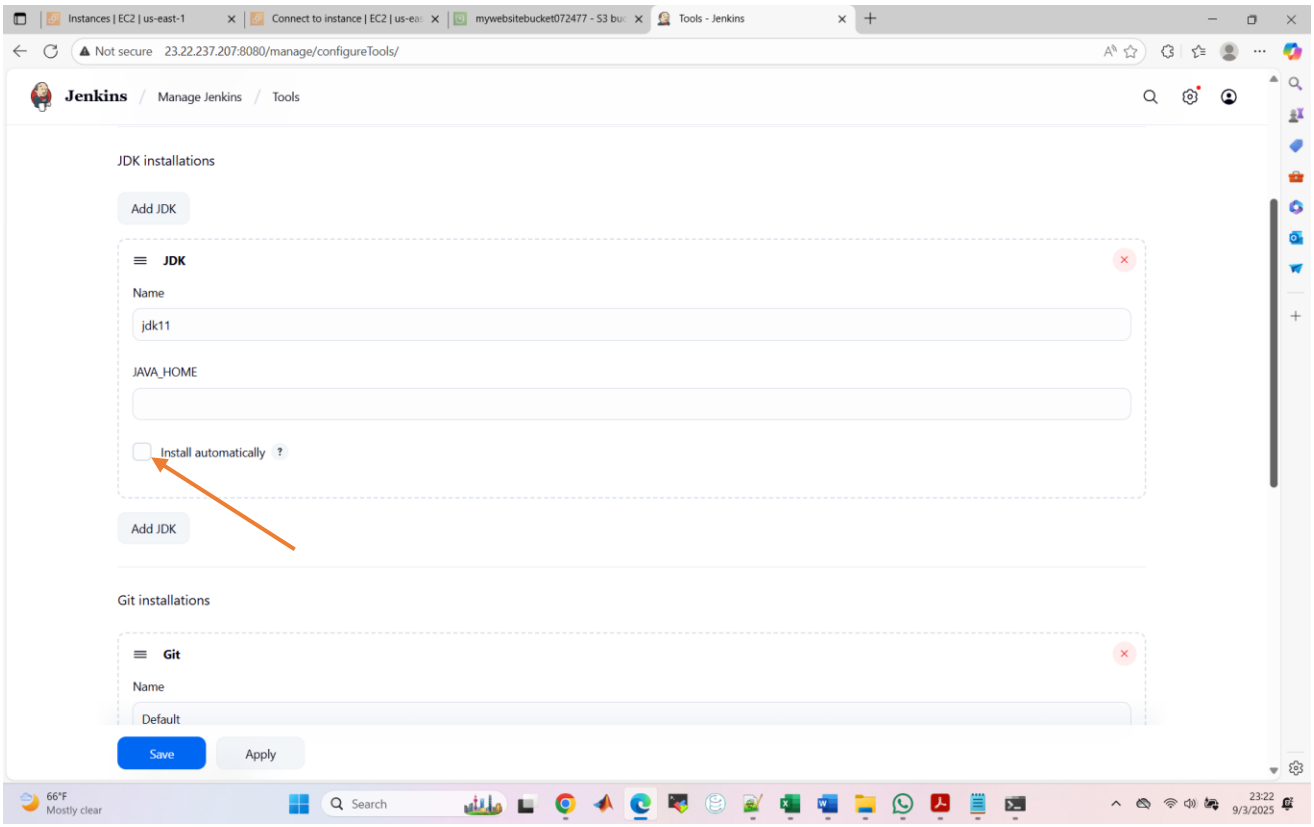
Click on "**tools**"



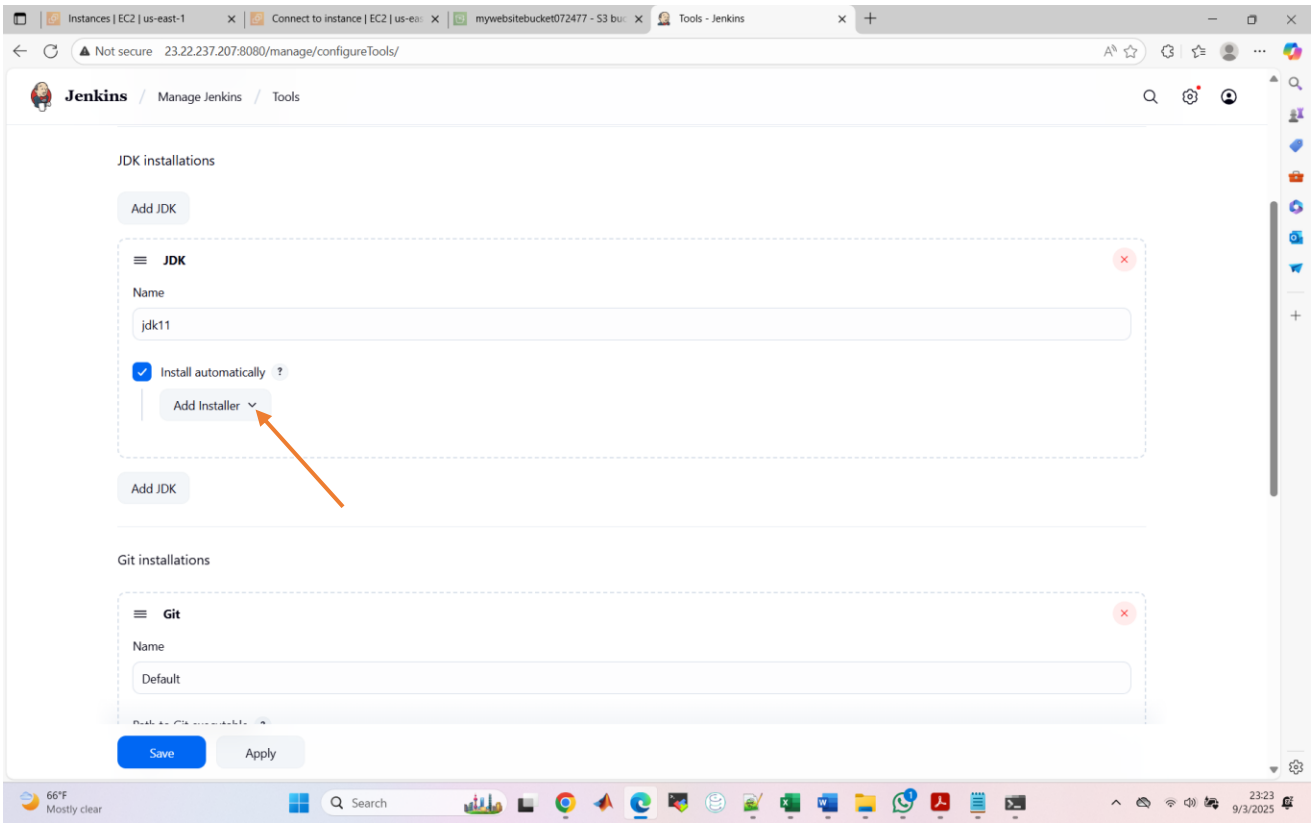
Click on "**Add JDK**"



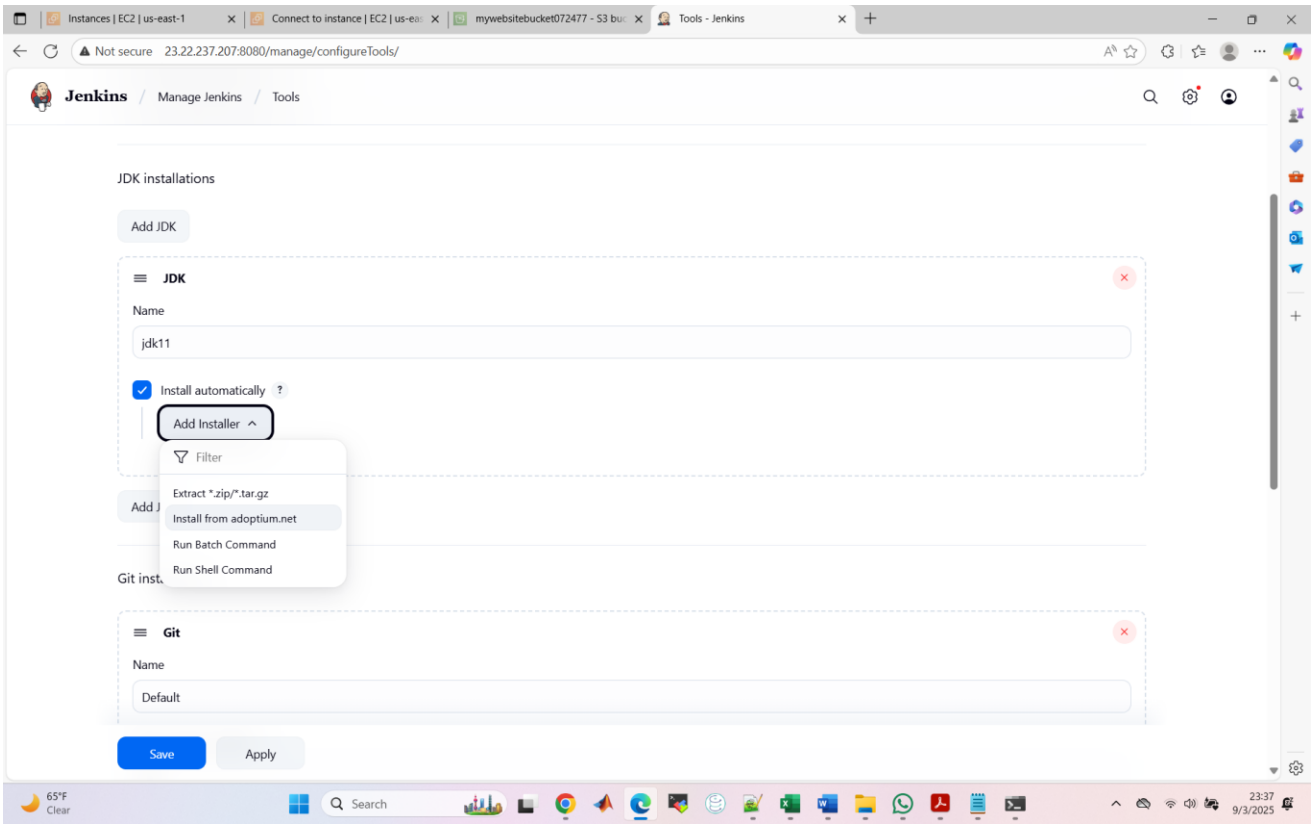
On Name enter "**jdk11**"



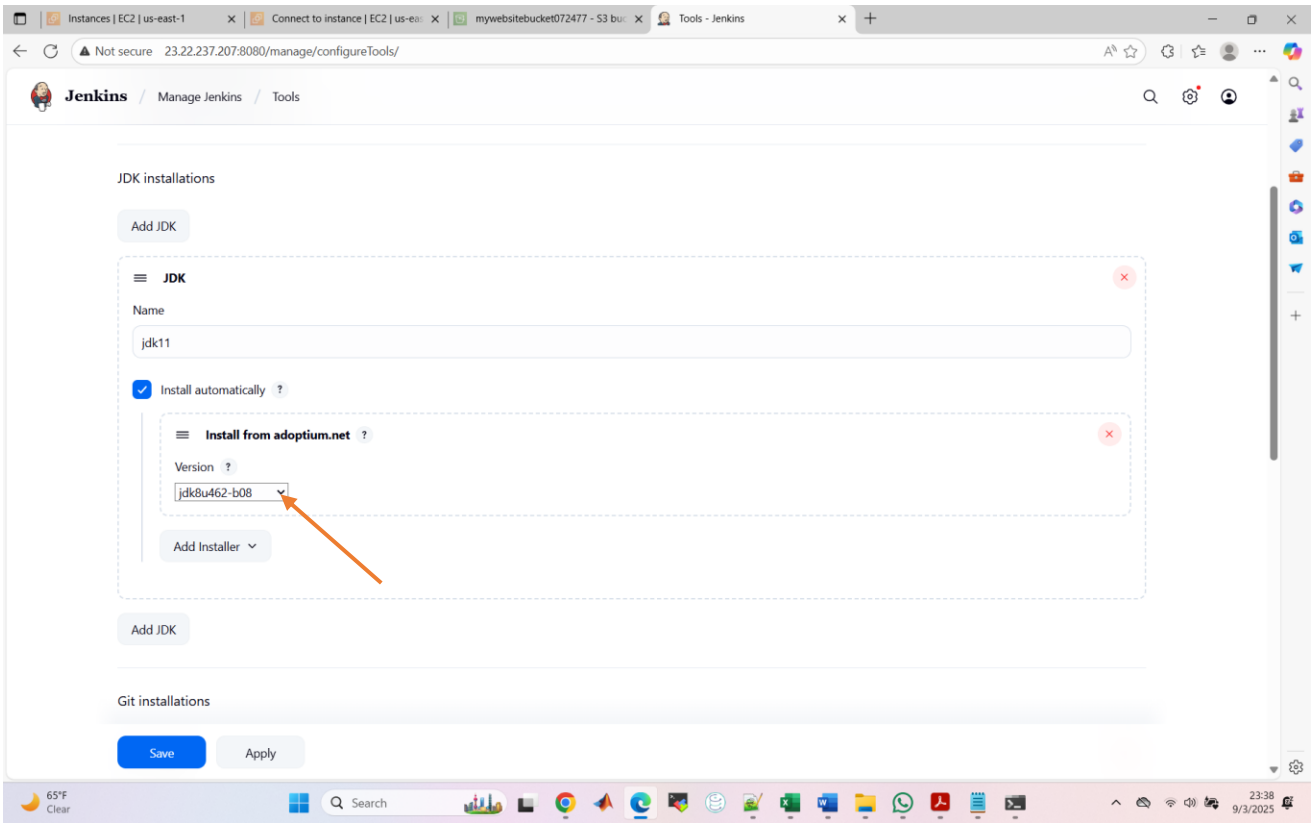
Select “**Install Automatically**”



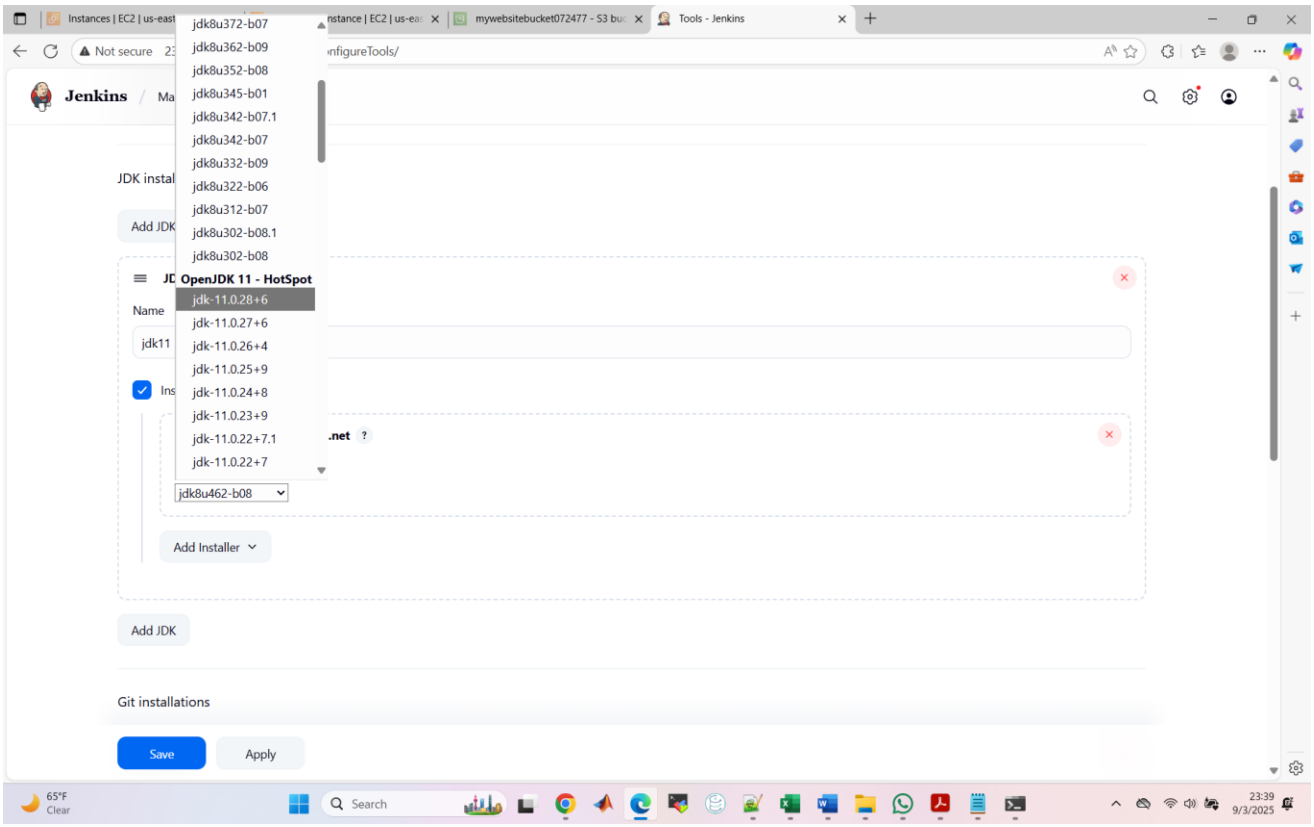
Click on the drop down on “**Add Installer**”



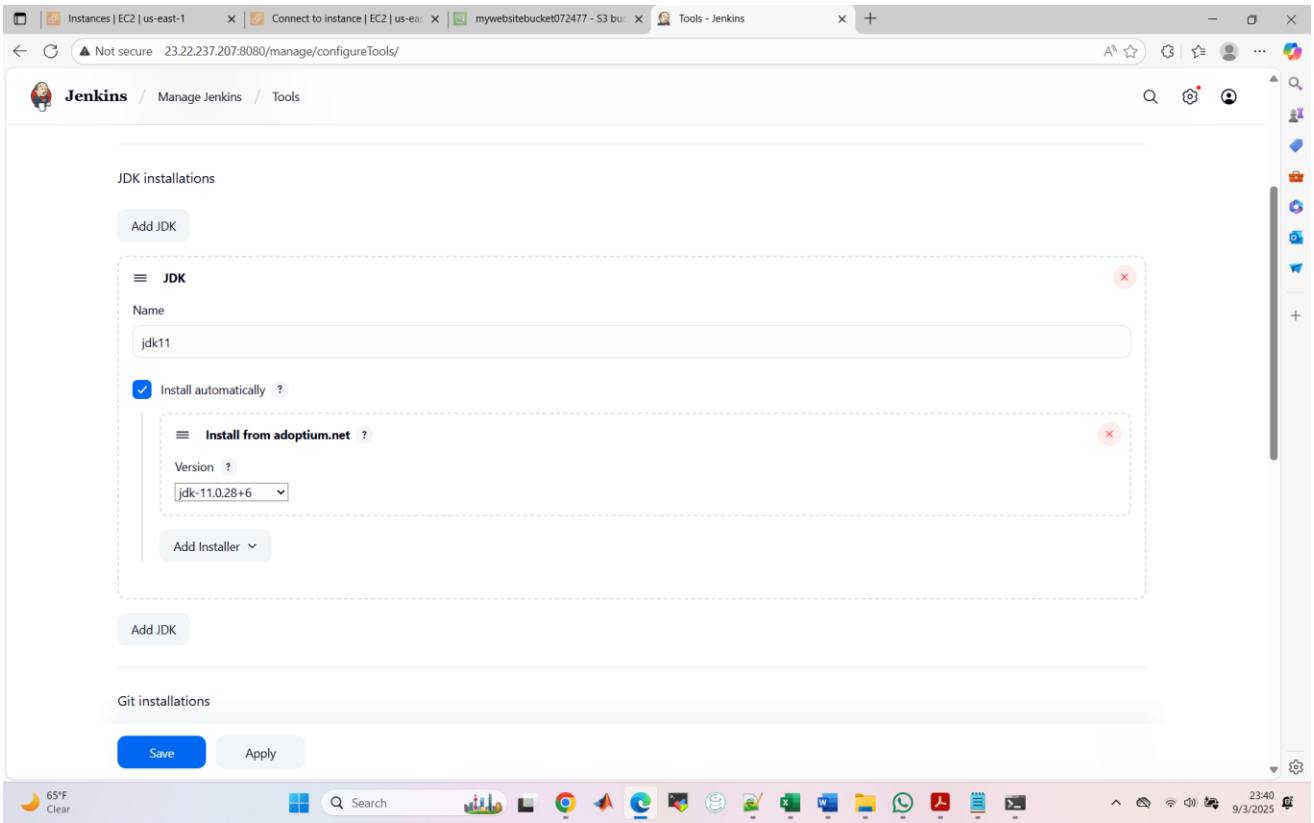
Select “**Install from adoptium.net**”



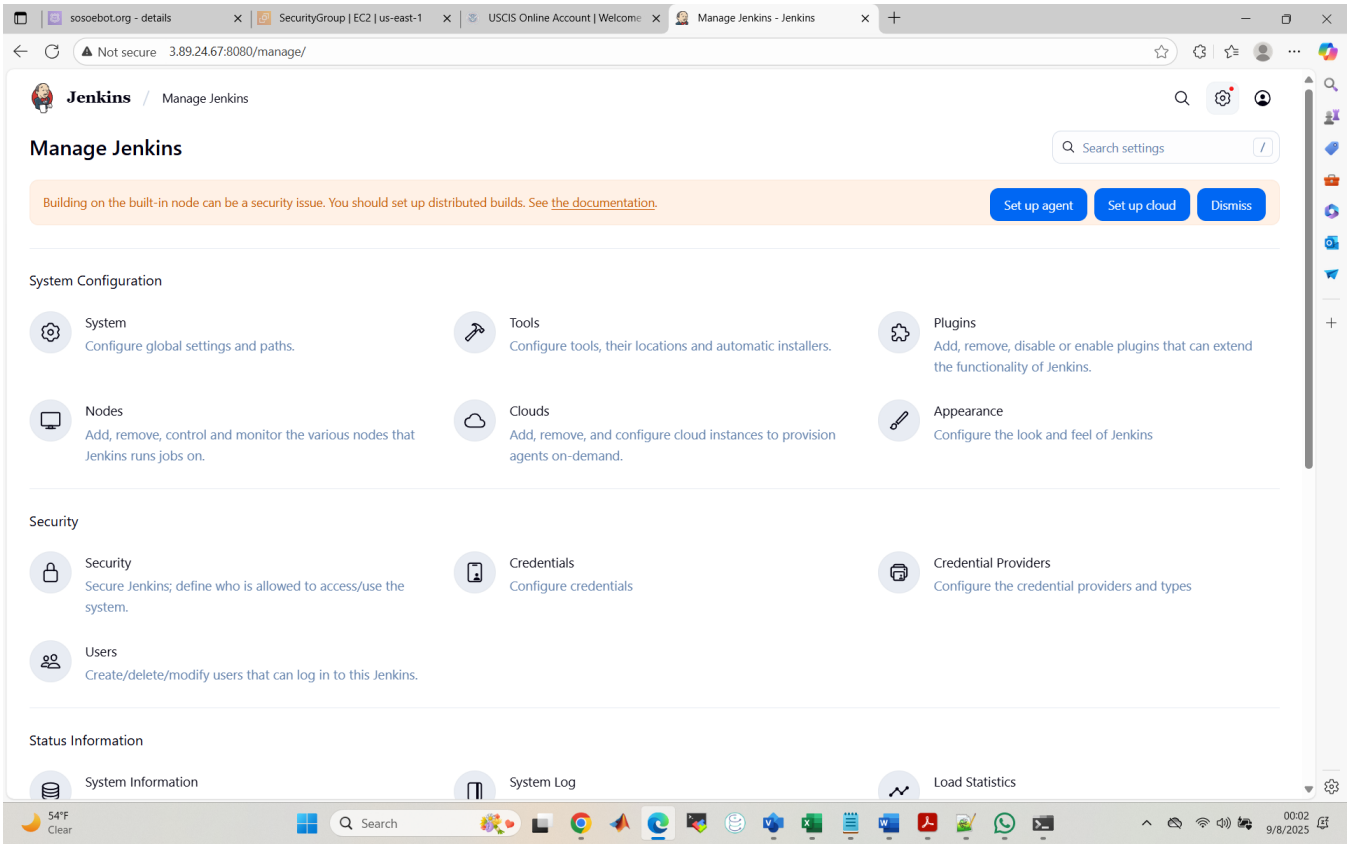
Click on the drop down on “**Version**”



Select version "**jdk-11.0.28+6**"



Click on "**apply**" then "**Save**"

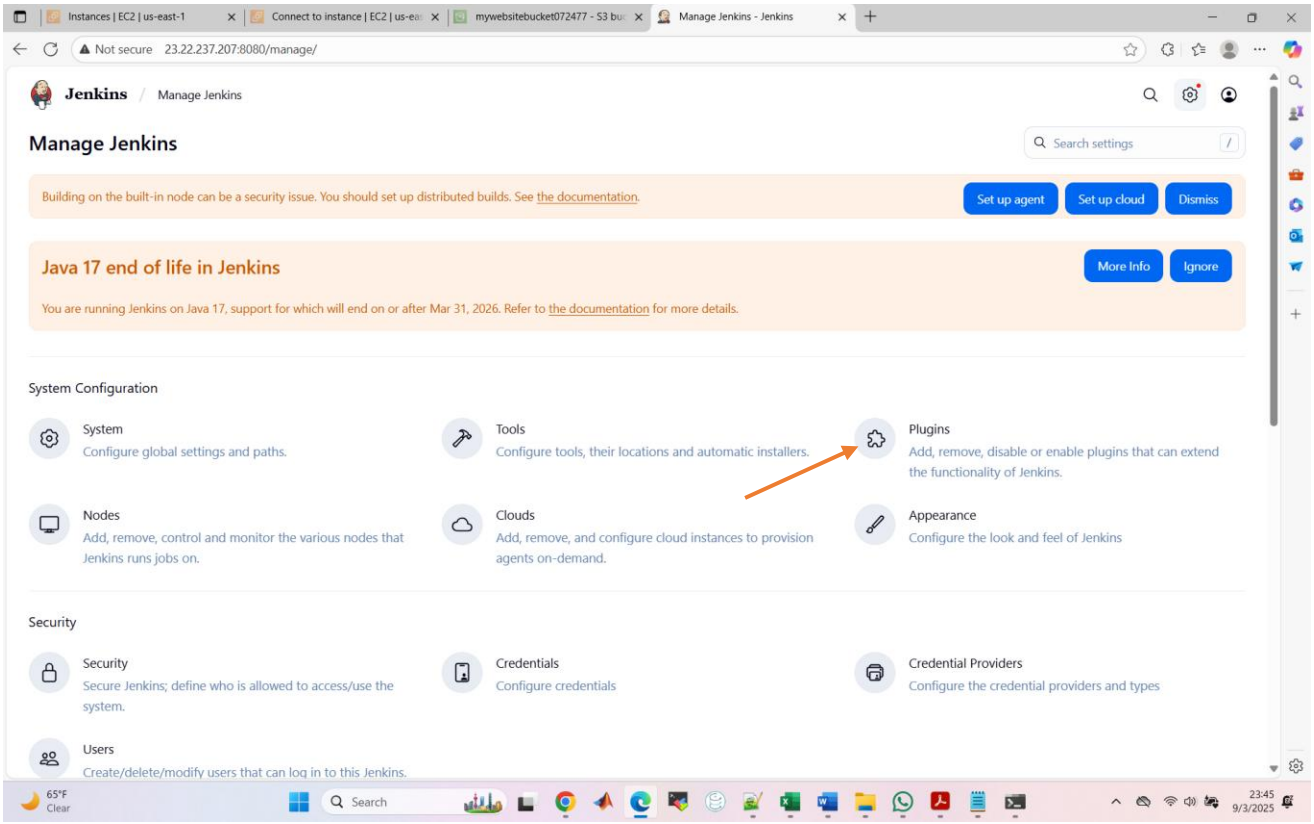

sosoebot.org - details
SecurityGroup | EC2 | us-east-1
USCIS Online Account | Welcome
Manage Jenkins - Jenkins
Not secure 3.89.24.67:8080/manage/
Jenkins / Manage Jenkins
Manage Jenkins
Search settings
Building on the built-in node can be a security issue. You should set up distributed builds. See the documentation.
Set up agent
Set up cloud
Dismiss
System Configuration
System
Configure global settings and paths.
Tools
Configure tools, their locations and automatic installers.
Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
Nodes
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.
Clouds
Add, remove, and configure cloud instances to provision agents on-demand.
Appearance
Configure the look and feel of Jenkins
Security
Security
Secure Jenkins; define who is allowed to access/use the system.
Credentials
Configure credentials
Credential Providers
Configure the credential providers and types
Users
Create/delete/modify users that can log in to this Jenkins.
Status Information
System Information
System Log
Load Statistics
54°F
Clear
Search
00:02
9/8/2025

STEP 9: Install Maven plugin and Configure Maven on Jenkins

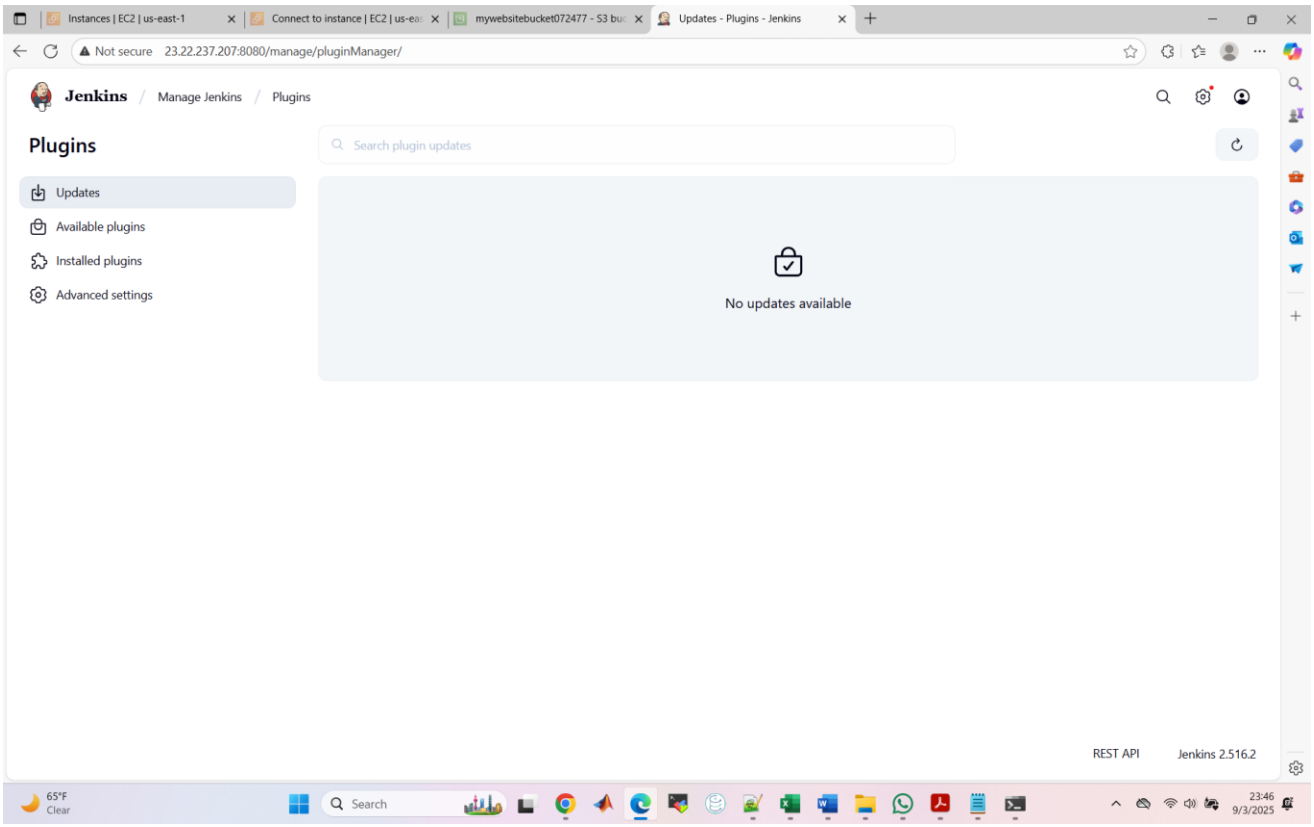
Here we will install Maven plugin and configure Maven on Jenkins.

PART 1: Install the Maven Plugin

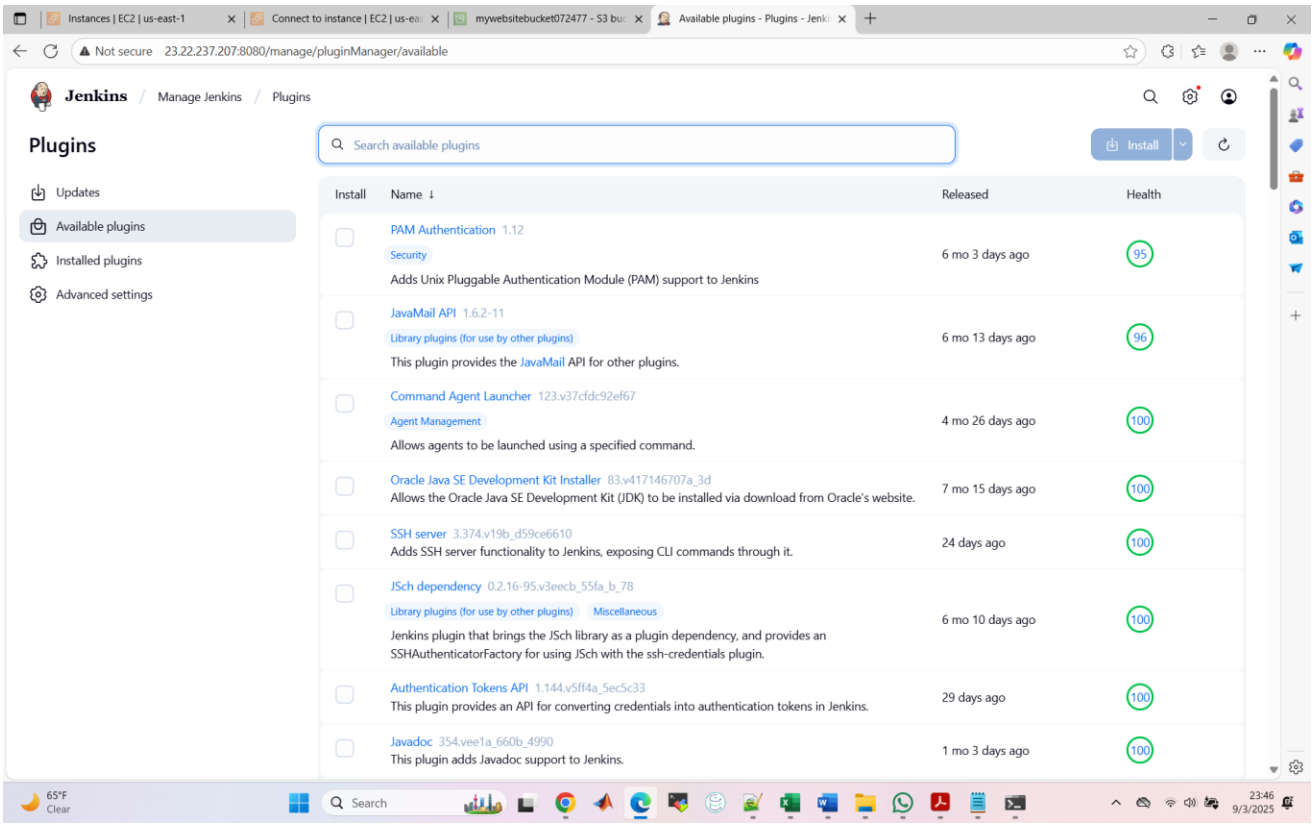
We will install the plugin "**Maven Integration**"



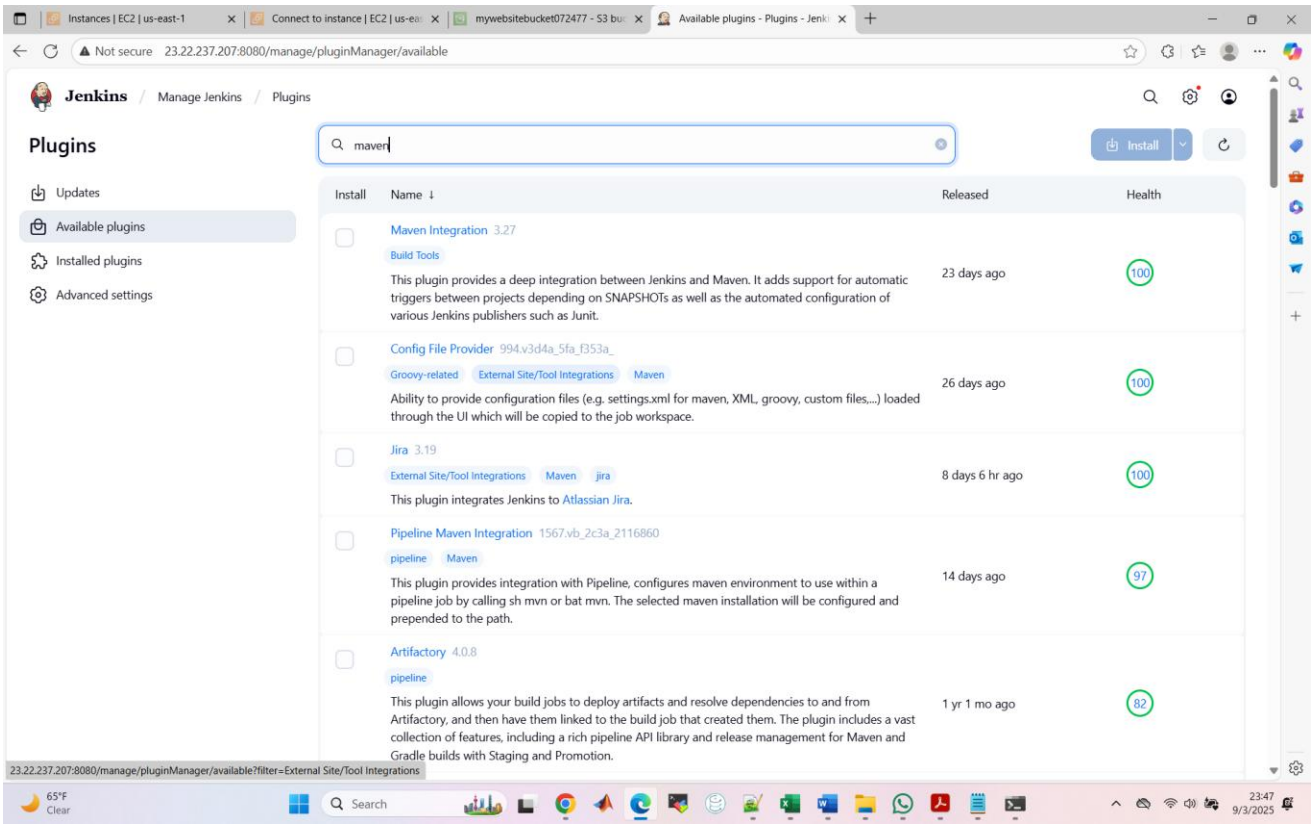
Click on "**Plugins**"



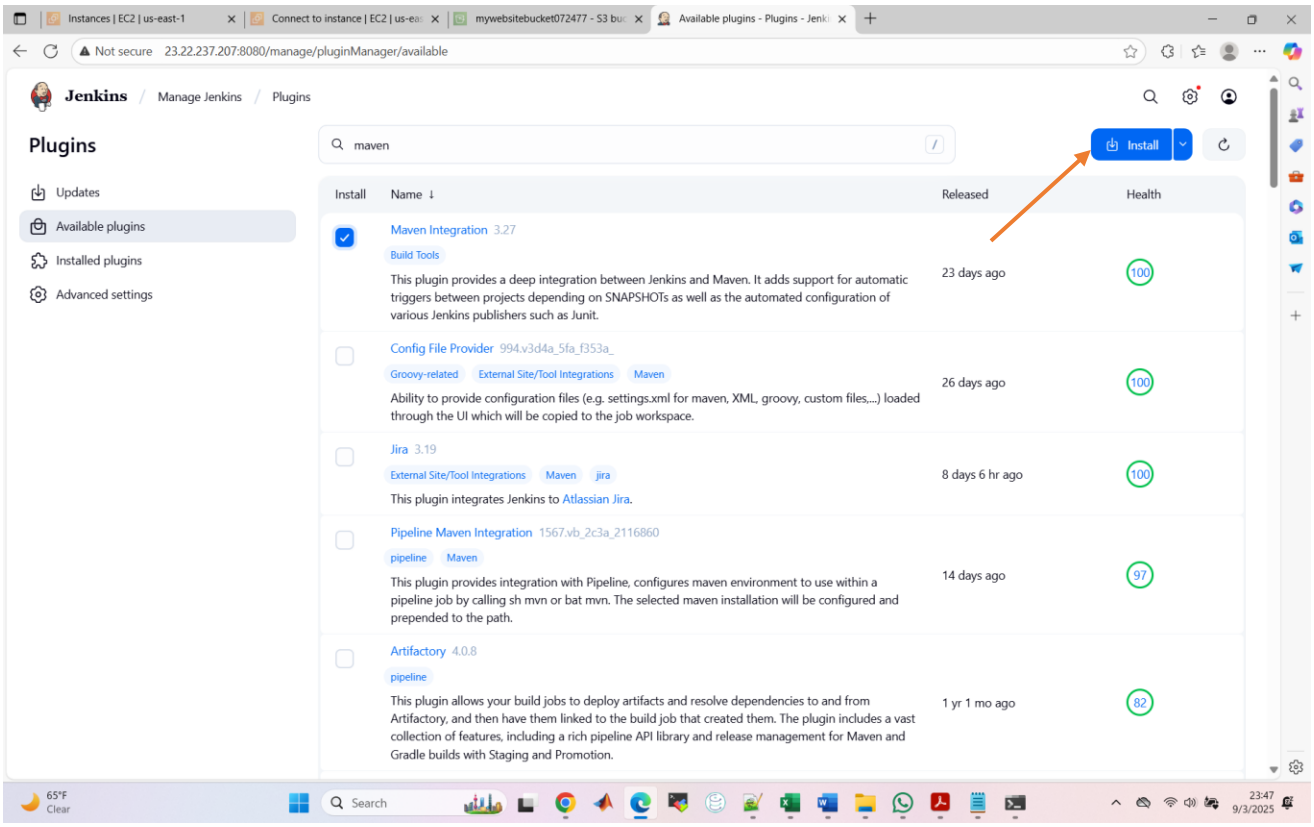
Click on "**Available Plugins**"



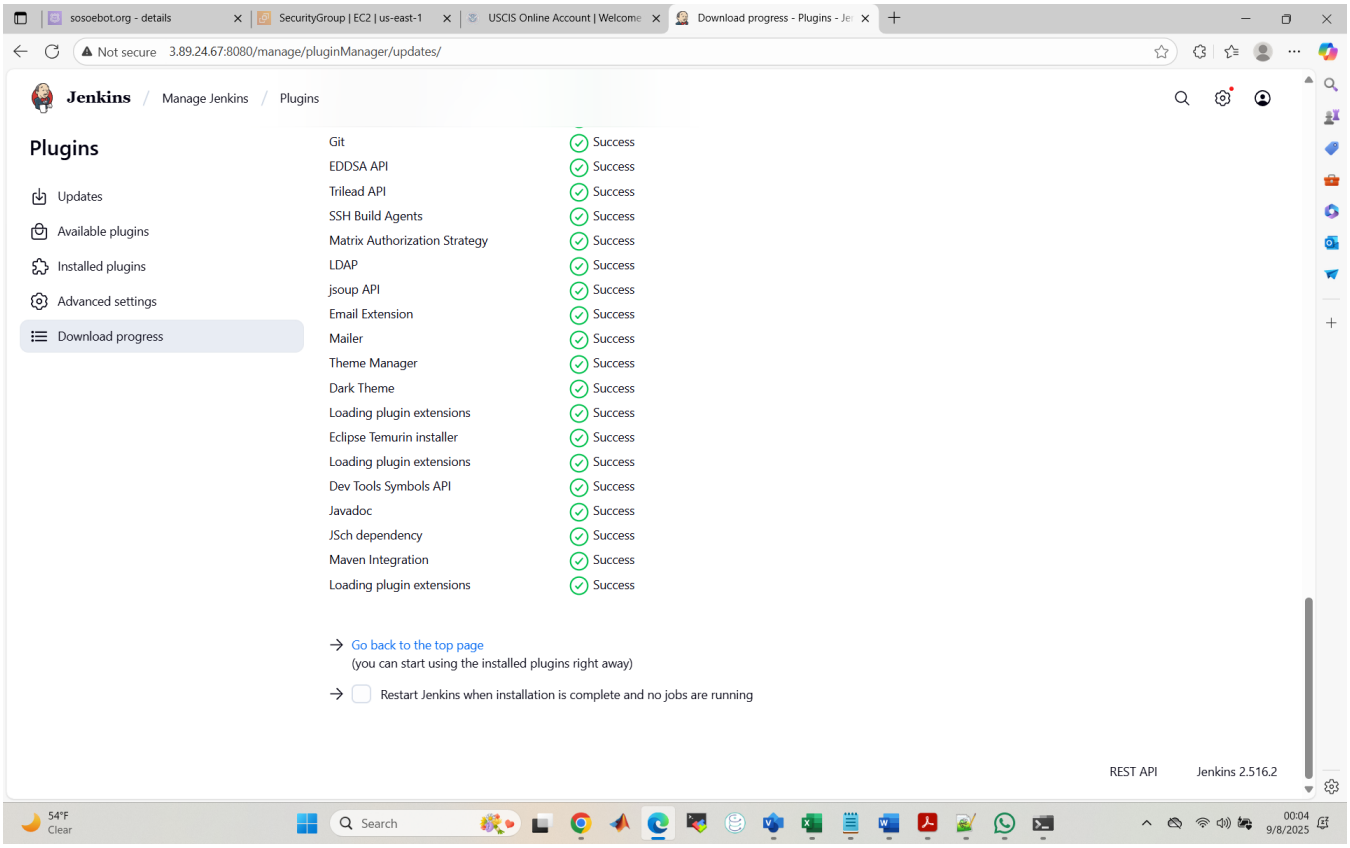
Search for "**Maven**"



Select “**Maven Integration**”

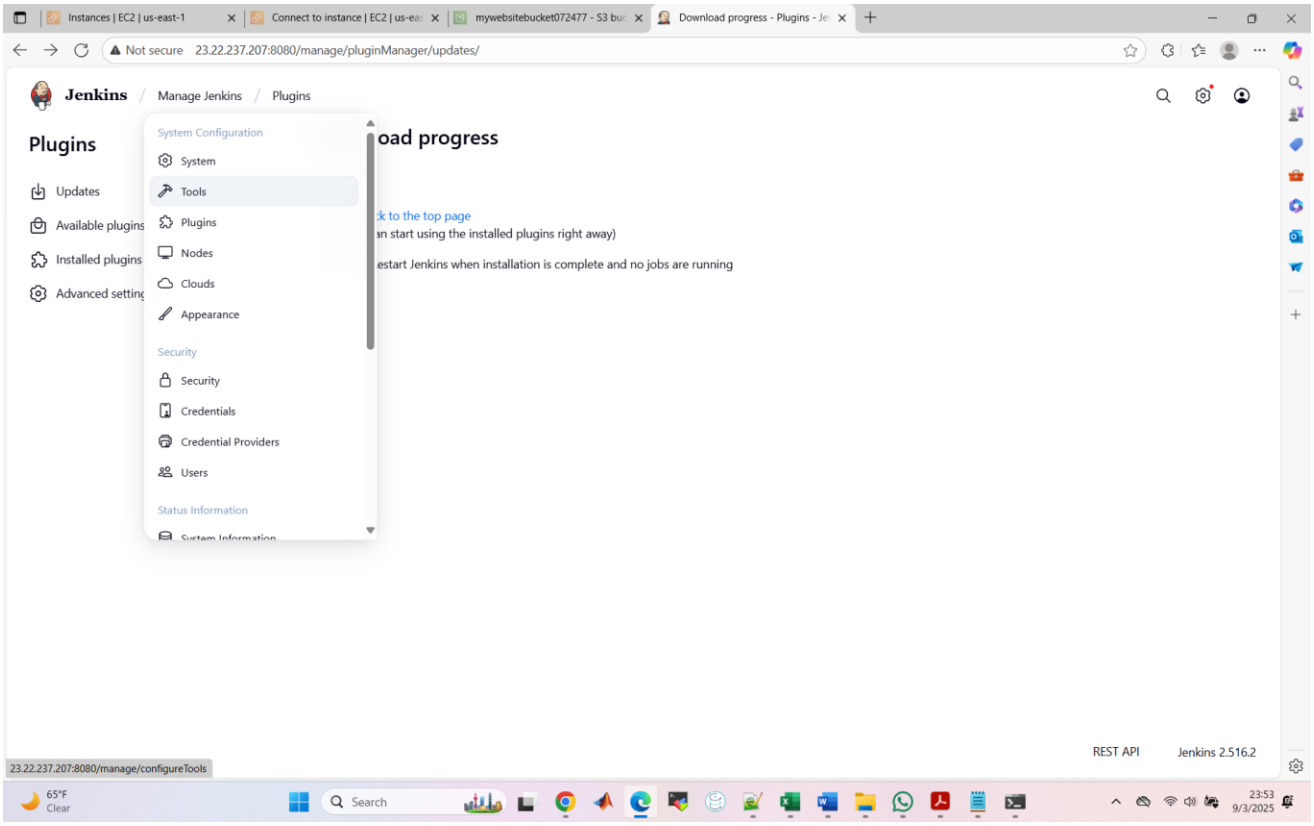


Click on “**Install**”

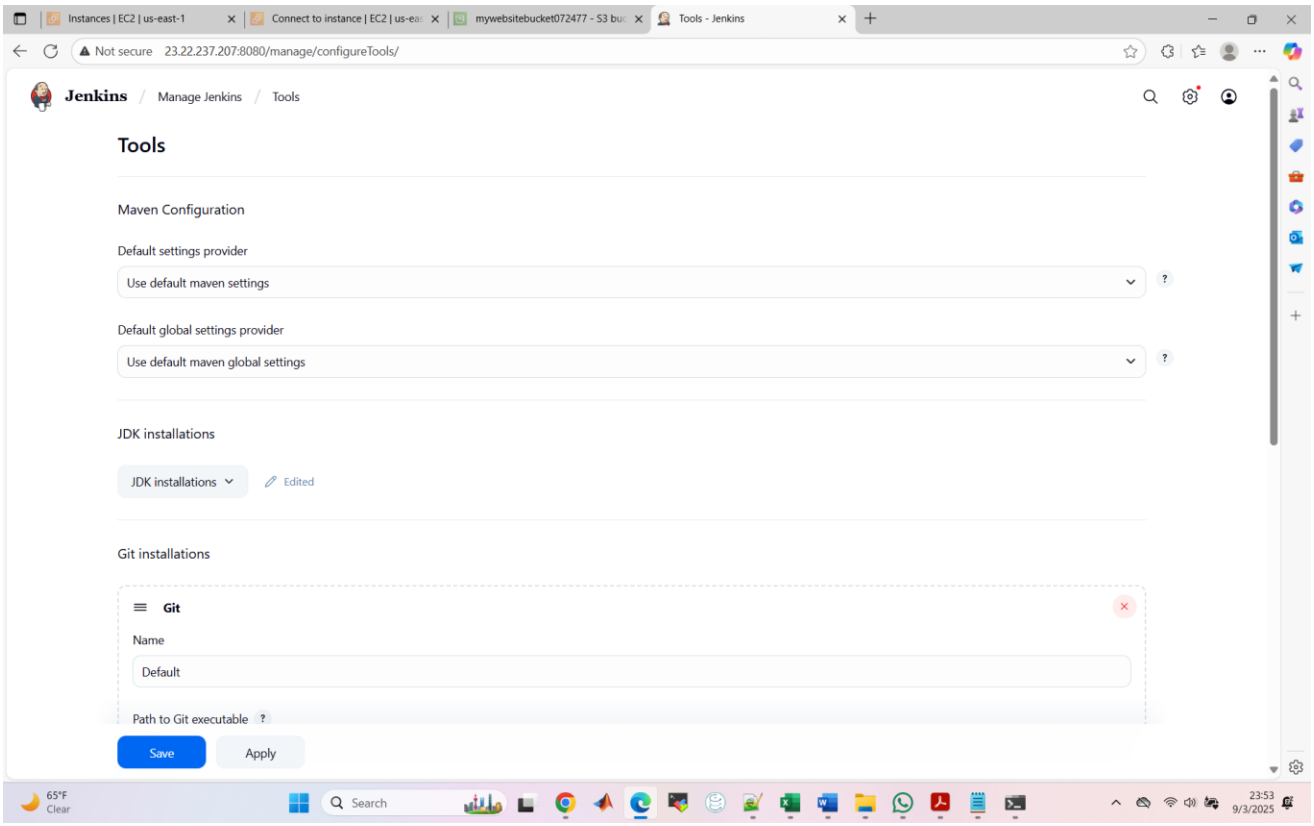


PART B: Configure Maven as a tool on Jenkins

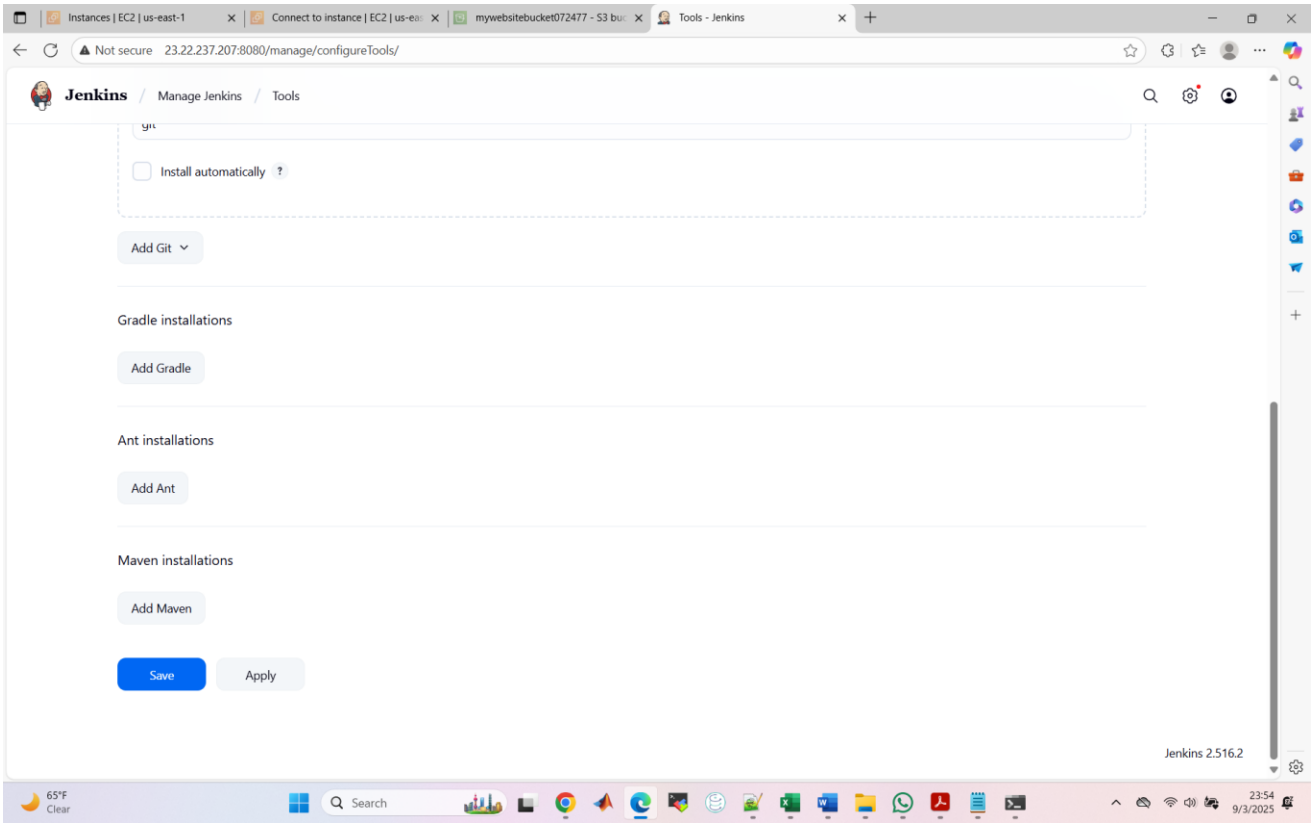
We will configure Maven on Jenkins. Go to "**Manage Jenkins**"



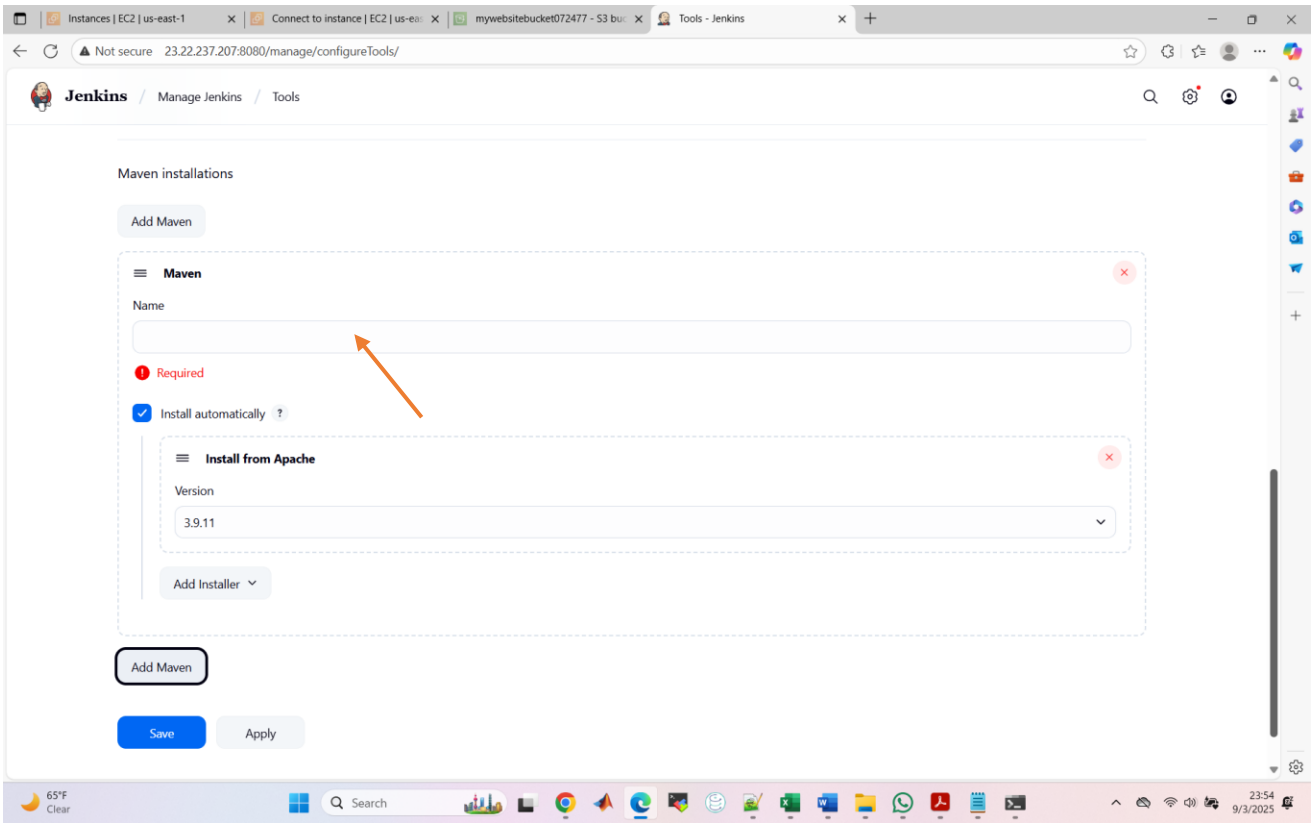
Select “**tools**”



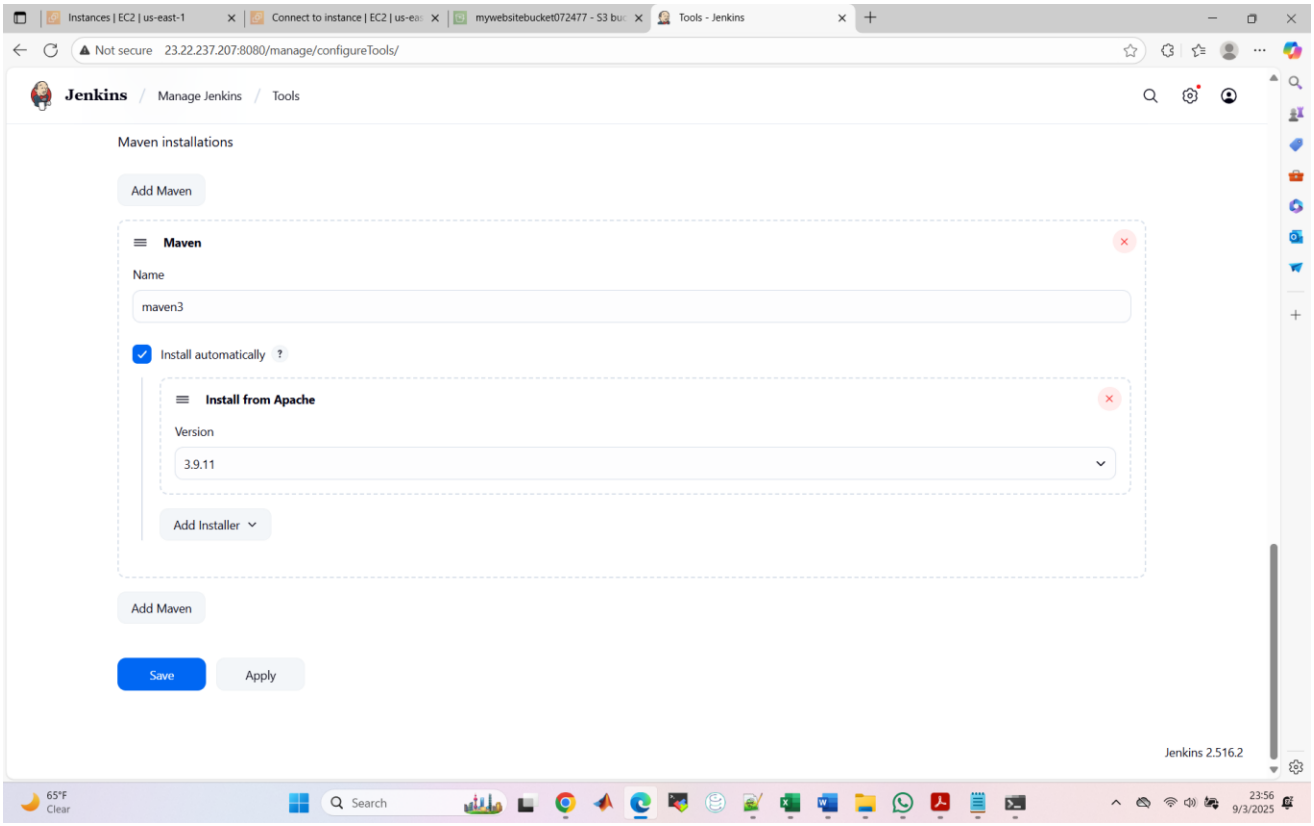
Scroll down to “**Maven Installations**”



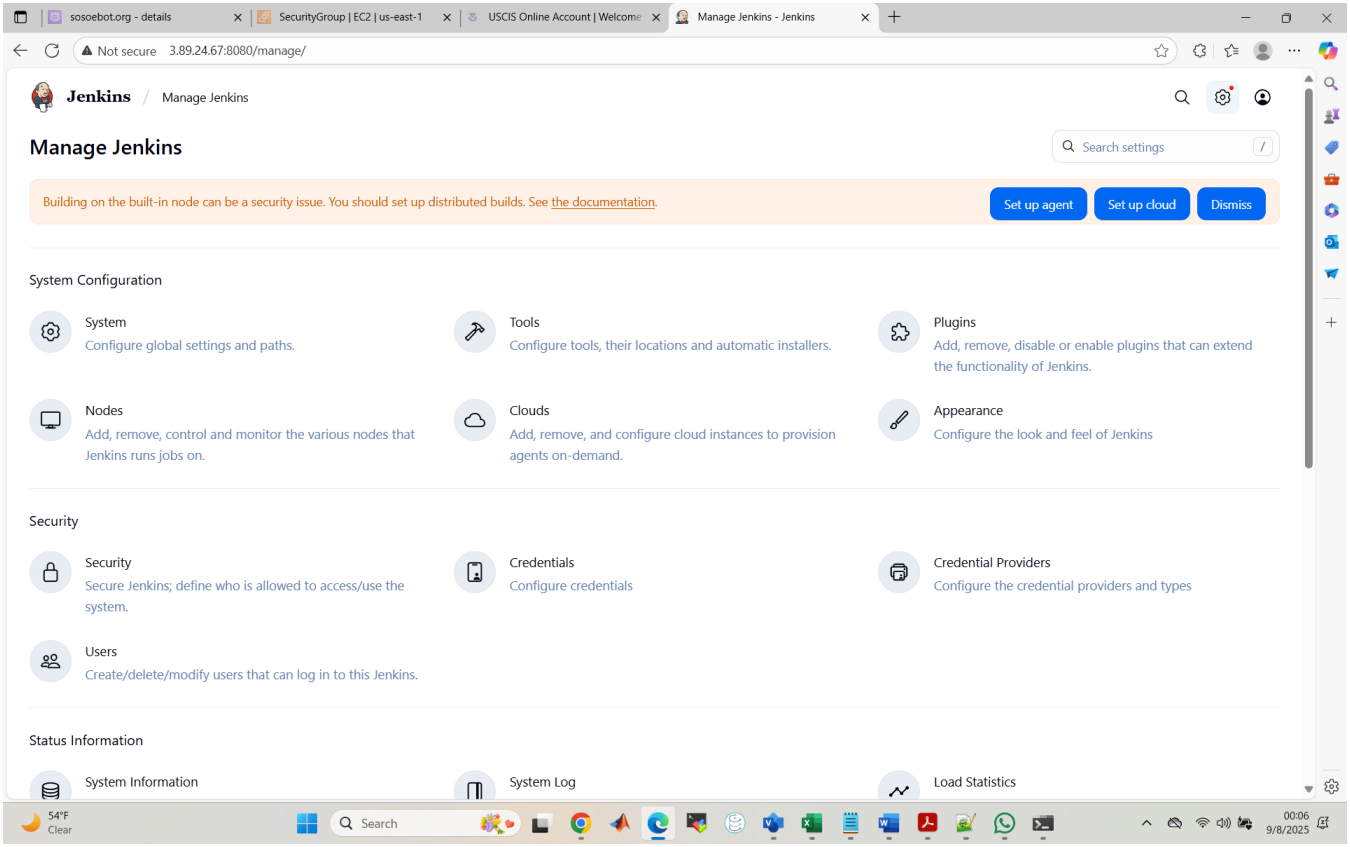
Click on “**Add Maven**”



On name, we will enter “**maven3**”



Click on “**Apply**” followed by “**Save**”

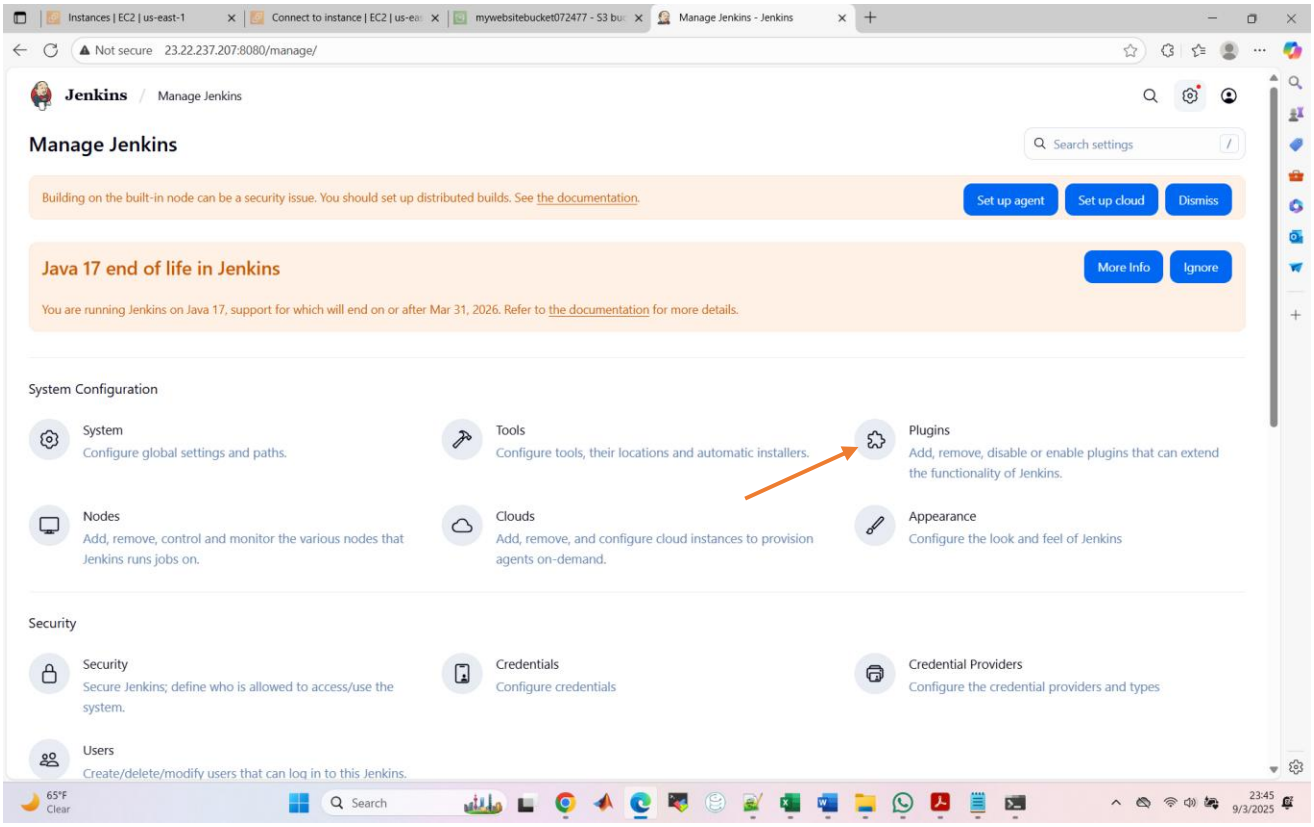


STEP 10: Install and Configure Docker on Jenkins

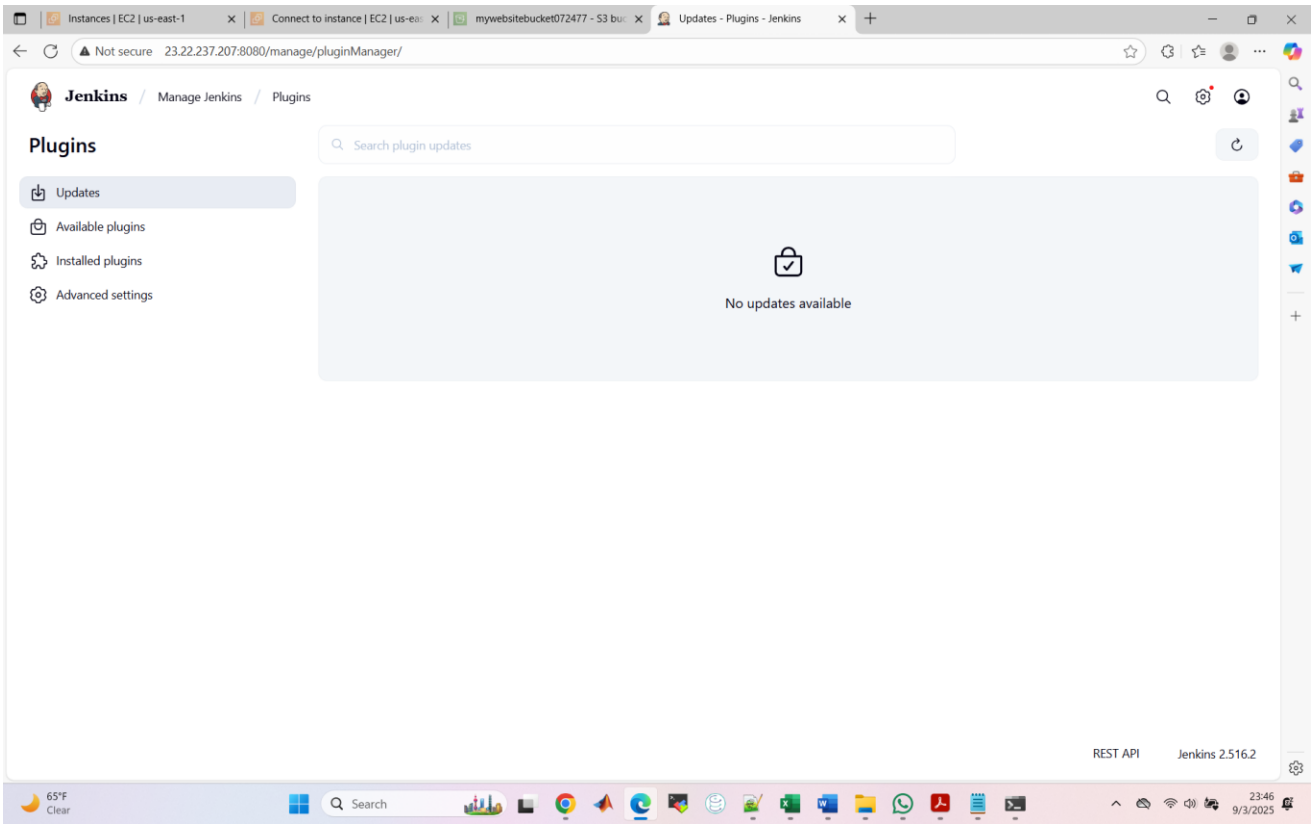
Here we will install “**Docker**”, “**Docker Commons**”, “**Docker Pipeline**” and “**Docker API**” plugins and configure Dependency Checks on Jenkins.

PART 1: Install all the Plugins

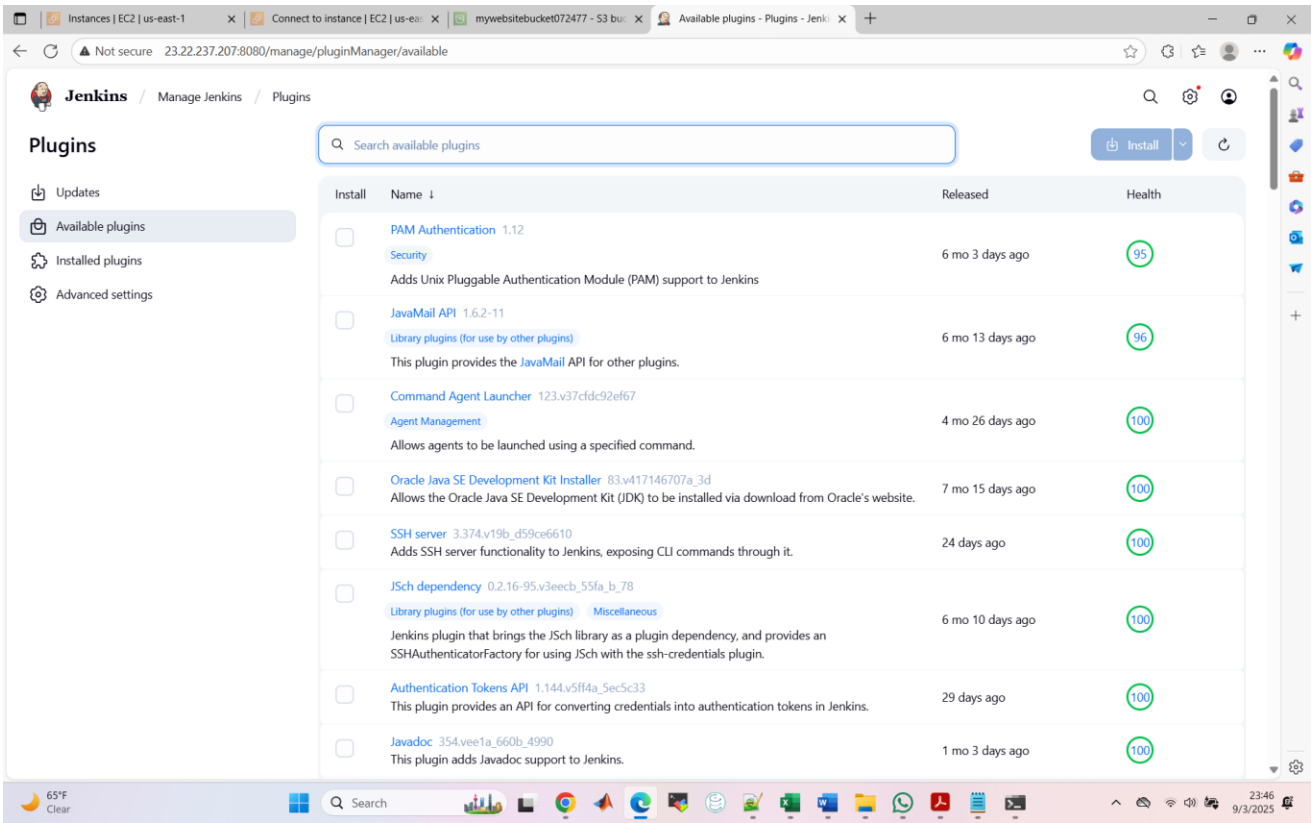
We will install all the docker plugins needed for this project



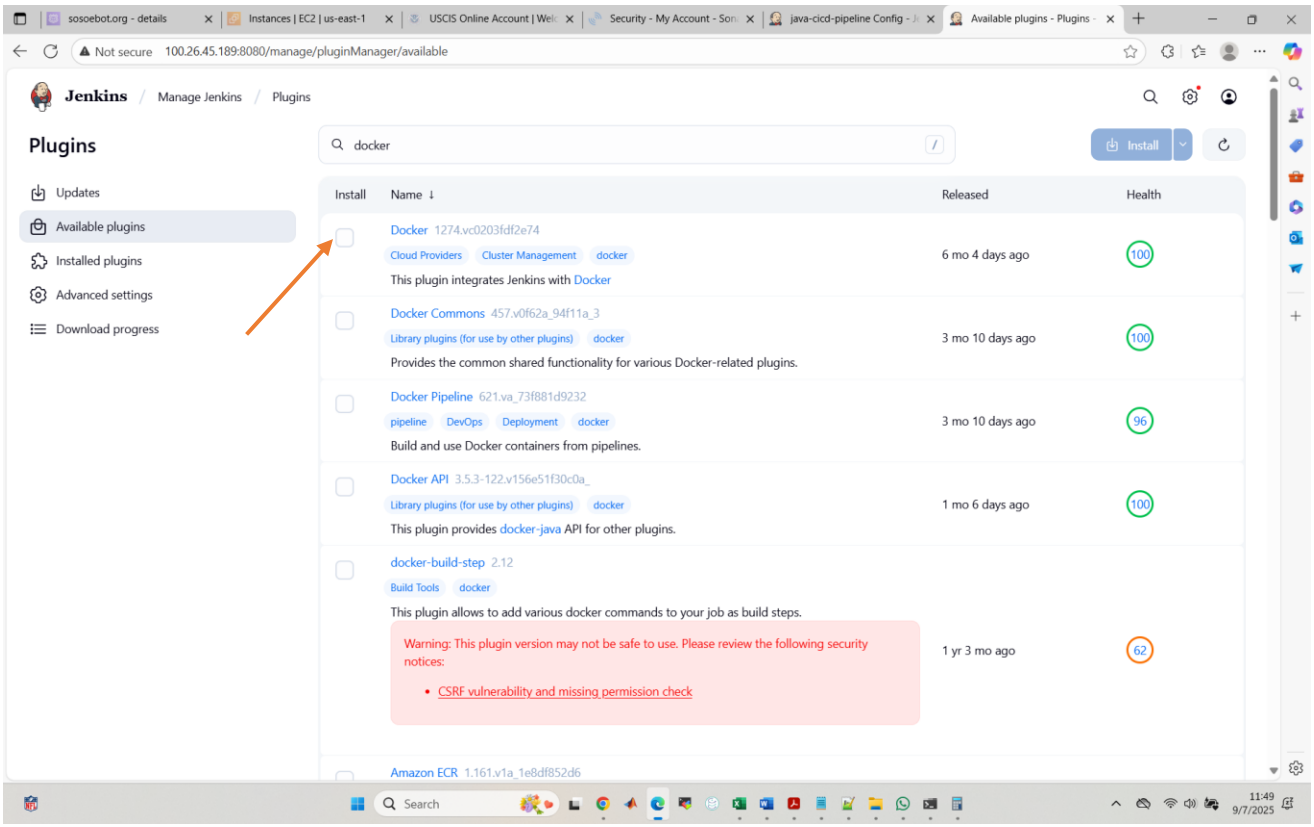
Click on “**Plugins**”



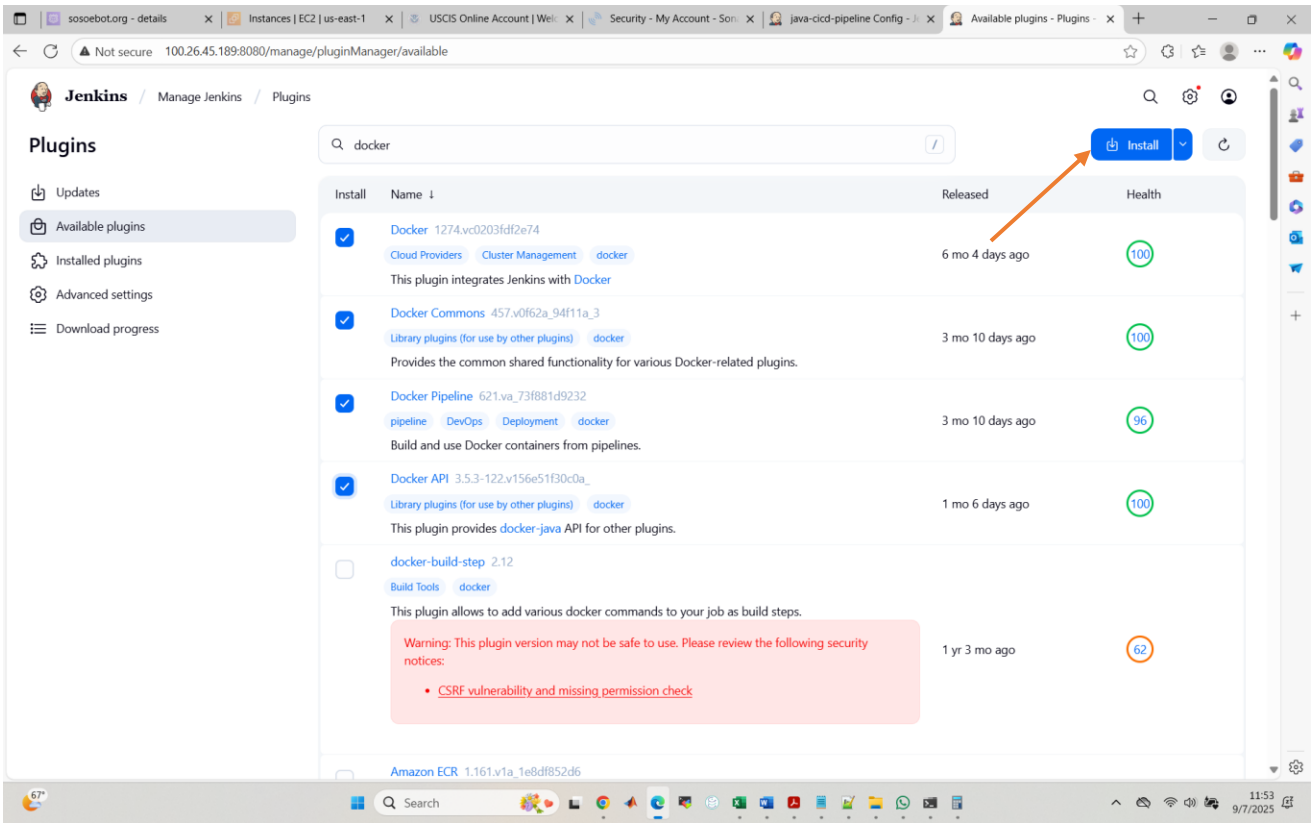
Click on "**Available Plugins**"



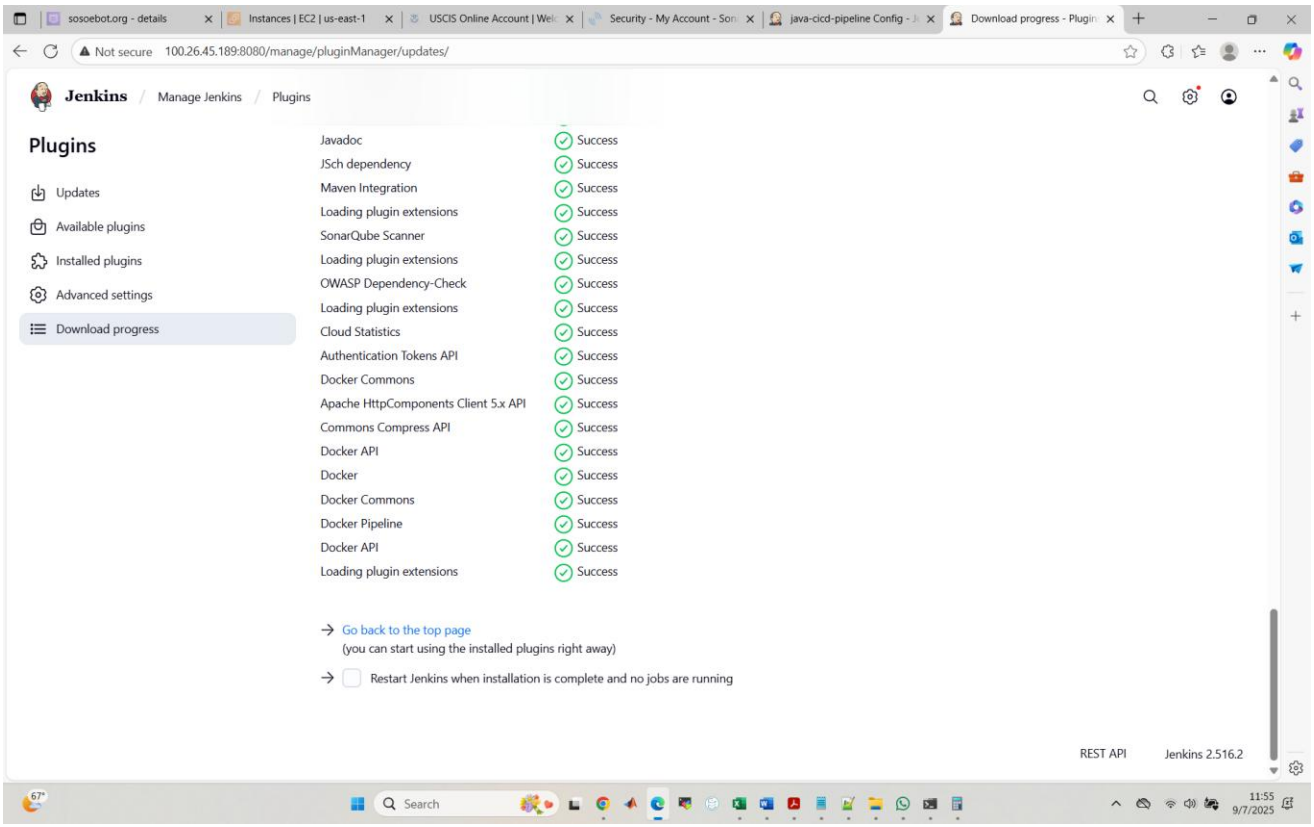
Search for "**Docker**"



Select “**Docker**”, “**Docker Commons**”, “**Docker Pipeline**” and “**Docker API**”

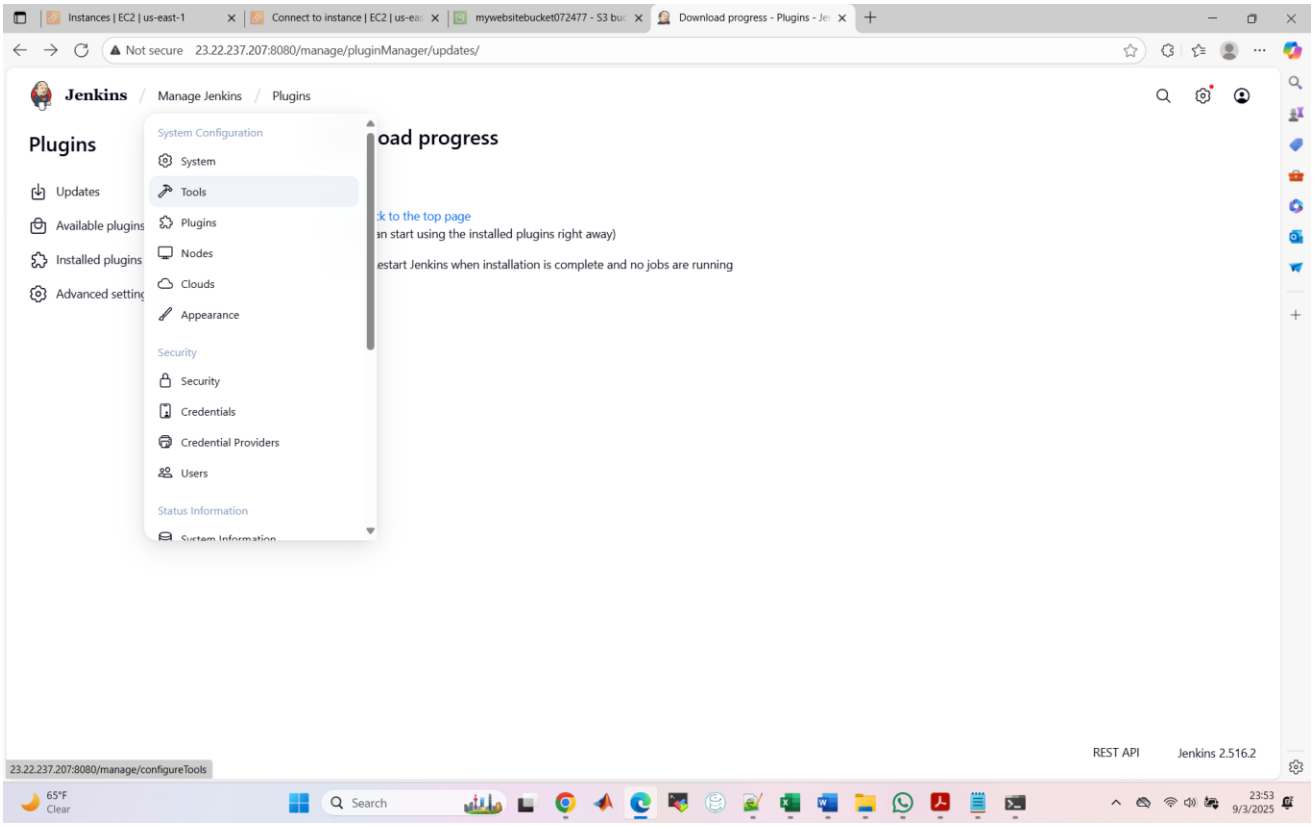


Click on “**Install**”

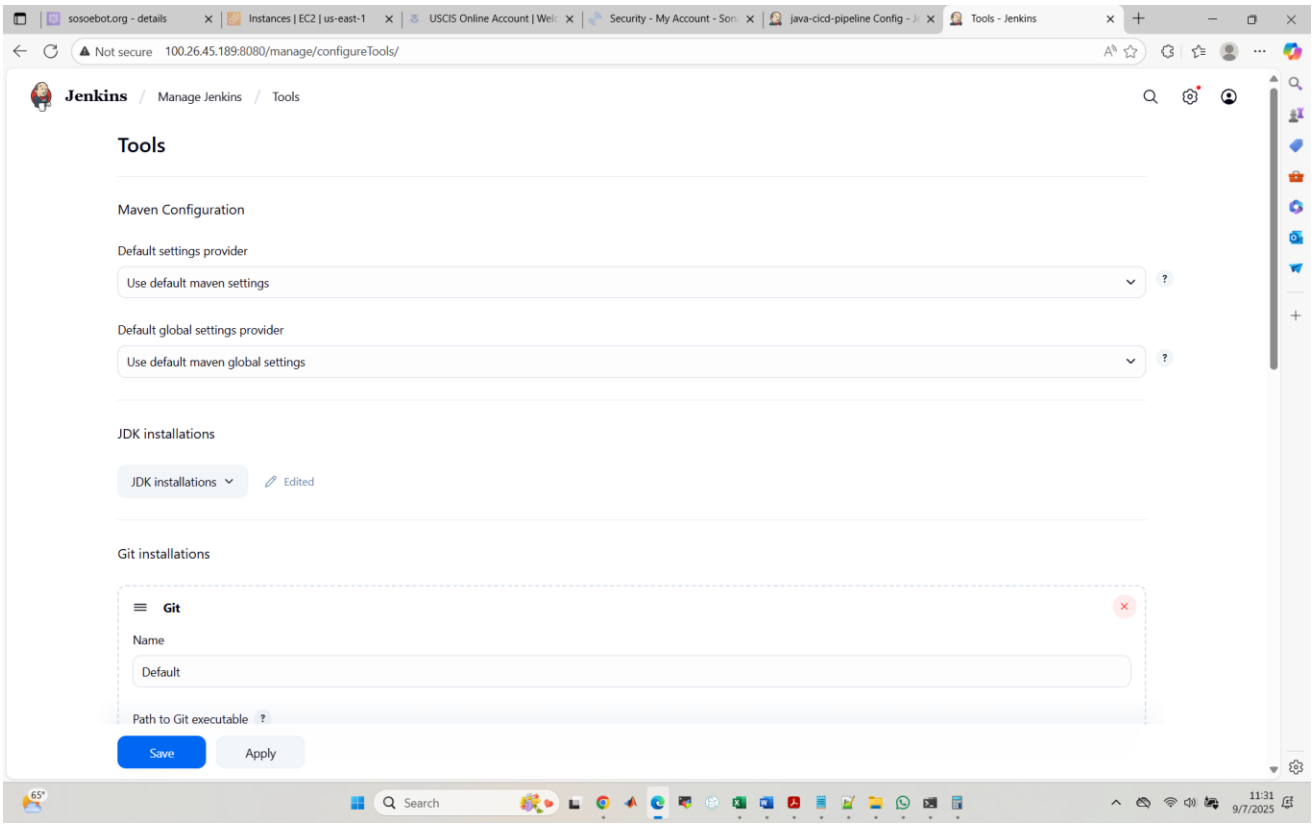


PART B: Configure Docker on Jenkins

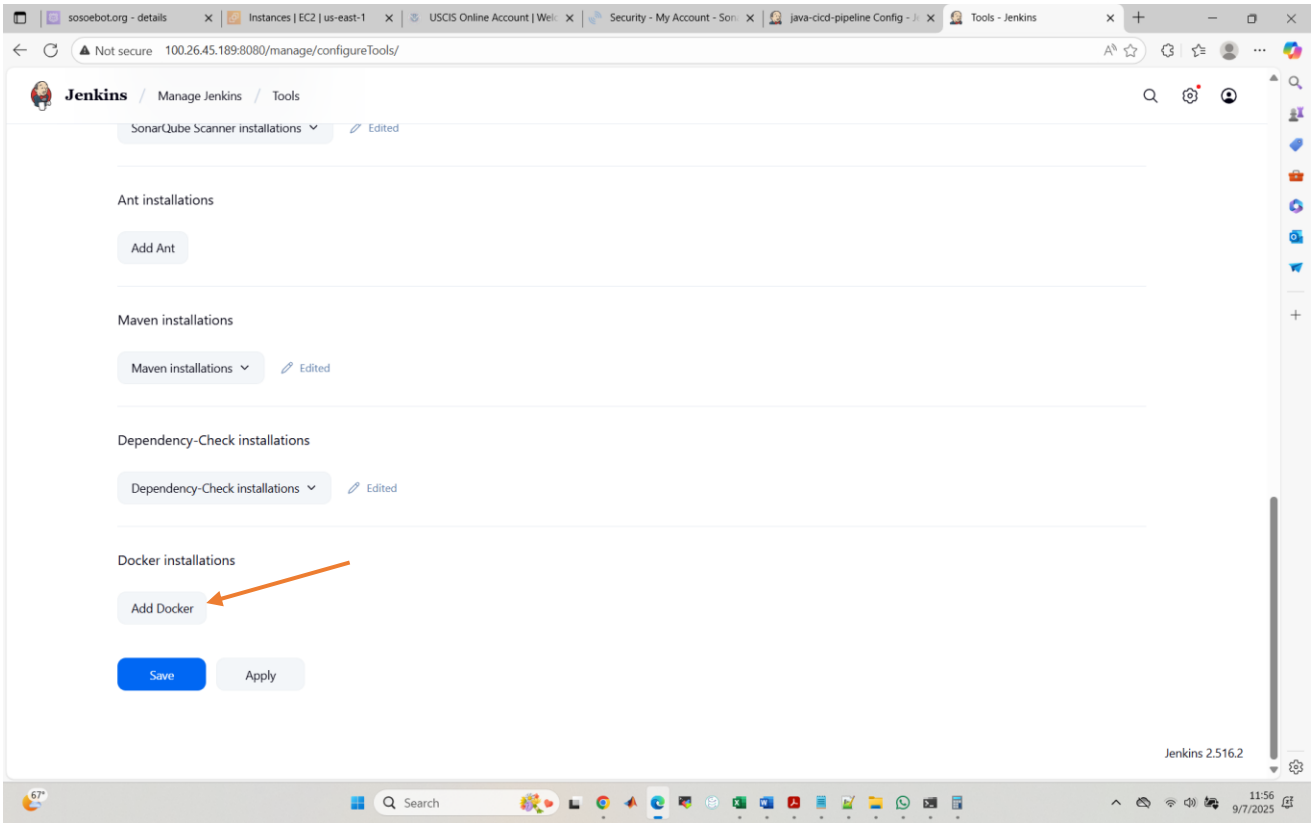
We will configure Dependency Check on Jenkins. Go to "**Manage Jenkins**"



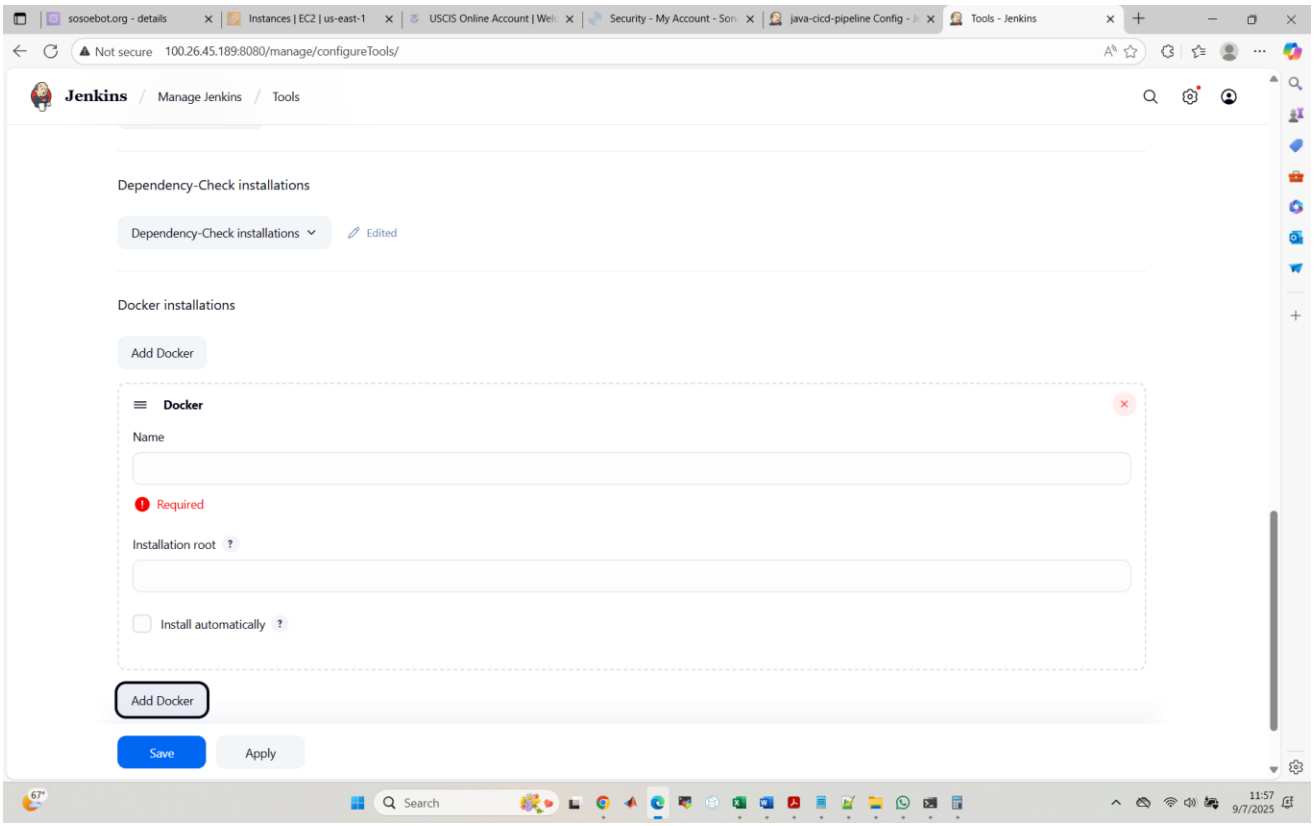
Select “**tools**”



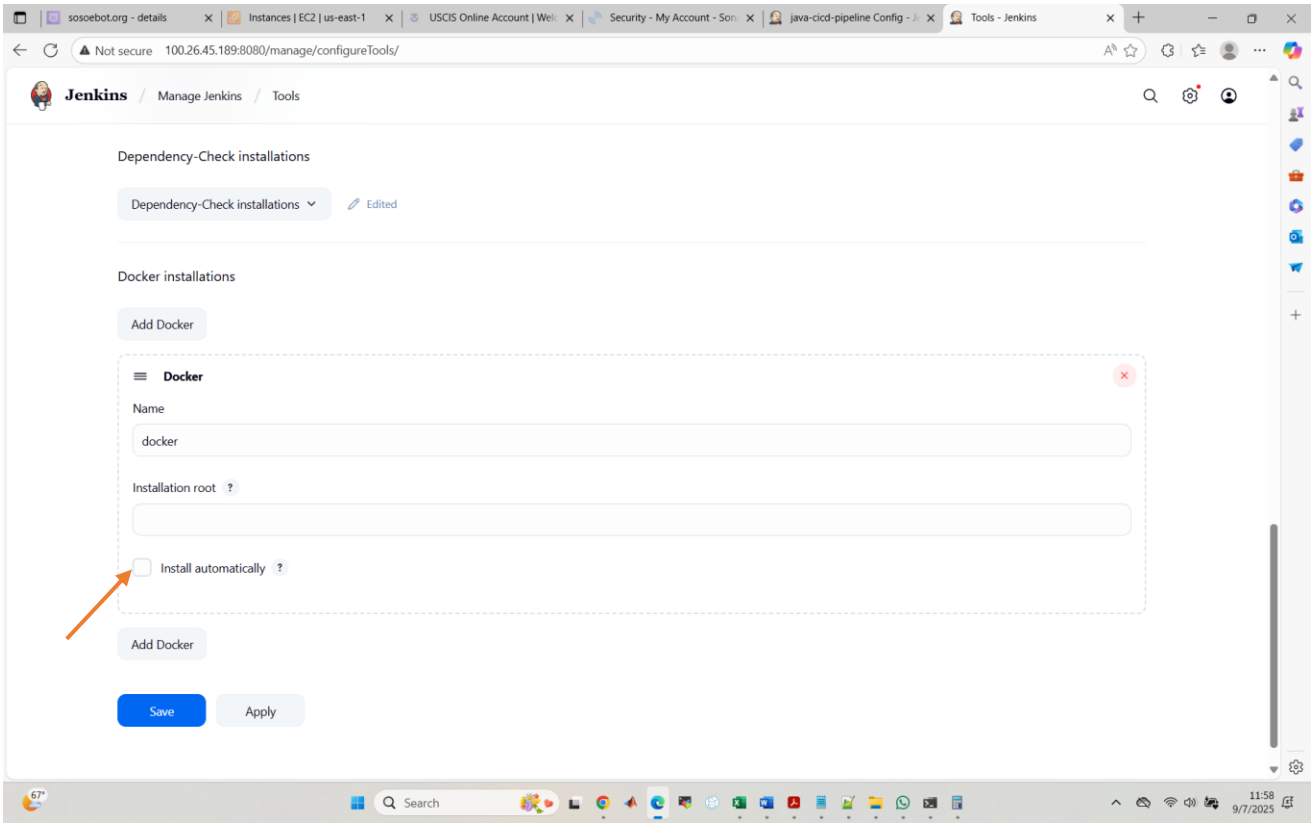
Scroll down to “**Docker Installations**”



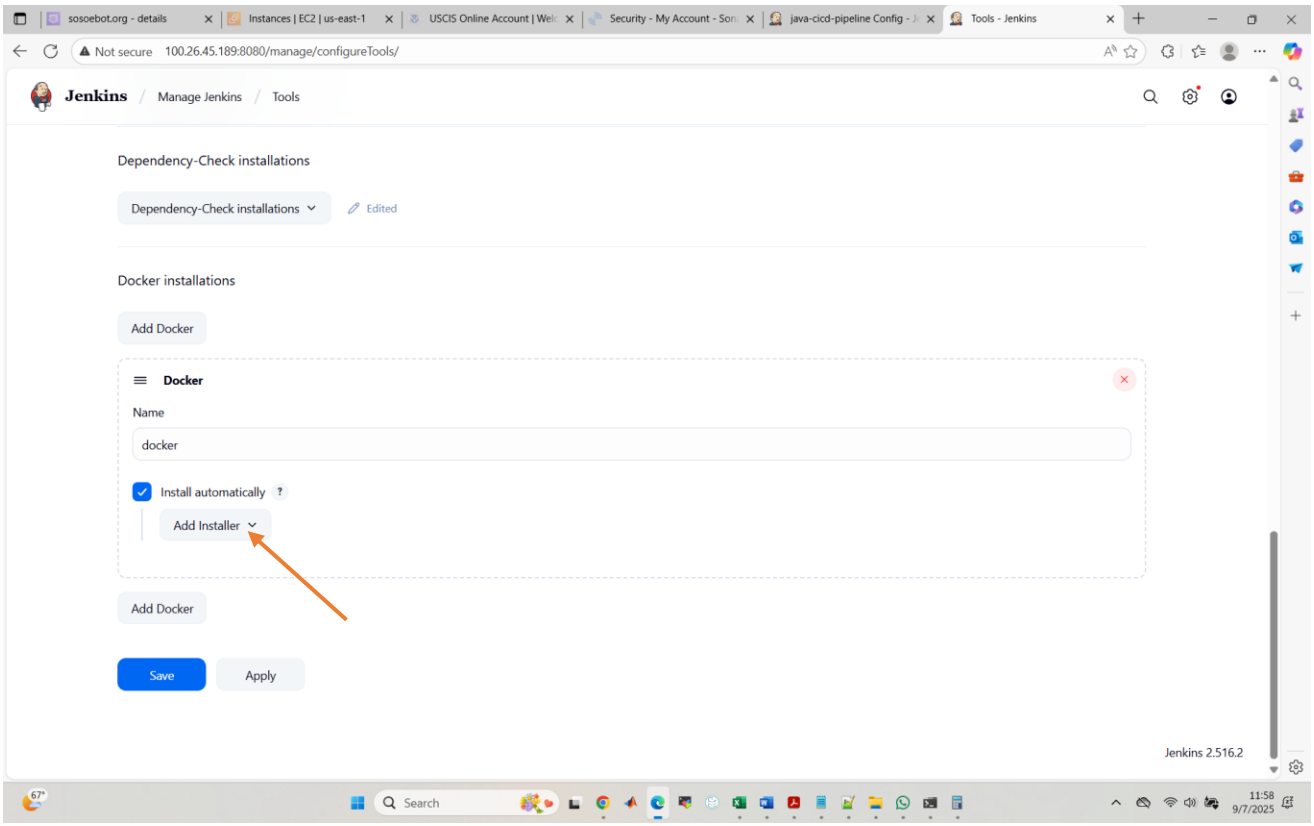
Click on “**Add Docker**”



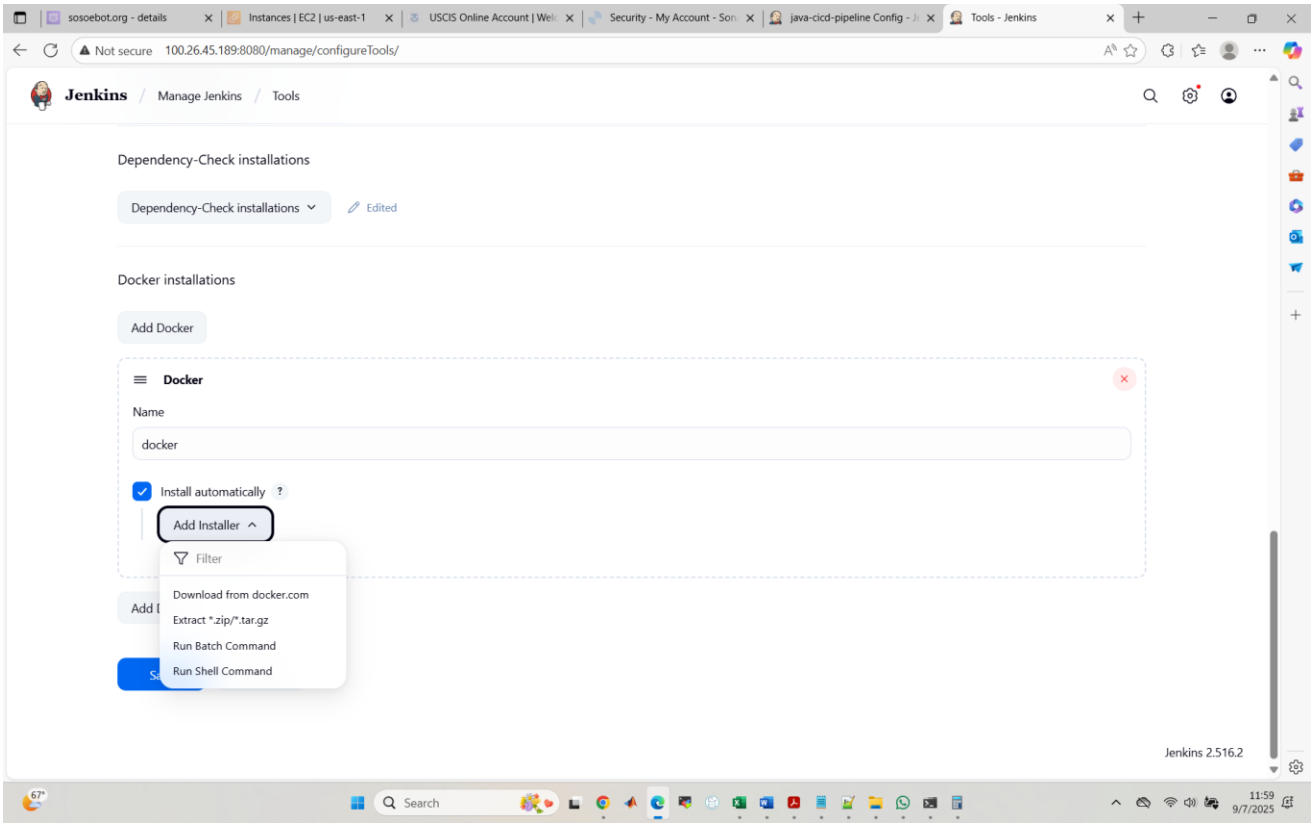
On name, we will enter “**docker**”



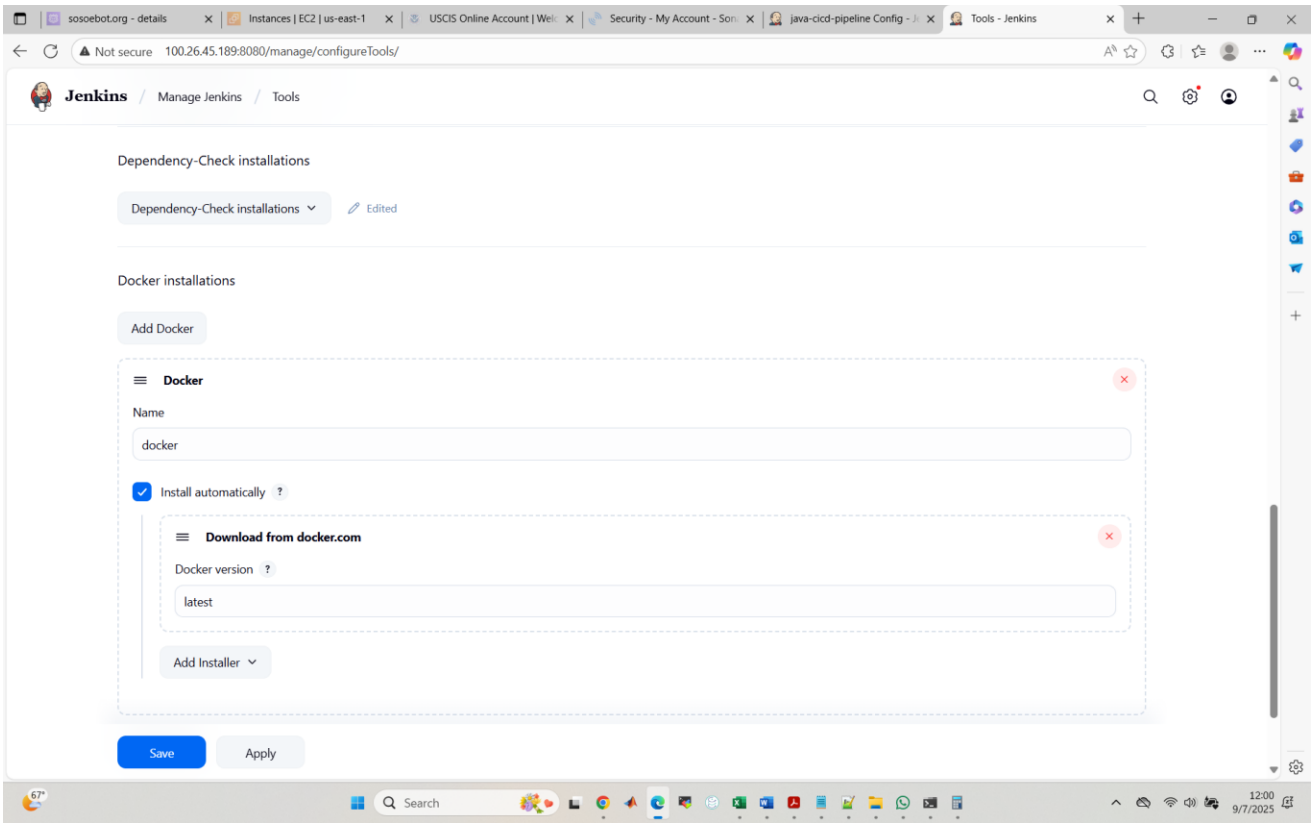
Select “**Install Automatically**”



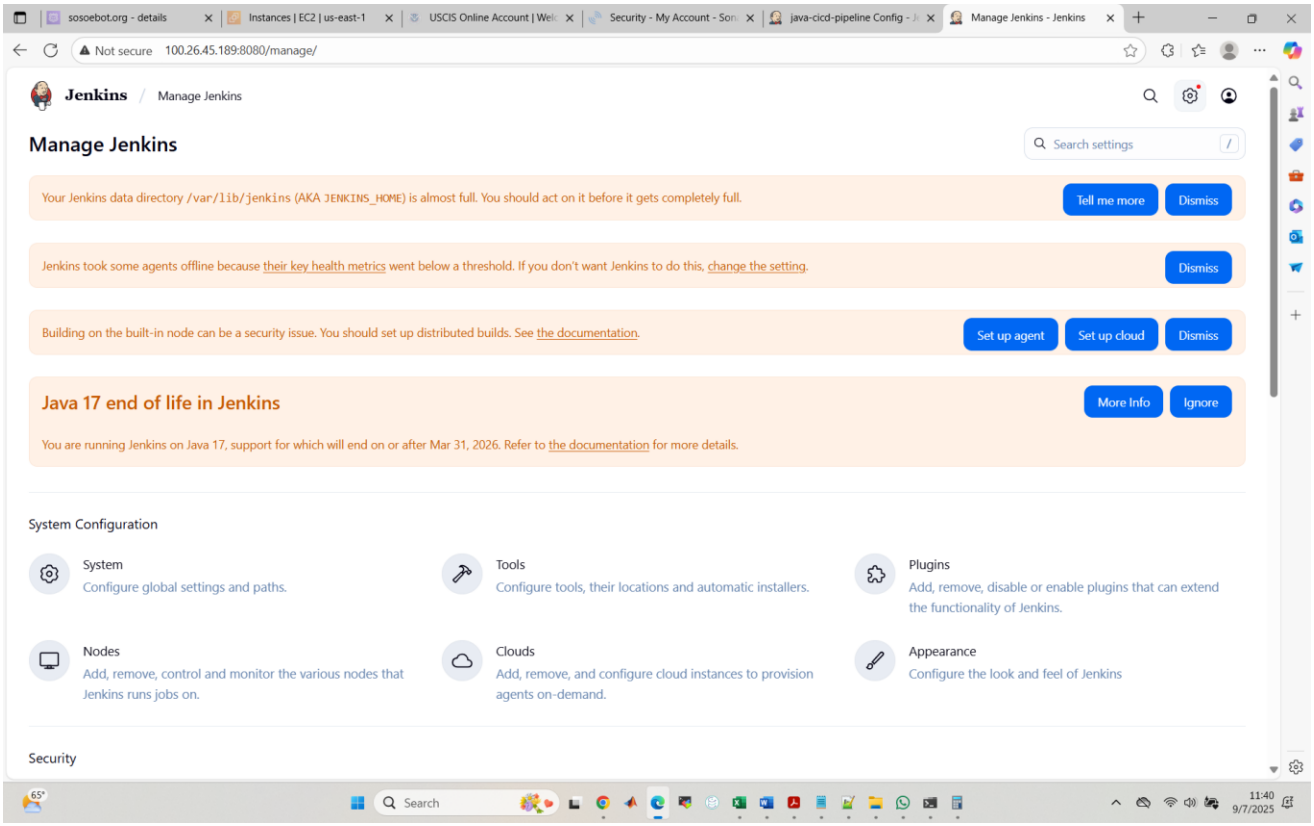
Click on the drop down on “**Add Installer**”



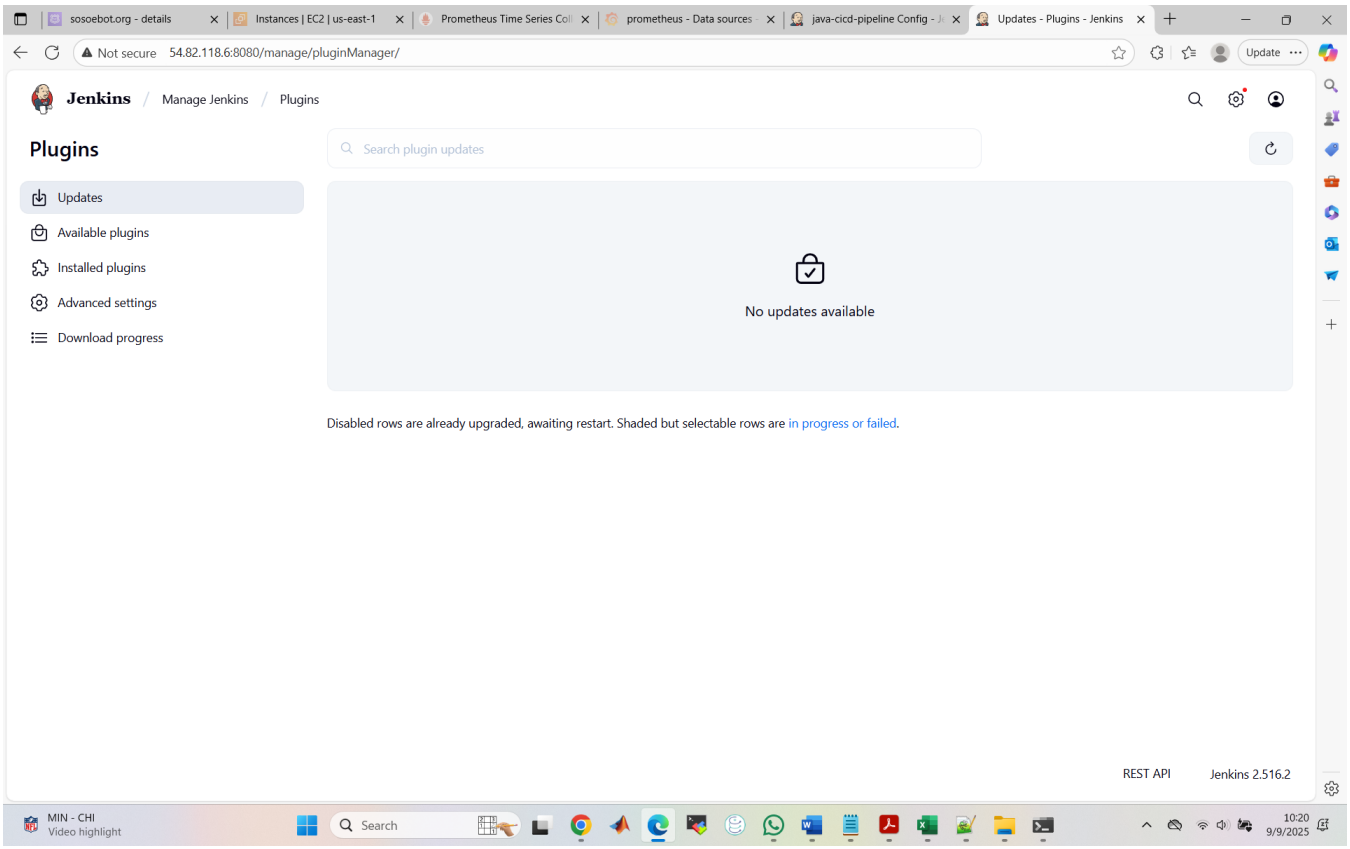
Select “**Download from docker.com**” and the Docker version should be “**latest**”



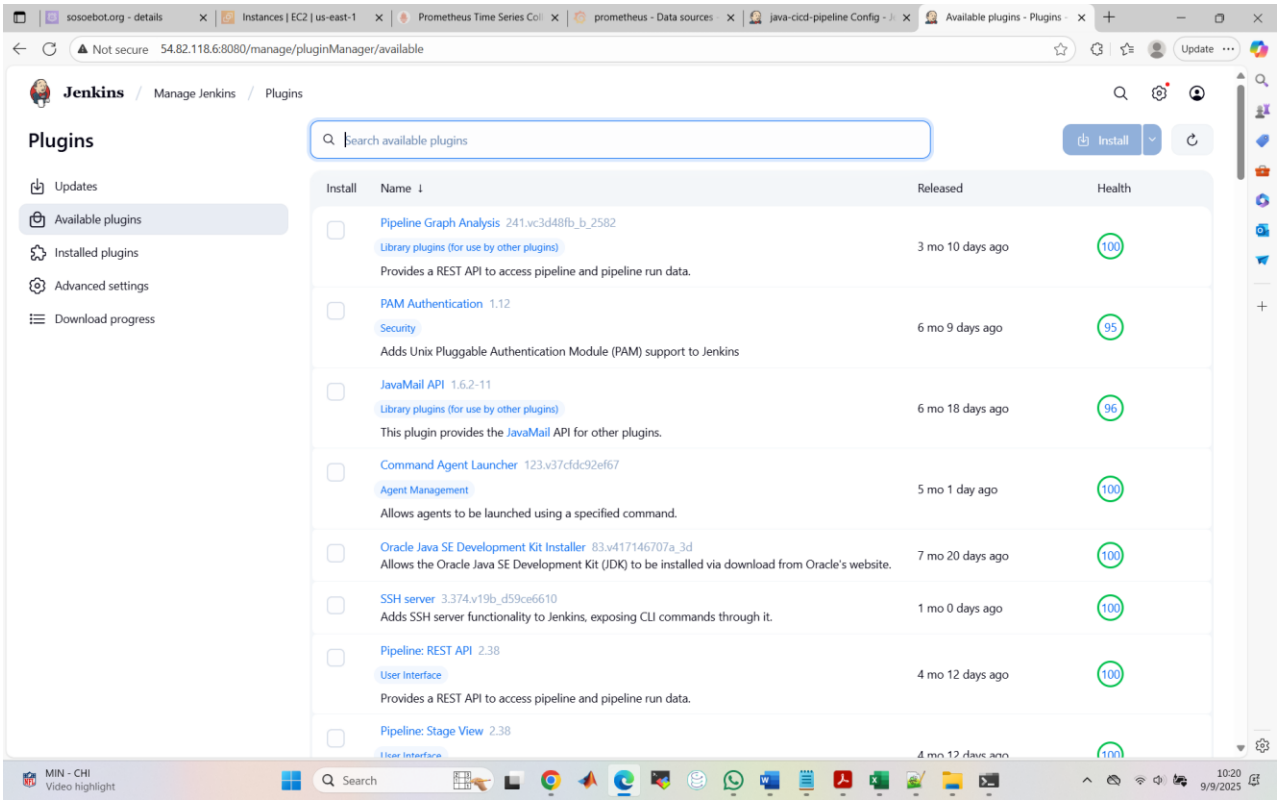
Click on “**Apply**” followed by “**Save**”



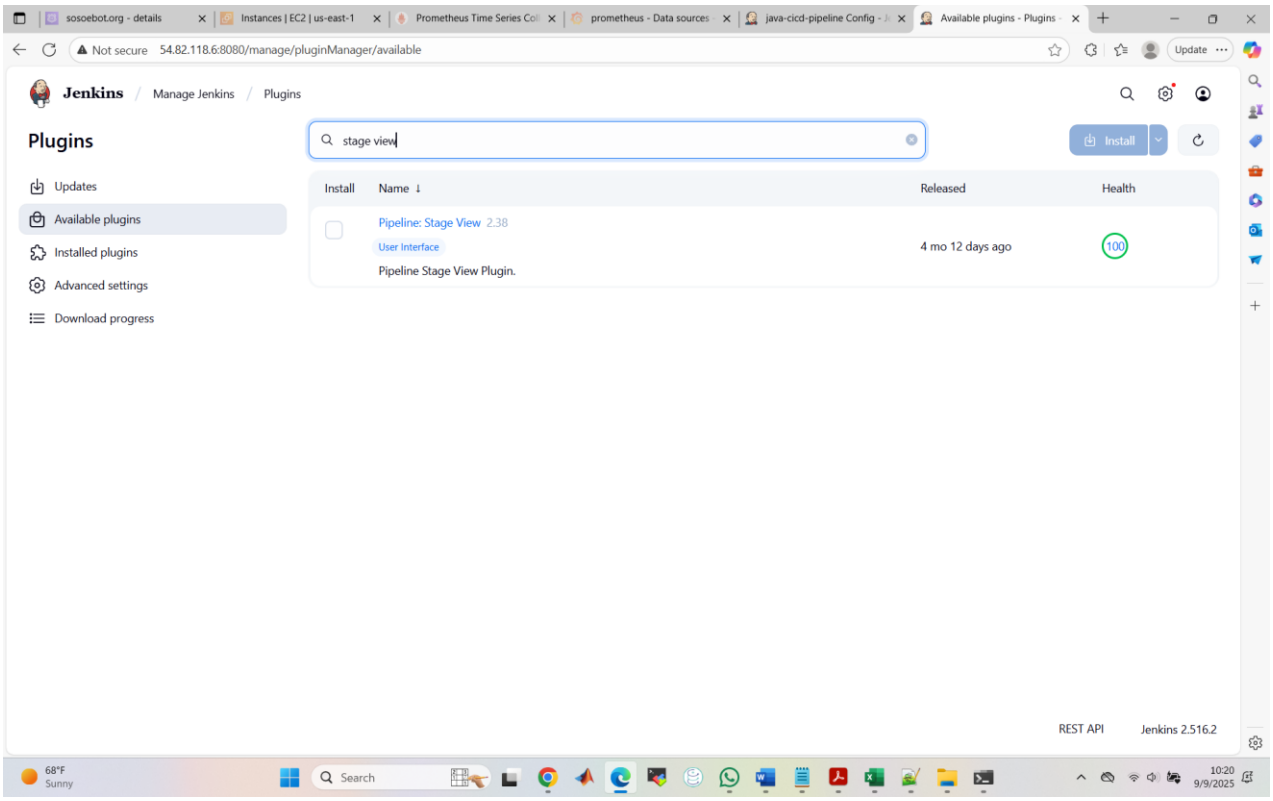
Let us install one more plugin called “**Pipeline Stage view**”. Click on “**plugins**”



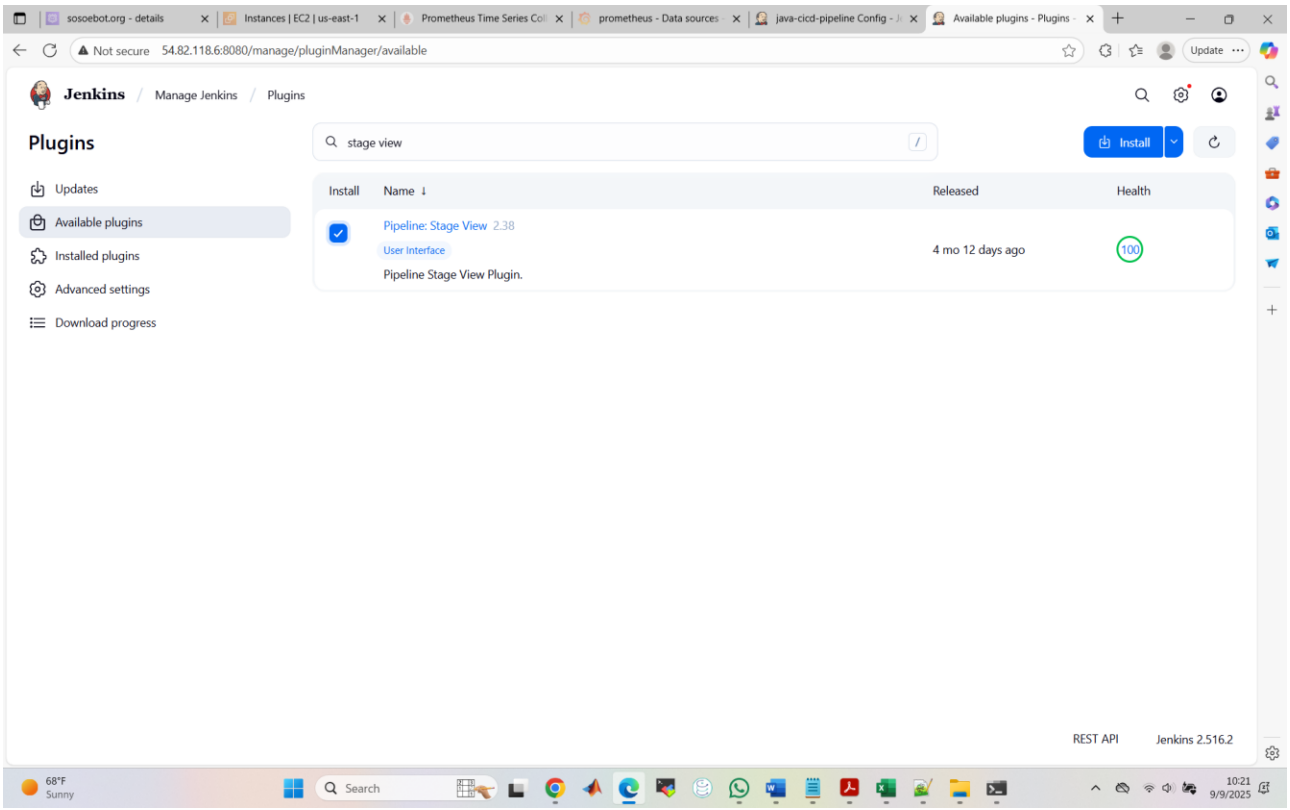
Then clikc on “**Available Plugins**”



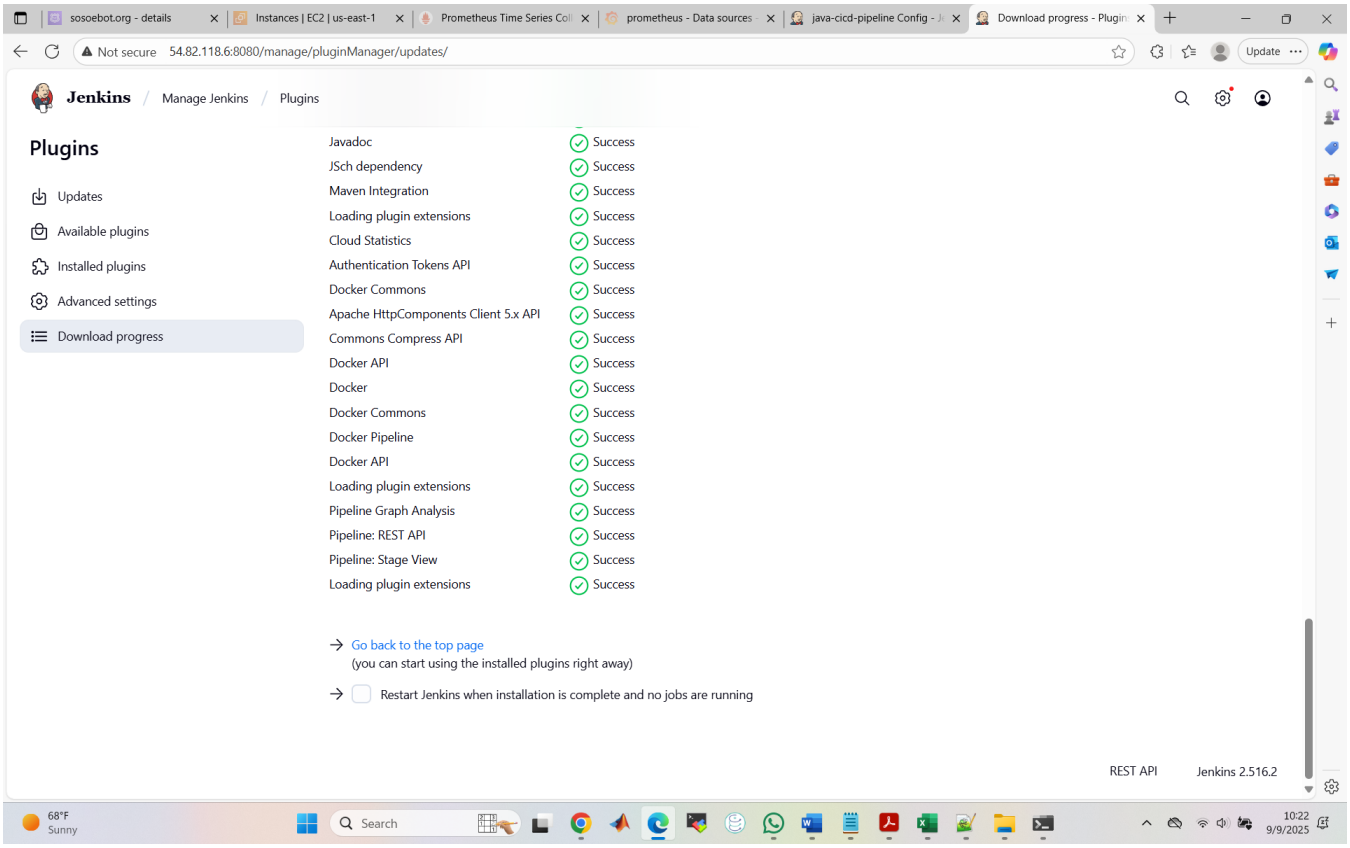
Search for “**Stage View**”



Select “**Pipeline: Stage View**”



Click on “**Install**”



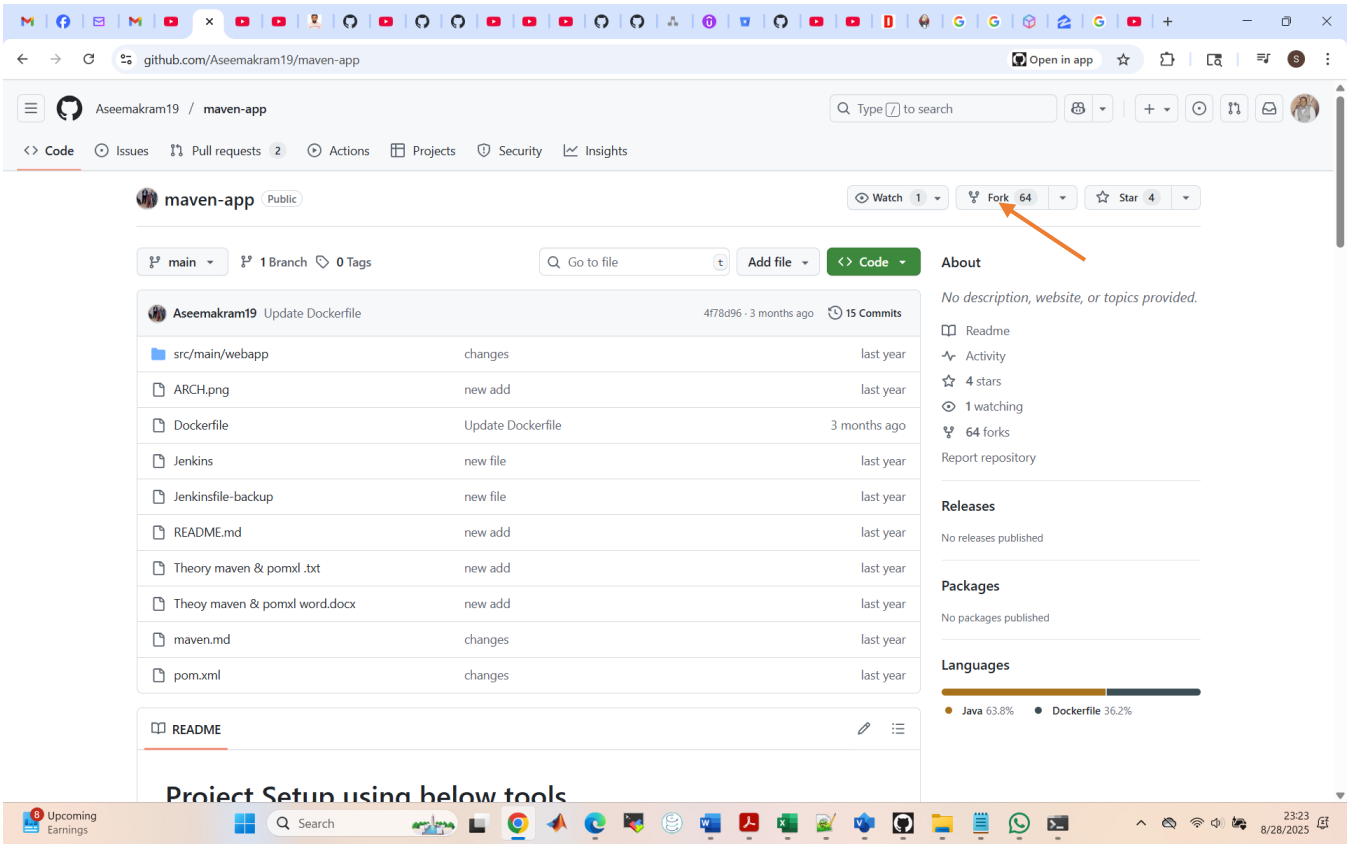
STEP 11: Add Stages to the Pipeline

IMPLEMENTATION

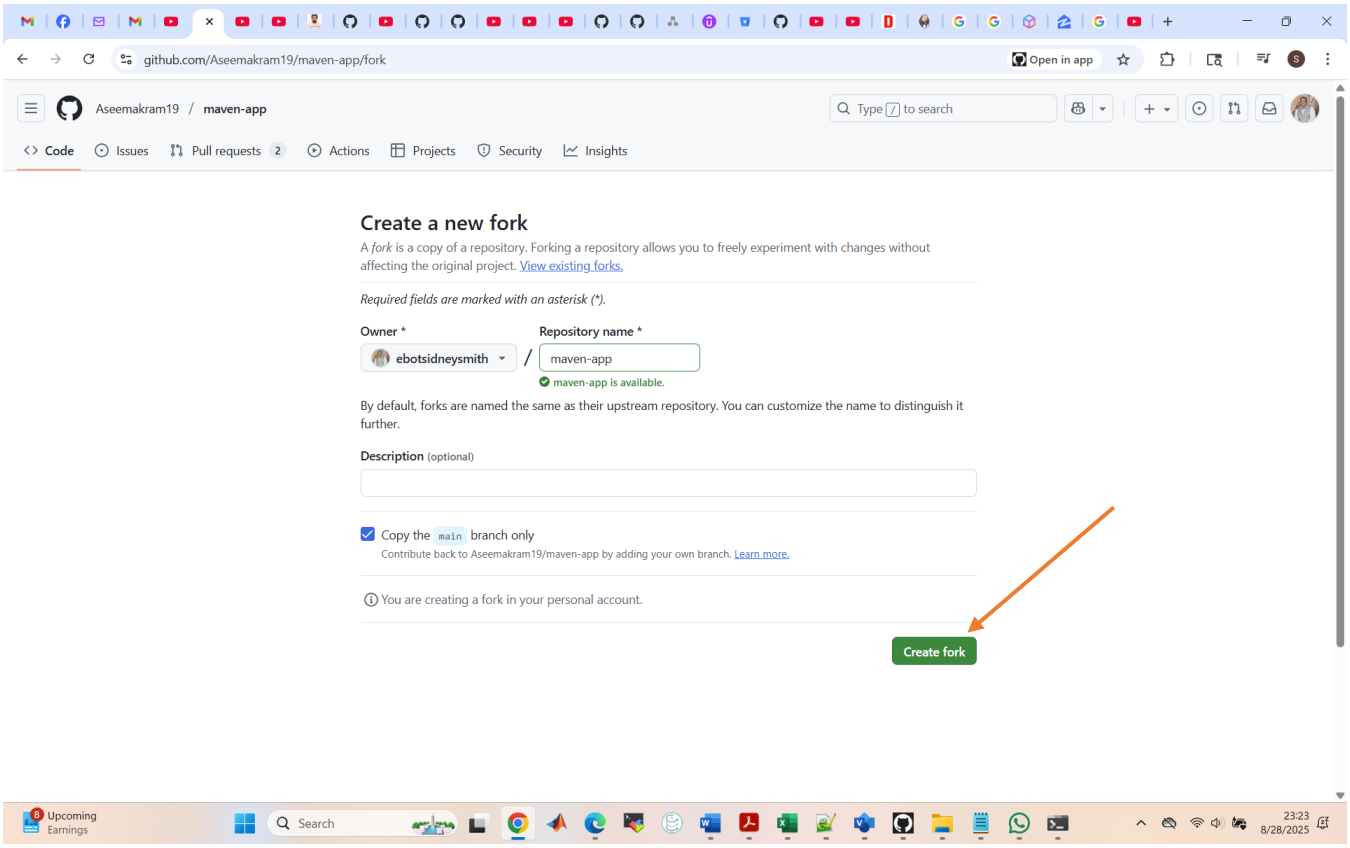
Let is now implement our pipeline following our architecture above.

Phase 1: Fork the repository

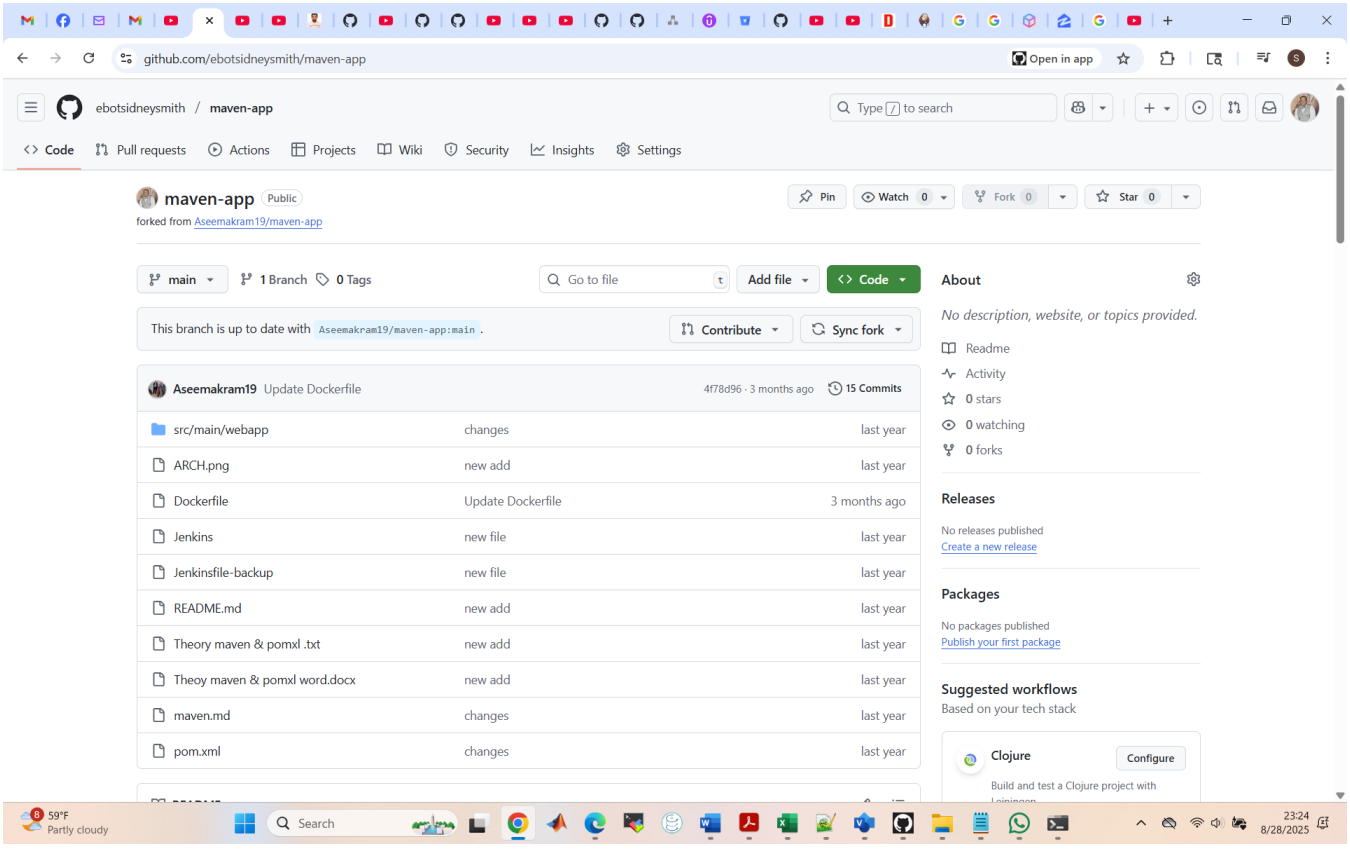
The first stage of our pipeline is to fetch the code from GitHub. Go to the GitHub repository: https://github.com/Aseemakram19/maven-app



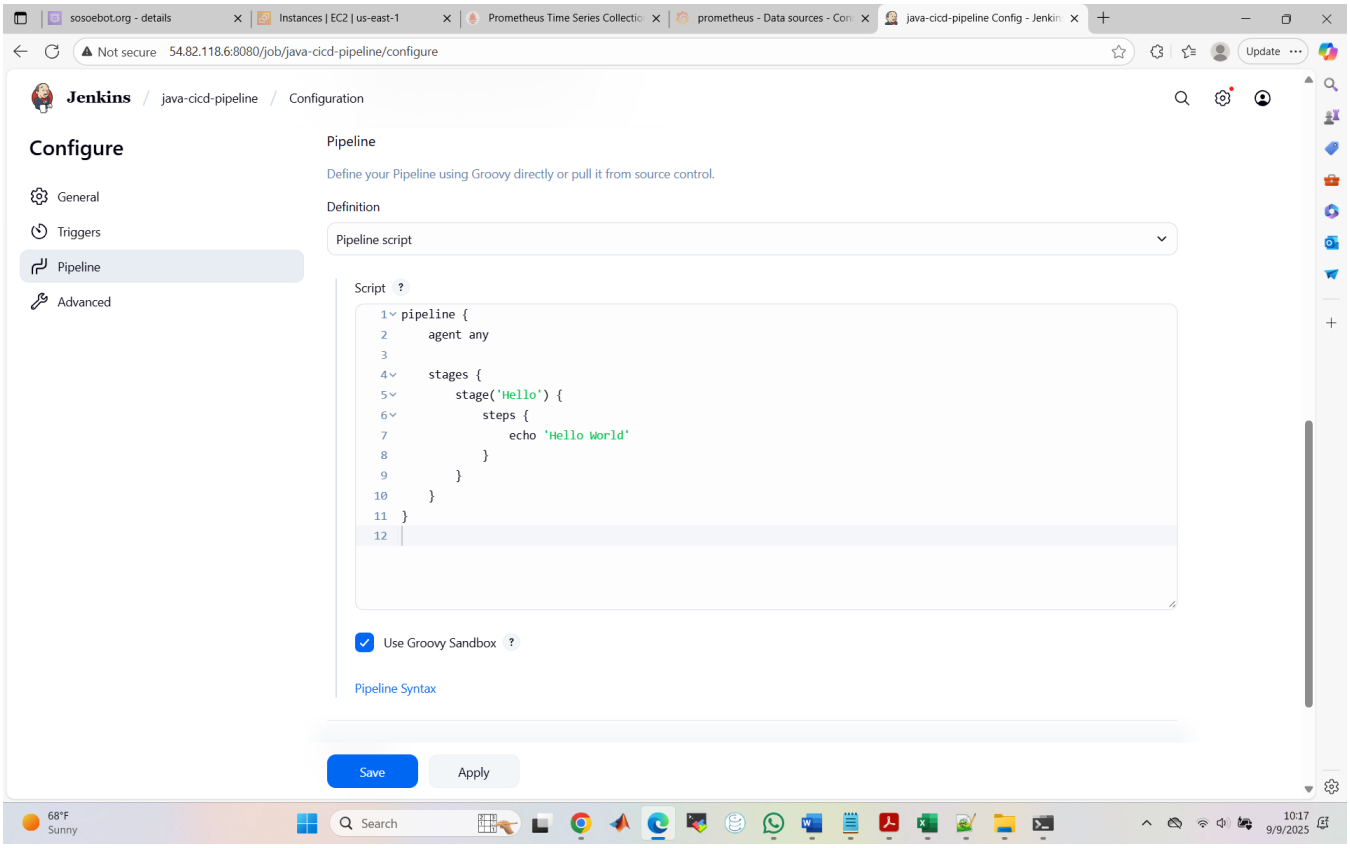
I will fork this repository by clicking on "**Fork**"



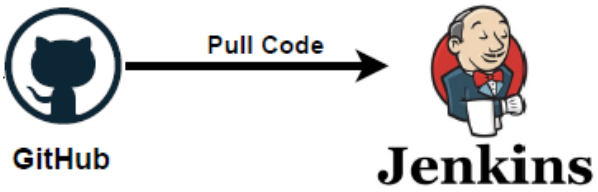
Click on “**Create Fork**”

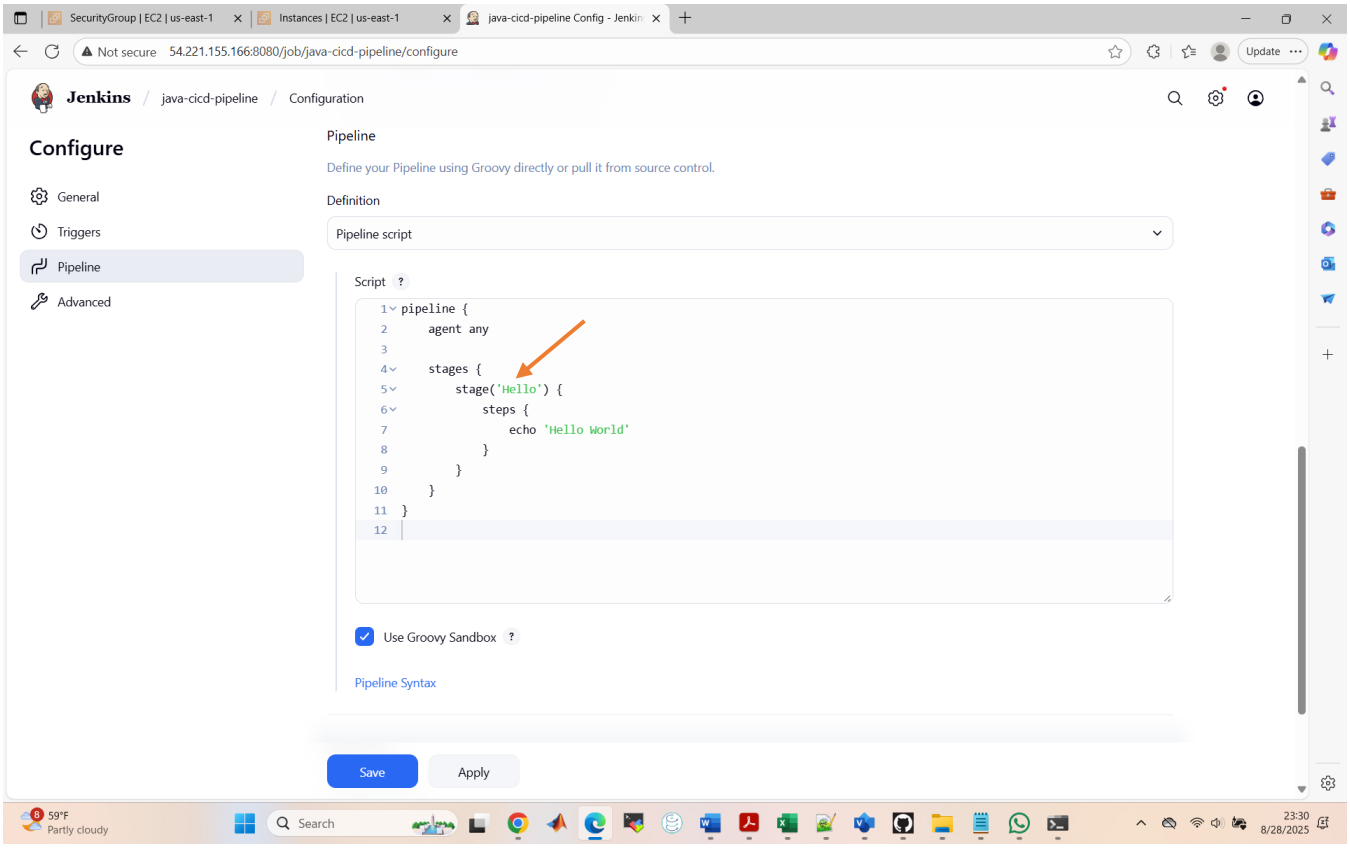


The repository has been forked. Now go to our pipeline code and start modifying it
Head back to Jenkins GUI and click on “**Pipeline**”

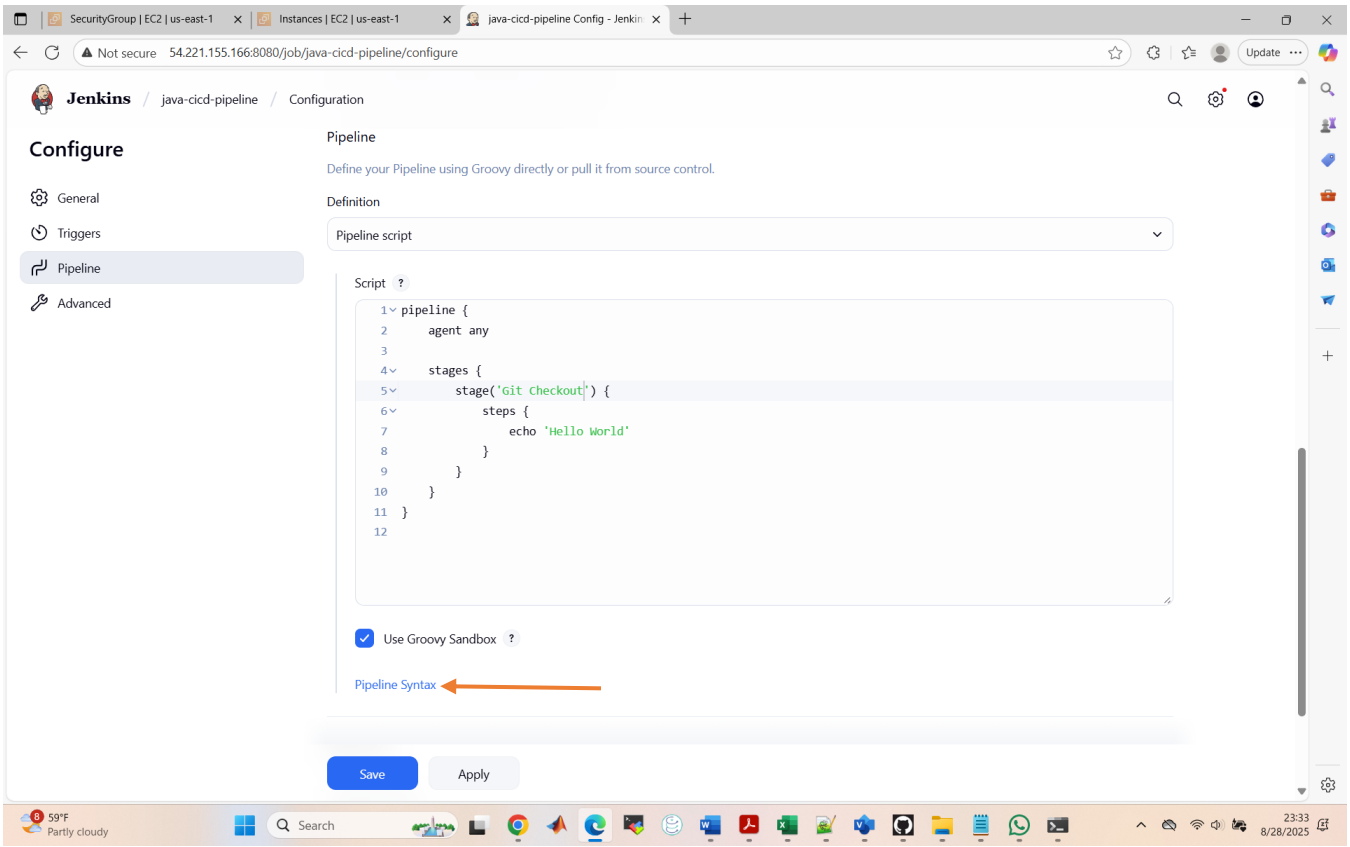


Phase 2: Clone the Repository to Jenkins

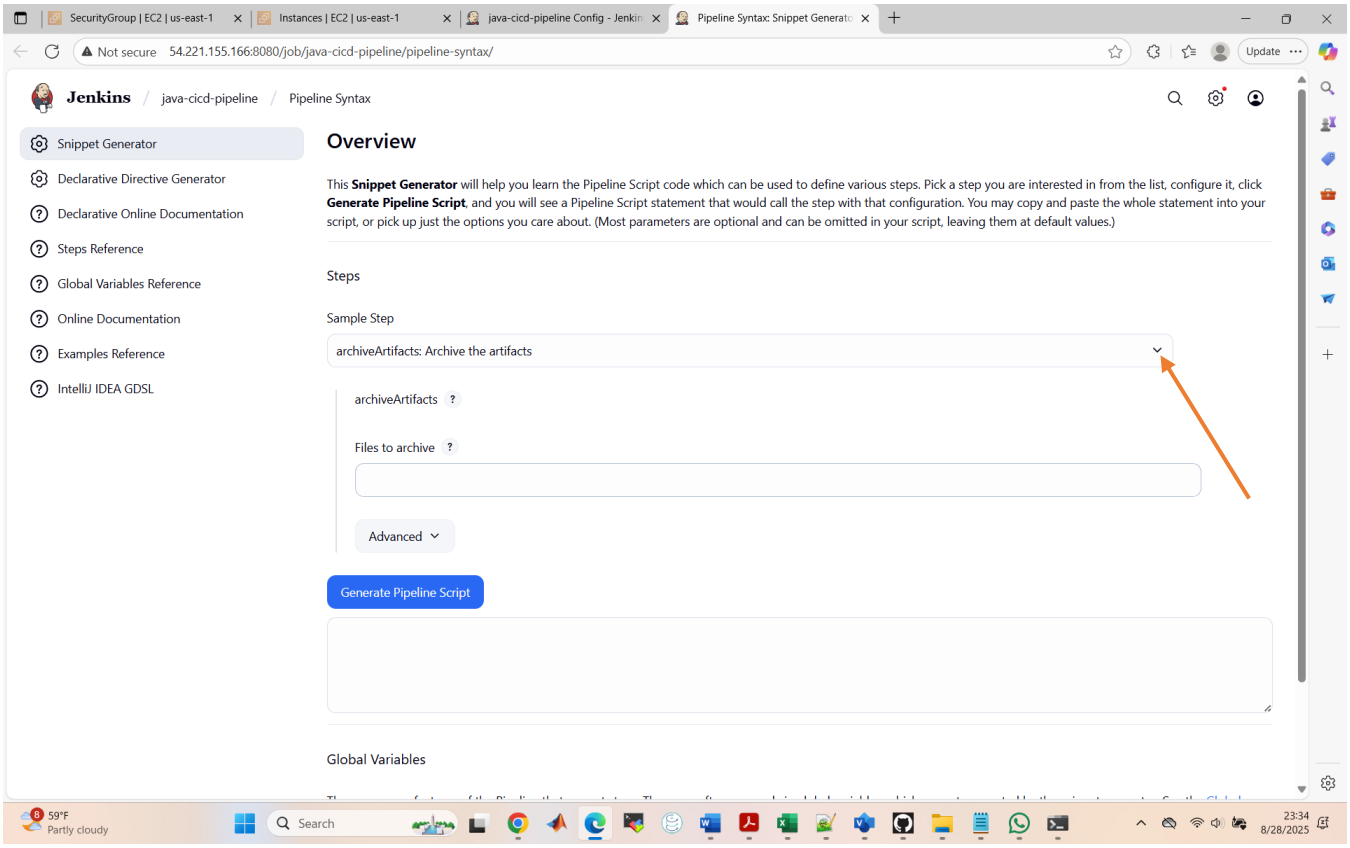




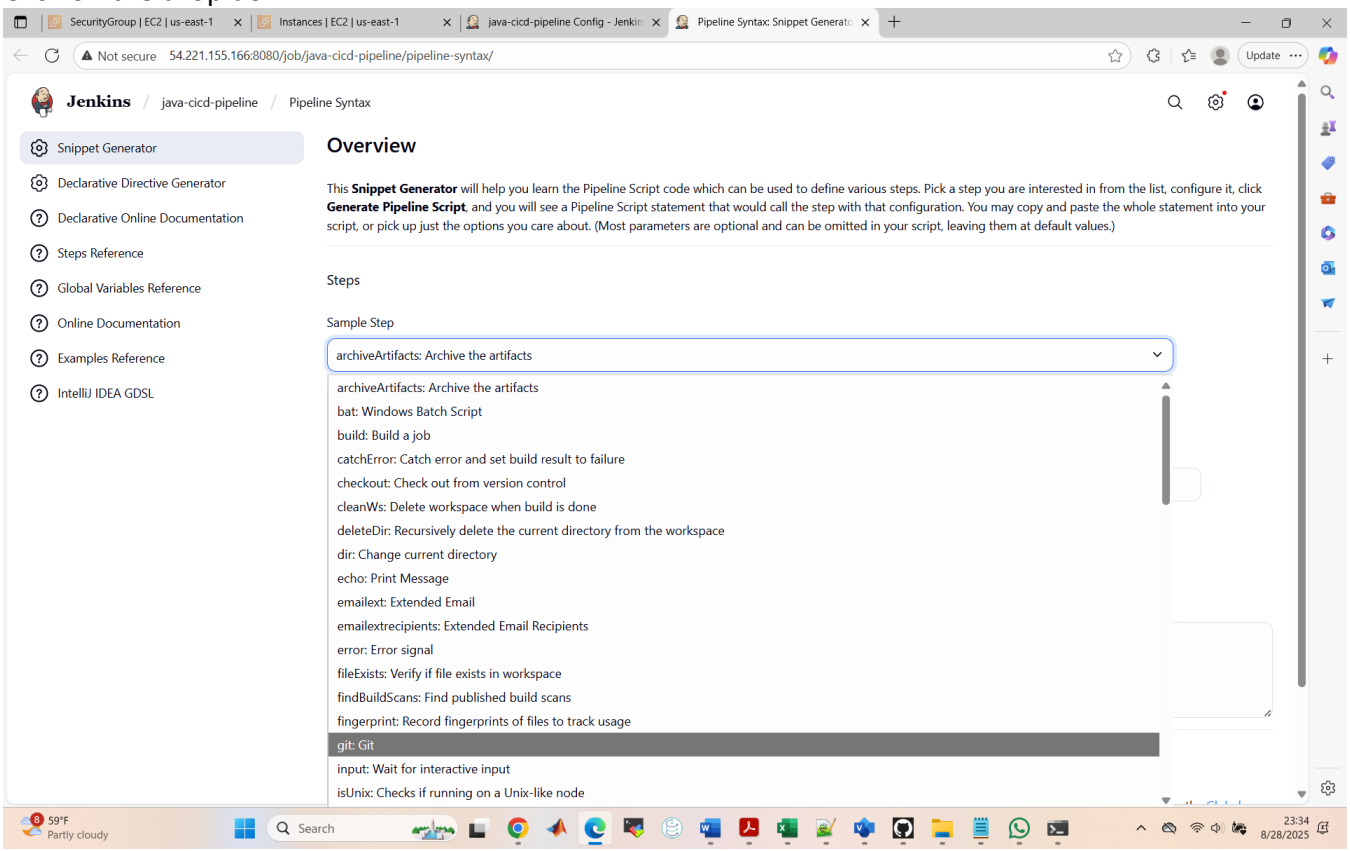
In the first line on the first stage, change the “Hello” to “**Git Checkout**”



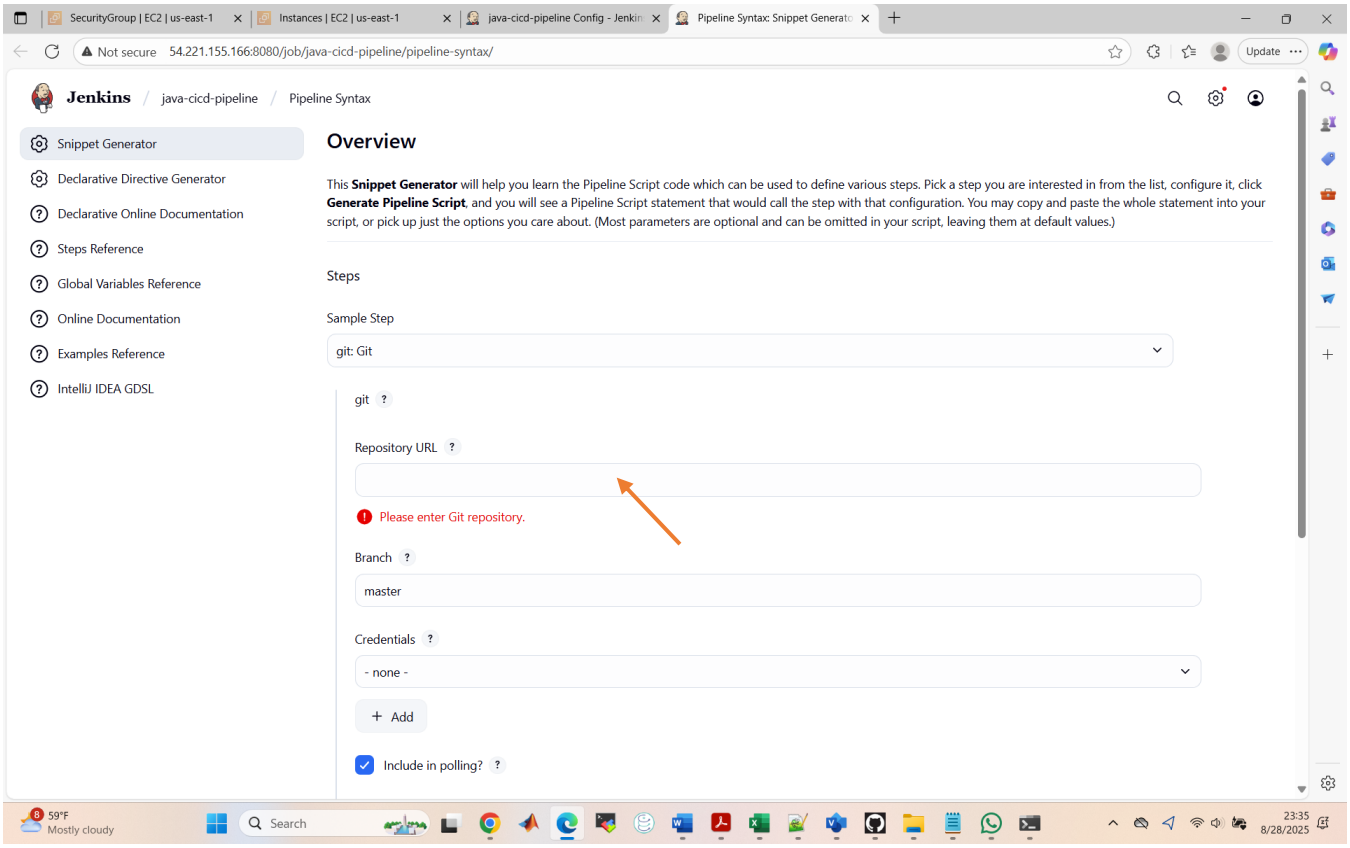
Then click on “**Pipeline Syntax**”, a new window will open



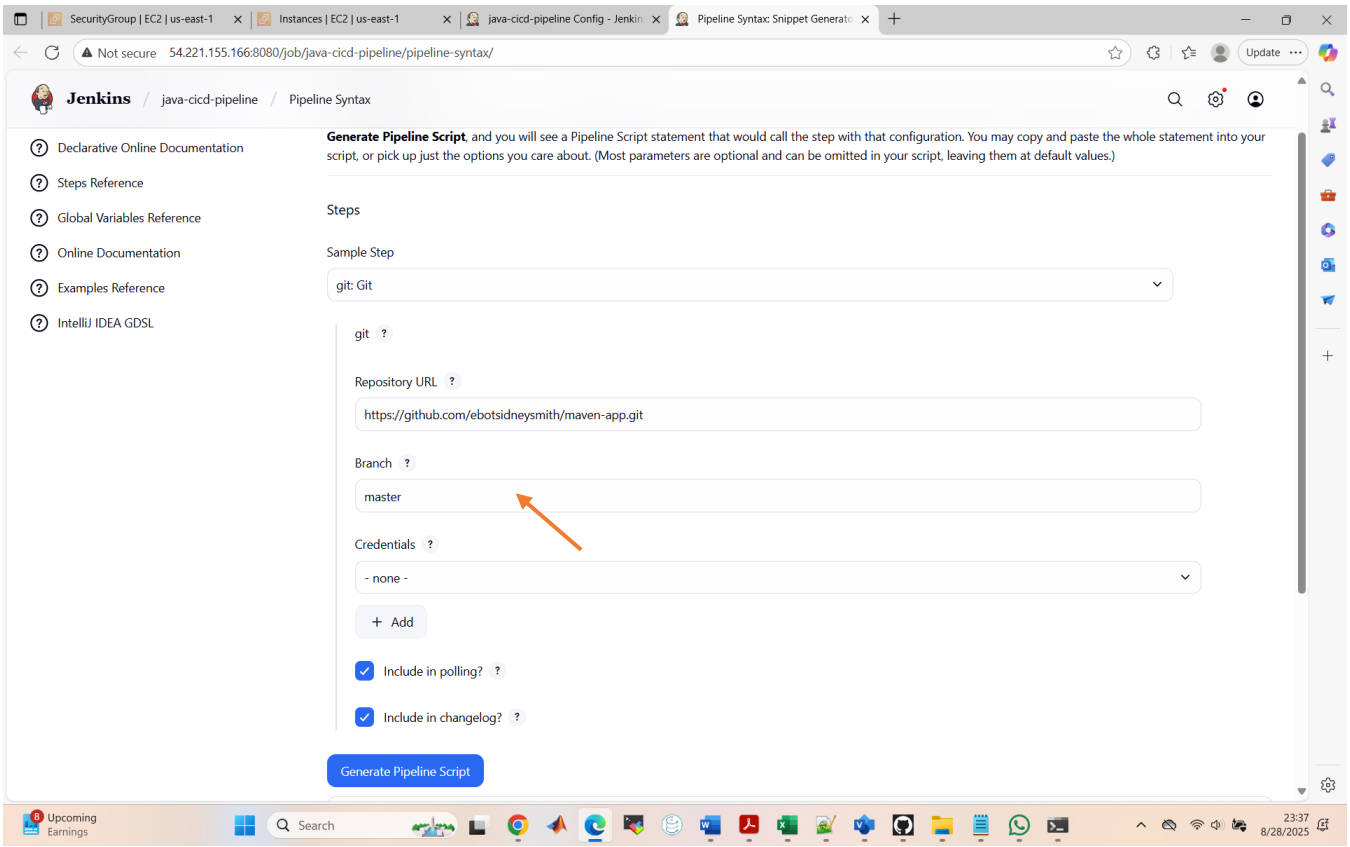
Click on the drop down



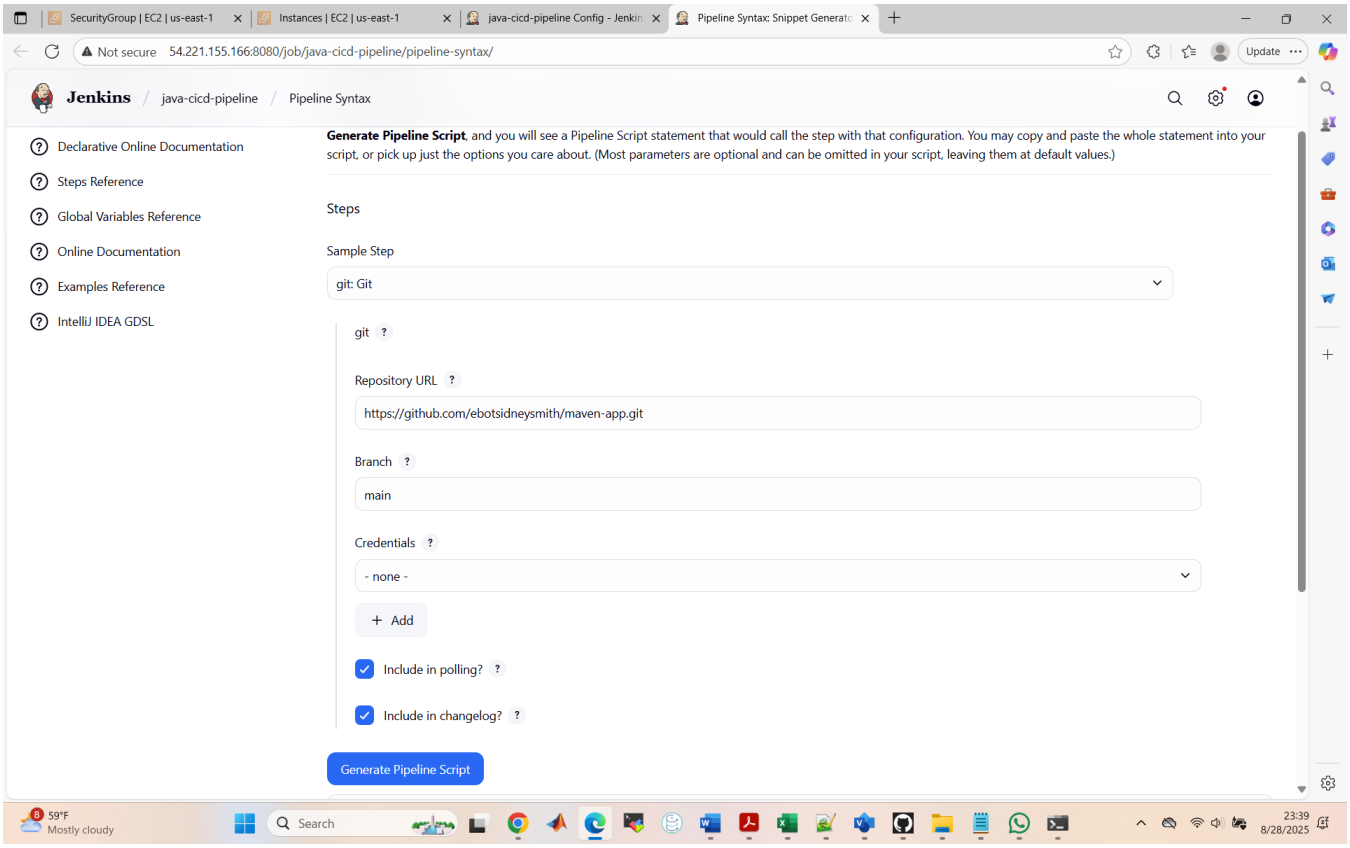
Select “**git:Git**”



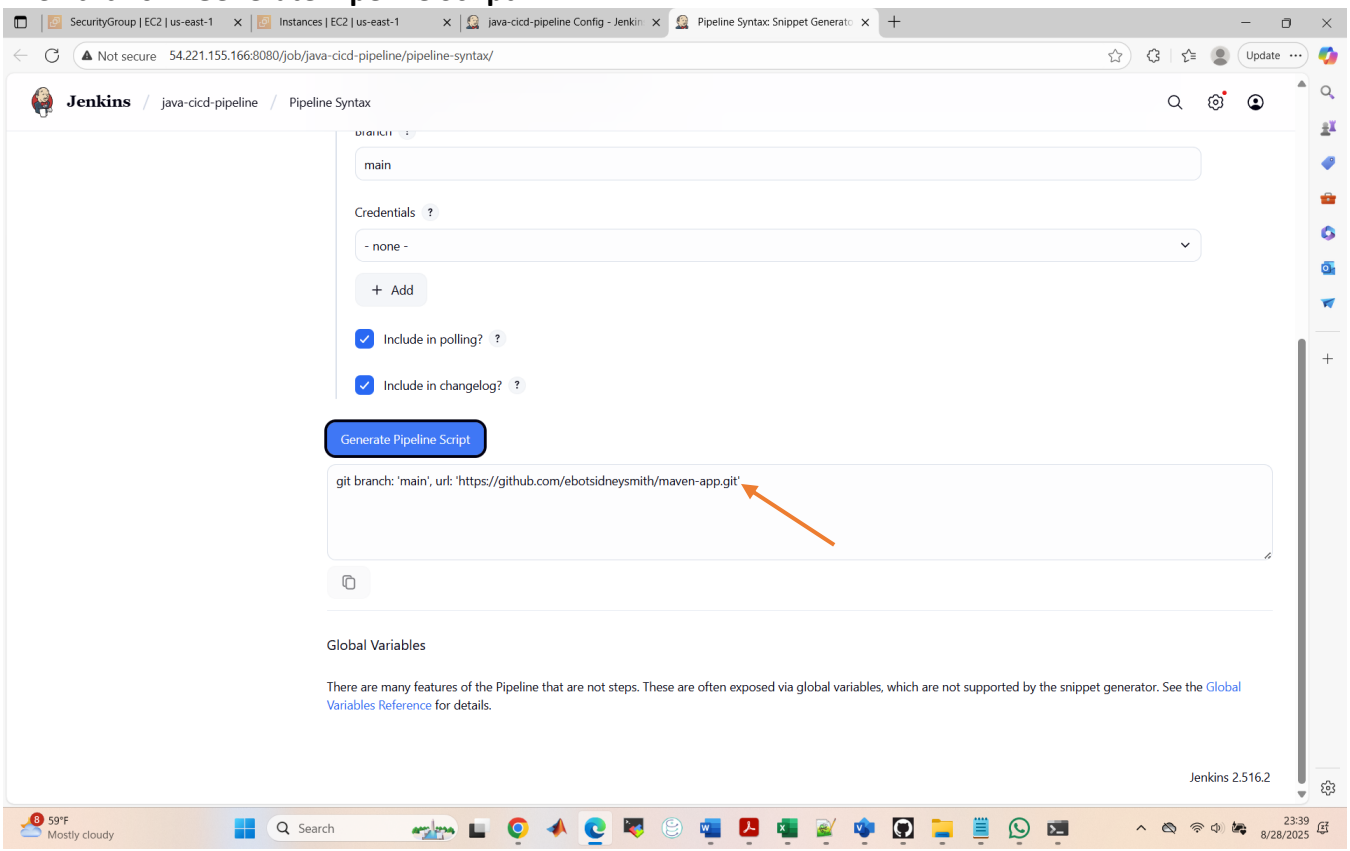
Copy the URL of the GitHub repository and paste on "**Repository URL**"



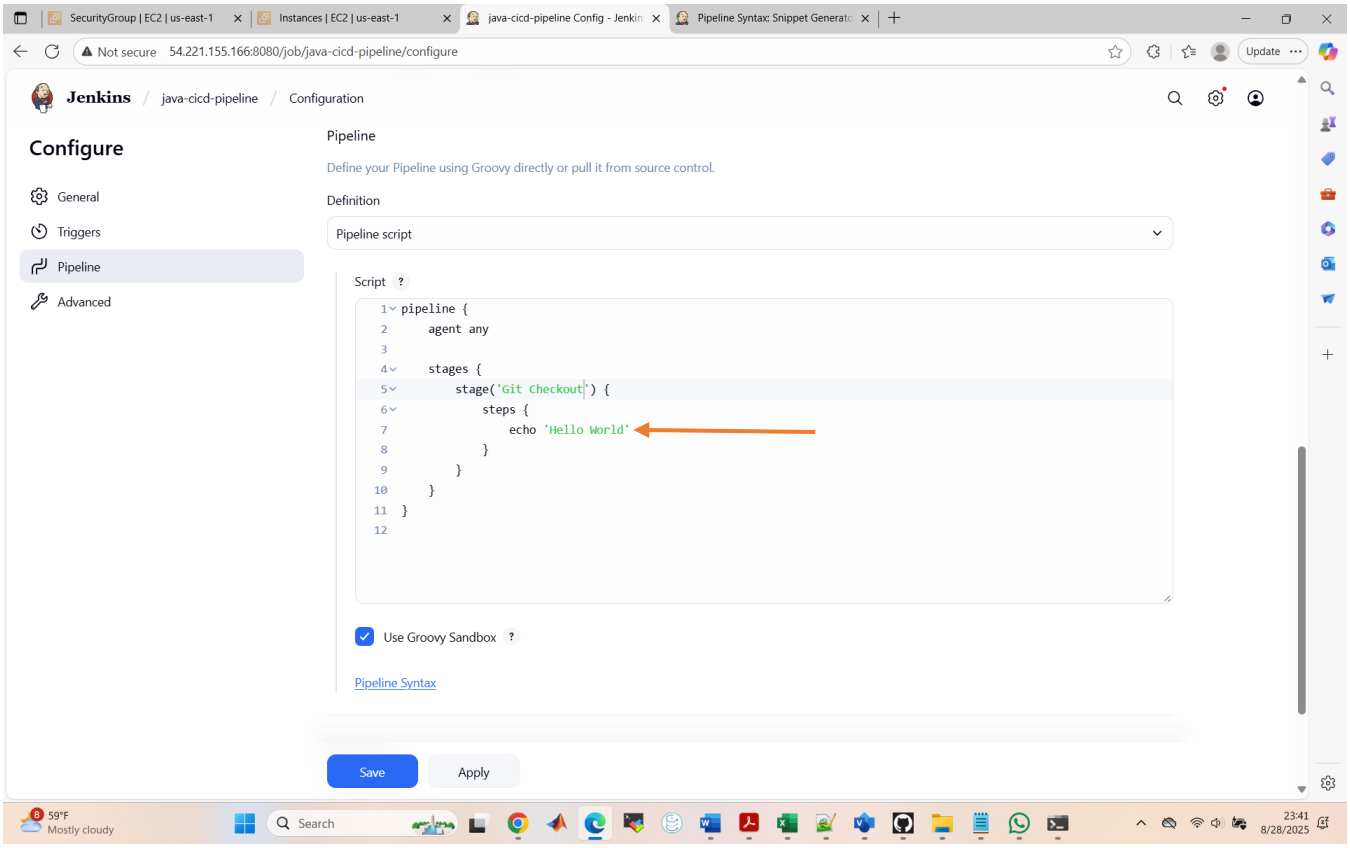
Change "**Master**" to "**Main**" since our code is in the main branch of our repository



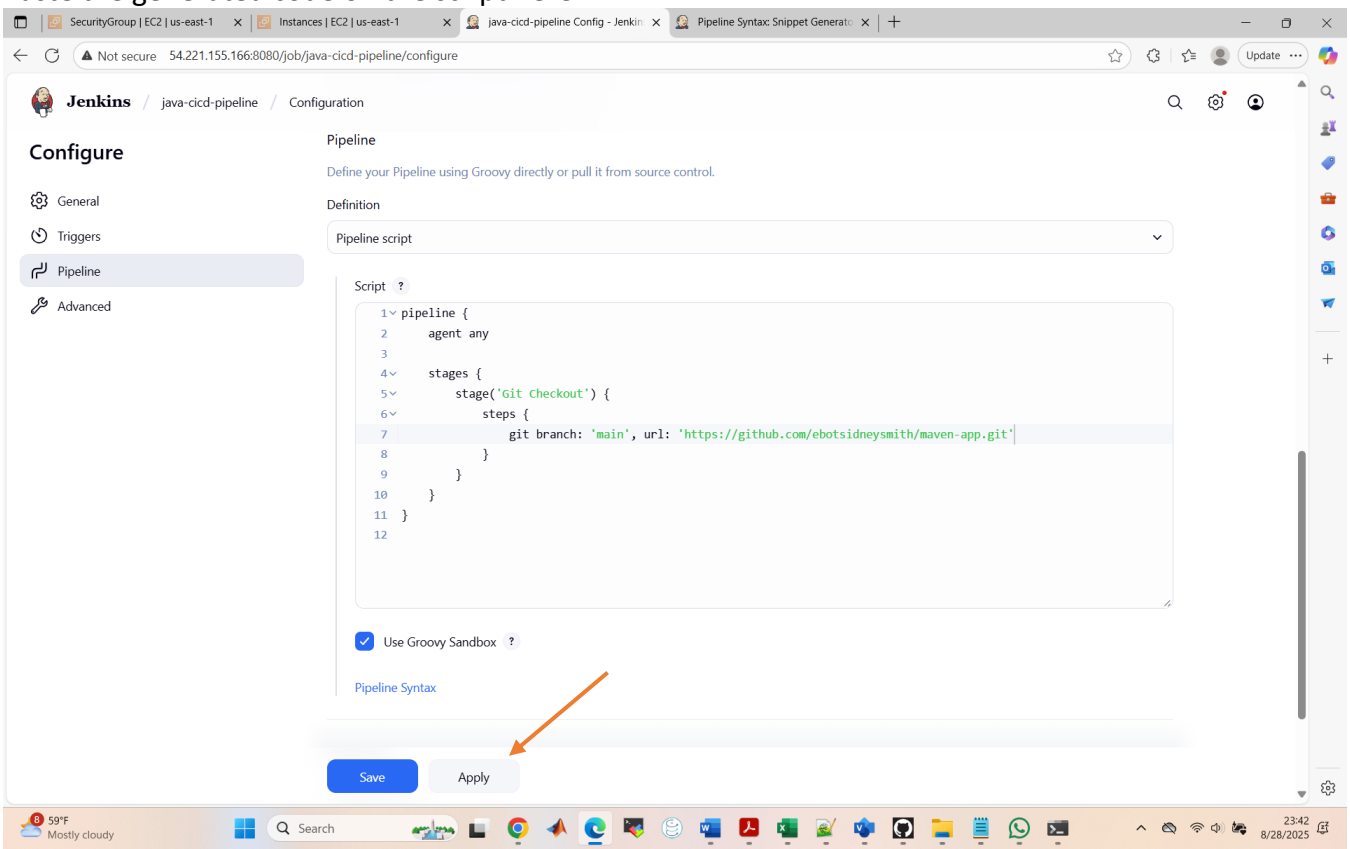
Then click on "**Generate Pipeline Script**"



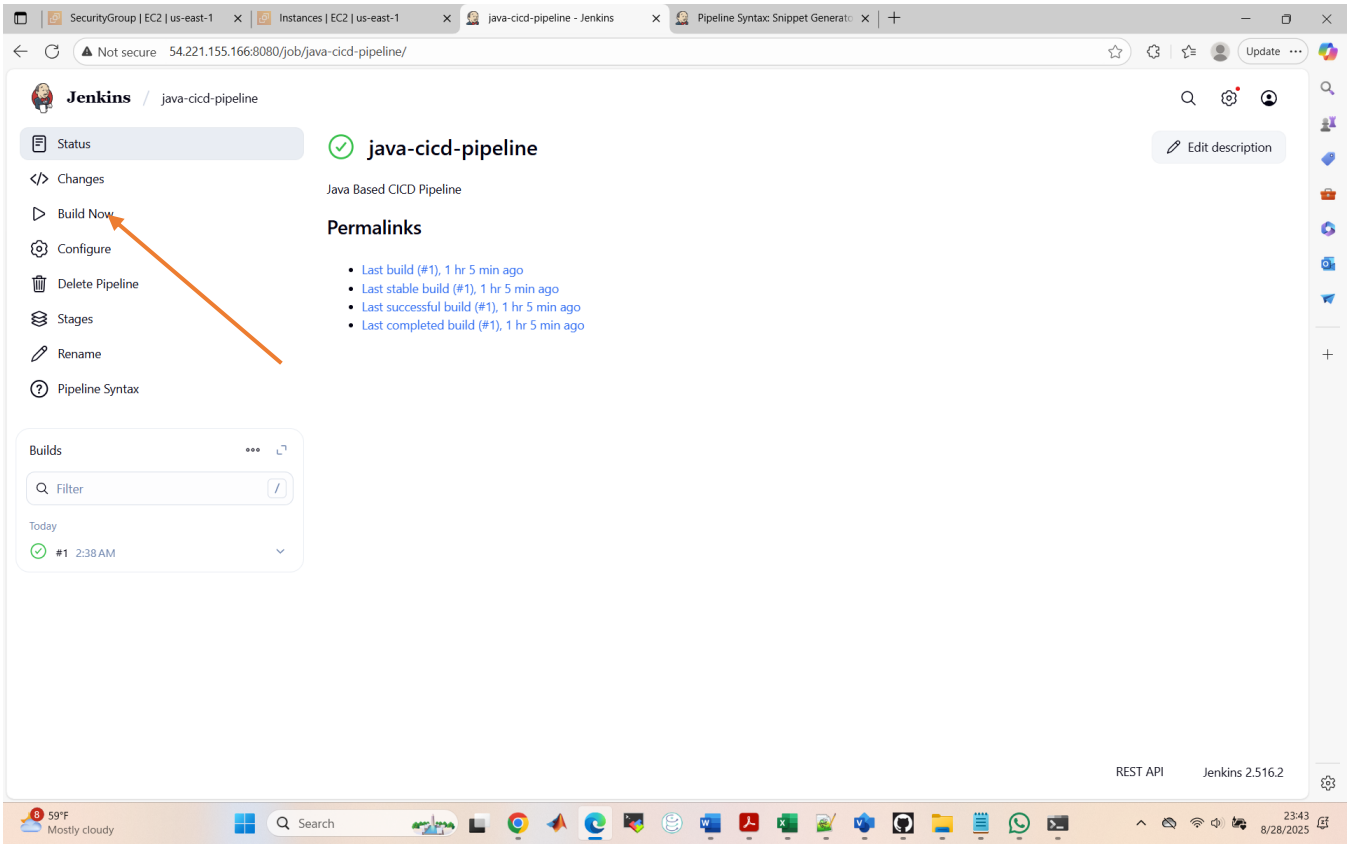
The script has been generated. Copy the script and go back to our previous Jenkins window



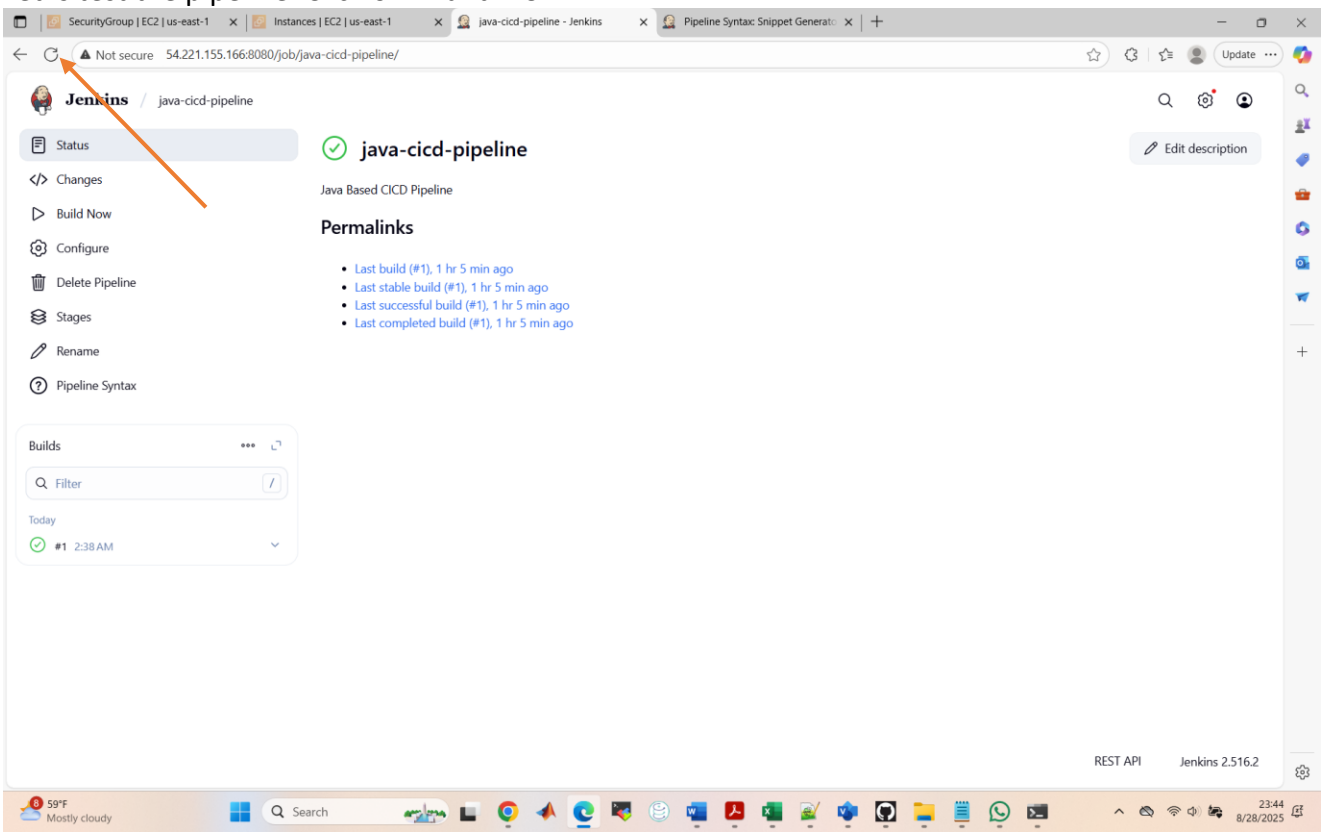
Paste the generated code on the script here



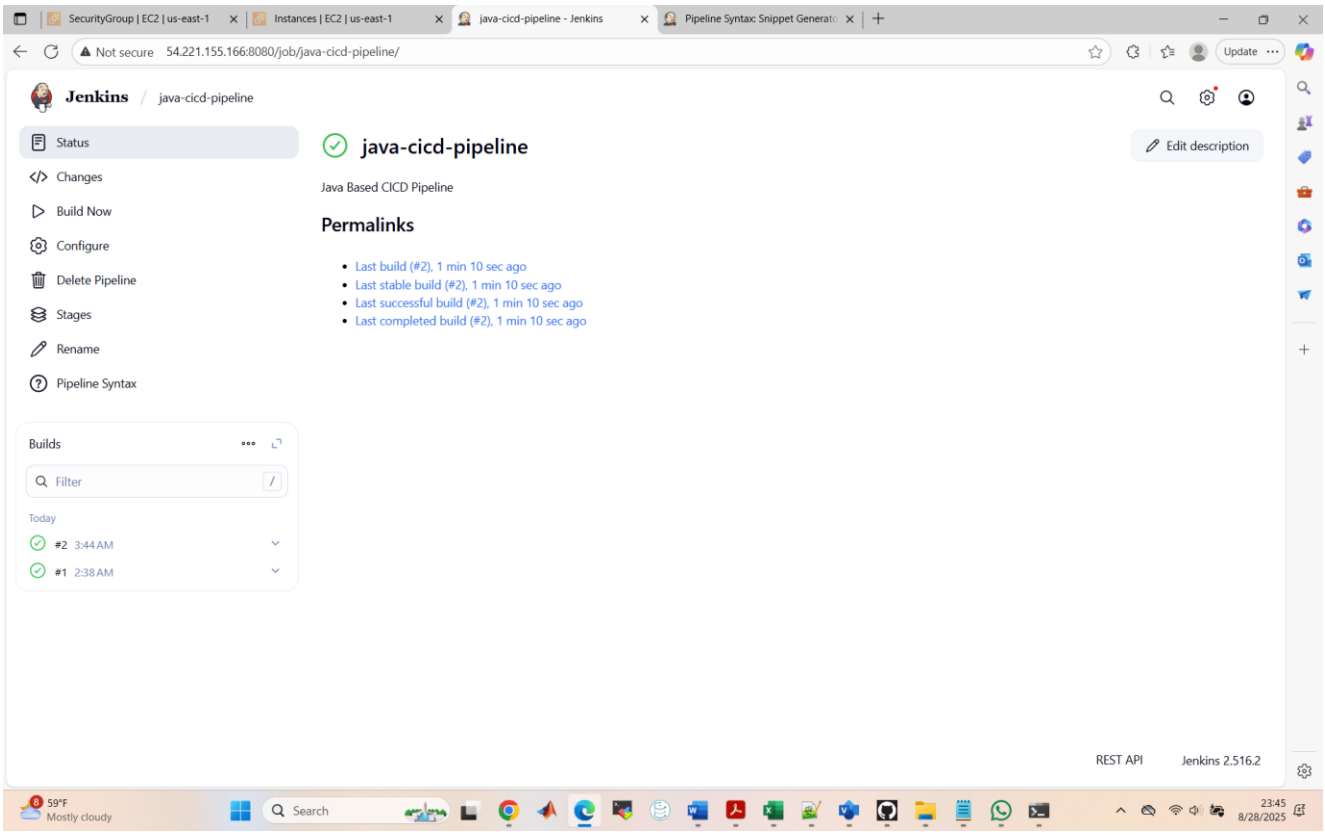
Click on “**Apply**”, followed by “**Save**”



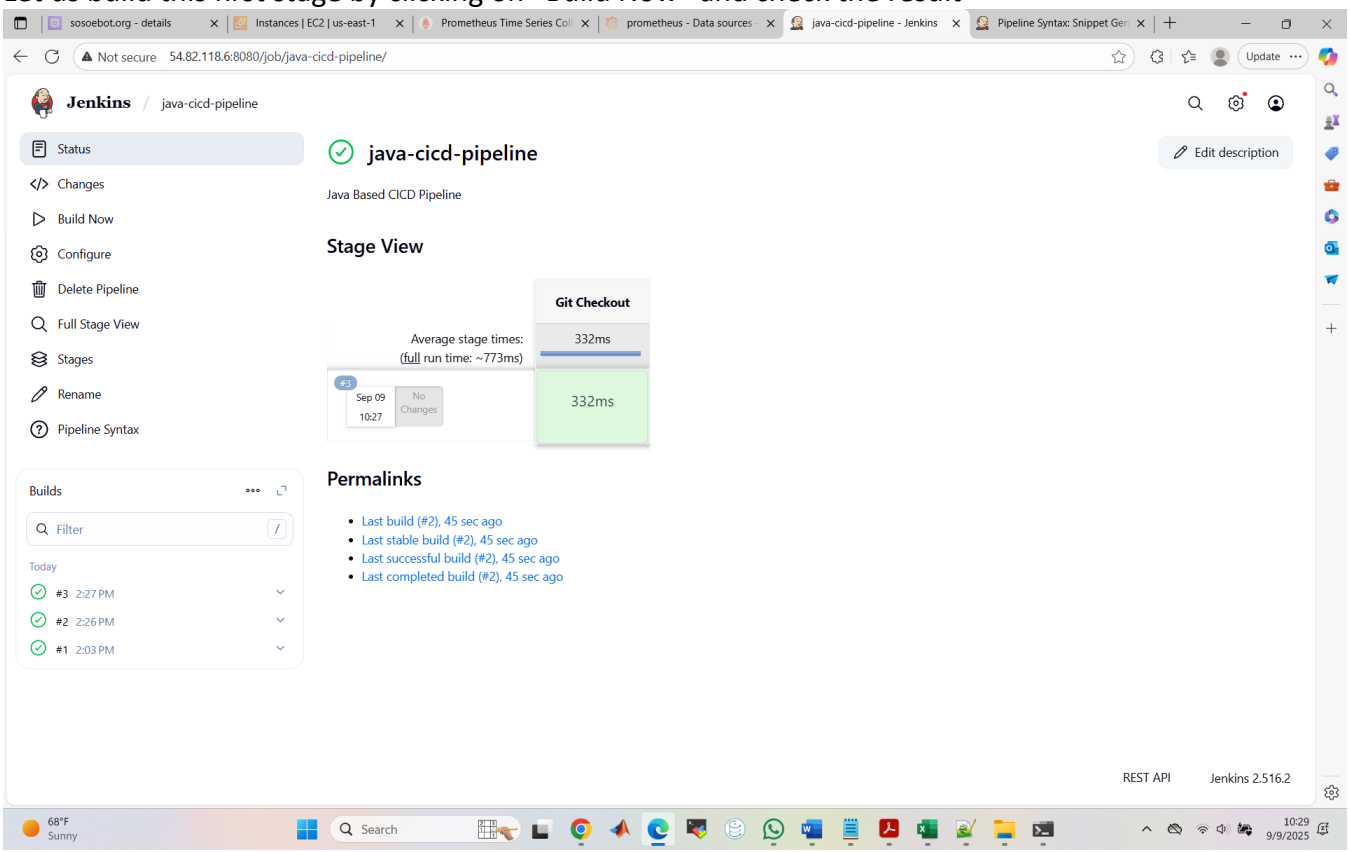
Let is test the pipeline. Click on "**Build Now**"



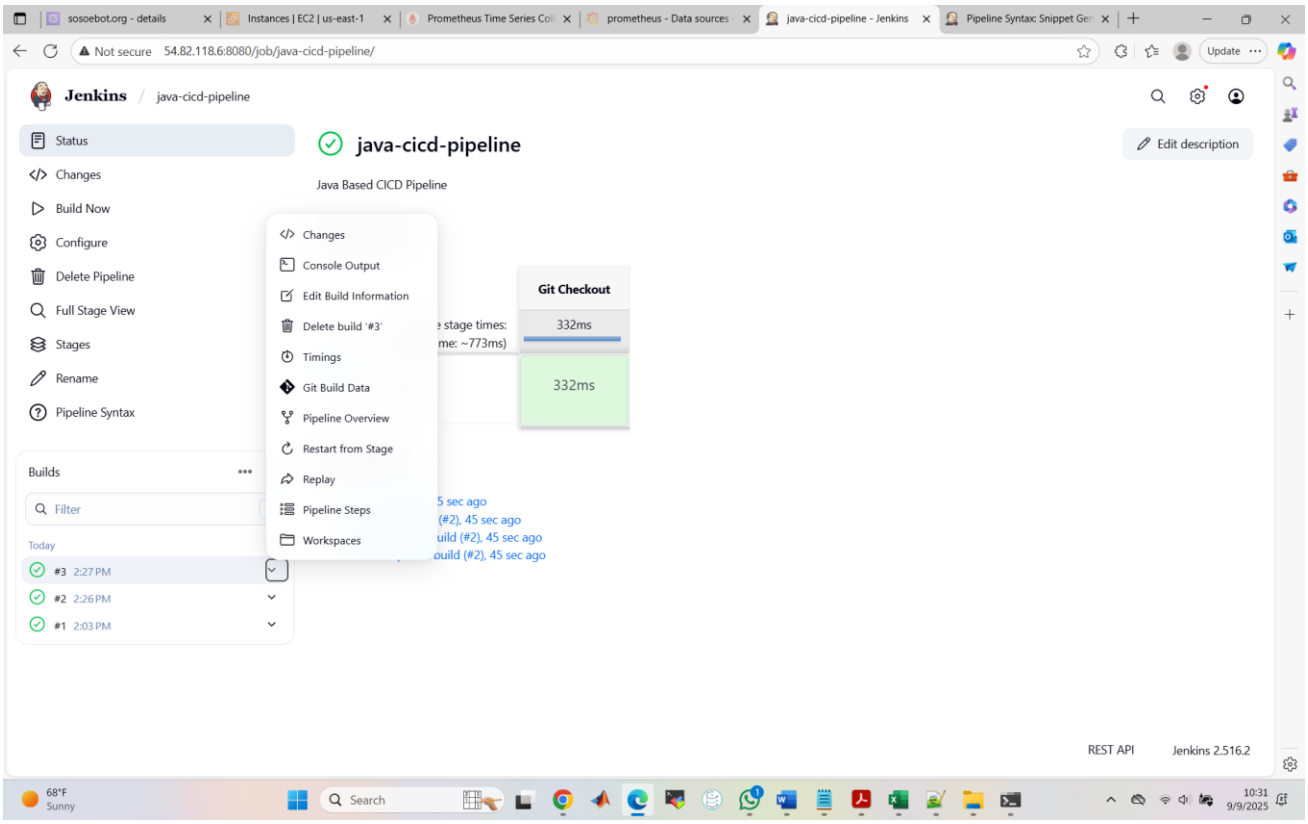
Refresh the page



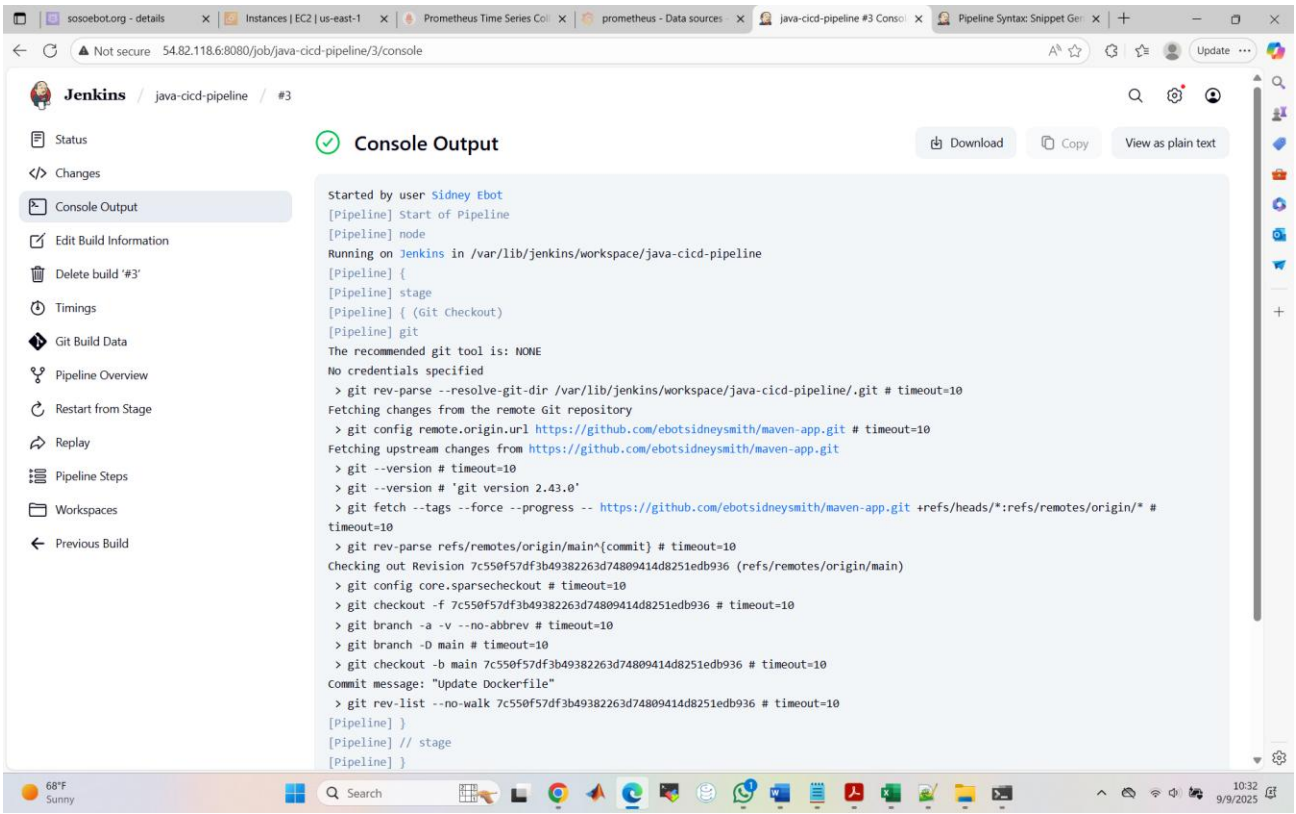
Let us build this first stage by clicking on “Build Now” and check the result



You can see that the build is successful. Click on the drop down on “#3”



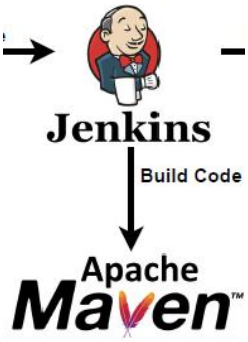
Select "**Console Output**"



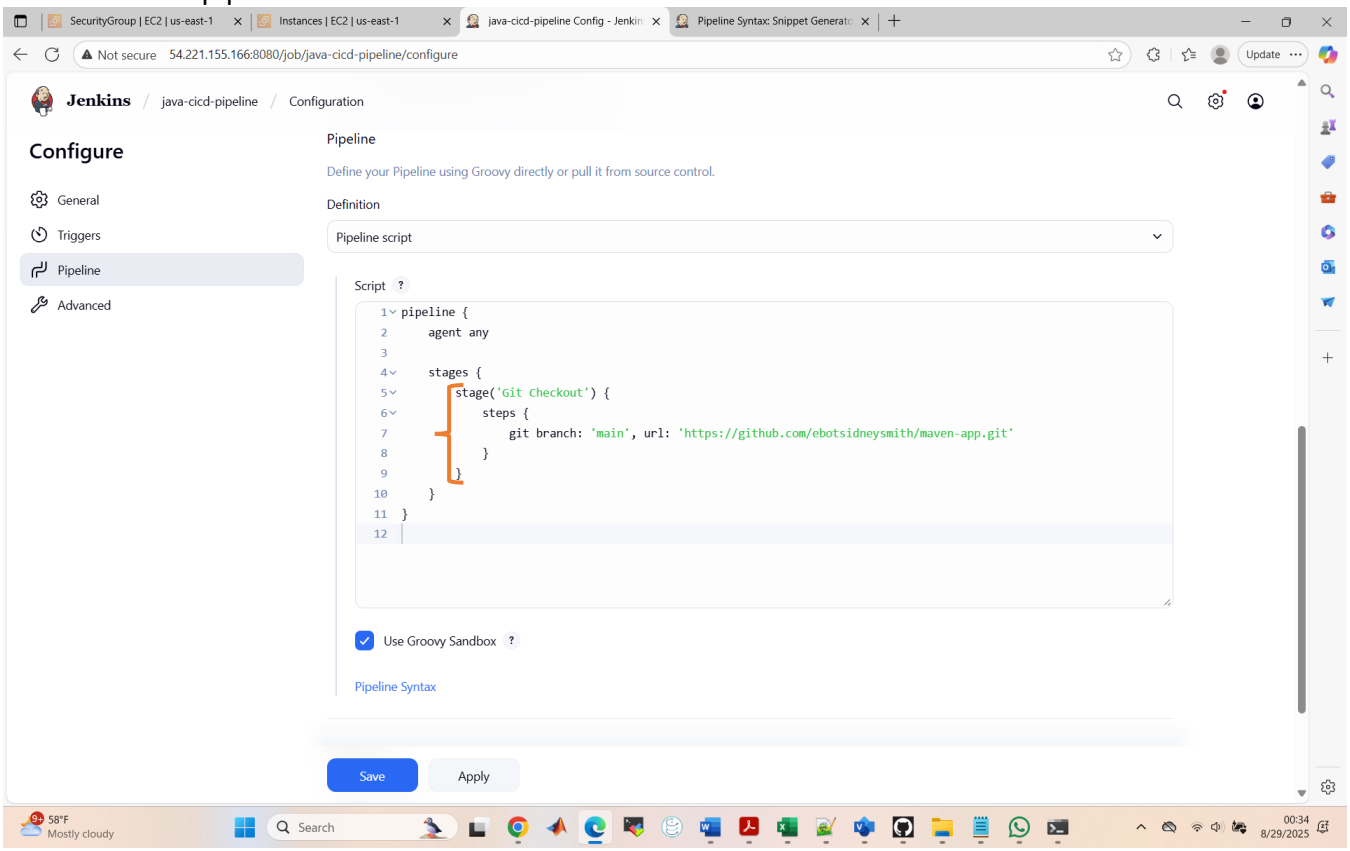
You can see that the build is successful. Git has cloned the repository. So, we are done with this part of the architecture

Phase 3: Build the code with Maven

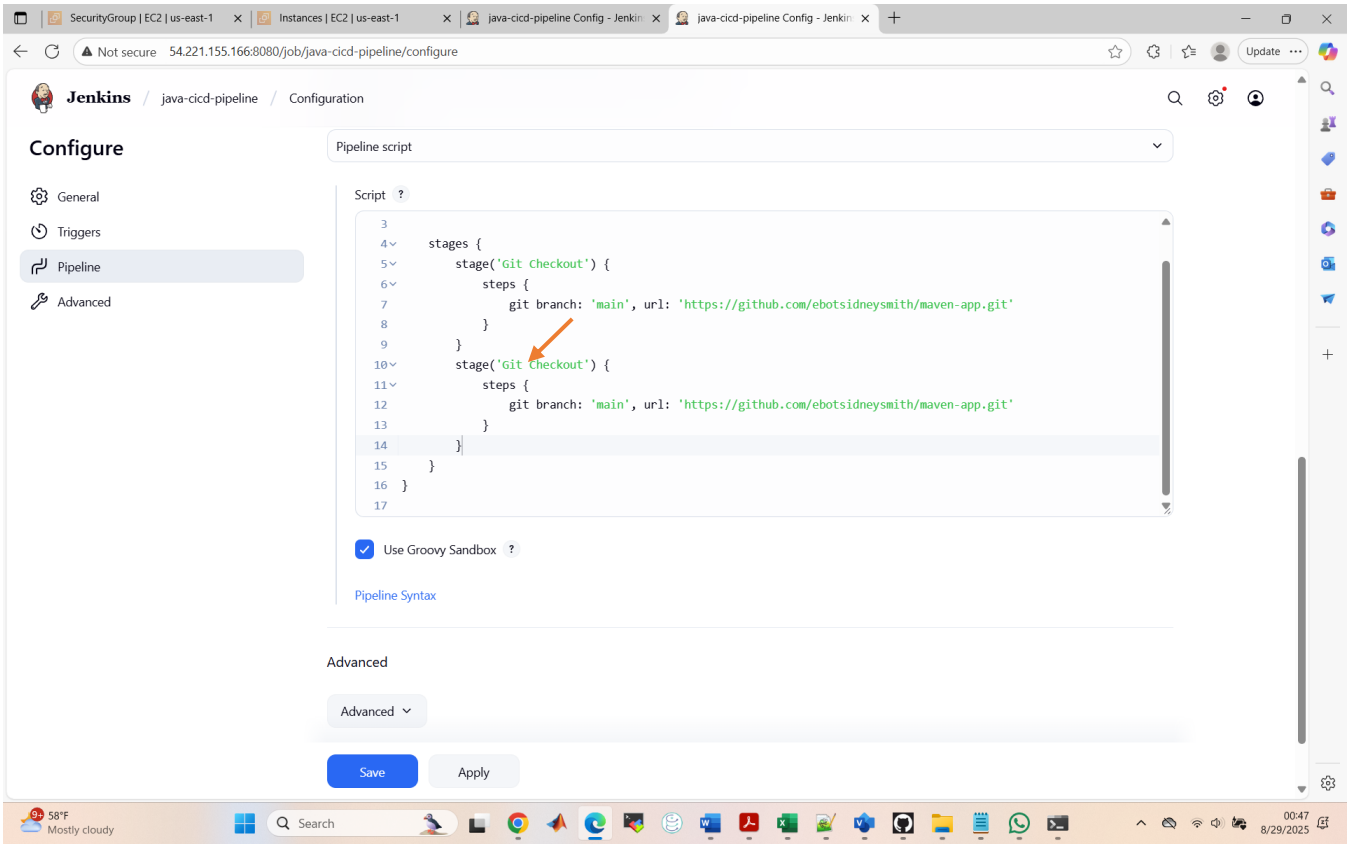
The next thing to do is to Build the code using Maven



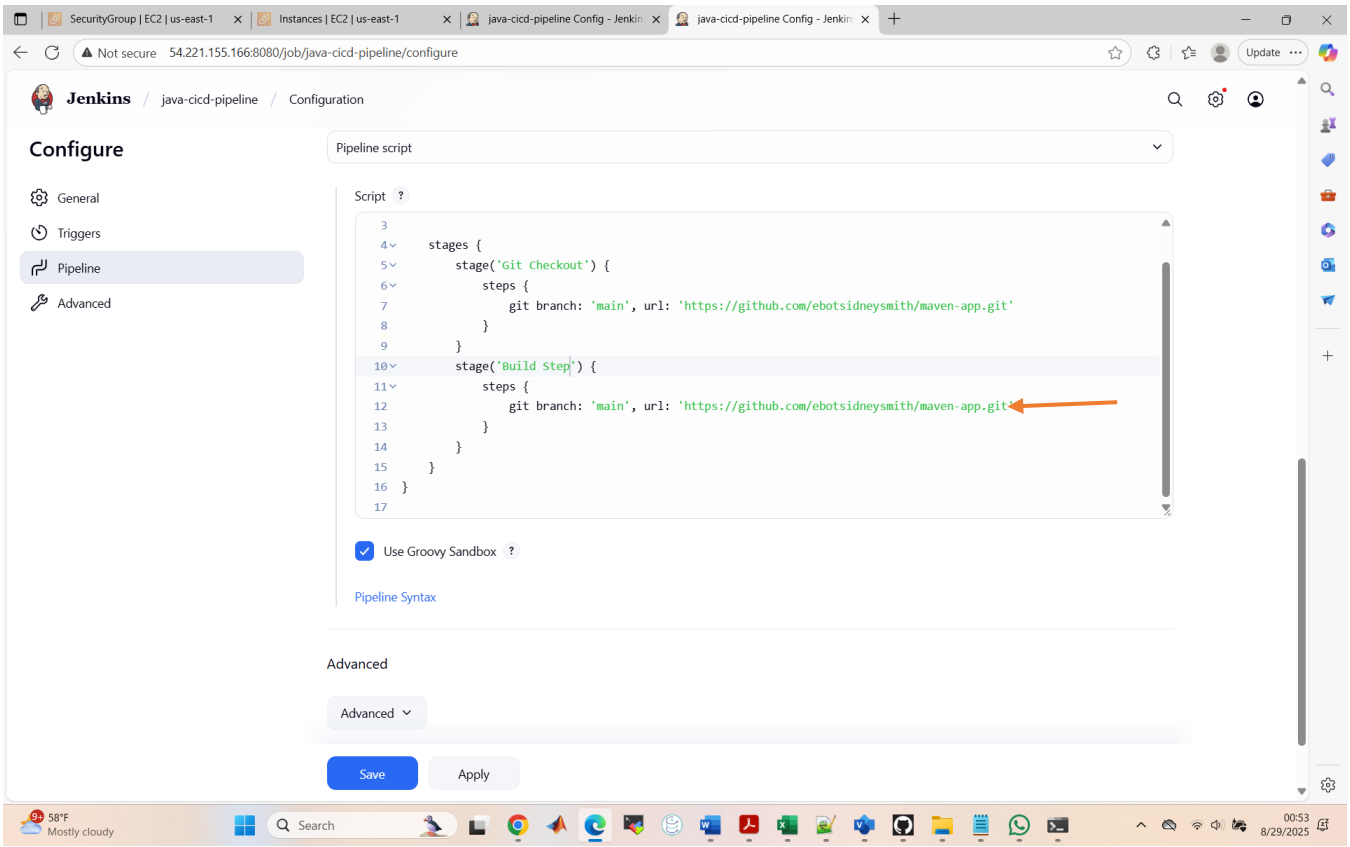
Go back to our pipeline code



Copy this stage and paste for the second stage

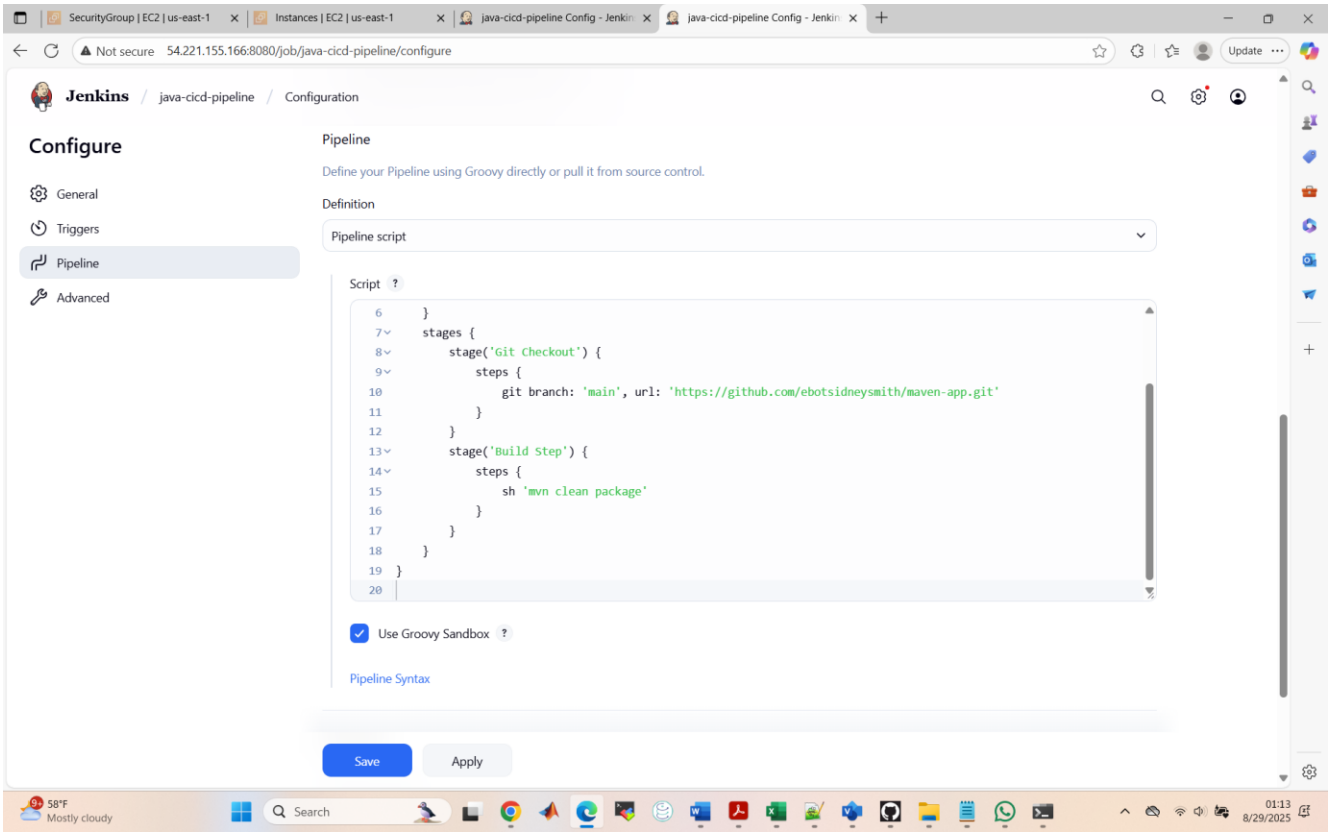


Now, let us modify the second stage. We will replace "**Git Checkout**" with "**Build Step**"

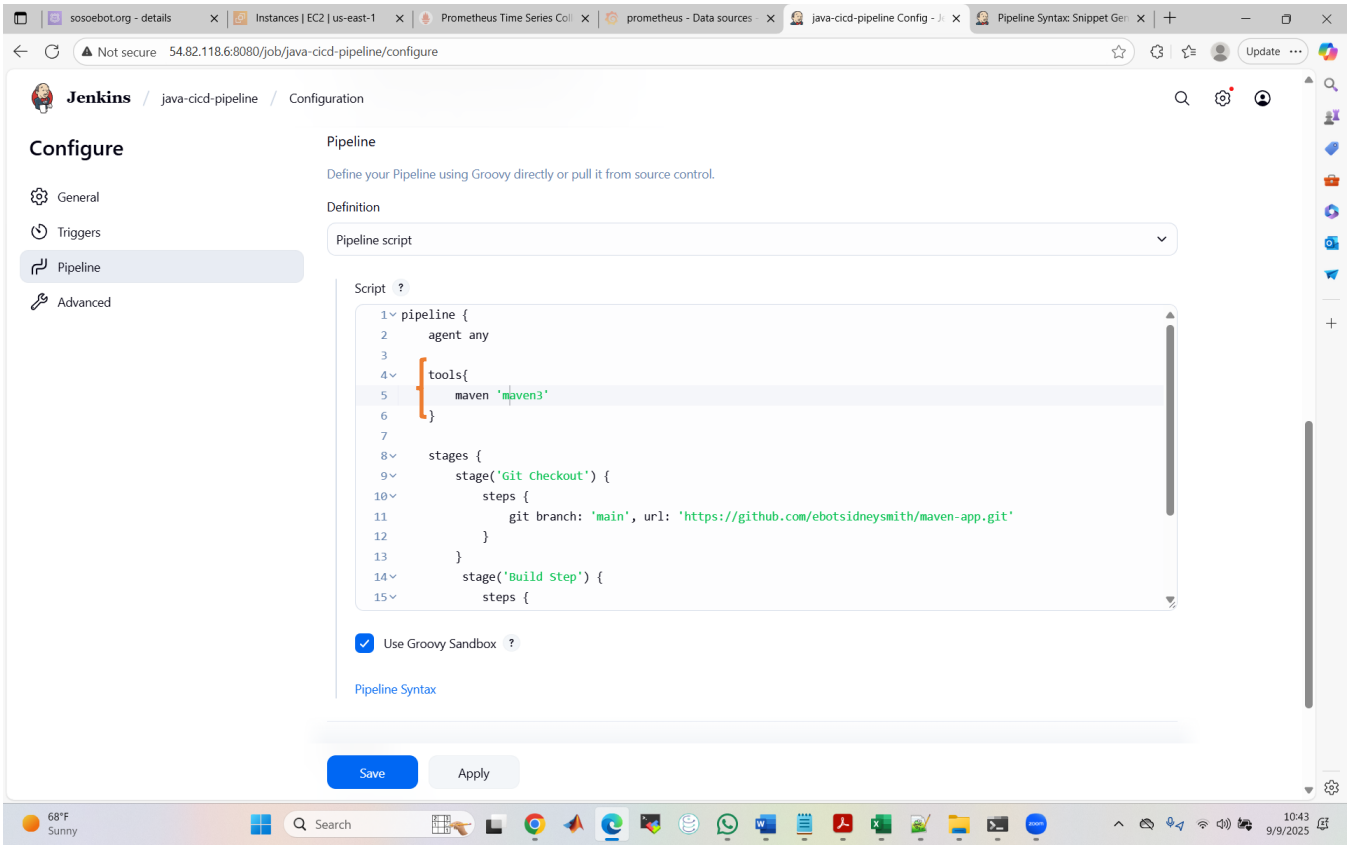


Then we modify the second line as follows:

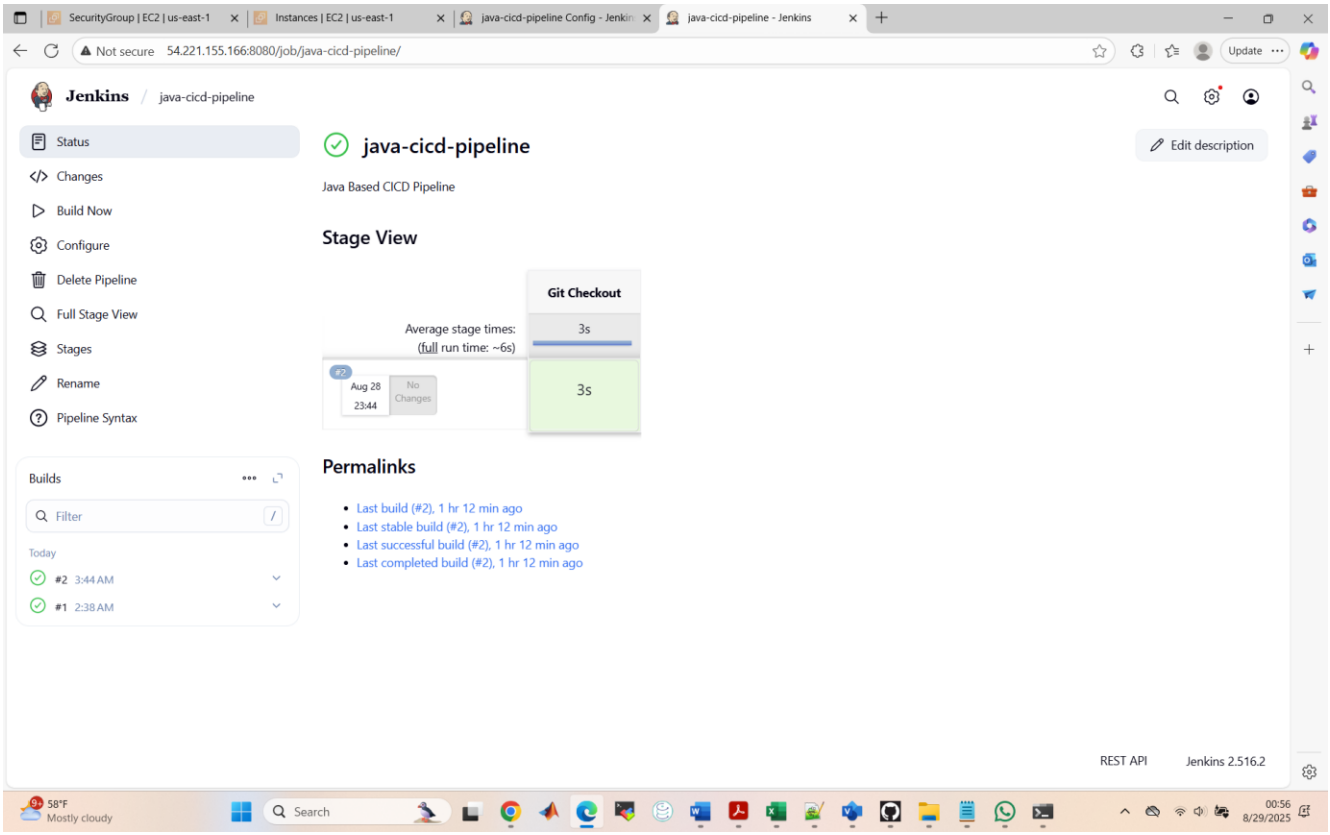
```
sh 'mvn clean package'
```



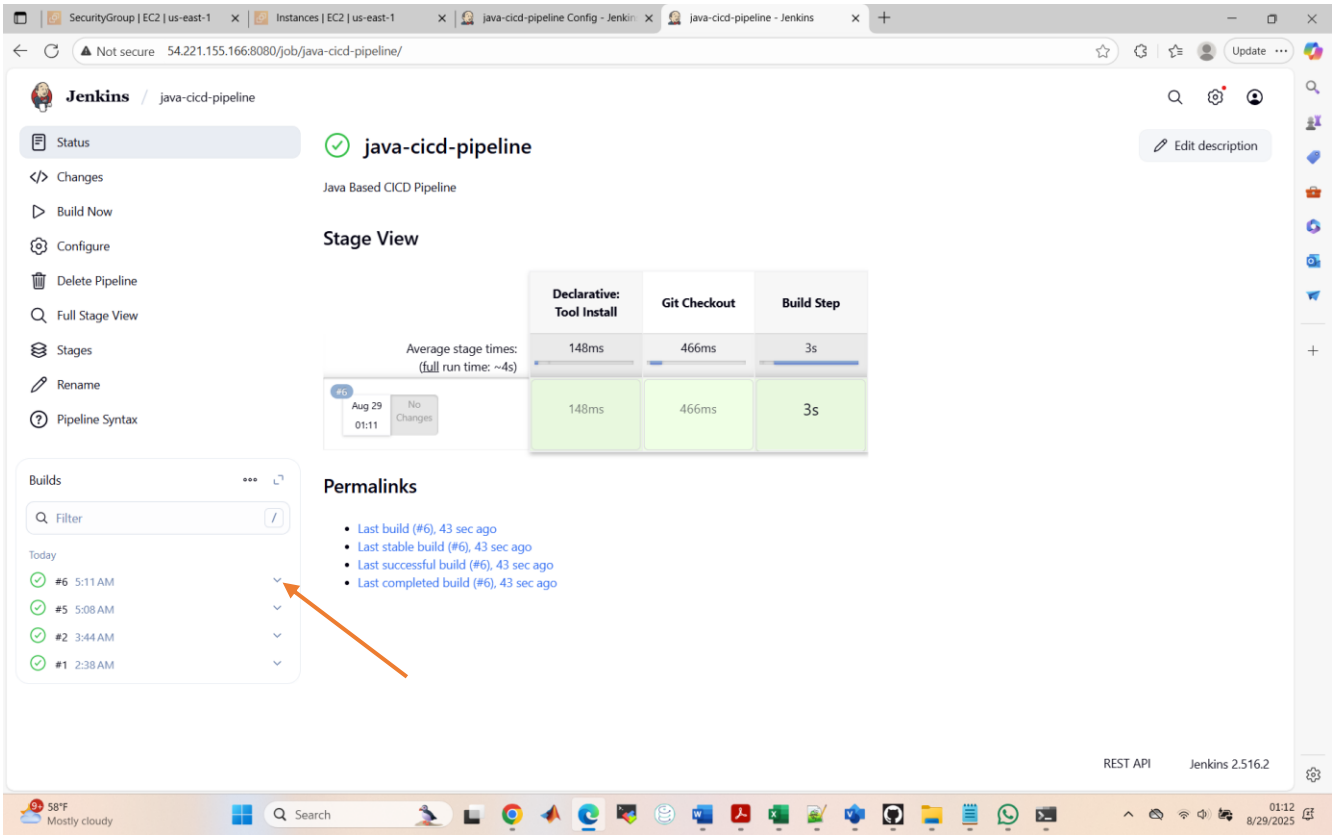
Then we have to add the **tool** Maven at the top of the code with name "**maven3"**



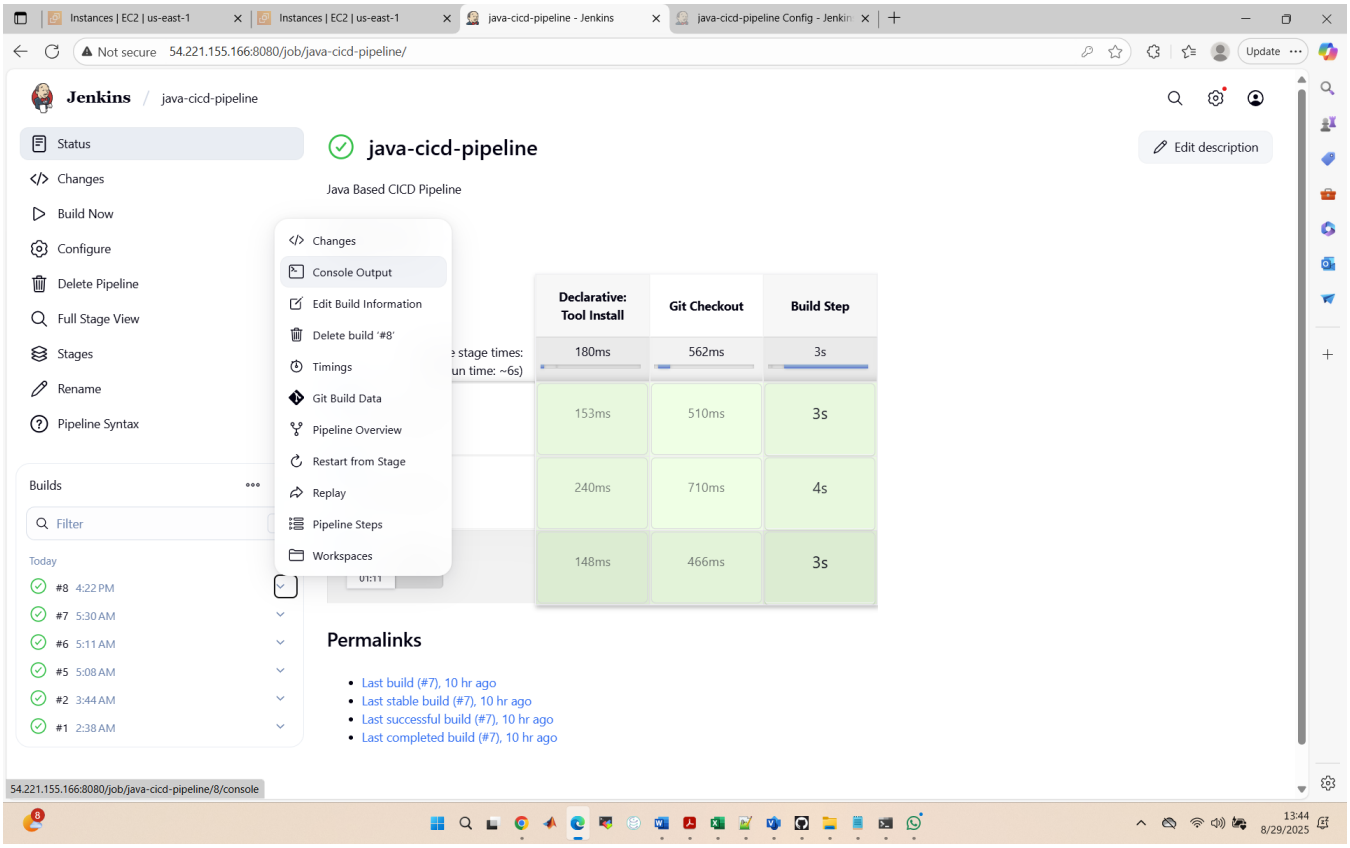
Click on "**Apply**" followed by "**Save**"



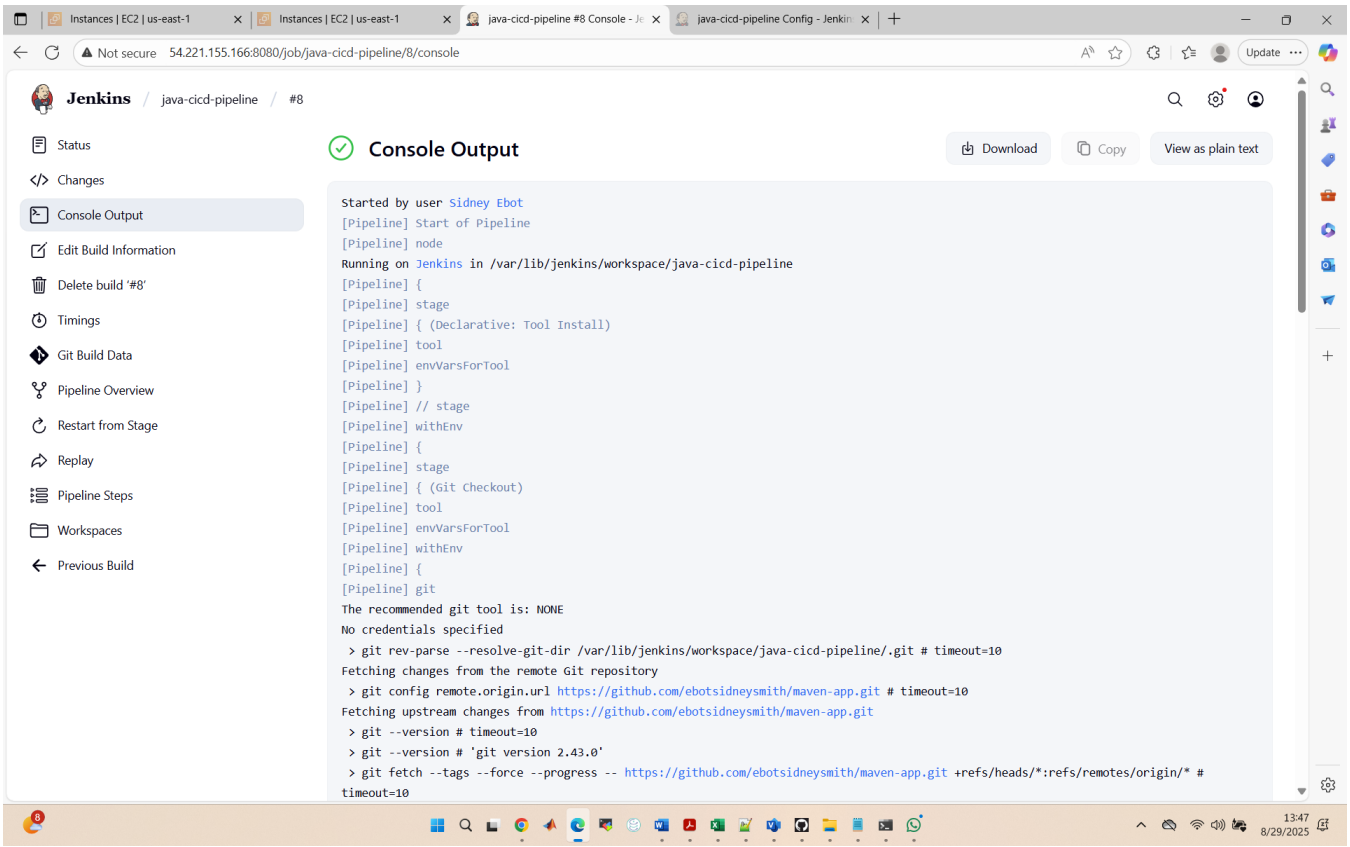
Then click on "**Build Now**"



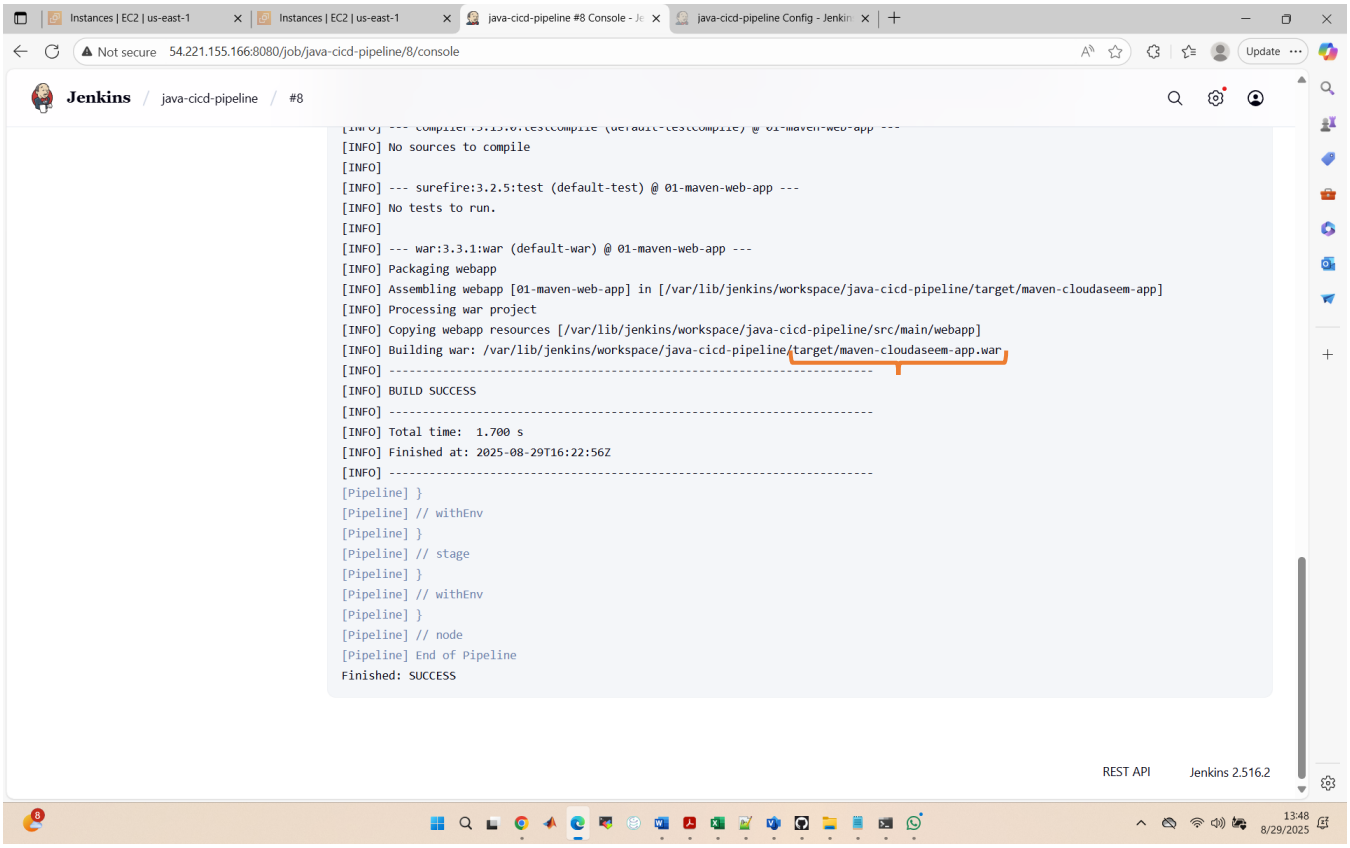
Click on the drop down on "**#6**"



Select “**Console Output**”



Scroll down to the end

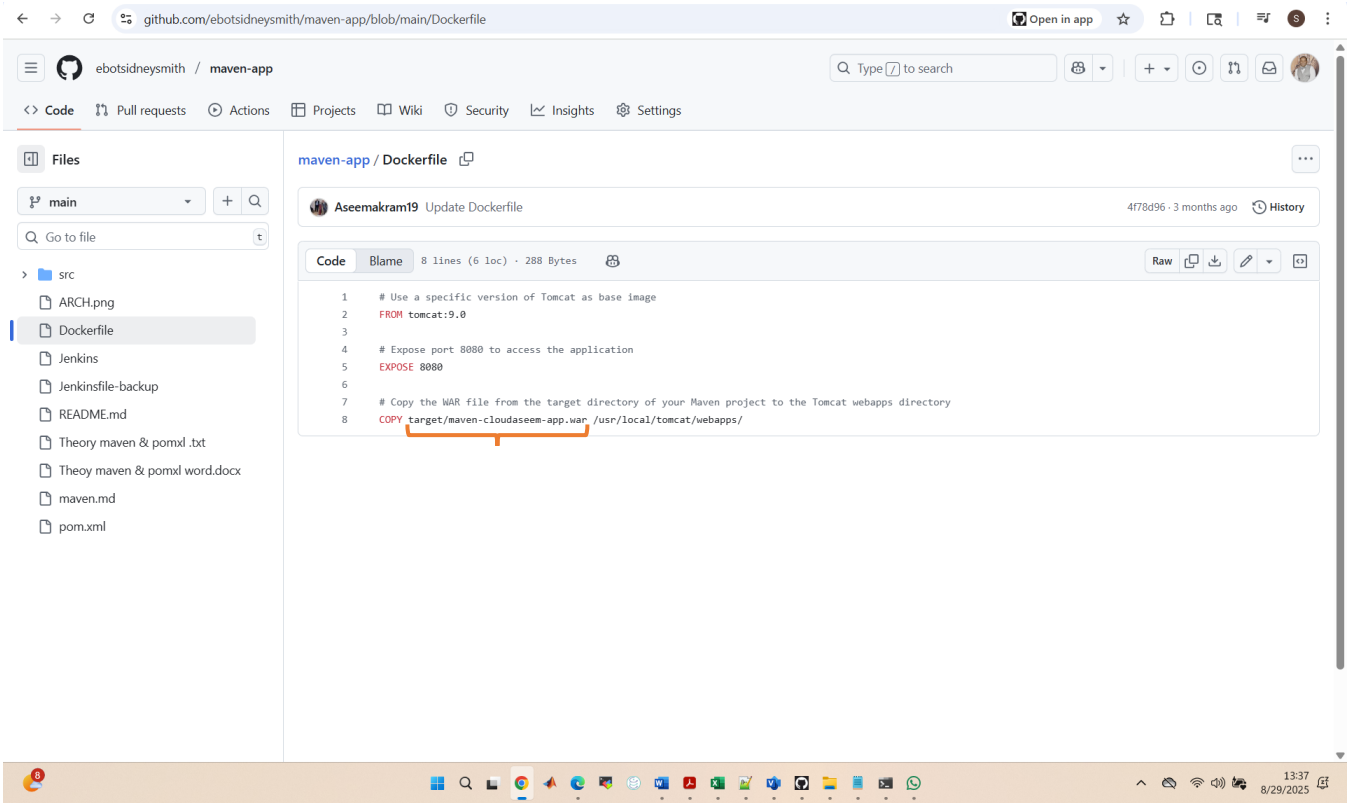


Copy this part of the code, we will use it later
`target/maven-cloudaseem-app.war`

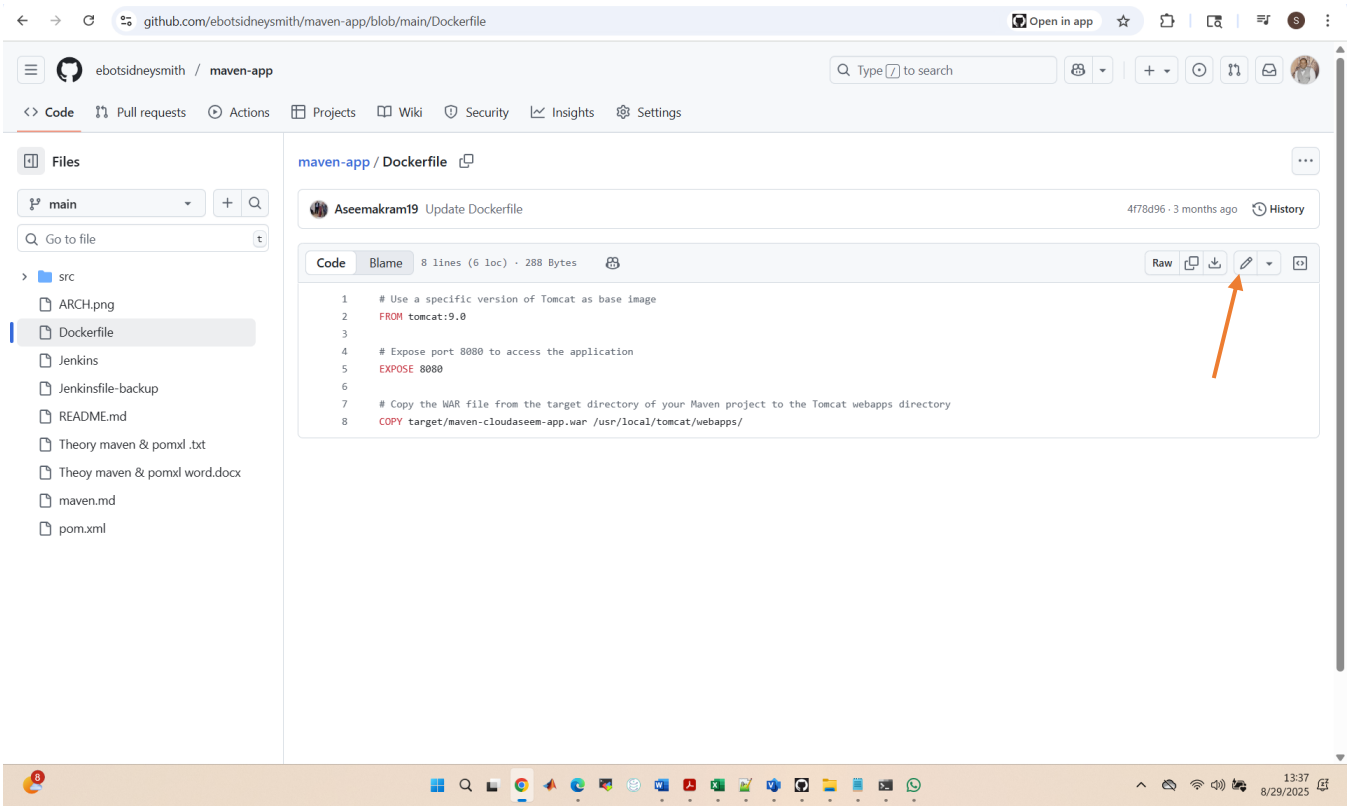
So, we have completed this step. We now go to the next step

Phase 4: Create Docker Image

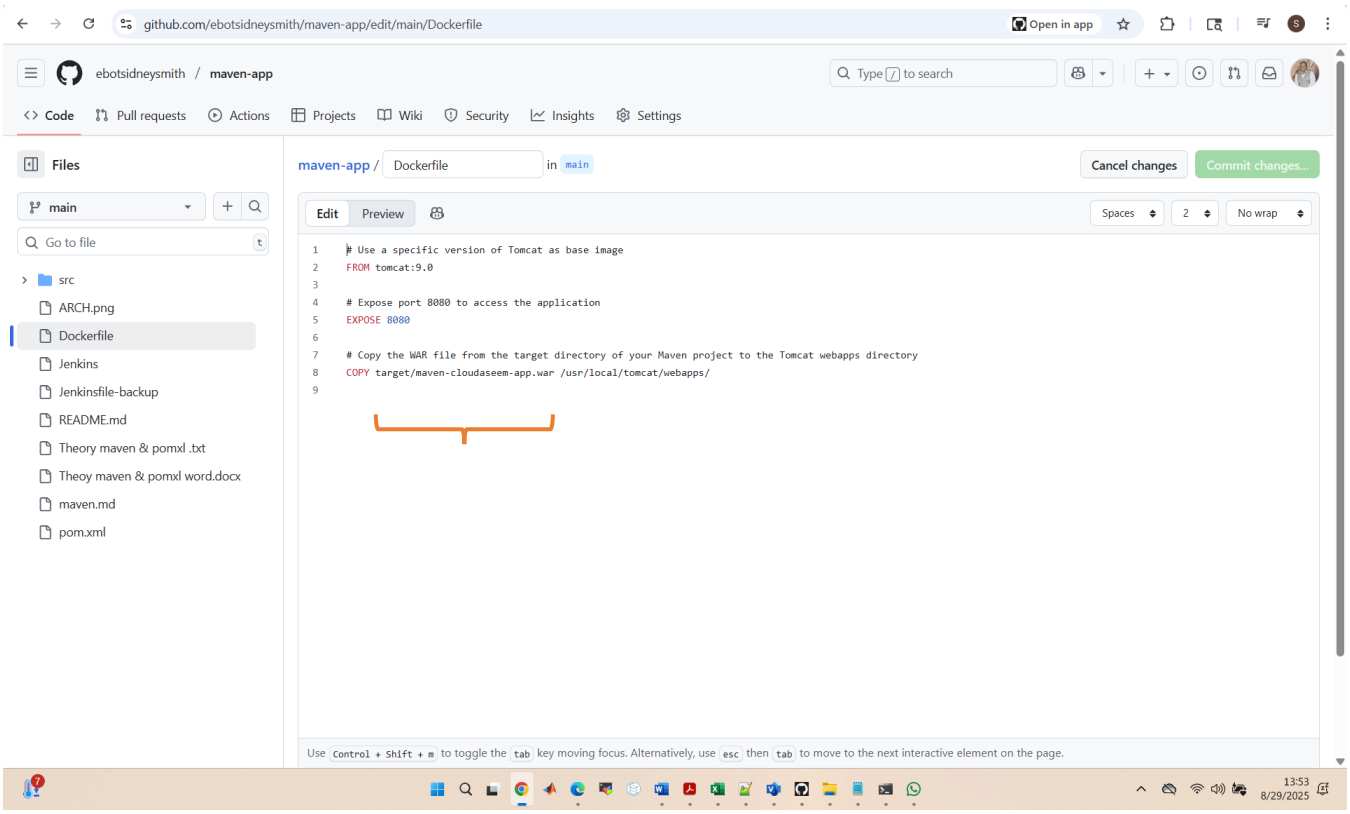
In this step, we have to build the docker image. We will start by opening our Dockerfile and modify it.



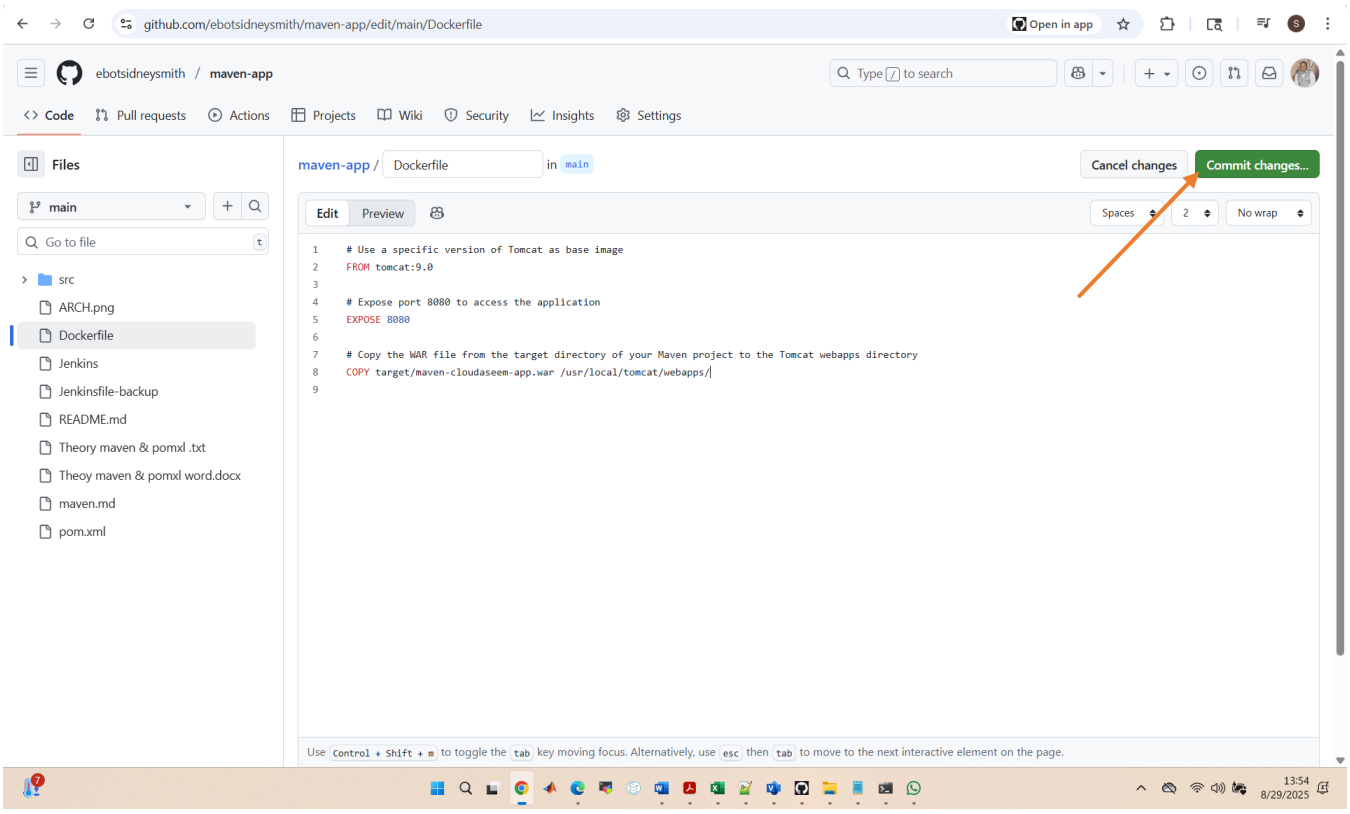
We have to modify **line 8** of our Dockerfile code. We will replace the part of the code show above with the part of the code we copied previously.



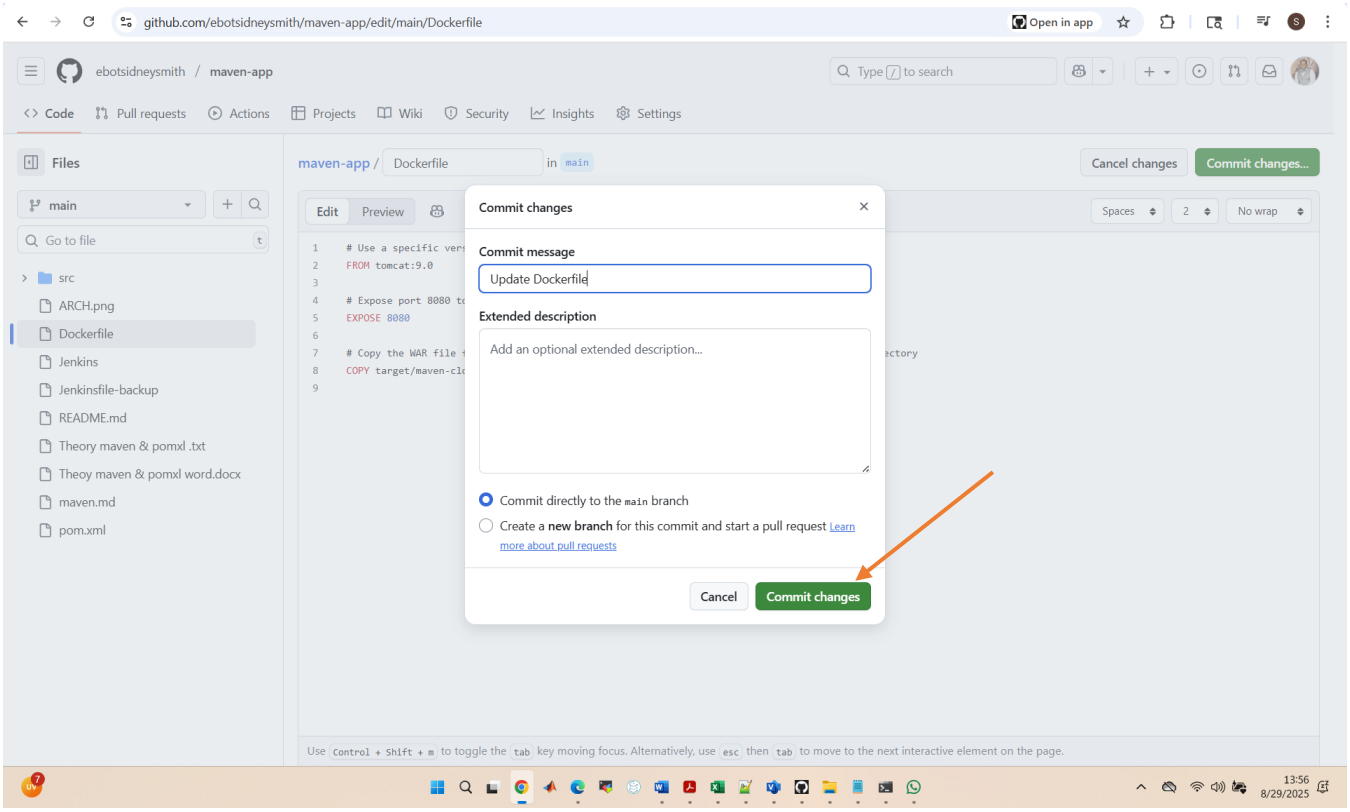
To do this click on "**Edit**"



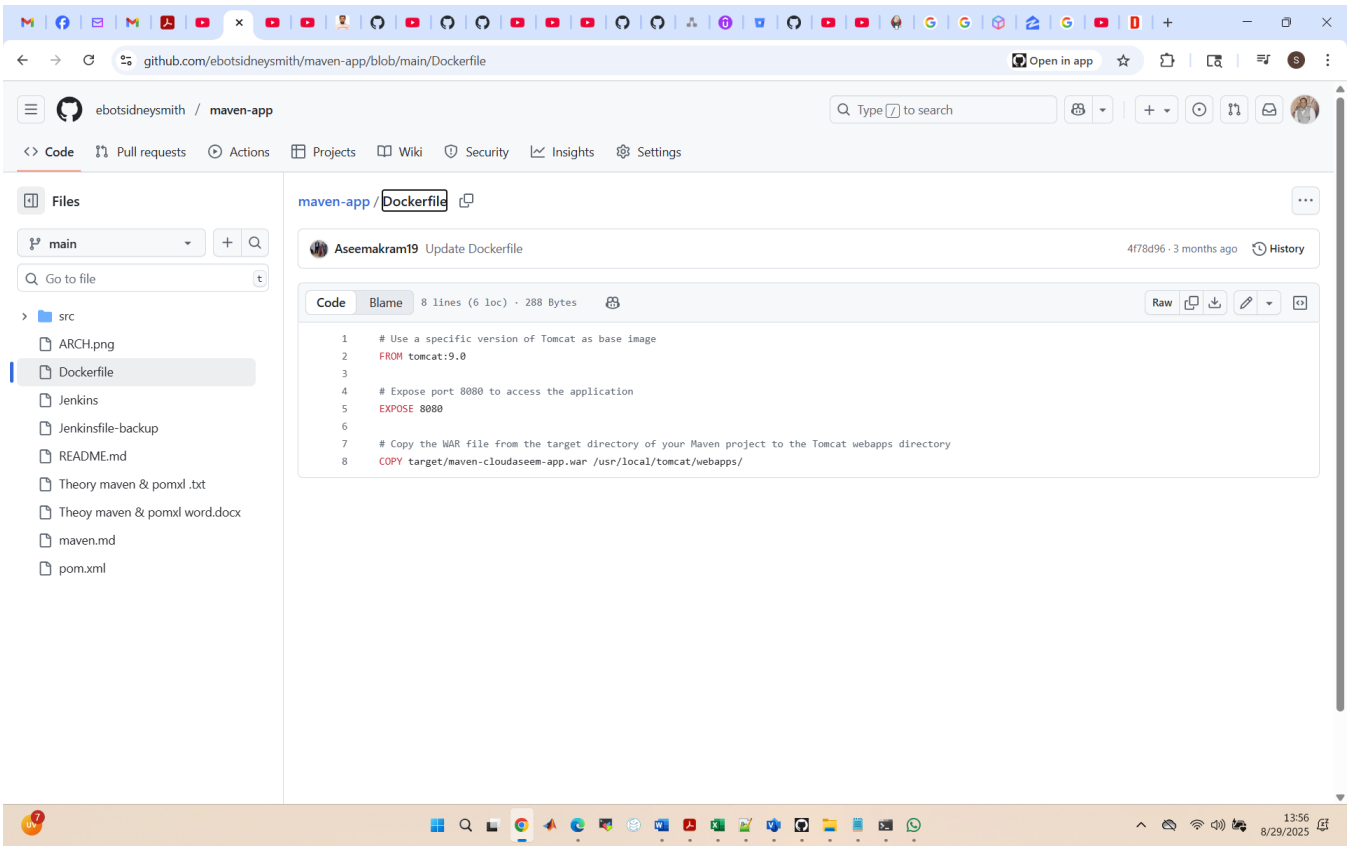
Paste the code here



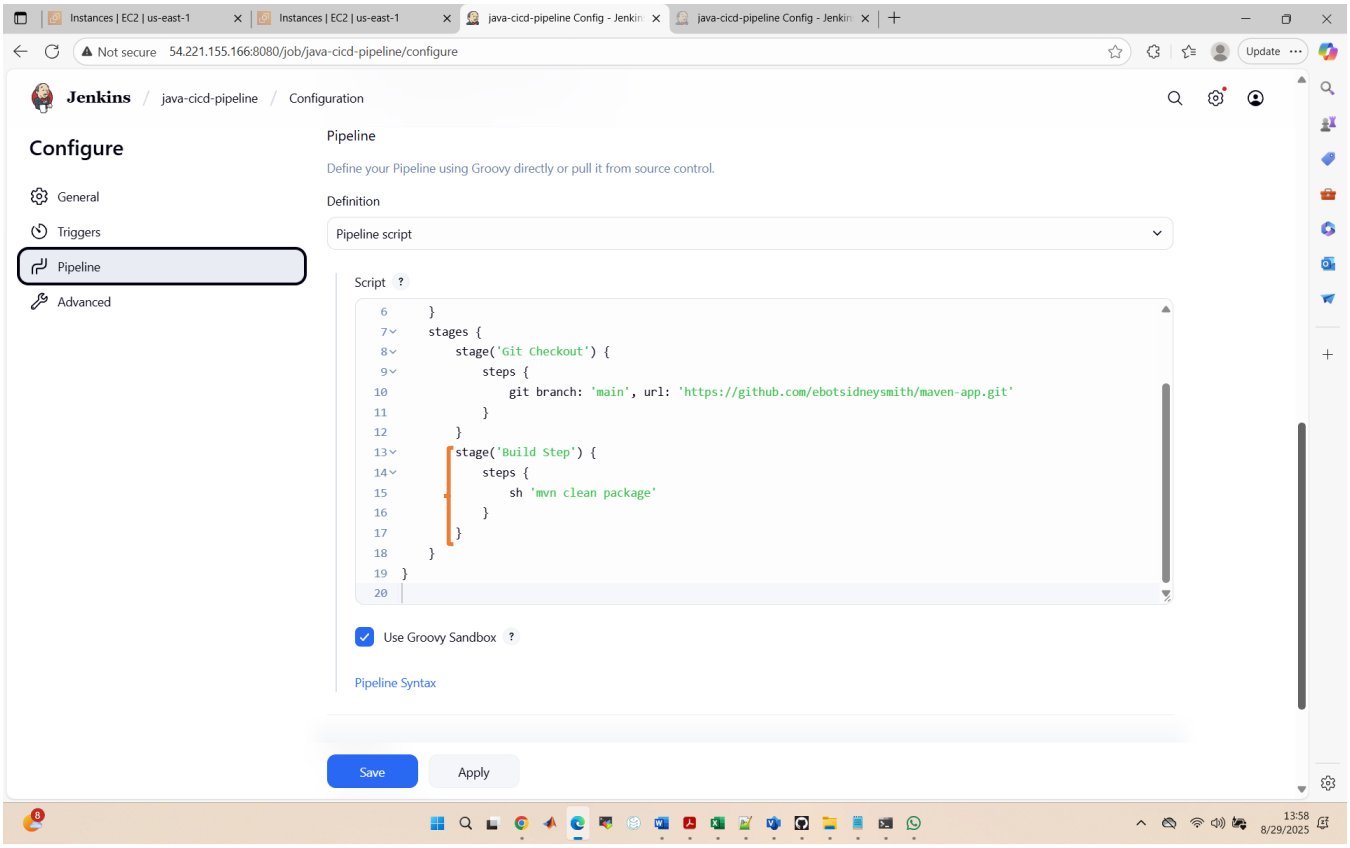
Then click on "**Commit Changes**"



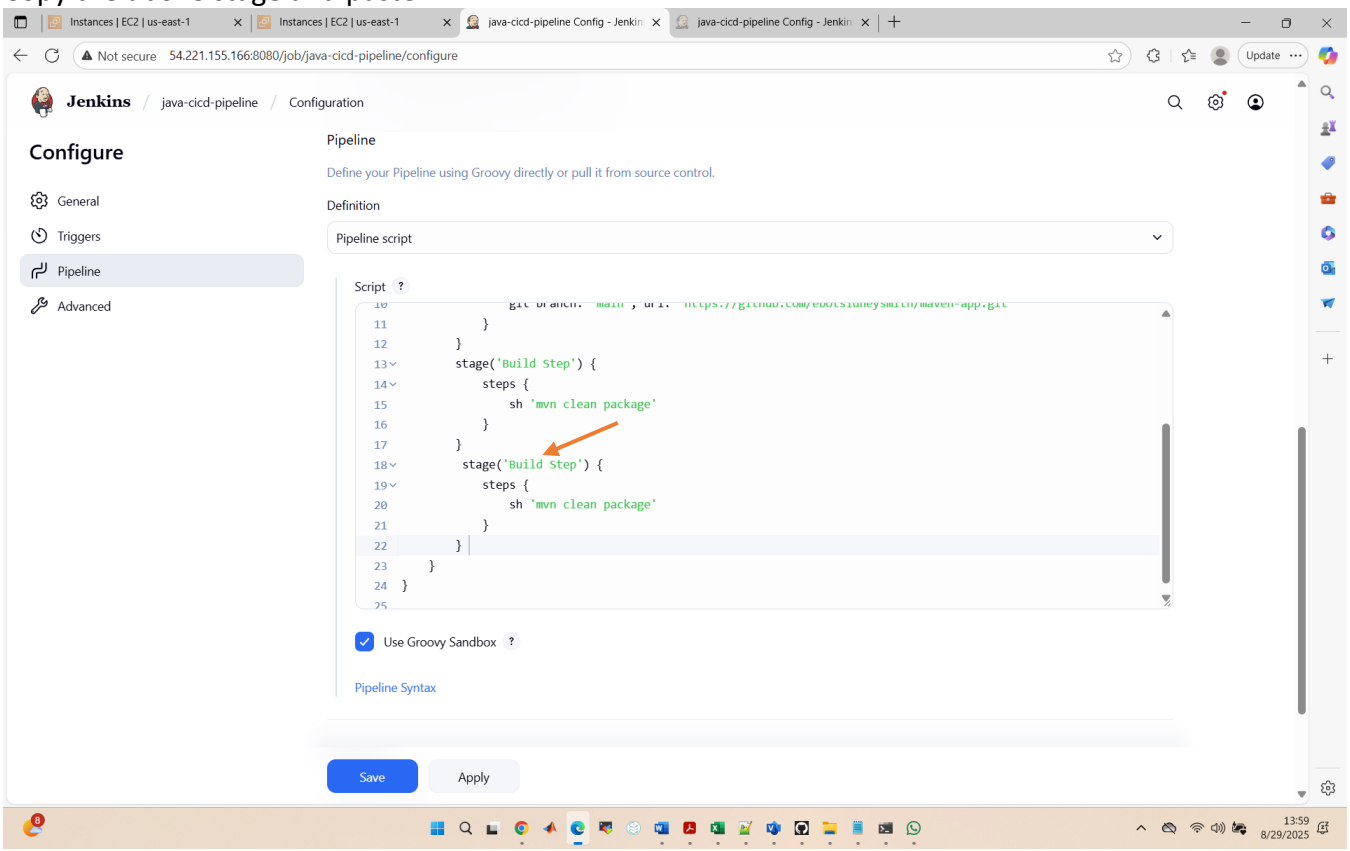
Confirm by clicking on "**Commit Changes**" again



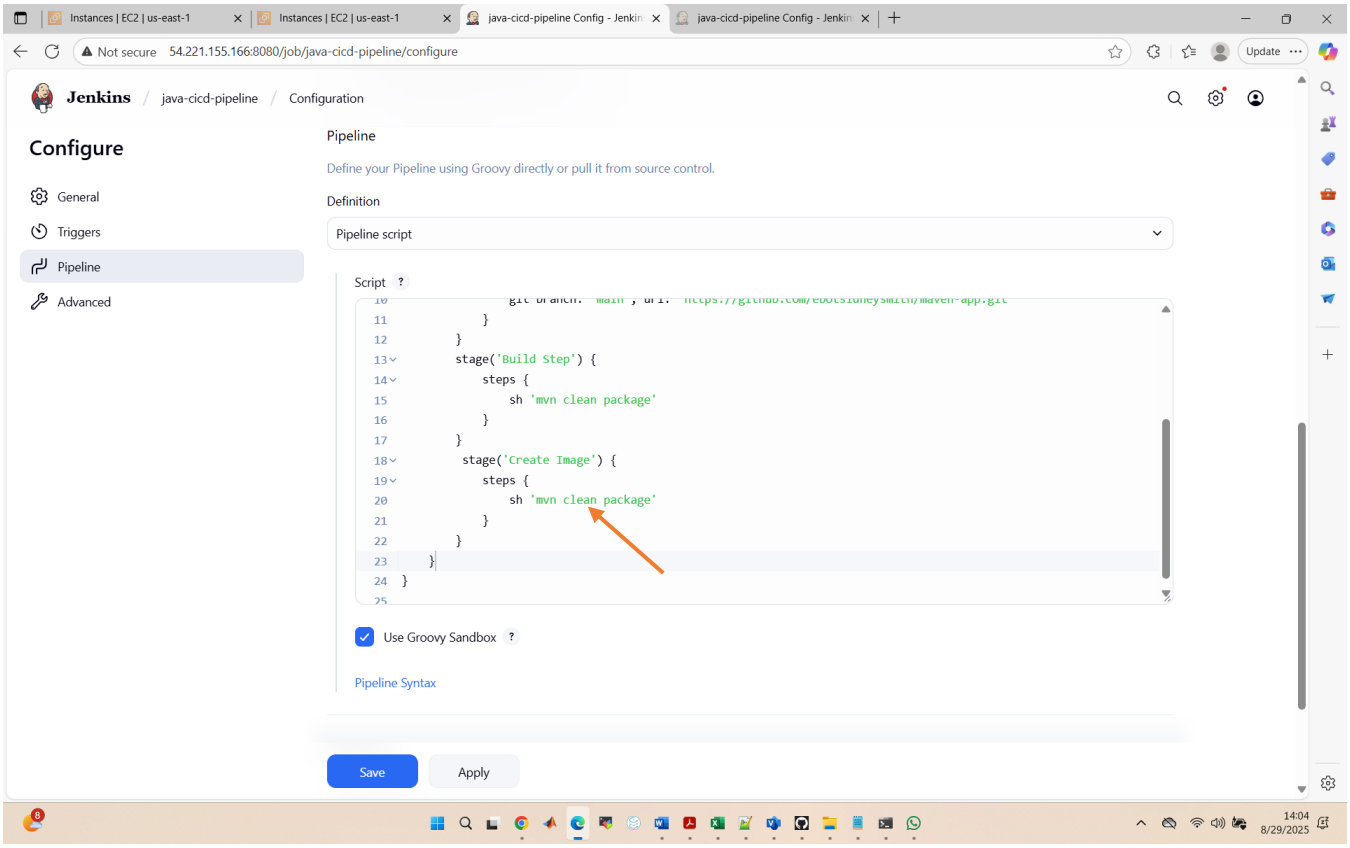
Now, let us go back to our pipeline and add one more stage



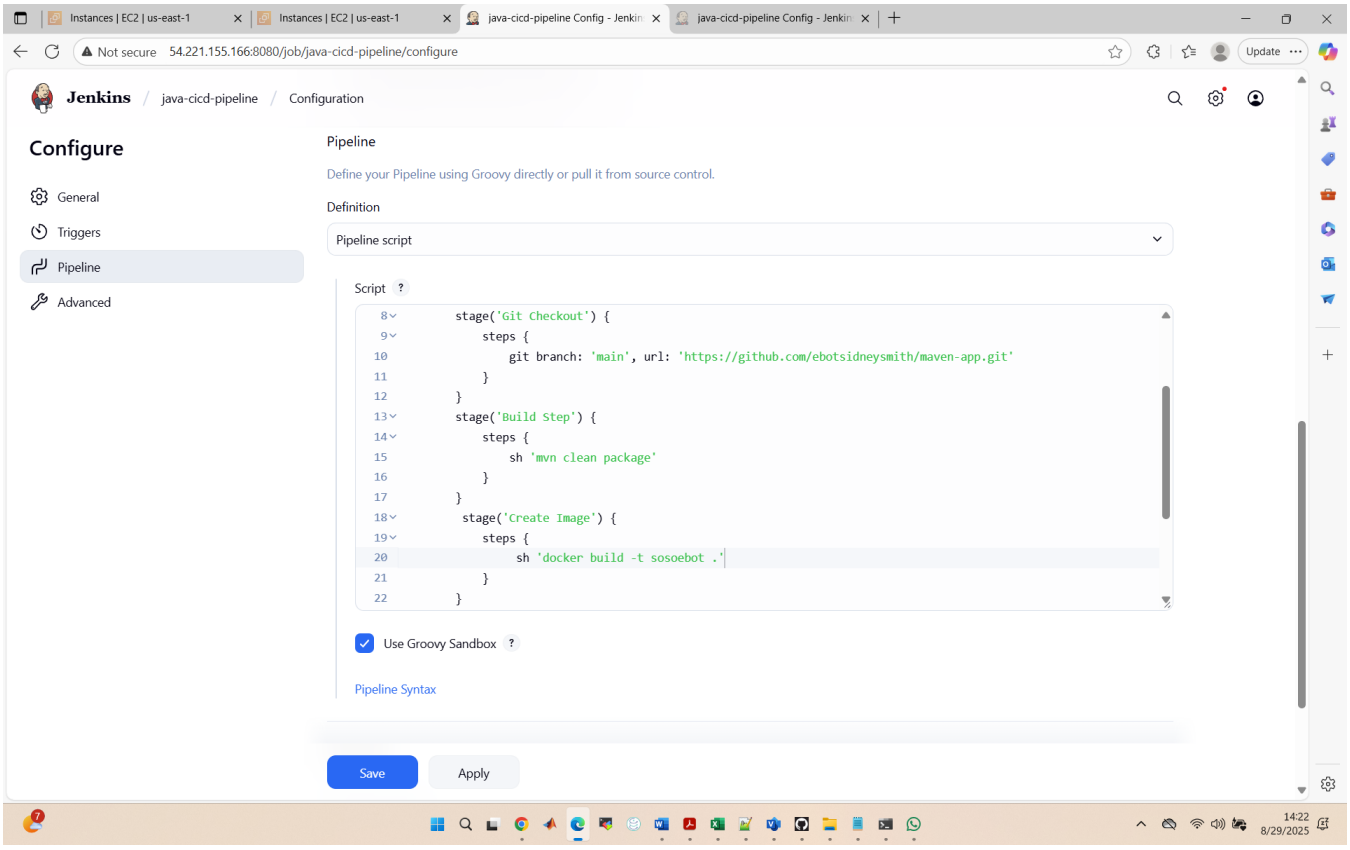
Copy the above stage and paste



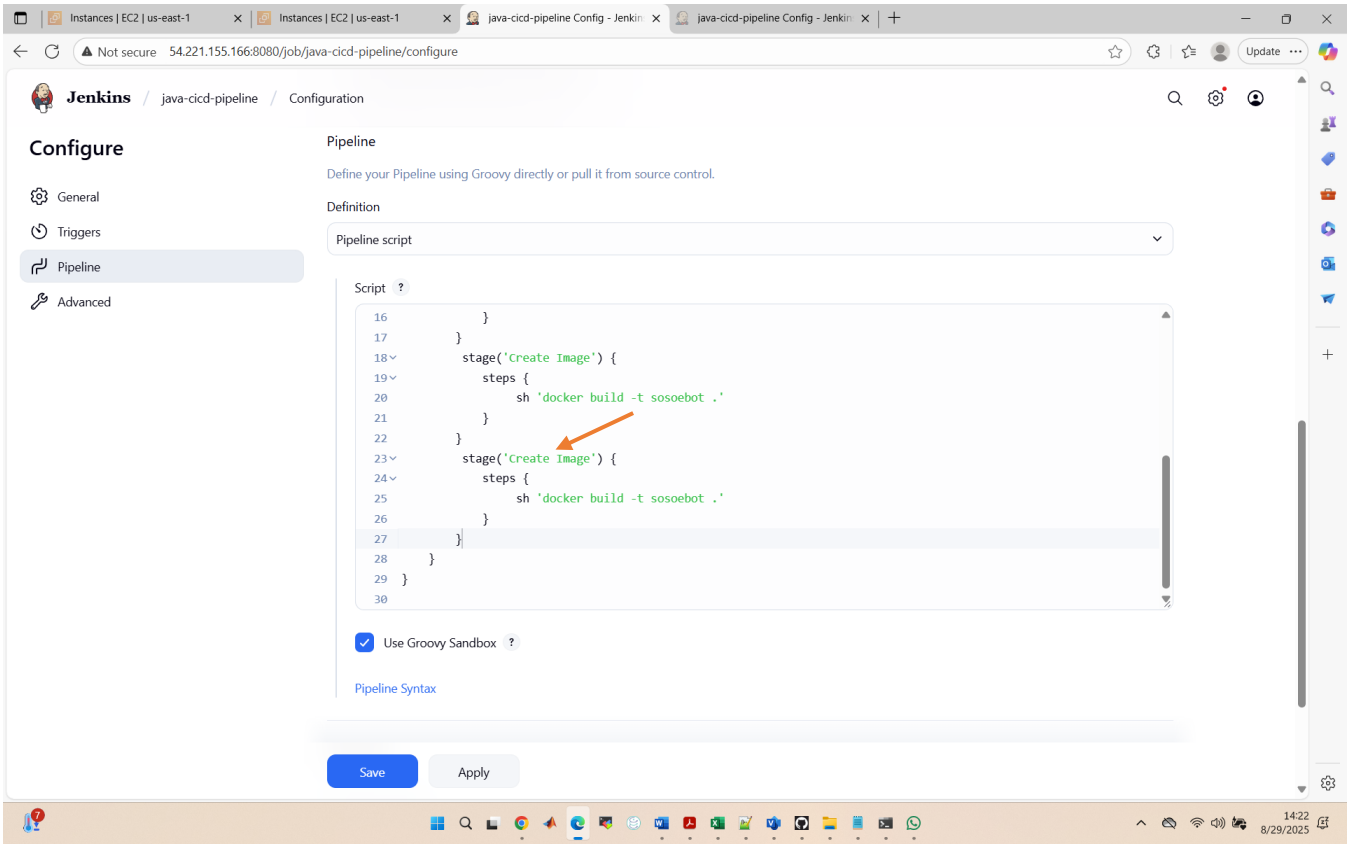
Let us now modify the added stage. We will change "**Build Step**" on this added stage with "**Create Image**". Take note that you can decide to use any name for the stage.



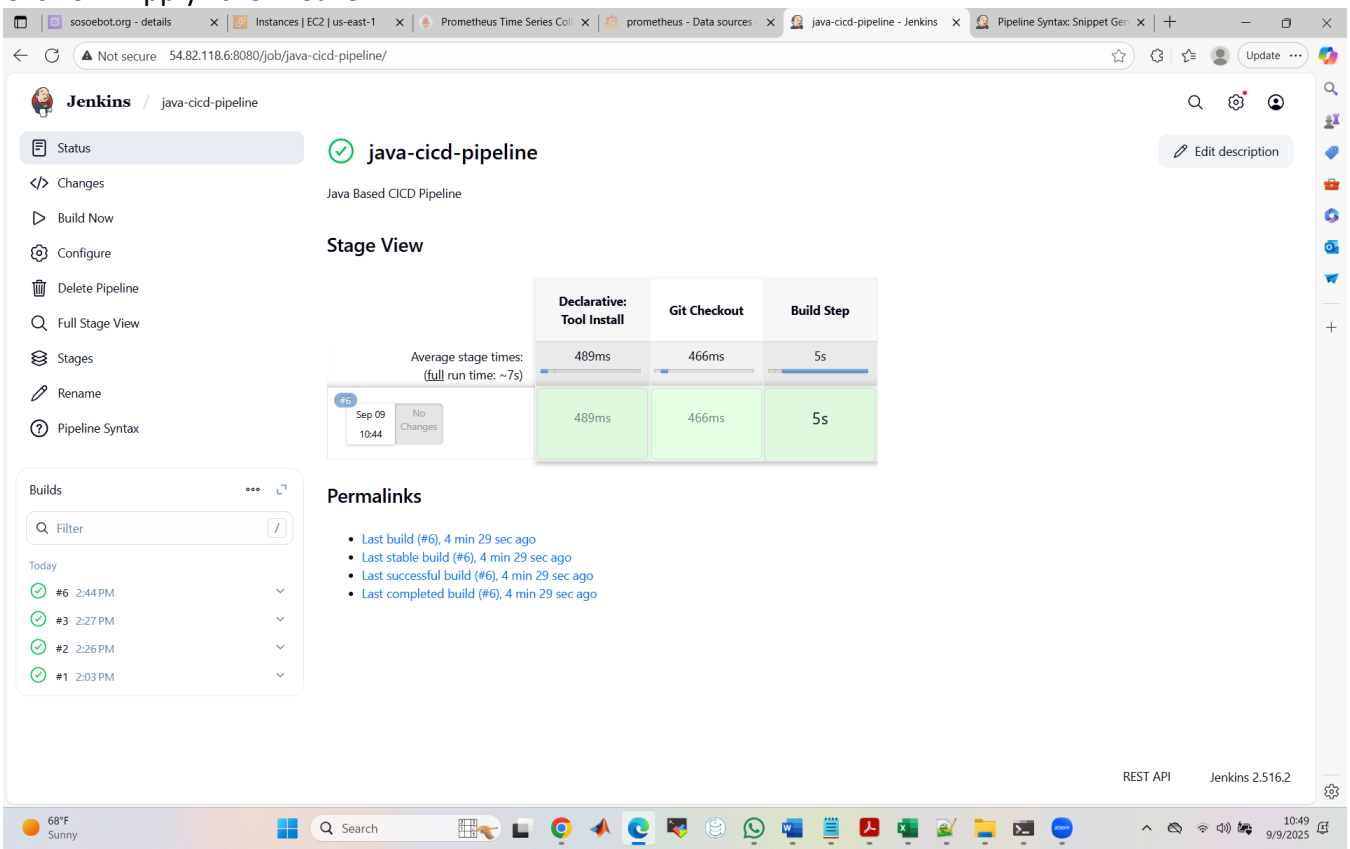
Then, let us modify the step. We will replace "**sh 'mvn clean package'**" with "**sh 'docker build -t sosoebot .'**". Where "**sosoebot**" is our image name.



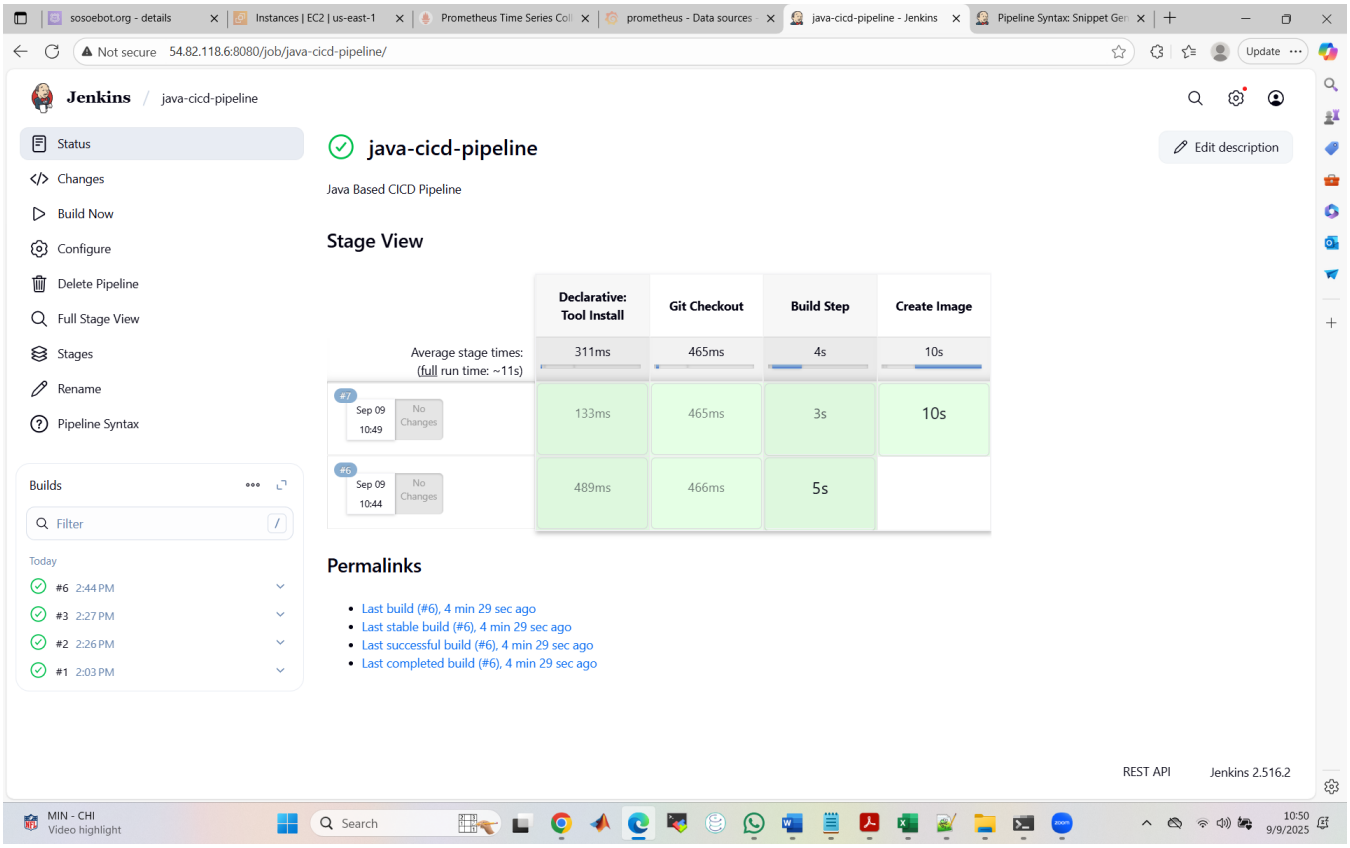
Now, let us add another stage in our pipeline code to containerize our application



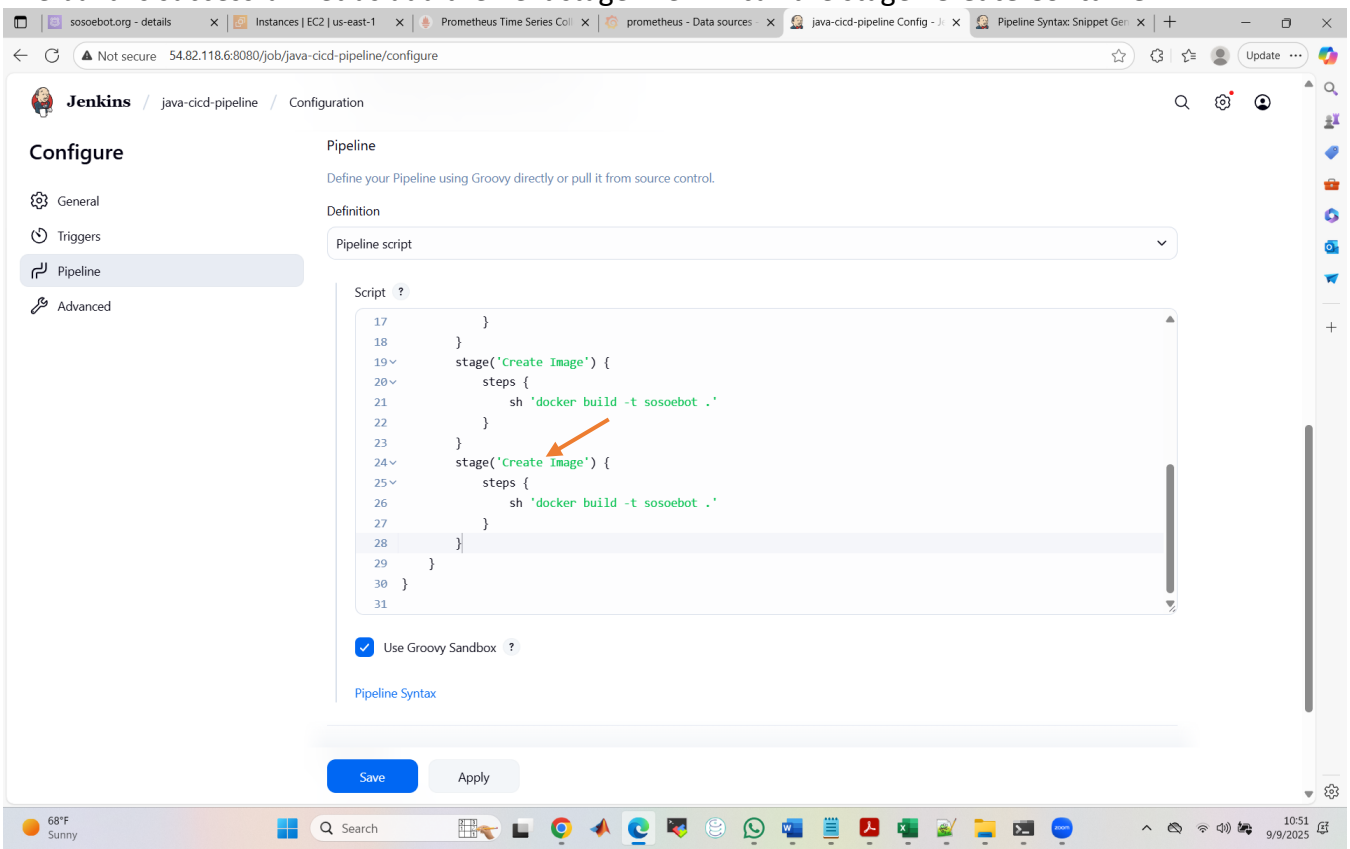
Click on “Apply” then “Save”



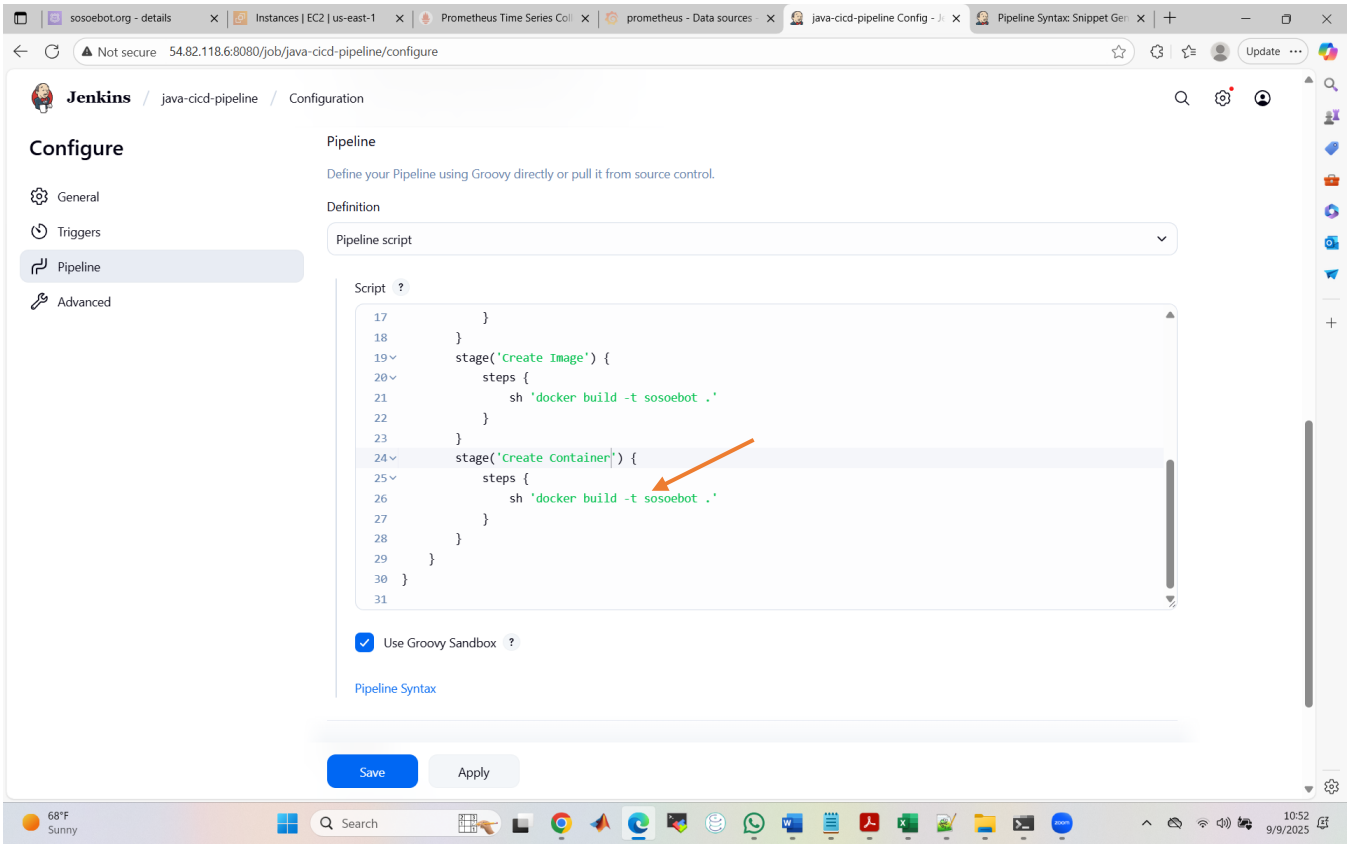
Click on “Build Now”



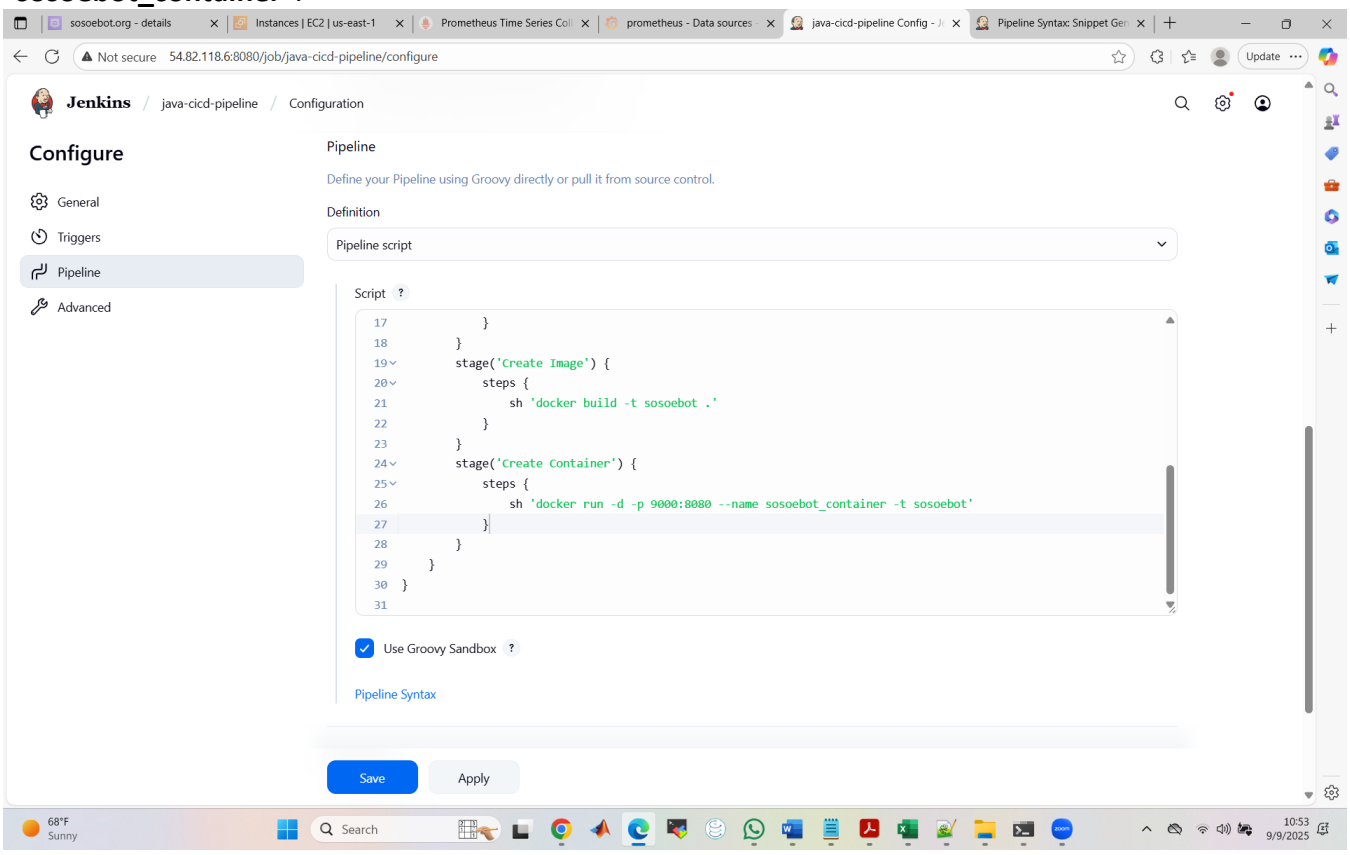
The build is successful. Let us add the next stage We will call the stage "**Create Container**".



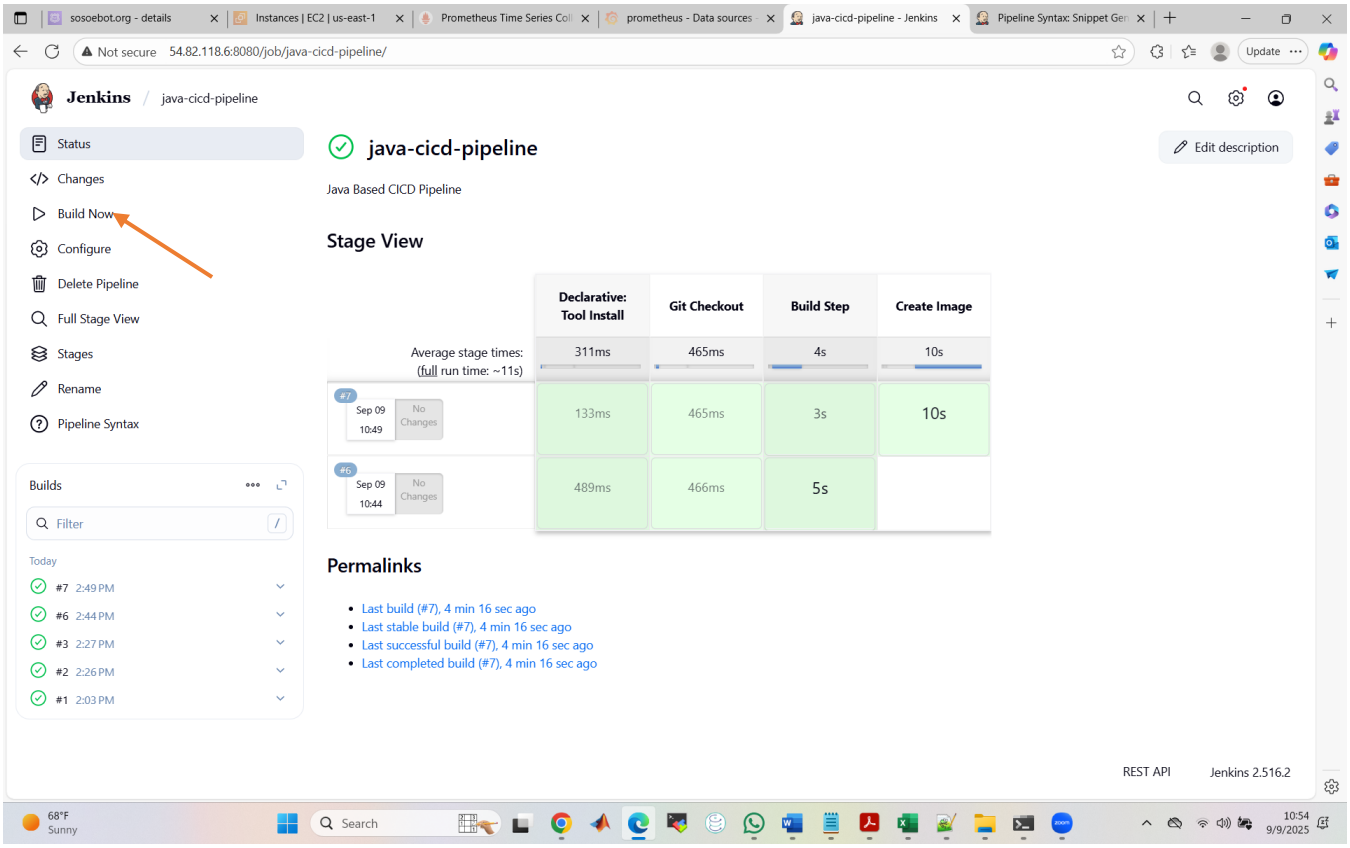
So, we will change "**Create Image**" to "**Create Container**"



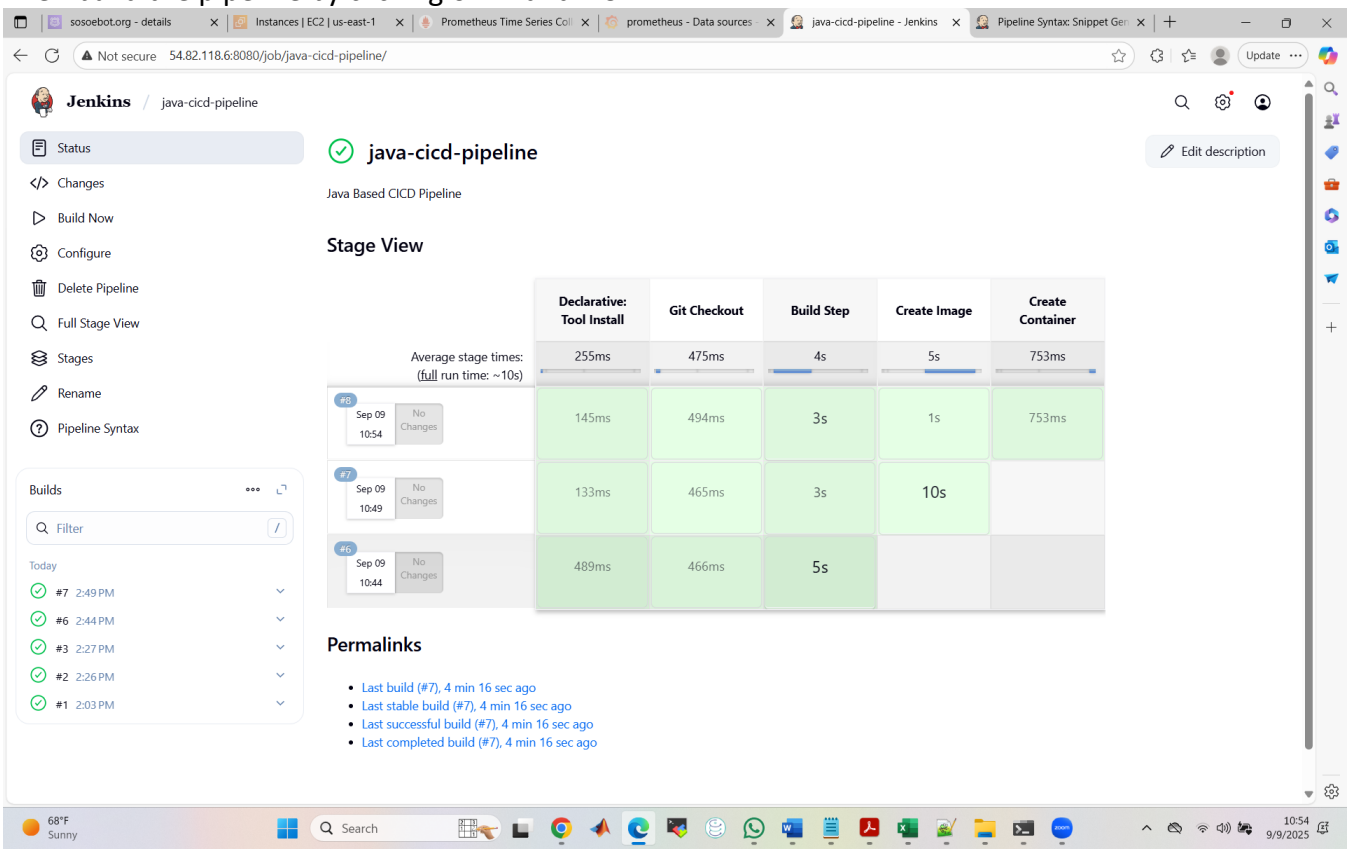
Then, we will modify "**sh 'docker build -t sosoebot'**" to "**sh 'docker run -d -p 9000:8080 --name sosoebot_container -t sosoebot'**". Where the image name is "**sosoebot**" and the container name is "**sosoebot_container**".



Click on "**Apply**" followed by "**Save**"



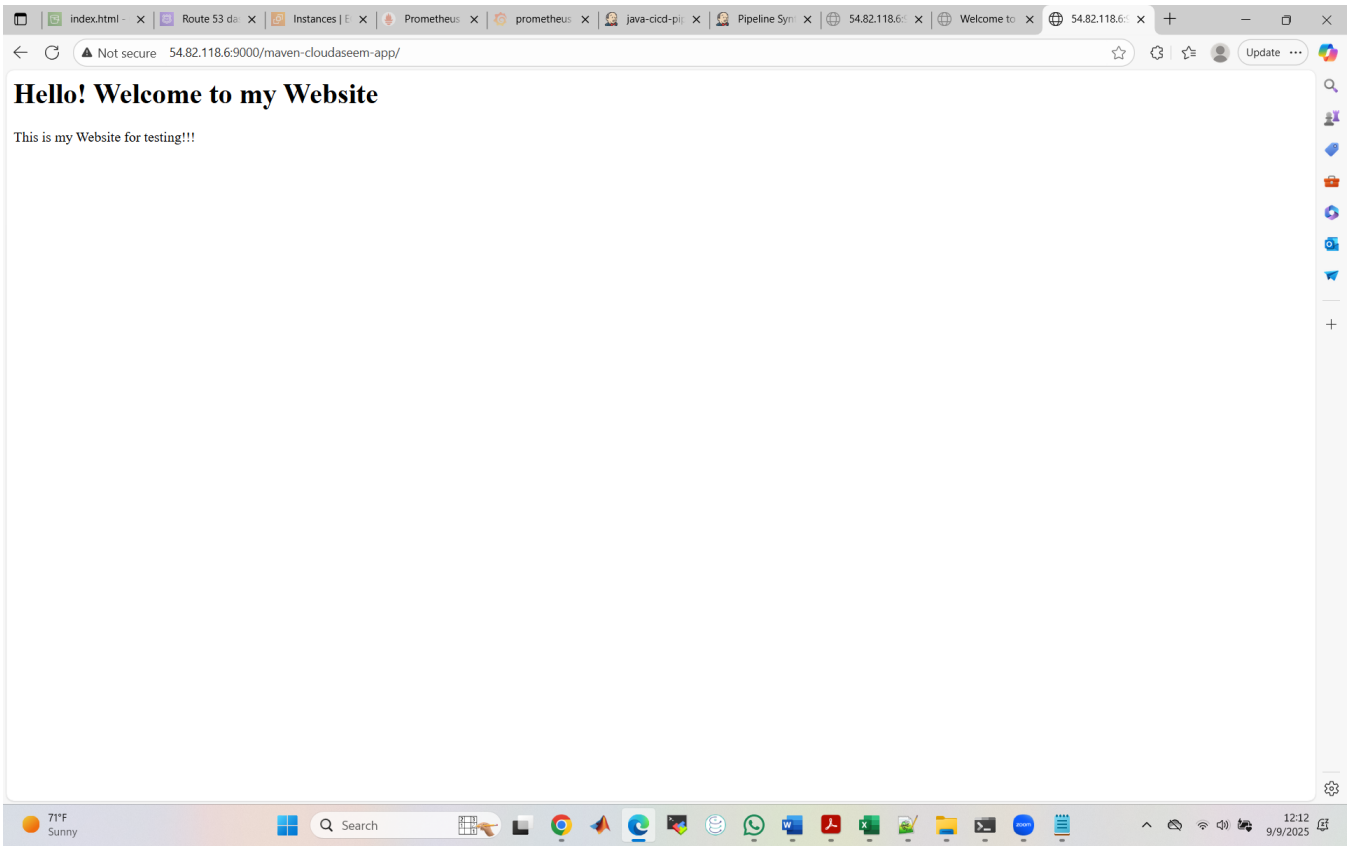
Then build the pipeline by clicking on "**Build Now**"



You can see that the build is successful.

Now, let us check if we can access our docker container by typing this one our browser

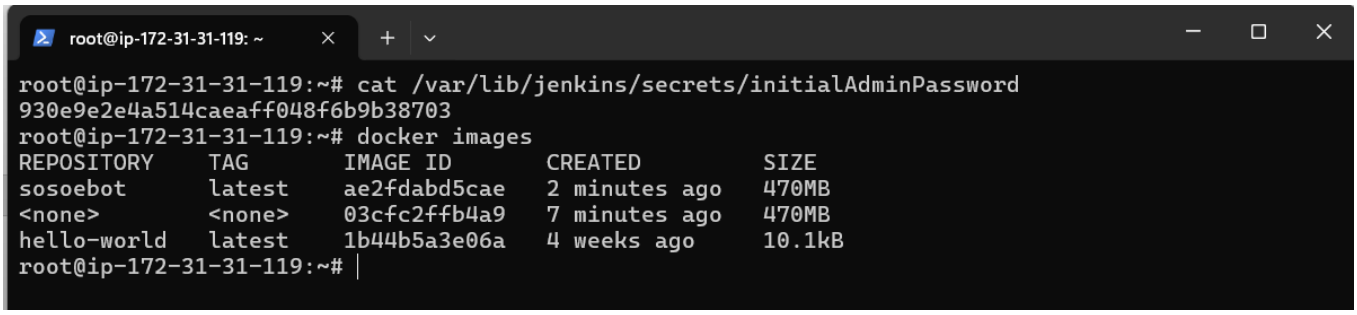
`http://Public IPv4 Address:9000/maven-cloudaseem-app`
That is `http://54.82.118.6:9000/maven-cloudaseem-app`



The application is working. So, we are able to access our Java application.

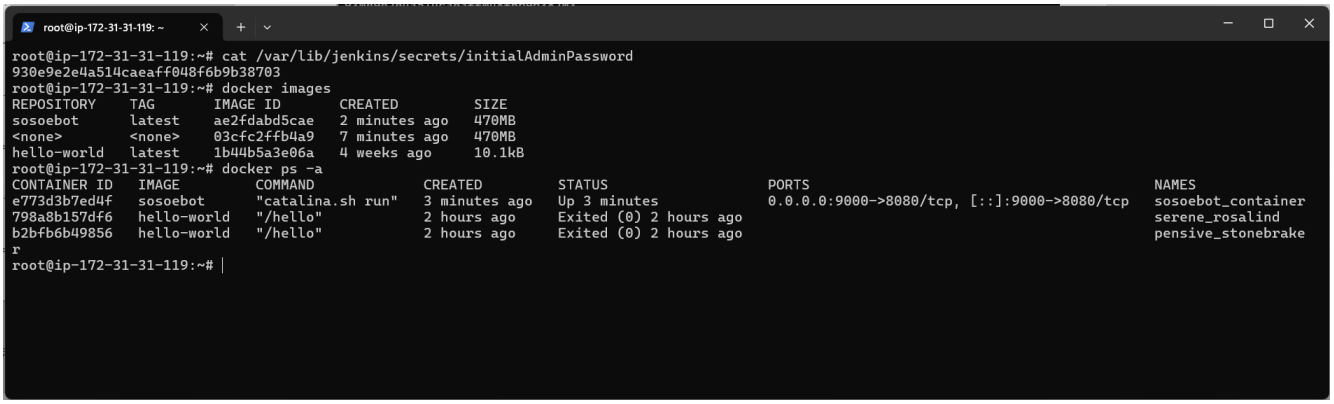
We can now verify if we have created an image using the command on the terminal:

`docker images`



You can see that we have created an image as expected. Let us also check if we have created a container using the command:

`docker ps -a`



You can see our container.

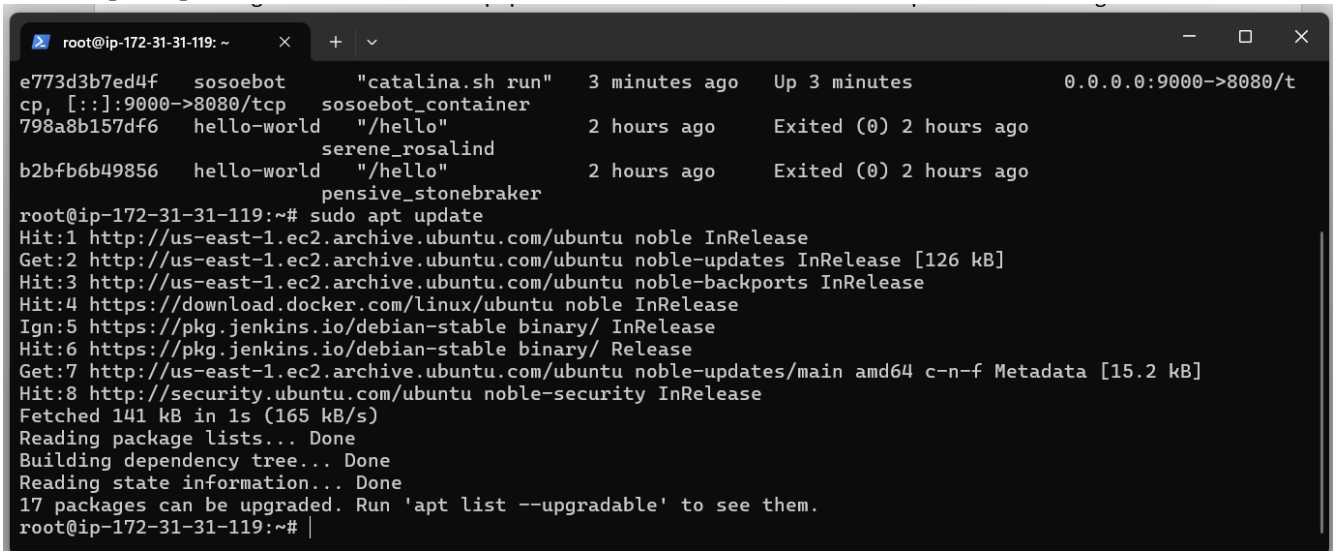
STEP 12: Install and configure Nginx and Certbot on EC2 server

Nginx is one of the most popular web servers in the world and is responsible for hosting some of the largest and highest-traffic sites on the internet. It is a lightweight choice that can be used as either a web server or reverse proxy.

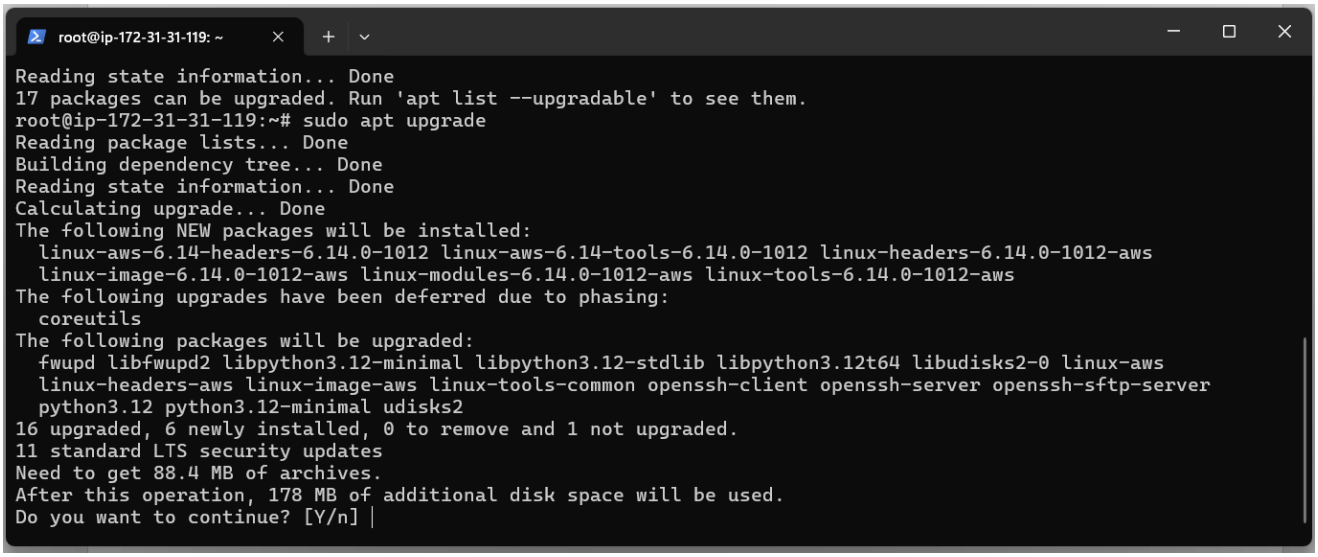
Part 1: Update and upgrade package

In this step, we have to install Nginx. To do that we first Update Package Repository and Upgrade Packages using these commands:

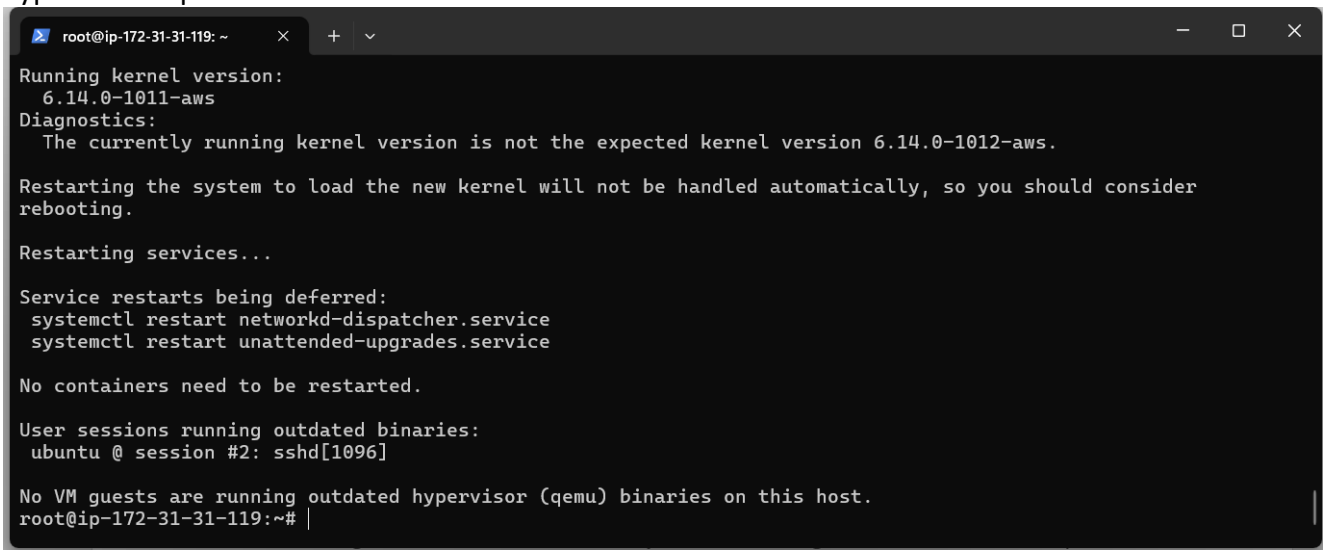
```
sudo apt update
```




```
sudo apt upgrade
```



Type "**Y**" and press ENTER

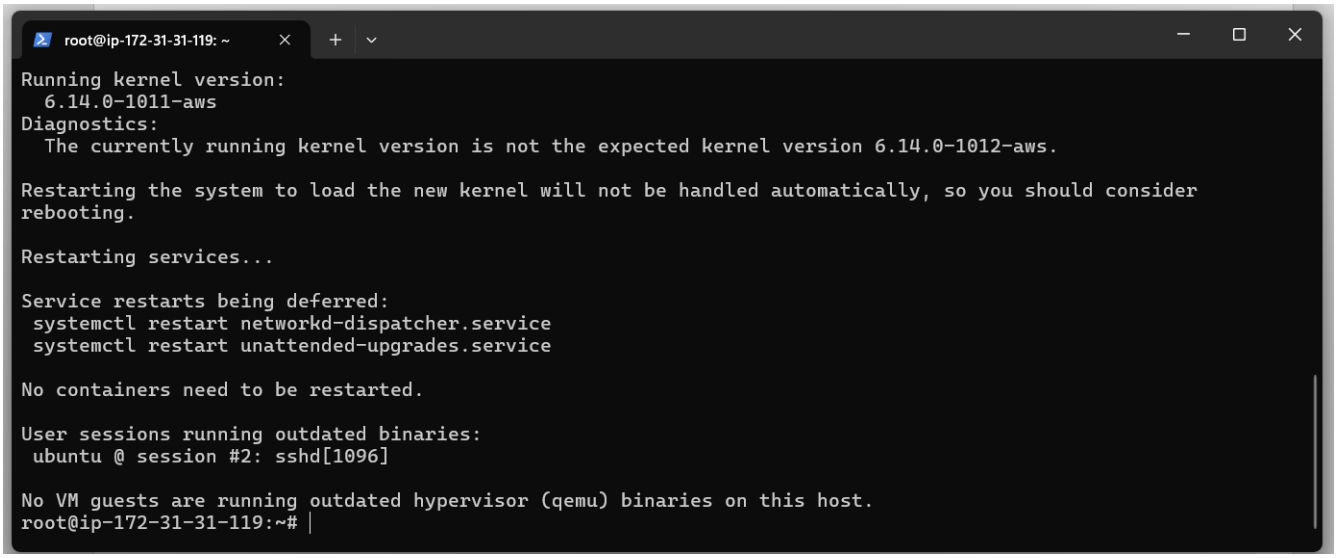


Now we want a domain name to take over the IP of the server so that instead of hitting the IP we can use the domain name that is TLS enabled. Let's see how to do so.

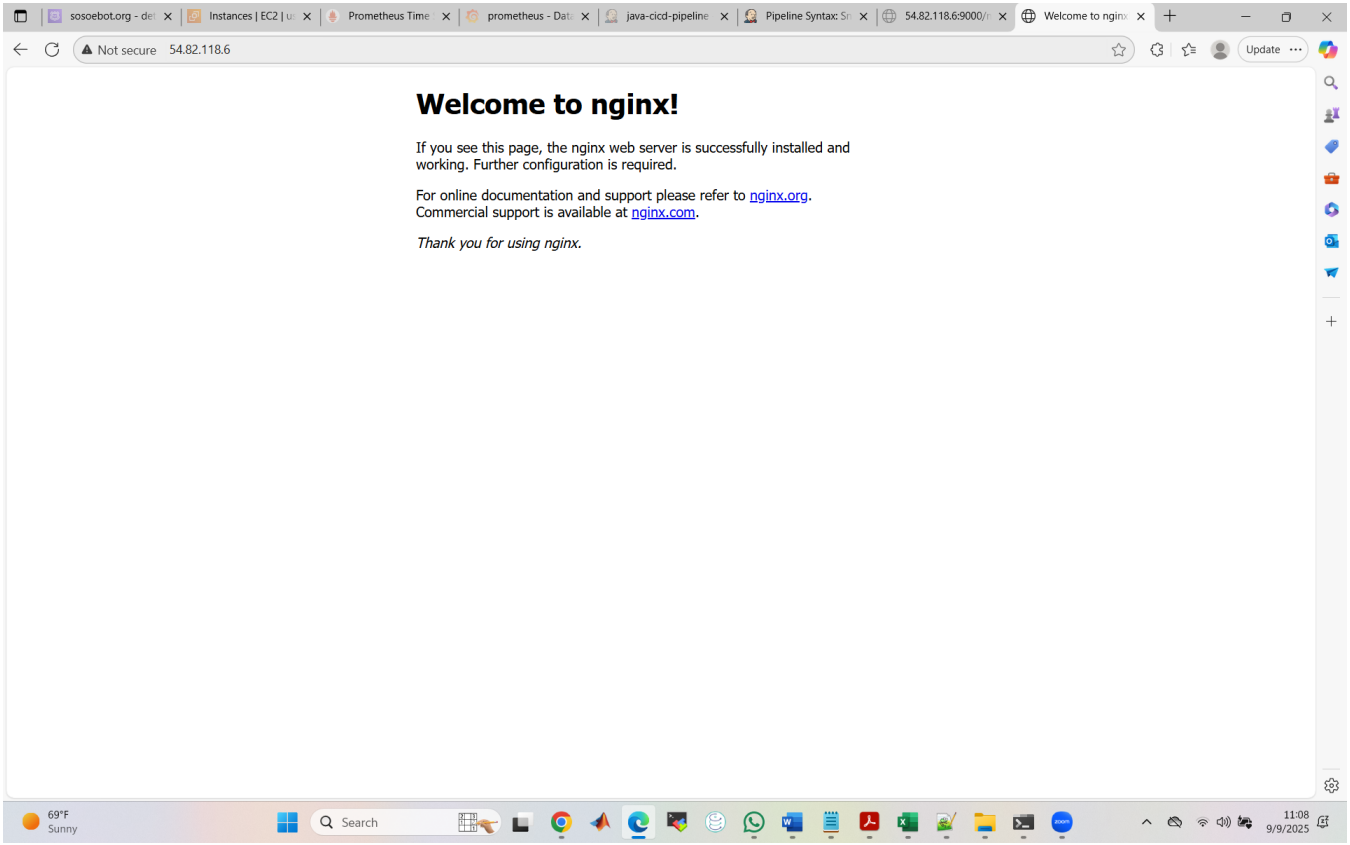
I will be using the sub-domain name **sonarqube.sosoebot.org**. This is mine. Please use yours.

Install Nginx

```
sudo apt install nginx -y
```



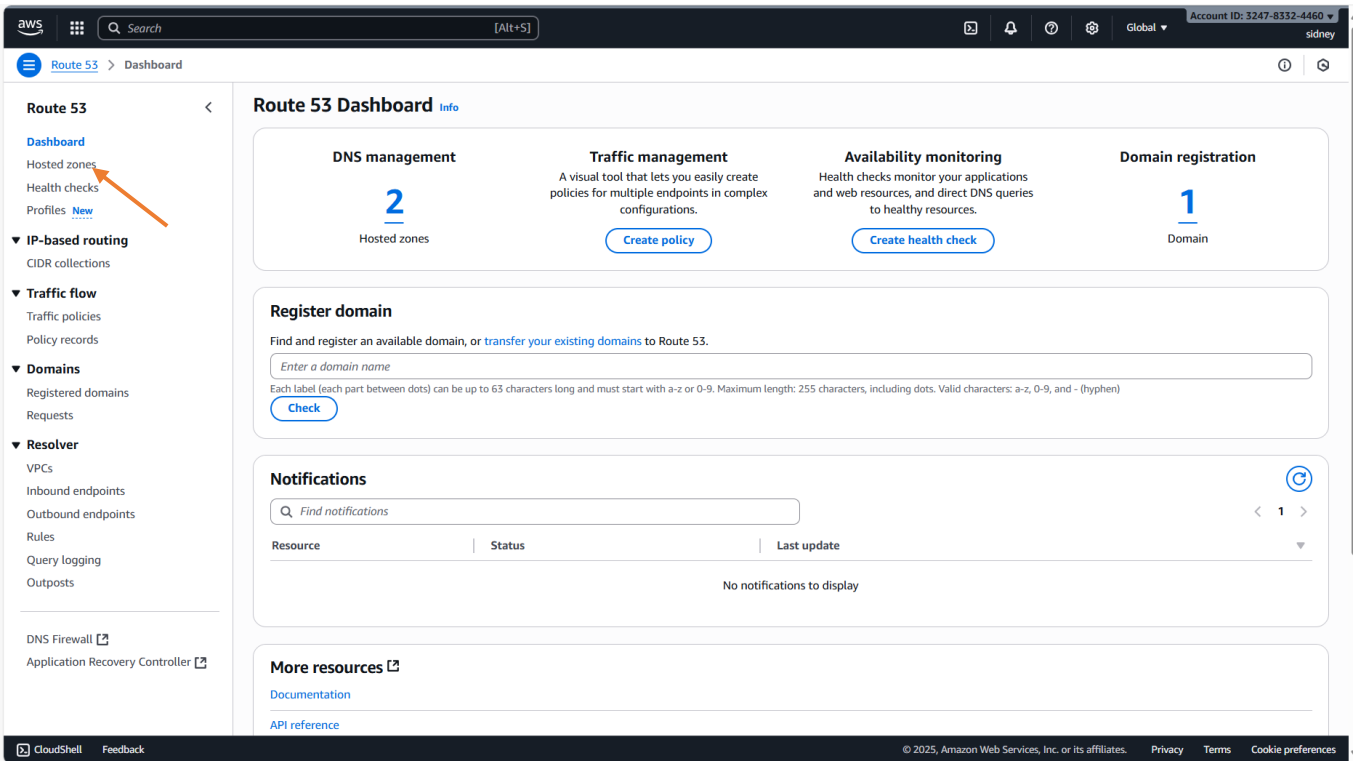
Verify installation by using: http://Public IPv4 Address, that is **http://54.82.118.6**



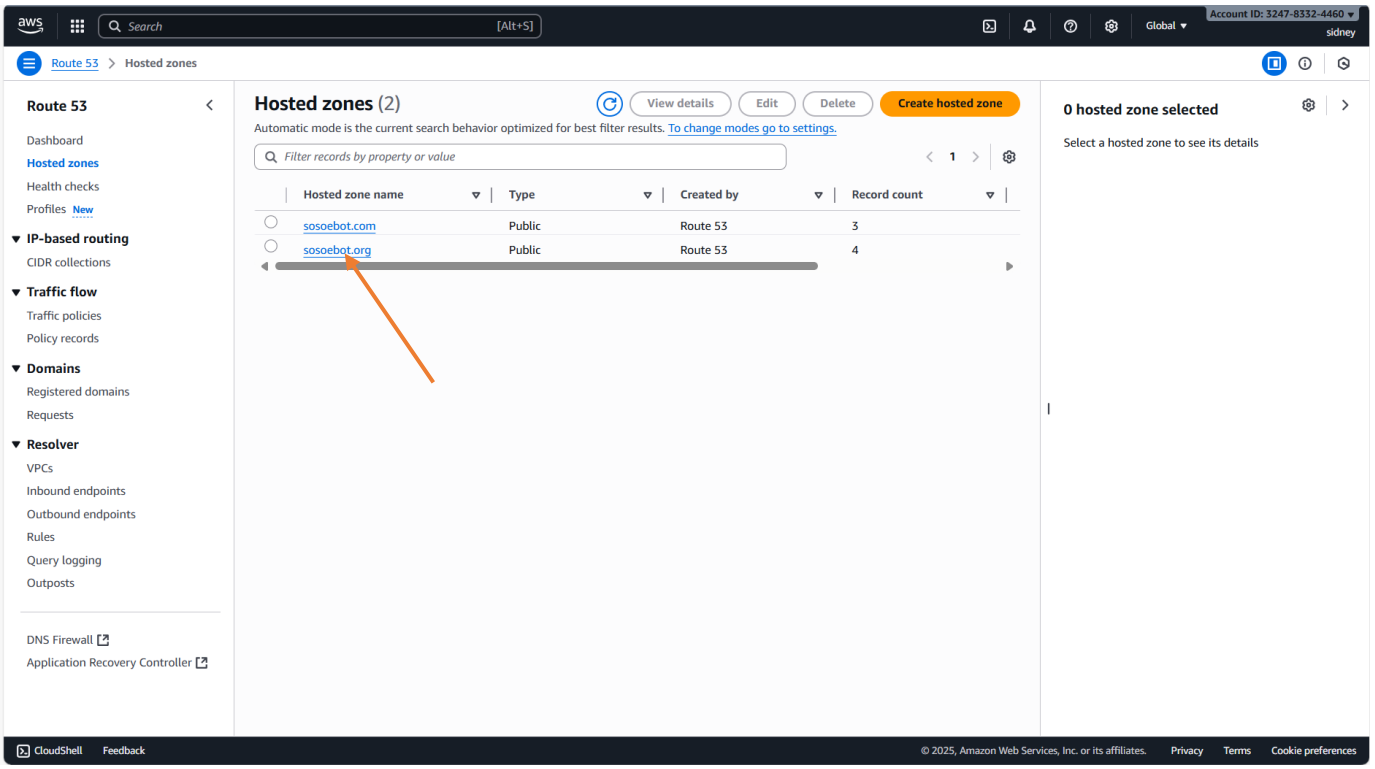
This shows that Nginx has been installed successfully

Add the DNS records

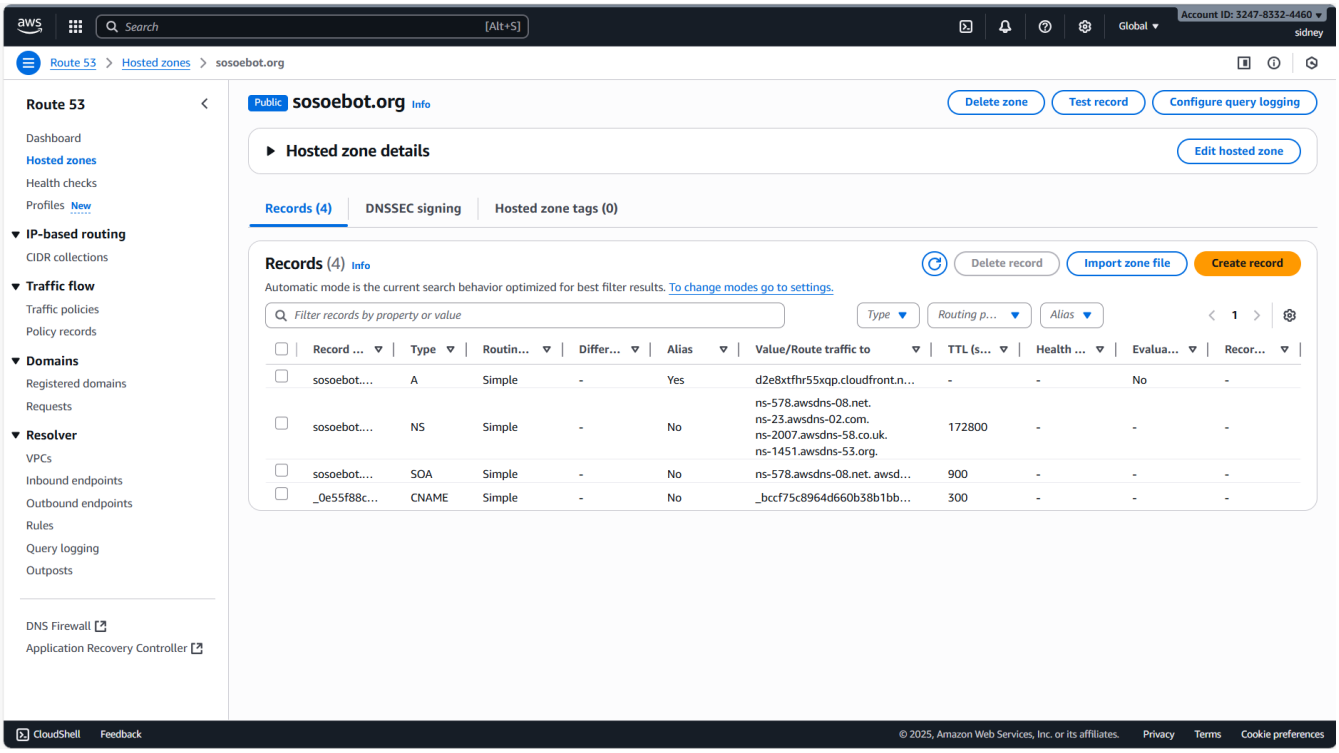
My domain is hosted on the AWS Route53, thus I will be adding the records in there. Go to AWS Console and search for "**Route53**"



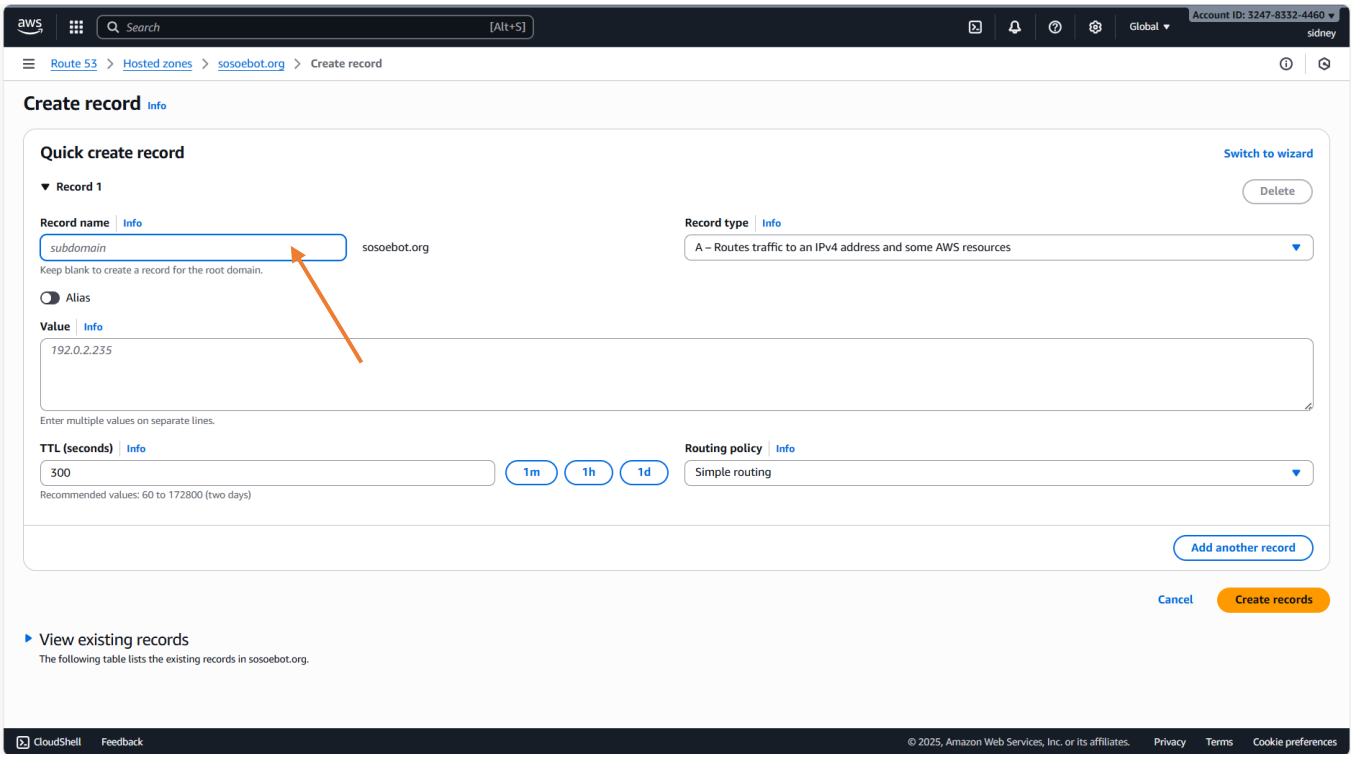
Adding the IP of the server on the DNS zone. Click on "**Hosted Zones**"



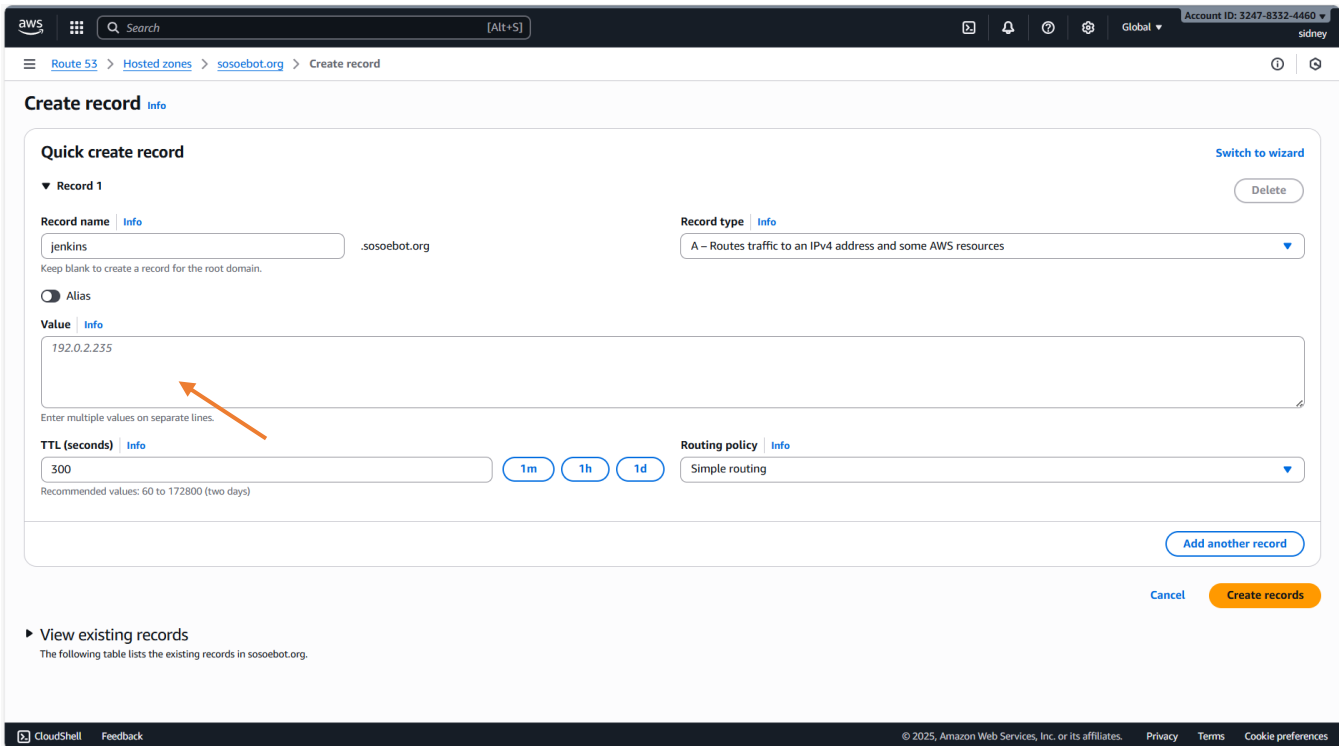
And now our domain sonarqube.sosoebot.org is ready to be used for the nginx configuration setup. Click on “**sosoebot.org**”



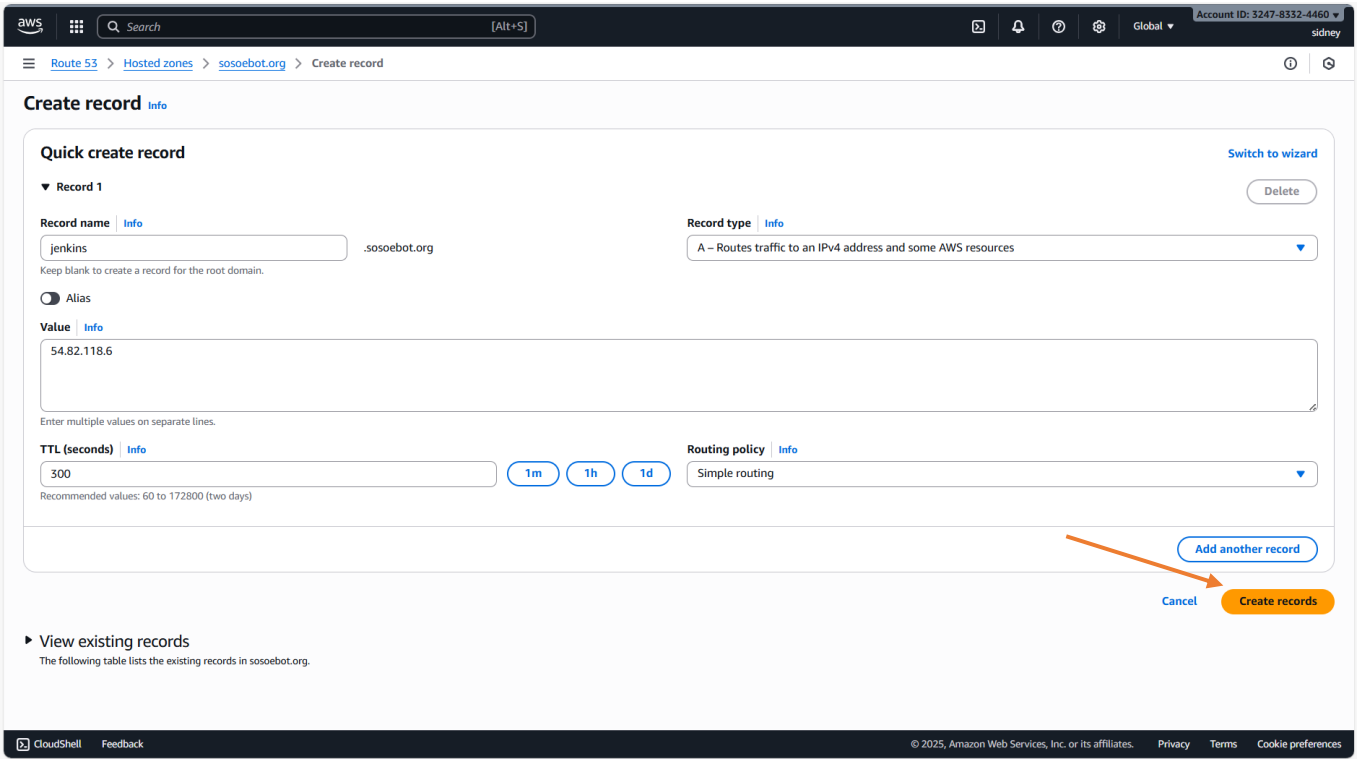
Click on “**Create Record**”



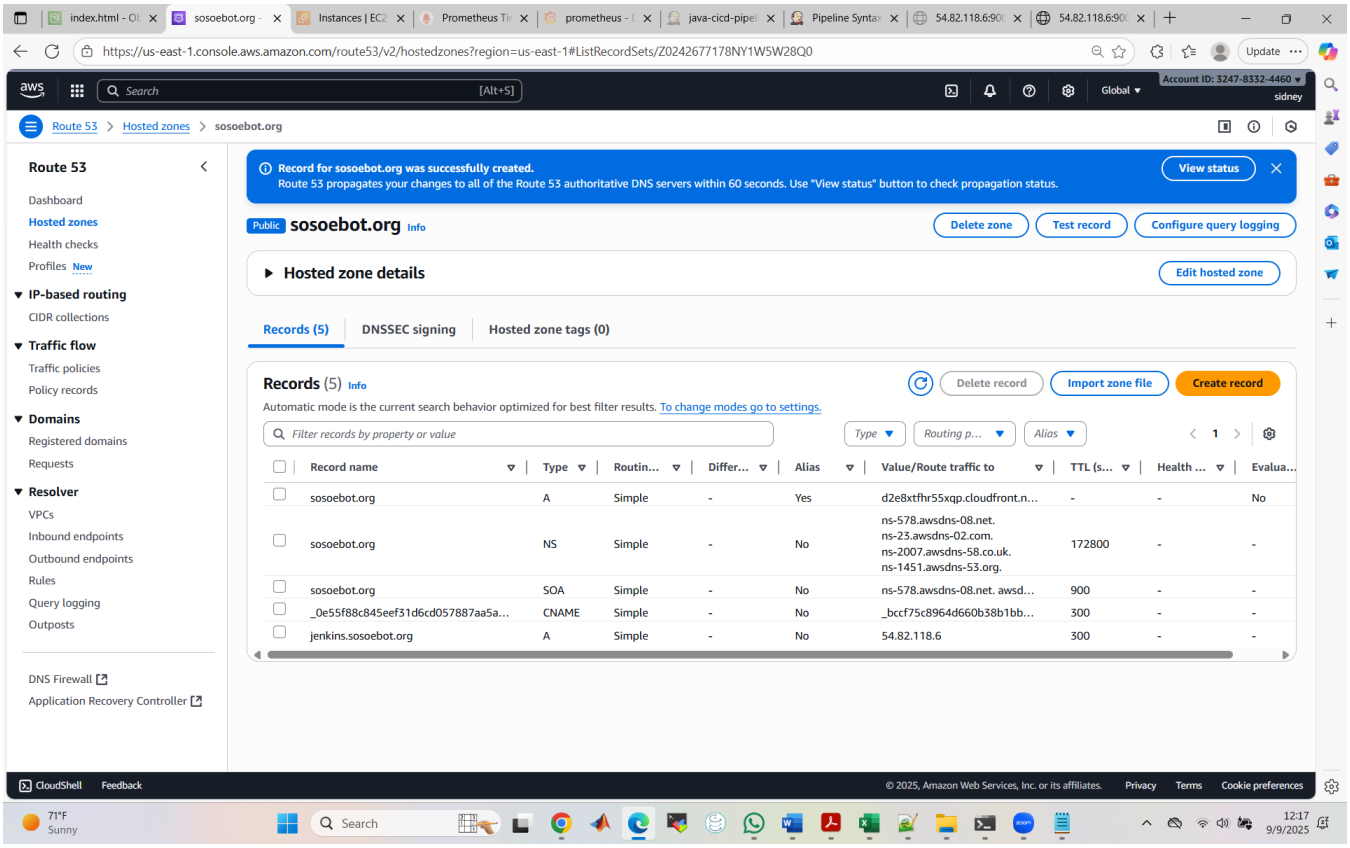
Enter “**jenkins**”



Add the IP address



Click on “**Create records**”



Create a new Nginx configuration file for the site

```
sudo vim /etc/nginx/sites-available/jenkins
```

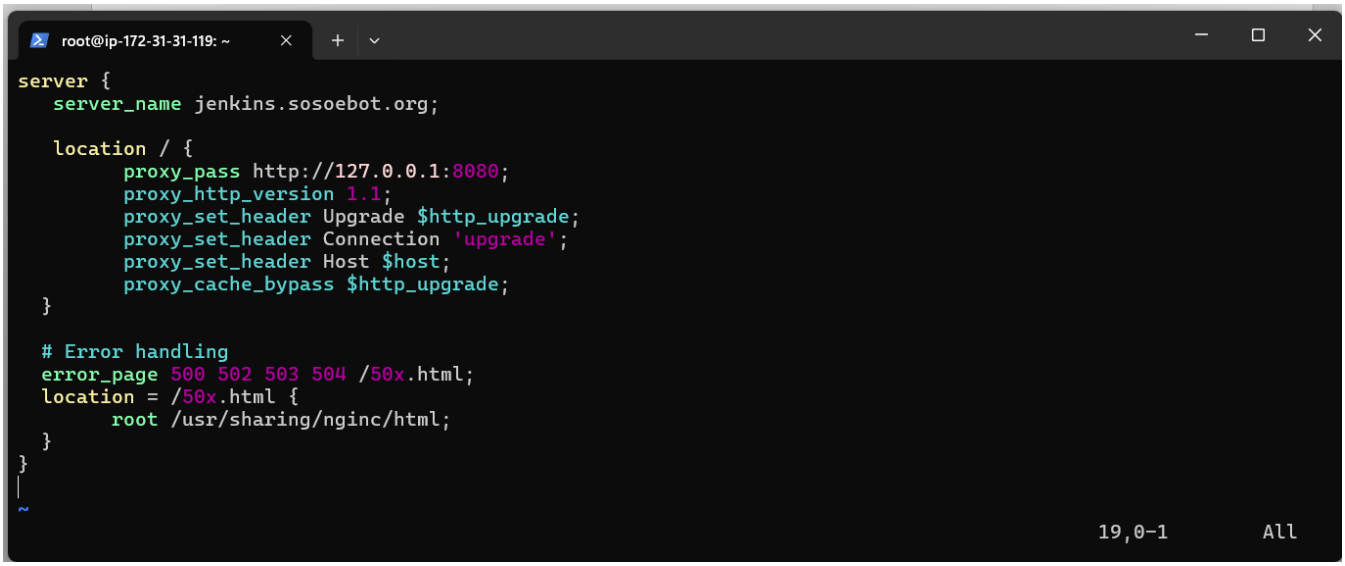
```
sudo vim /etc/nginx/sites-enabled/ddsonarqube
```

Add this configuration so that Nginx will route incoming traffic to Jenkins.

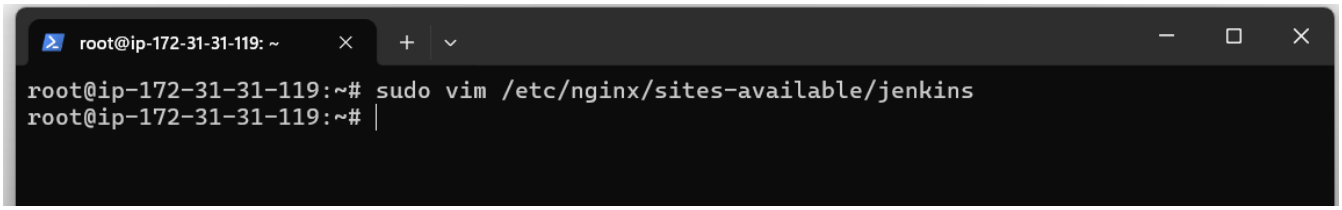
```
server {
```

```
    server_name jenkins.sosoebot.org;

    location / {
        proxy_pass http://127.0.0.1:8080;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }

    # Error handling
    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
        root /usr/sharing/nginc/html;
    }
}
```

Paste this code

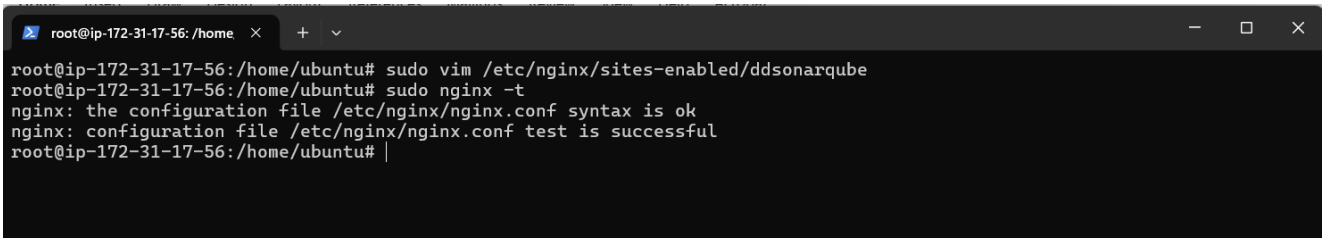


Save it by Pressing ESC, followed by :wq and press ENTER



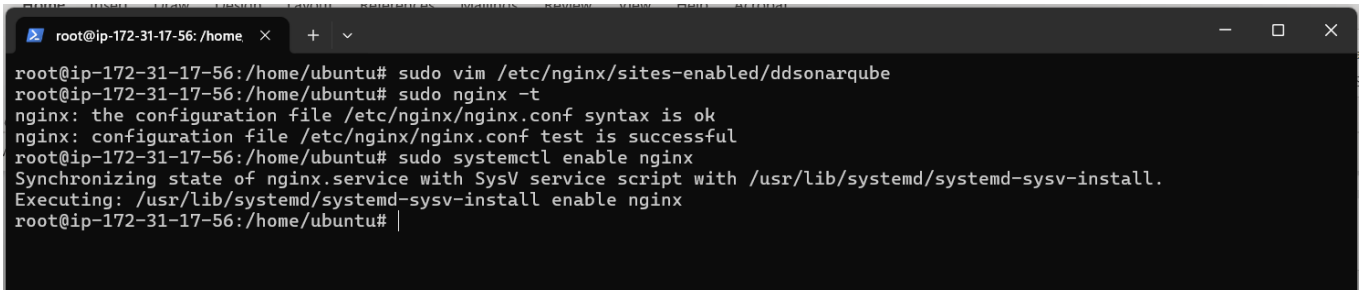
Make sure your configuration file has no syntax errors.

```
sudo nginx -t
```



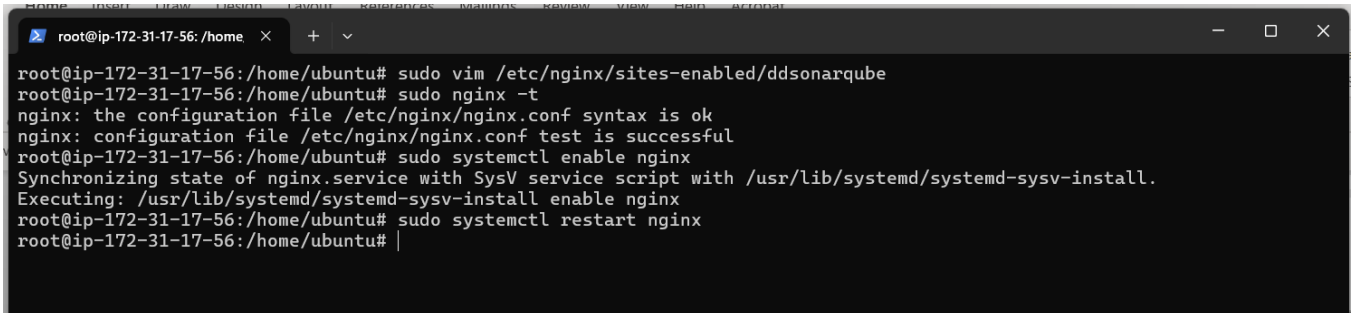
Enable the Nginx site and restart Nginx

```
sudo systemctl enable nginx
```



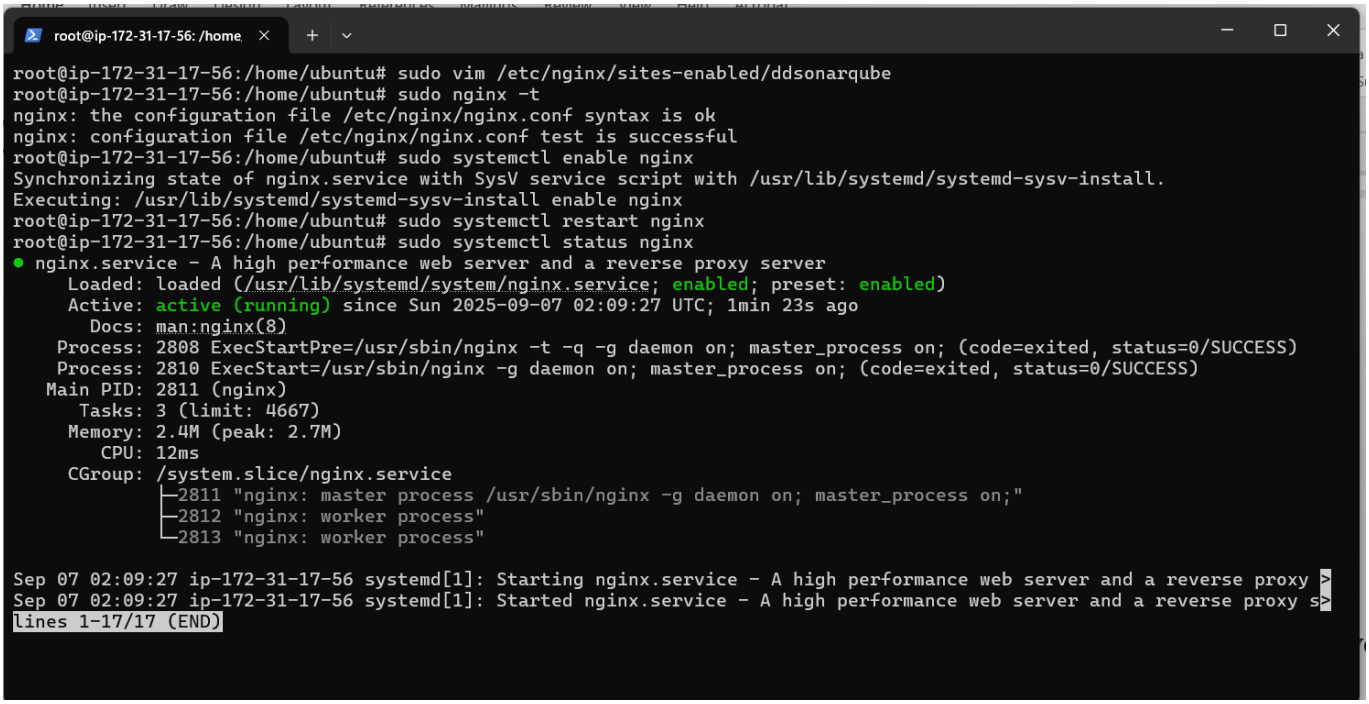
Restart the nginx server

```
sudo systemctl restart nginx
```



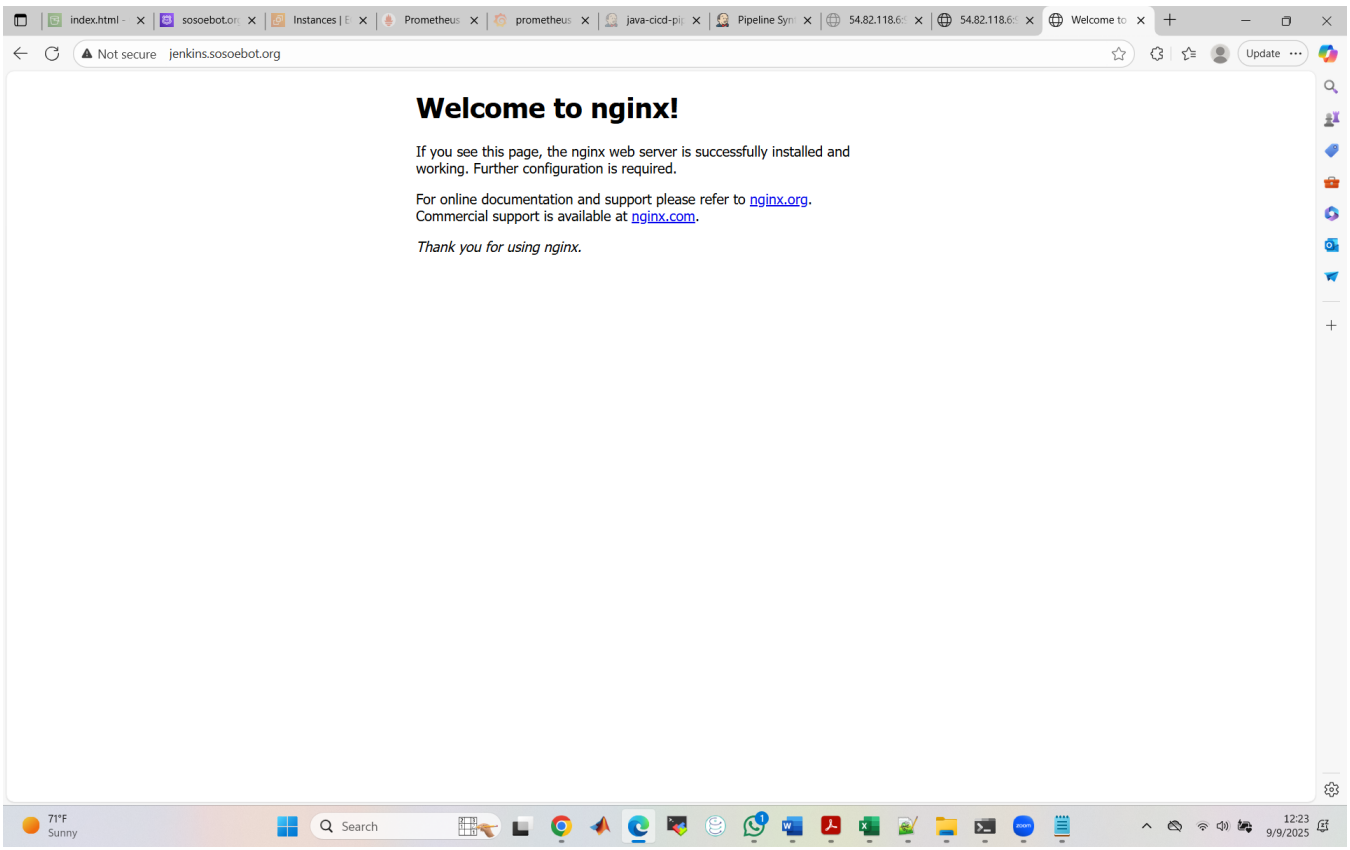
Check the status of the nginx server

```
sudo systemctl status nginx
```



Use the configured domain

You can now visit http://jenkins.sosoebot.org in your web browser. You will be greeted with the Jenkins web interface.

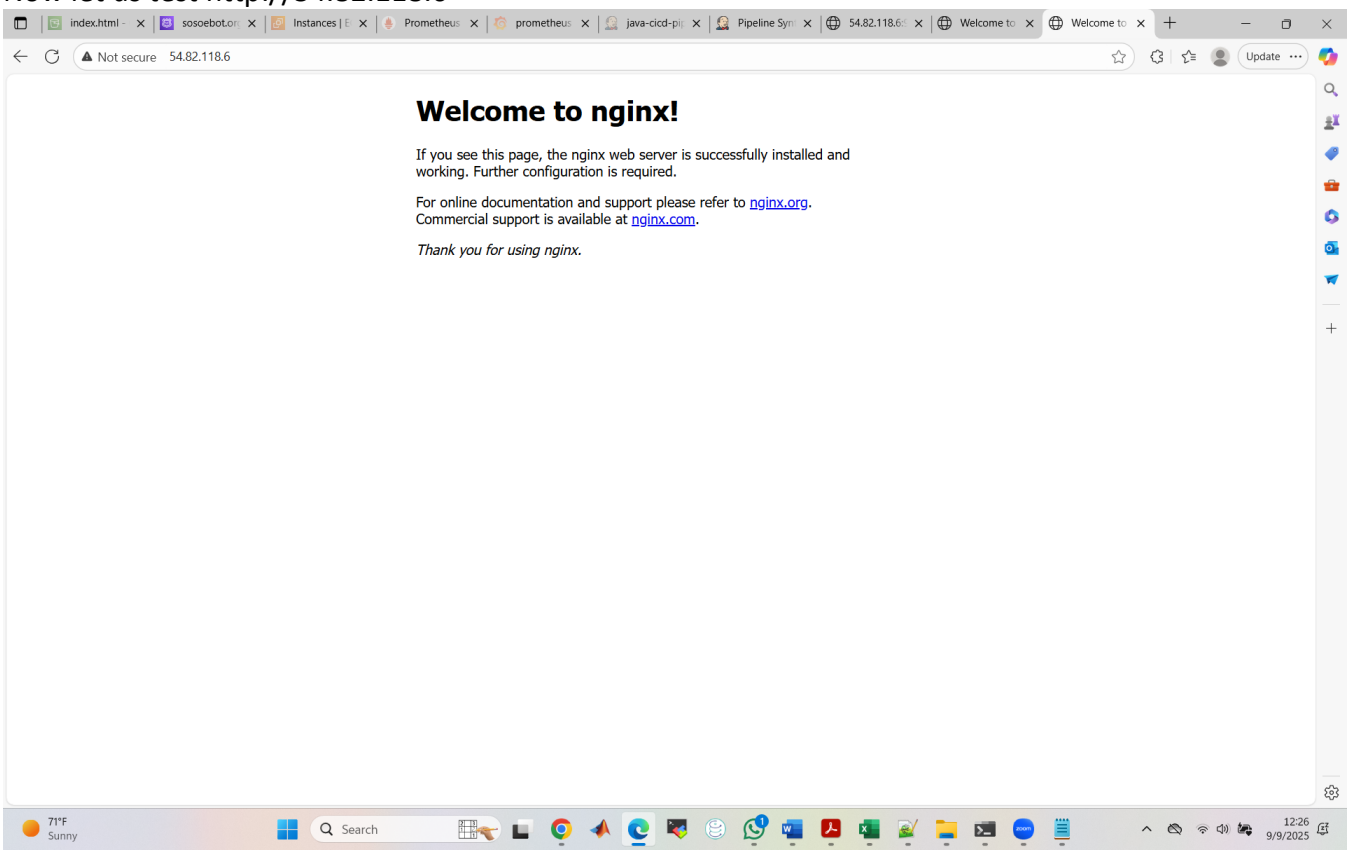


The configured domain sonarqube.sosoebot.org is working perfectly. Wait, seems there is a problem ahead, let's solve it. There seems to be a problem ahead

Problem:

But the domain is working on HTTP protocol and we need to make it secure by making it work with the HTTPS protocol with SSL certificate.

Now let us test http://54.82.118.6

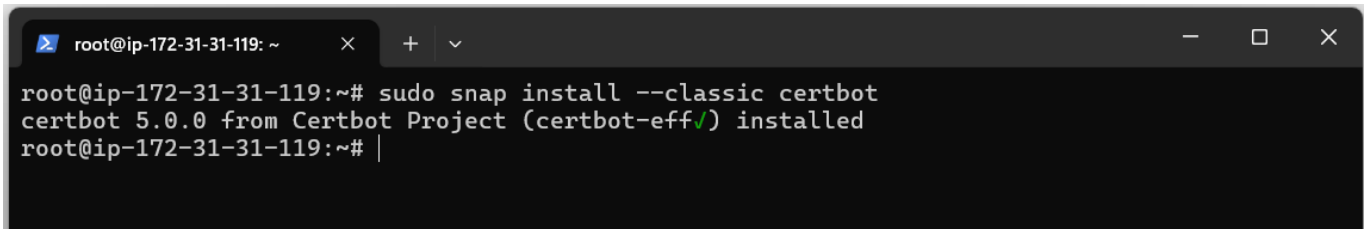


Part 2: Secure with SSL using Certbot

We will Install SSL certificate for maximum security. We will use the free SSL certificate provided by Certbot for our sub-domain sonarqube.sosoebot.org. Let's see how we will do this.

Installing certbot

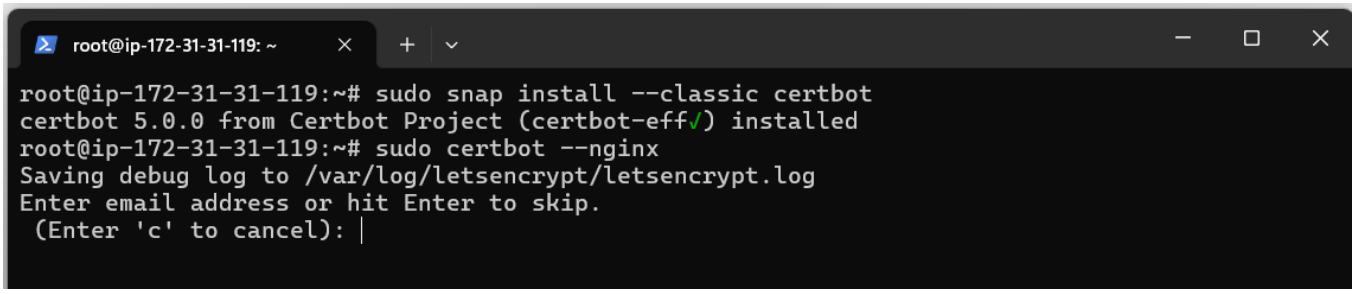
```
sudo snap install --classic certbot
```



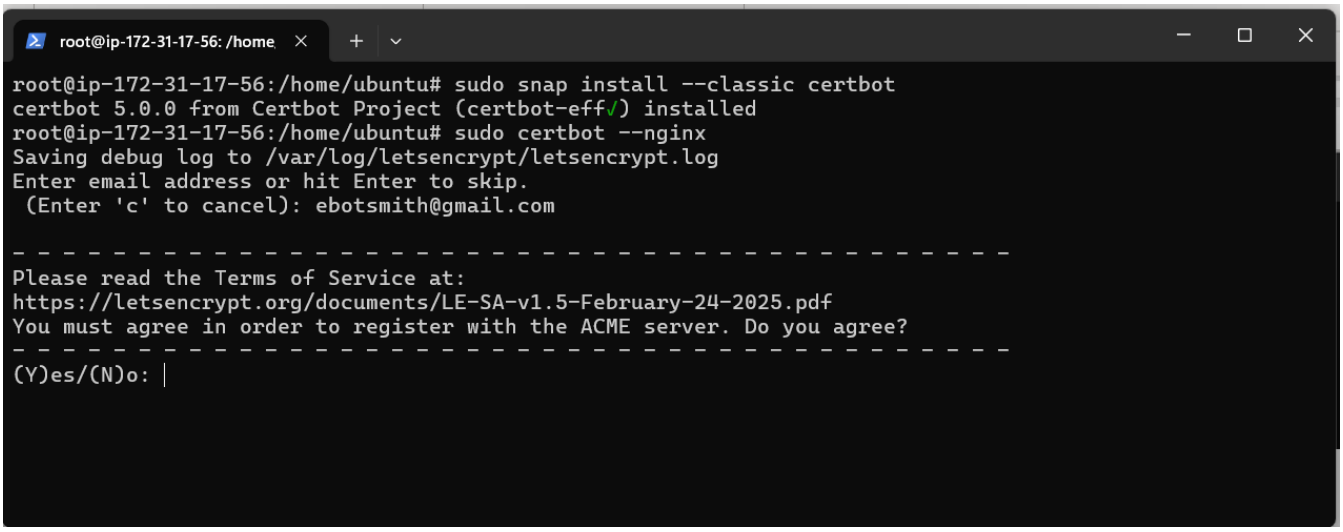
Obtain and install an SSL certificate on the subdomain

Generating SSL certificate for the required domain. Please read the asked question carefully and answer them accordingly.

```
sudo certbot --nginx
```

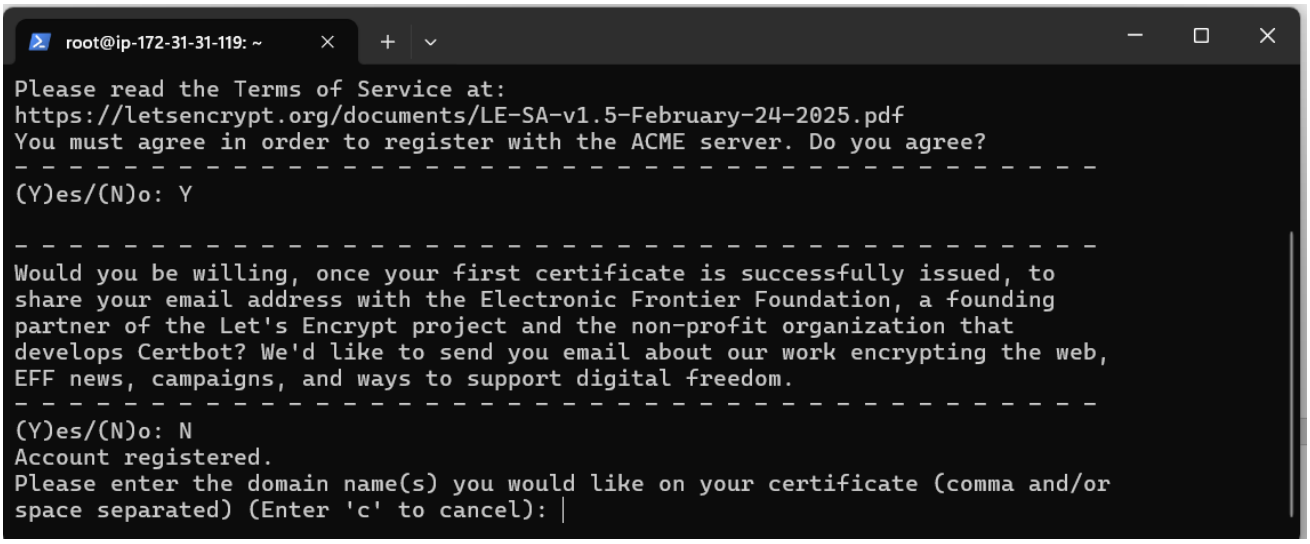


Enter your email address. I will enter “ebotsmith@gmail.com”

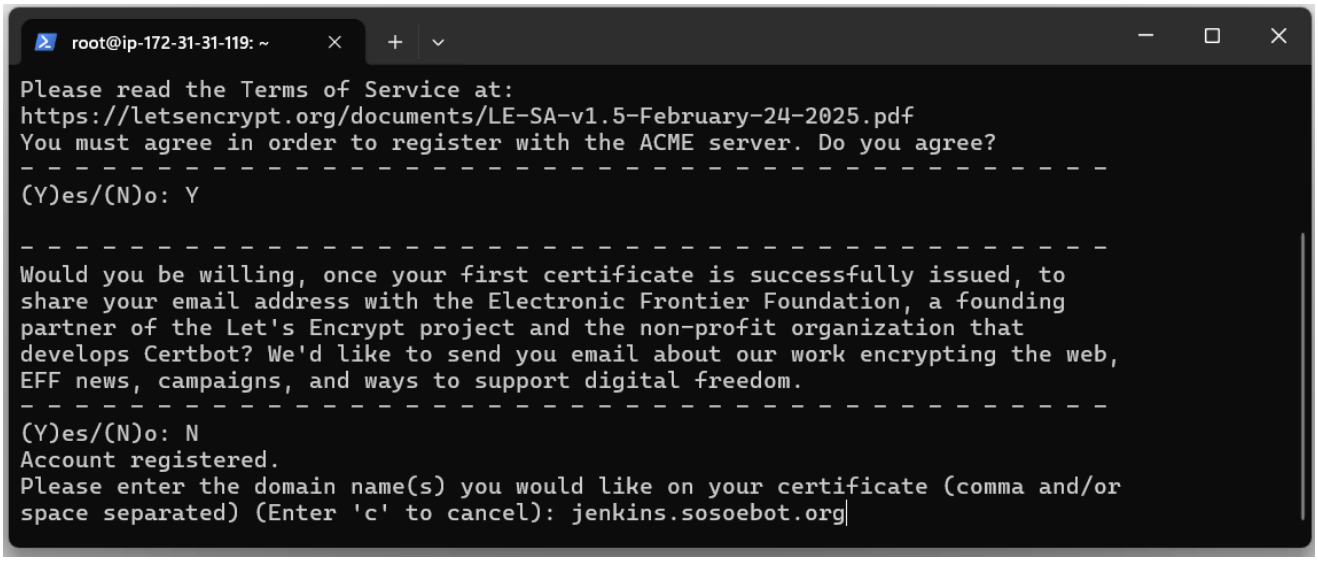


Enter “**Y**” and press ENTER

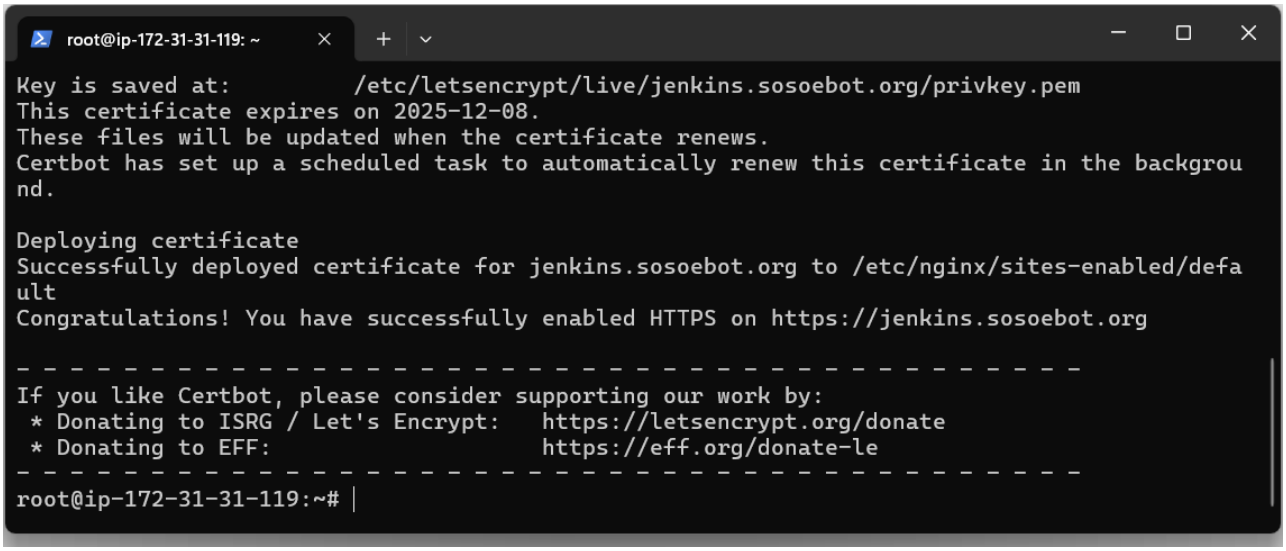
Enter “N” and press ENTER



Enter “**jenkins.sosoebot.org**”



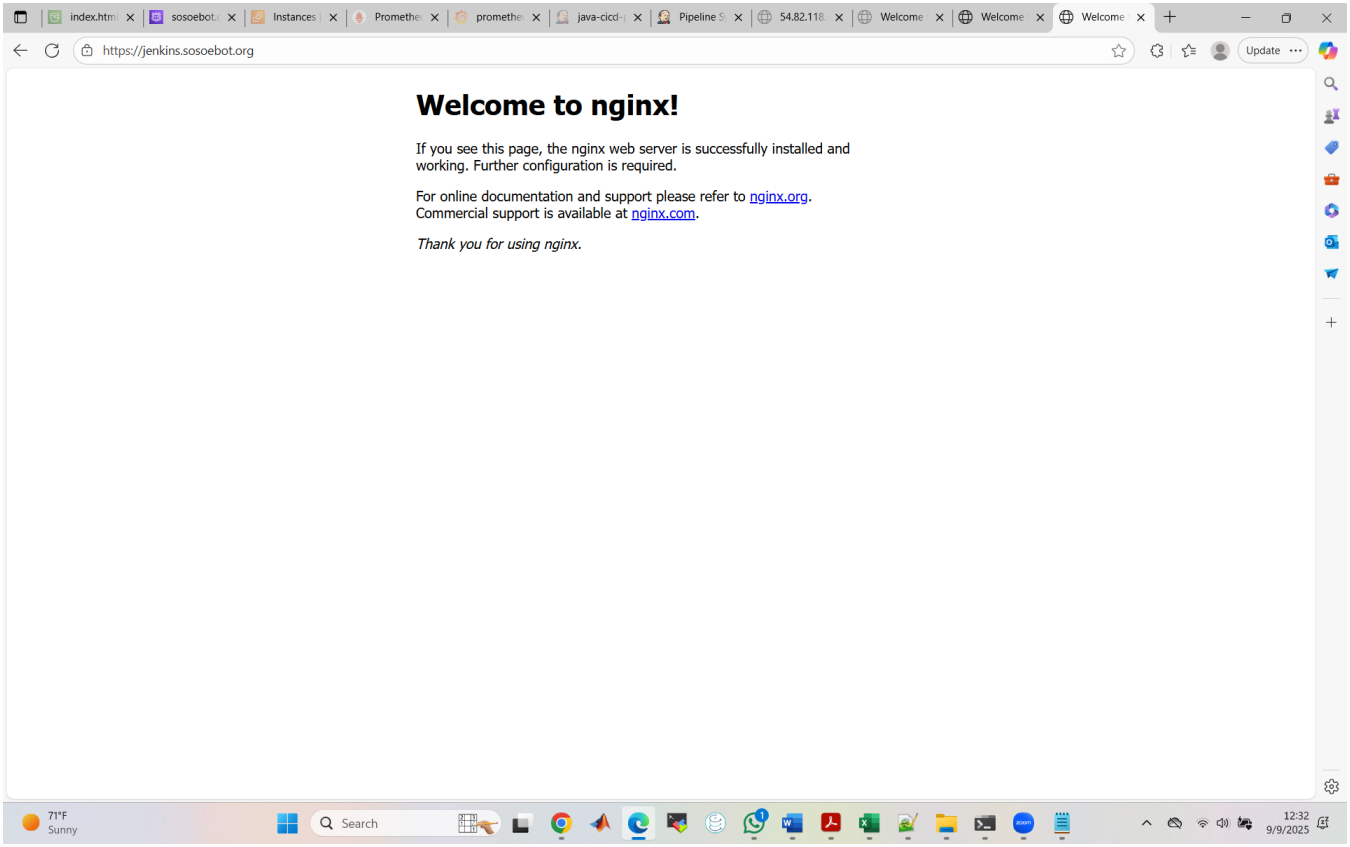
Press Enter



Success message that the SSL certificate has been deployed on the domain sonarqube.sosoebot.org

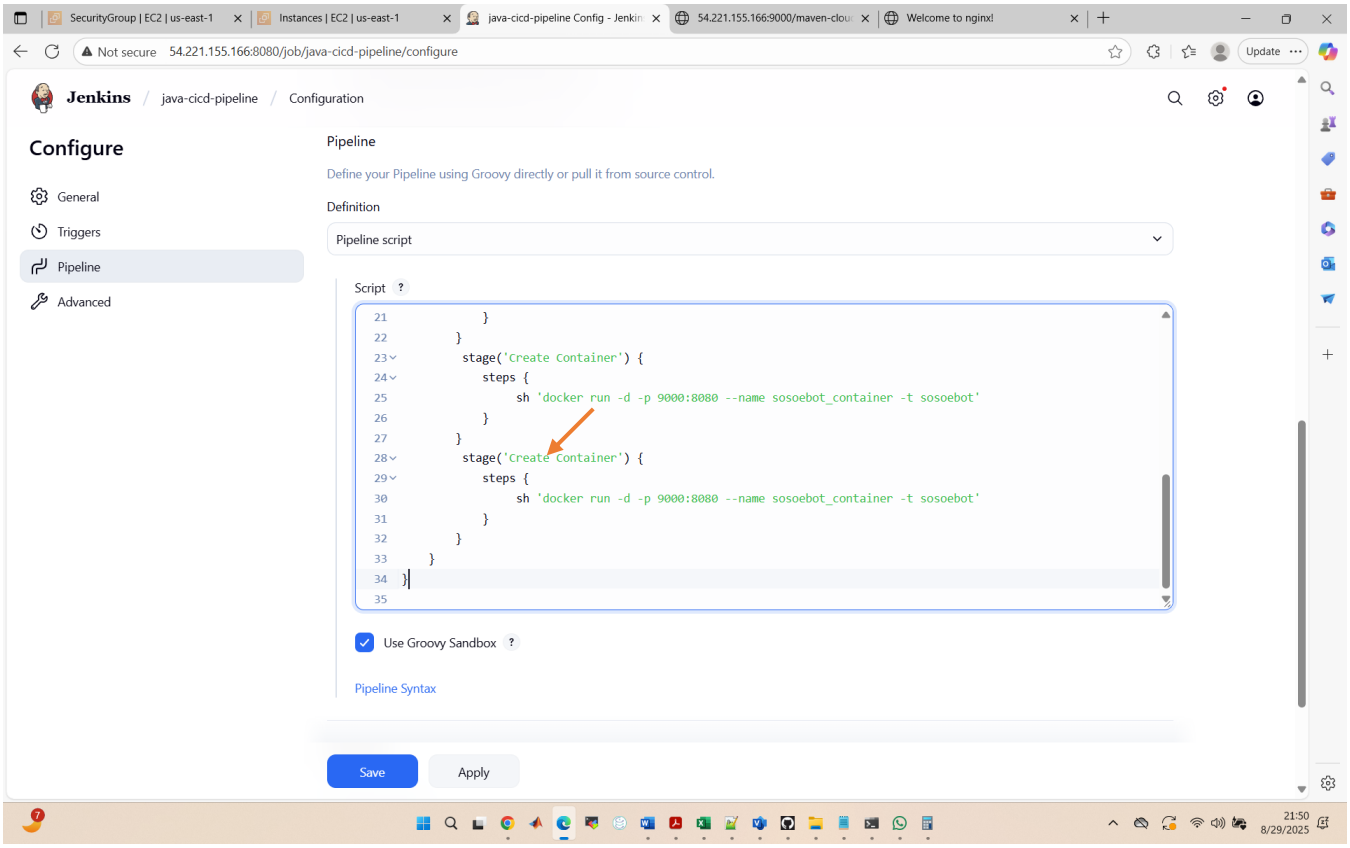
Verify if Certificate deployed successfully

Now if we hit the domain **jenkins.sosoebot.org**, it automatically redirects to HTTPS.

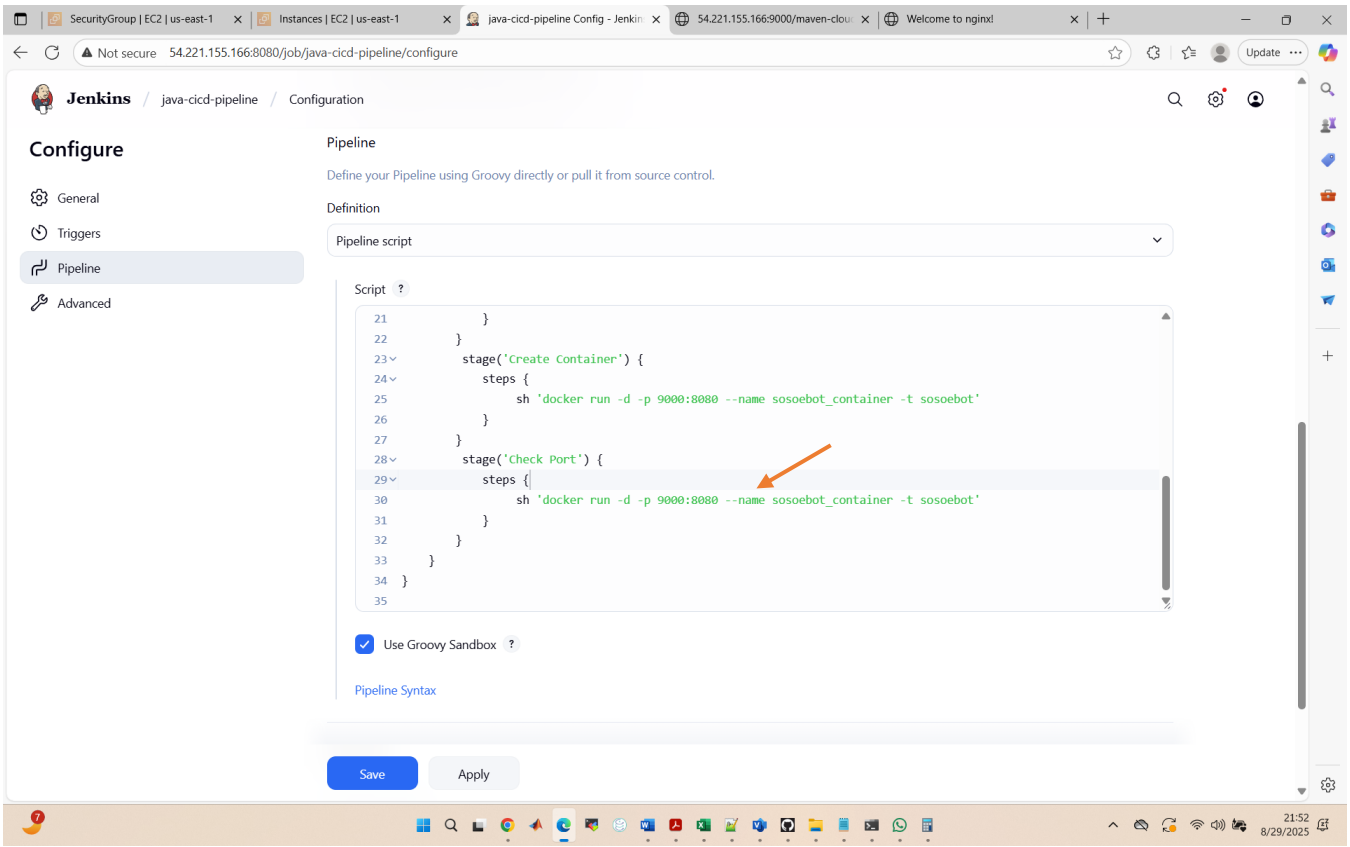


Now you can access SonarQube through your domain name or EC2 public IP address via Nginx (on port 80 or 443 since SSL has been configured.

It is directing us to the Nginx web page, but we want to access our webpage. We will fix this we have to give the port access. To do this, we have to add another stage in our pipeline.

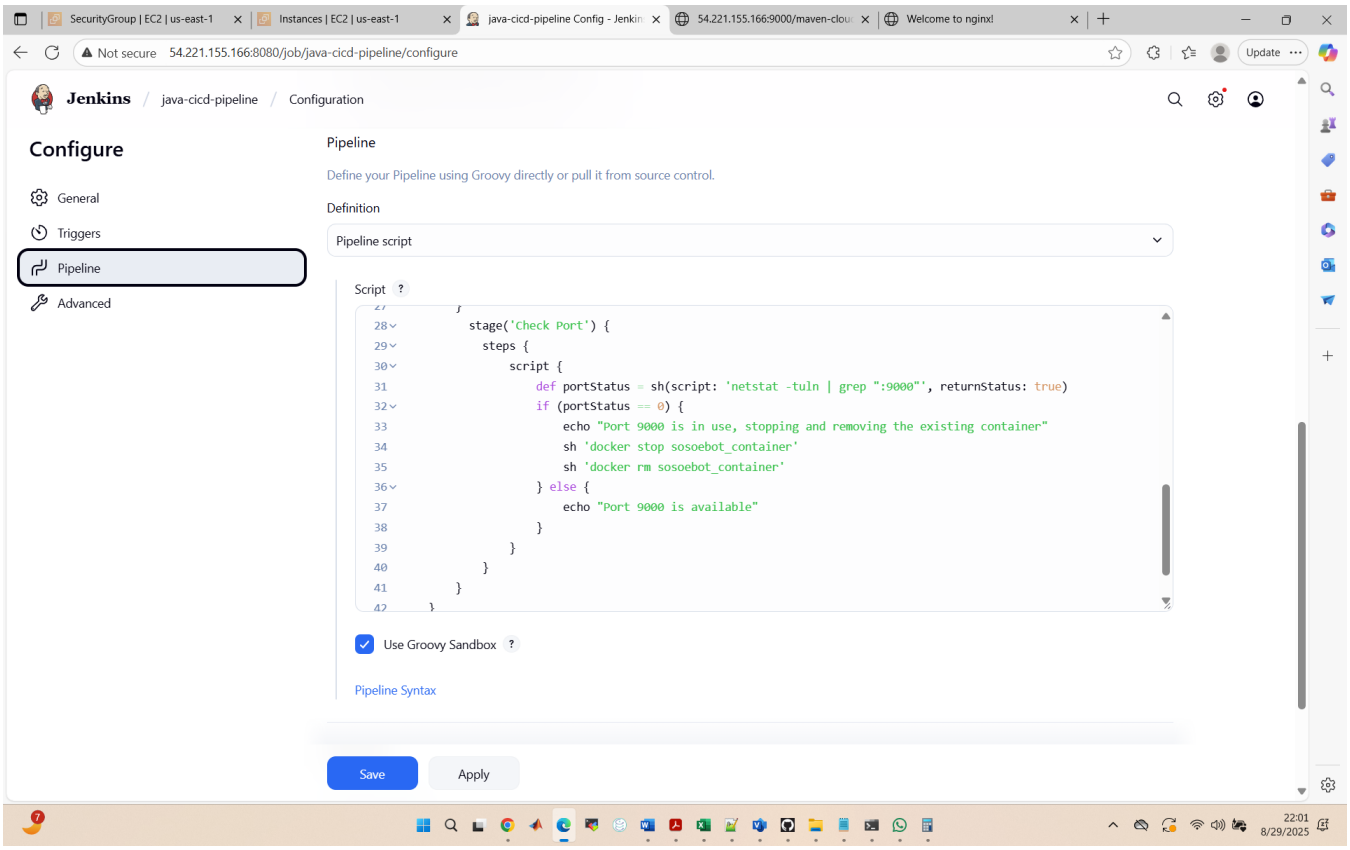


Change the stage name from "**Create Container**" to "**Check Port**"

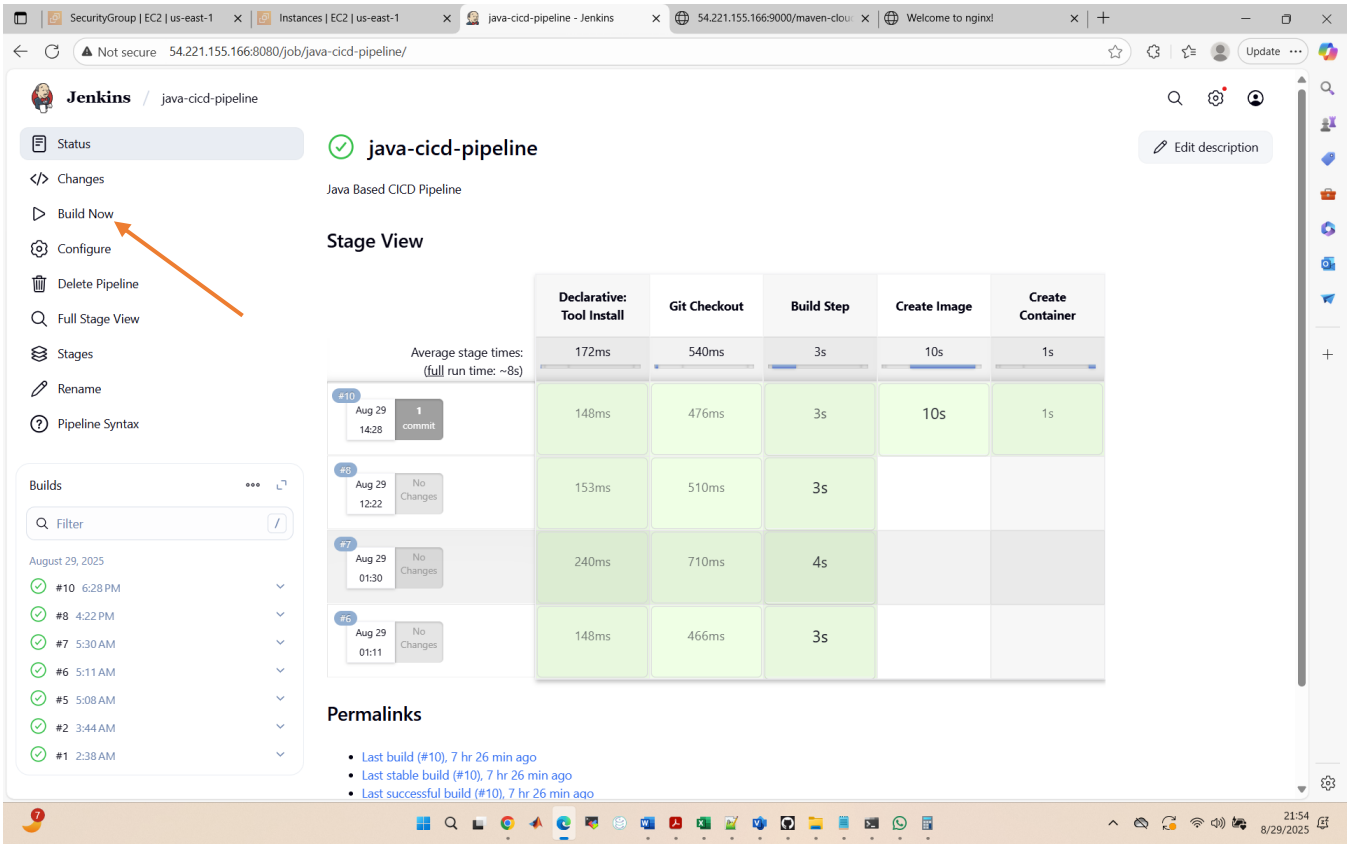


Then, modify the step as follows:
steps {

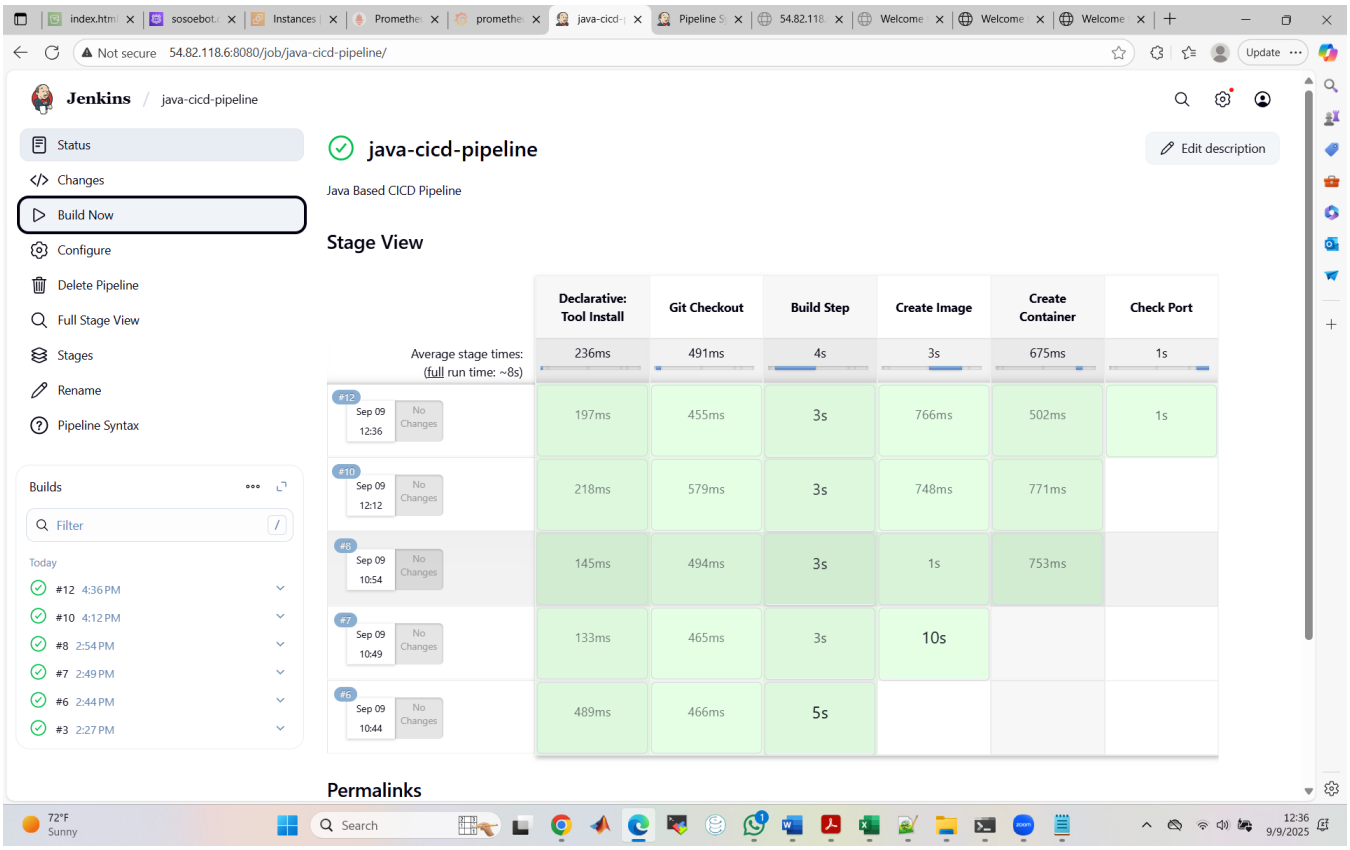
```
        script {
            def portStatus = sh(script: 'netstat -tuln | grep ":9000"', returnStatus: true)
            if (portStatus == 0) {
                echo "Port 9000 is in use, stopping and removing the existing container"
                sh 'docker stop sosoebot_container'
                sh 'docker rm sosoebot_container'
            } else {
                echo "Port 9000 is available"
            }
        }
    }
```



Then click on “**Apply**” followed by “**Save**”



Then click on “**Build Now**”



The build is successful

Jenkins Pipeline Code

```
pipeline {
    agent any

    tools{
        maven 'maven3'
    }

    stages {
        stage('Git Checkout') {
            steps {
                git branch: 'main', url: 'https://github.com/ebotsidneysmith/maven-app.git'
            }
        }
         stage('Build Step') {
            steps {
                sh 'mvn clean package'
            }
        }
        stage('Create Image') {
            steps {
                sh 'docker build -t sosoebot .'
            }
        }
        stage('Create Container') {
            steps {
                sh 'docker run -d -p 9000:8080 --name sosoebot_container -t sosoebot'
            }
        }
        stage('Check Port') {
            steps {
                script {
                    def portStatus = sh(script: 'netstat -tuln | grep ":9000"', returnStatus: true)
                    if (portStatus == 0) {
                        echo "Port 9000 is in use, stopping and removing the existing container"
                        sh 'docker stop sosoebot_container'
                        sh 'docker rm sosoebot_container'
                    } else {
                        echo "Port 9000 is available"
                    }
                }
            }

        }
    }
}
```