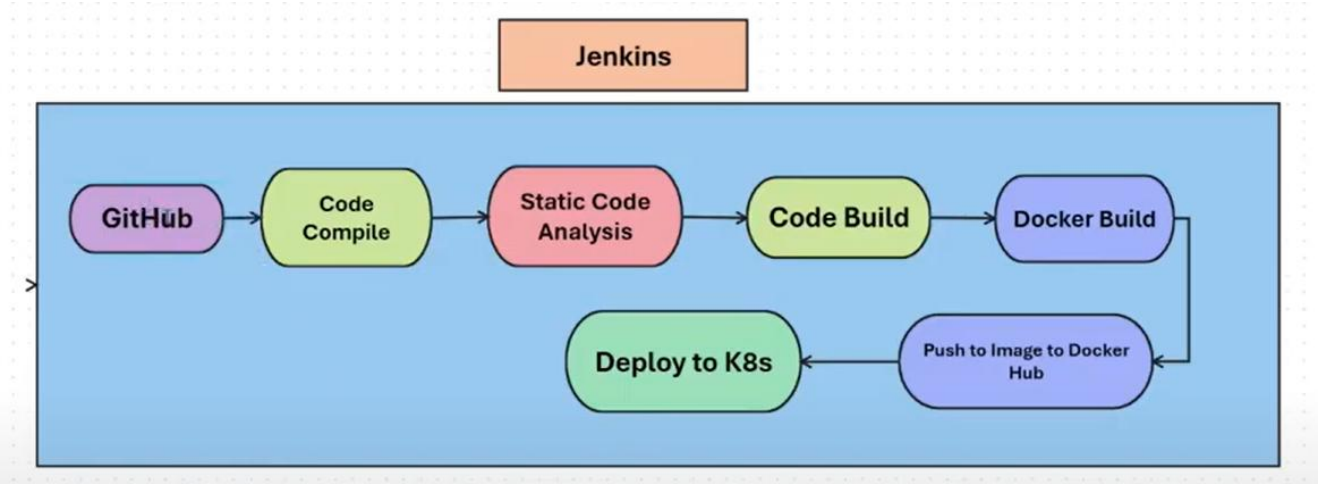
CI/CD Pipeline Implementation using Jenkins with Kubernetes, SonarQube, Docker, Maven



Initially, we will checkout the code in GitHub and then we will compile the code using Maven. Then we use SonarQube for static code analysis, then we build the code using Maven, then we will use Docker to build the image, once the image is built, we will push it into the Docker hub. From the Docker hub, we will be deploying it on the Kubernetes environment.

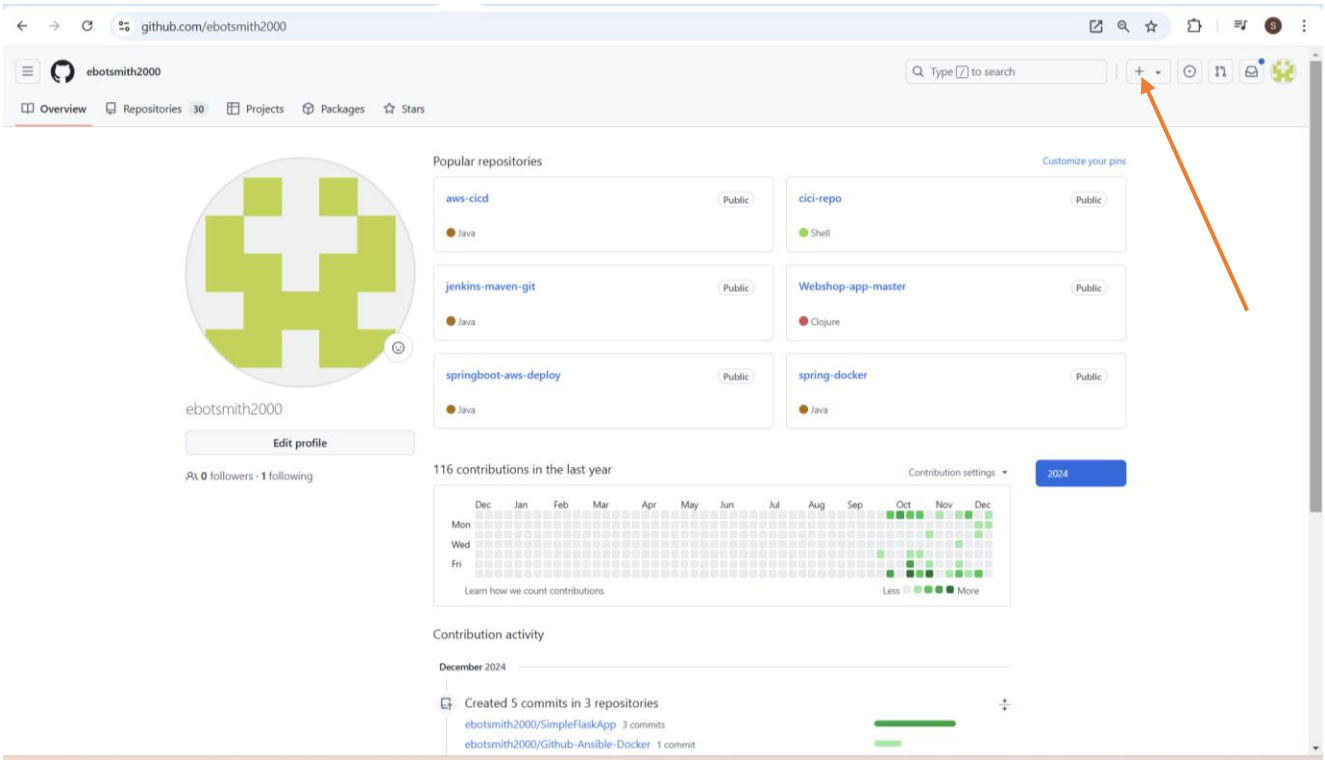
We will see all the plugins installation of all the tools and how the application runs. We will follow these steps:

STEPS:

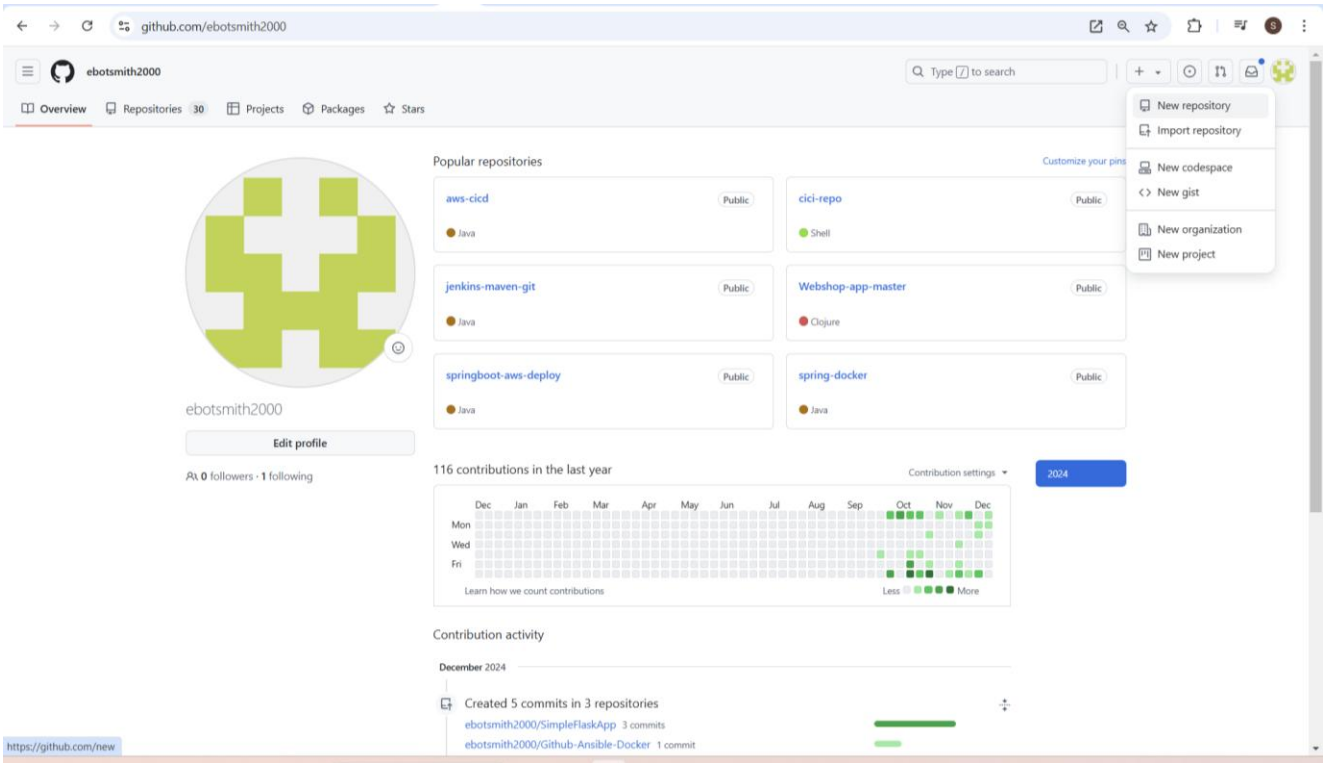
- **Step 1**: Create a GitHub repository and upload the project files to it
- **Step 2**: Create EC2 instance that will serve as Jenkins Server
- **Step 3**: Add Port 8080 in the inbound rule of the Instance for Jenkins
- **Step 4**: Connect to the Jenkins server
- **Step 5**: Install Java JDK 17
- **Step 6**: Install Jenkins
- **Step 7**: Access Jenkins GUI on browser
- **Step 8**: Create and Configure Pipeline on Jenkins
- **Step 9**: Install Pipeline Stage View plugin
- **Step 10**: Install and configure Maven with Jenkins
- **Step 11**: Install and Configure SonarQube with Jenkins
- **Step 12**: Configure Maven for Code Build
- **Step 13**: Install Docker and Build Docker Image
- **Step 14**: Push the Image to Docker Hub
- **Step 15**: Deploy Image to Kubernetes
- **Step 16**: Automating the process

Step 1: Create a GitHub repository and upload the project files to it

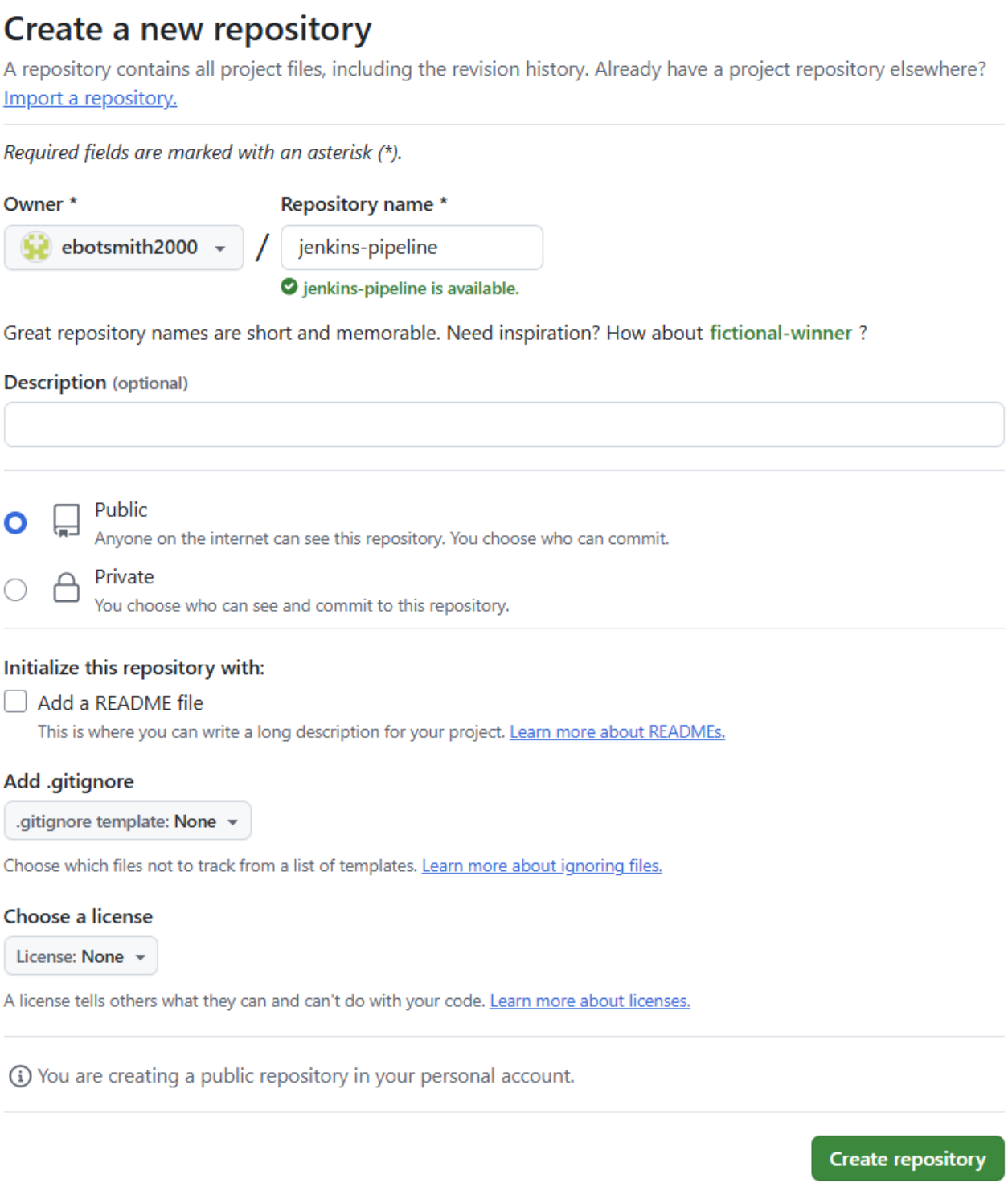
Now, we have to create a repository on GitHub and upload our project files from our laptop to the GitHub repository.



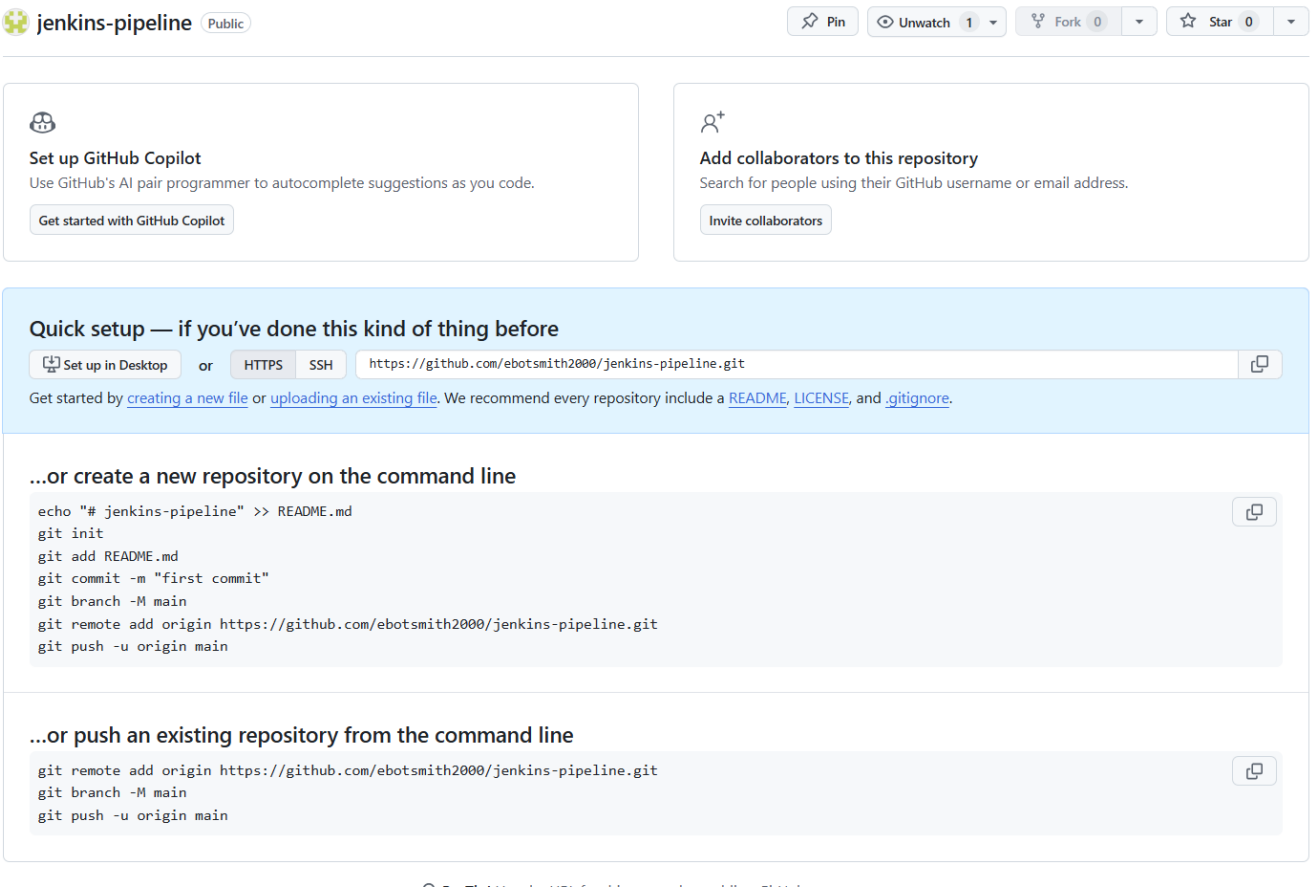
We will start by creating the repository. Click on "**+**"



Select “**New Repository**”. Enter the name of the repository, we will call it “**jenkins-pipeline**”

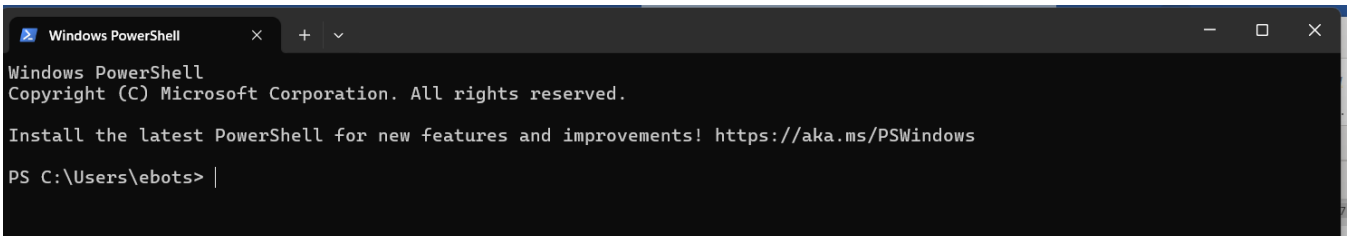


Click on “**Create Repository**”

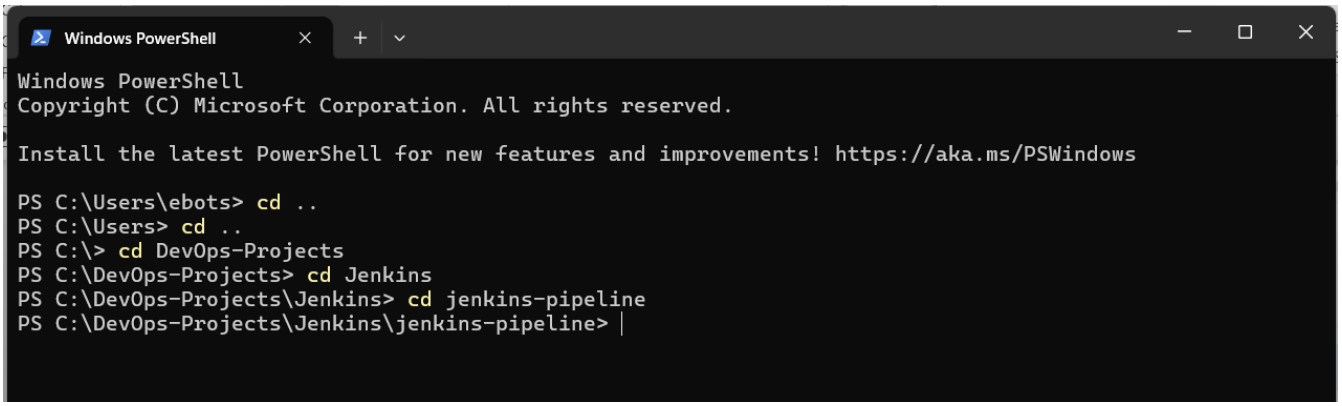


We have to navigate to the folder in our Local machine where we have our project files. The files are in this path: `C:\DevOps-Projects\Jenkins\jenkins-pipeline`

Open PowerShell



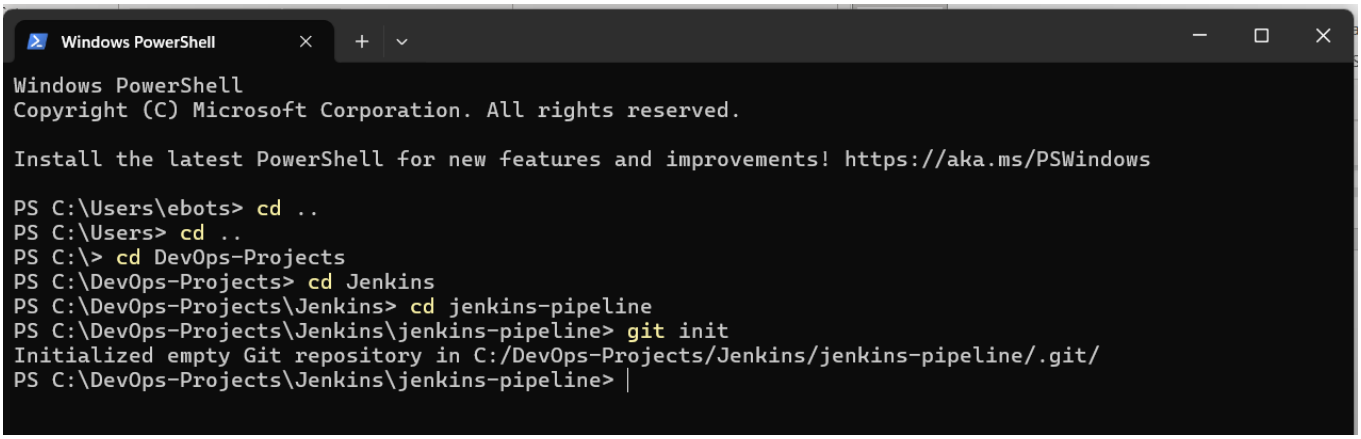
We now navigate to the folder



We are now in the repository in our local machine where the project files are stored

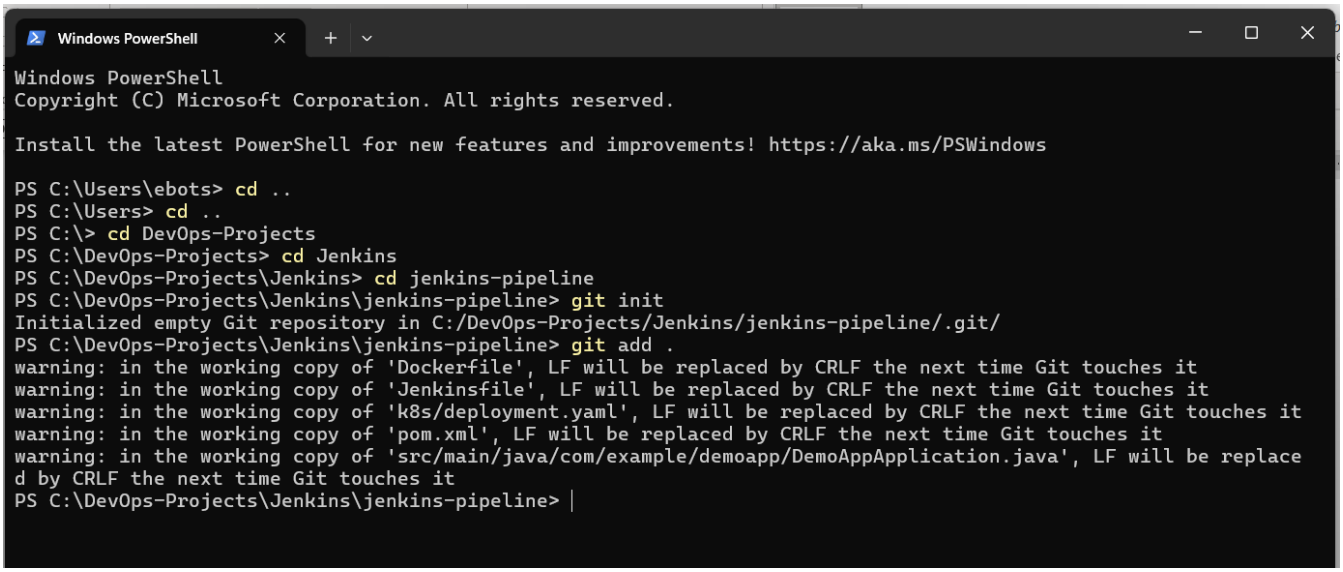
Initialize the repository in the local machine by using the command:

```
git init
```



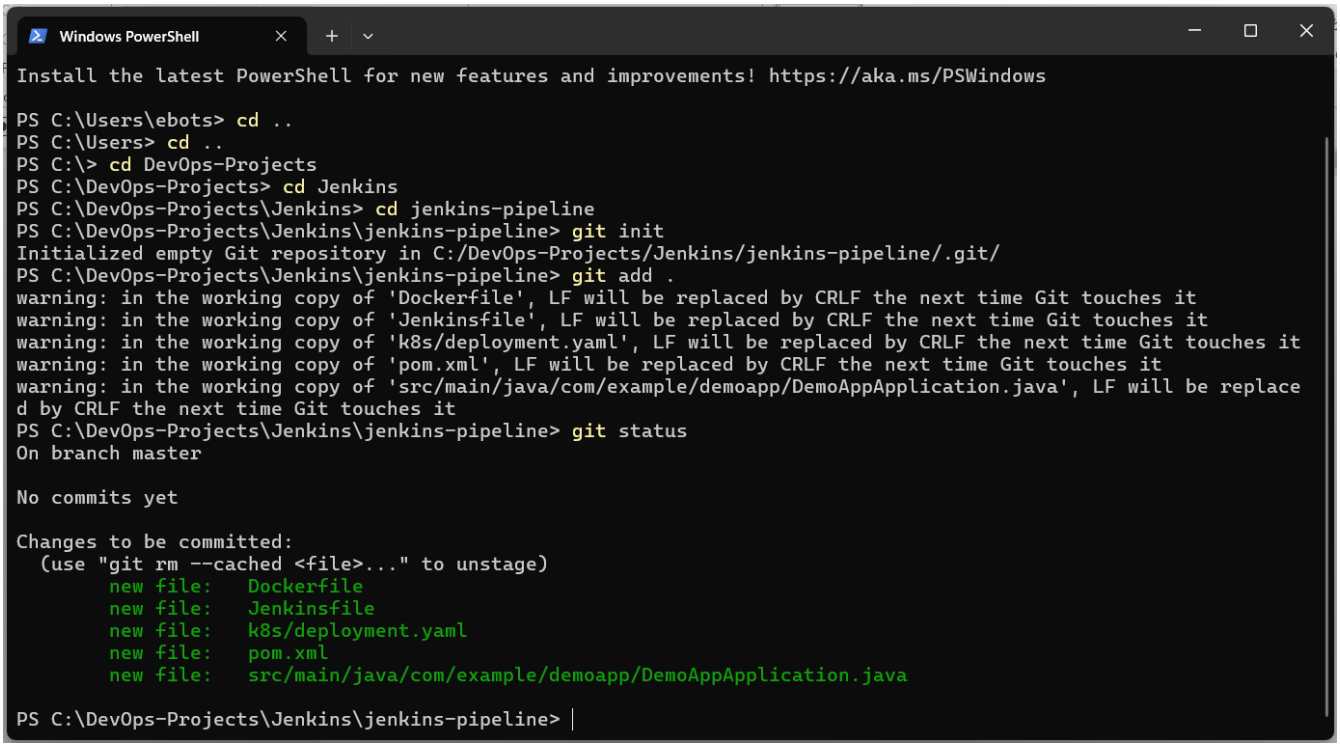
We stage the files by using the command:

```
git add .
```



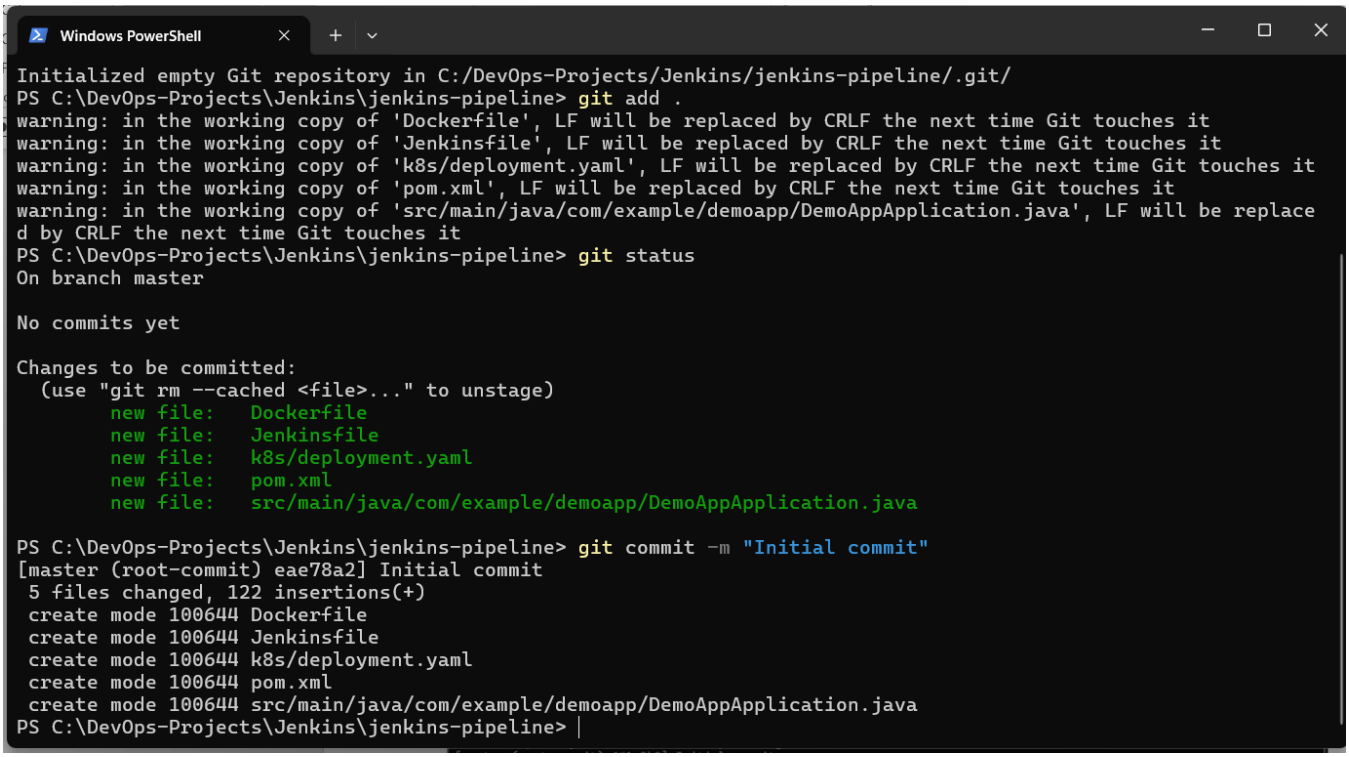
We check the status of the files by using the command:

```
git status
```



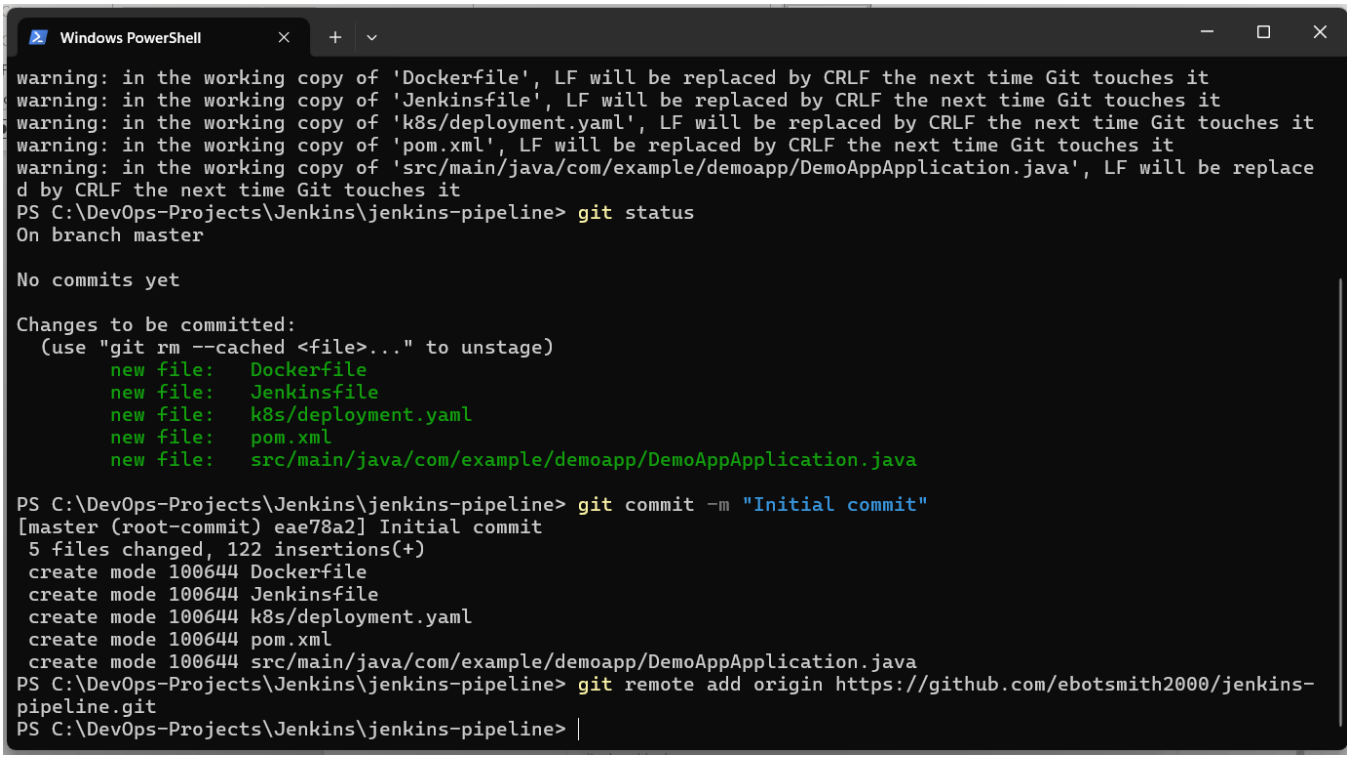
You can see the files that have been staged but not committed. Let us now commit the files by using the command:

```
git commit -m "Initial commit"
```



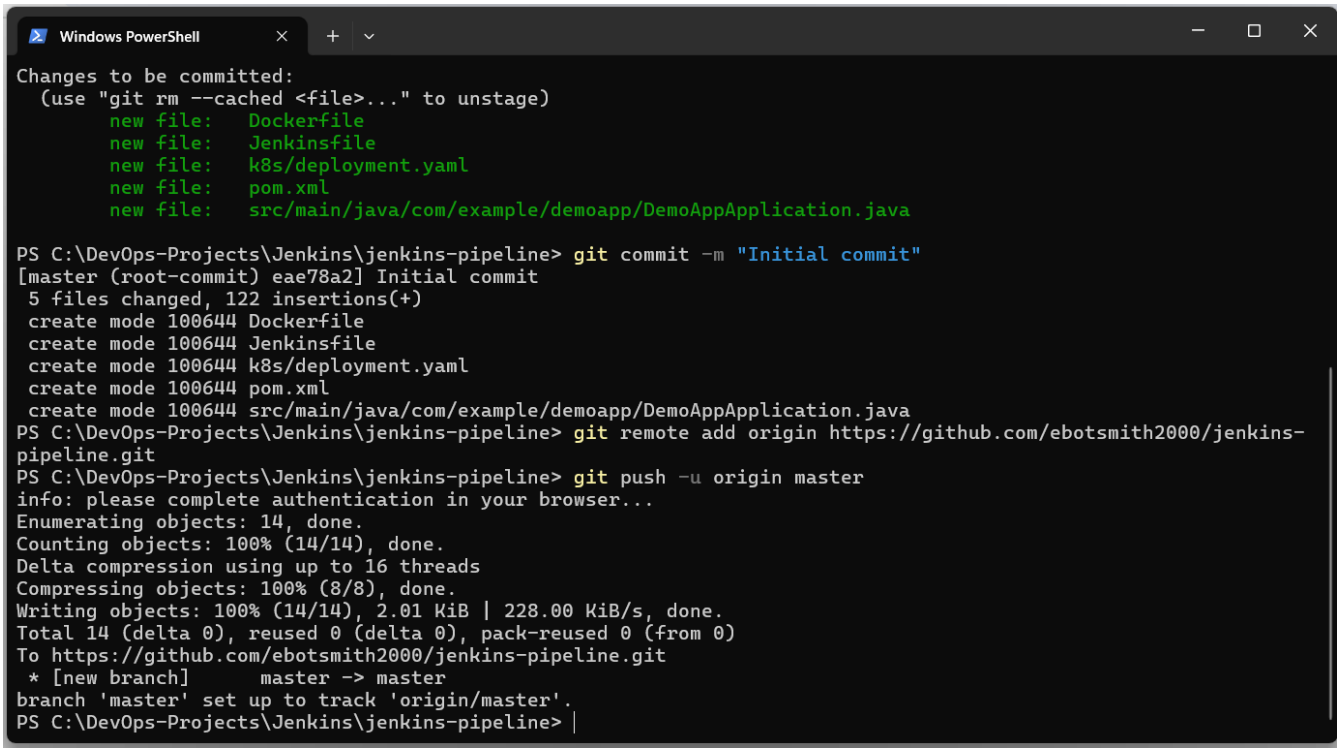
Let us add the files to our remote repository by using the command:

```
git remote add origin https://github.com/ebotsmith2000/jenkins-pipeline.git
```

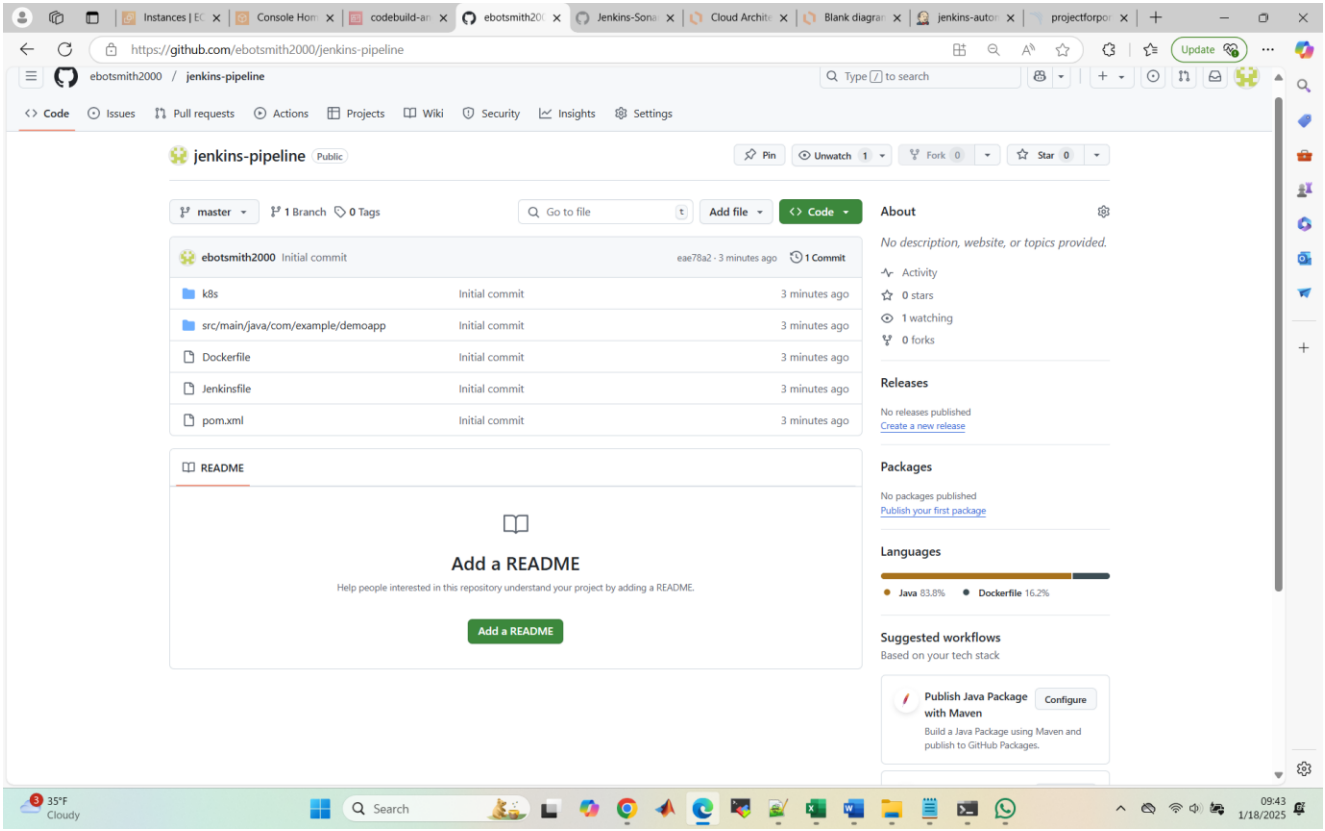


Let us push the files to our master branch in the GitHub repository by using the command:

`git push -u origin master`



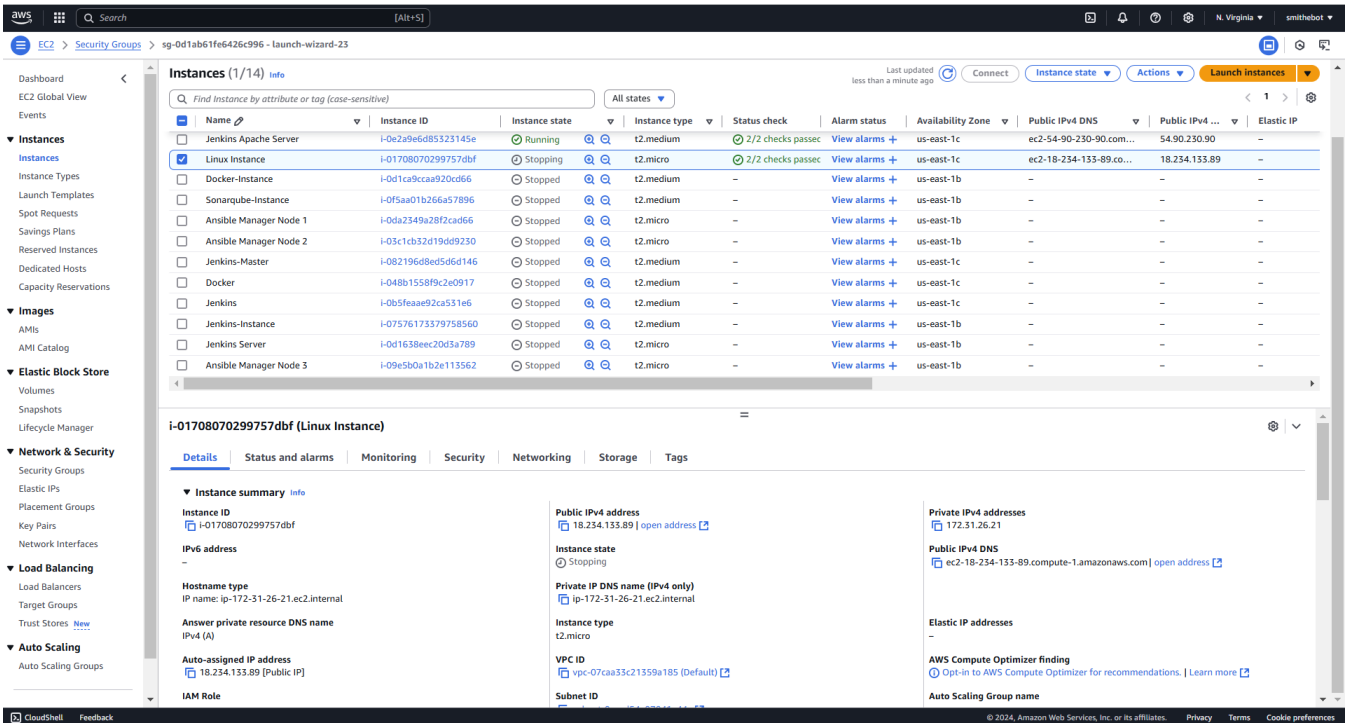
Refresh the GitHub page



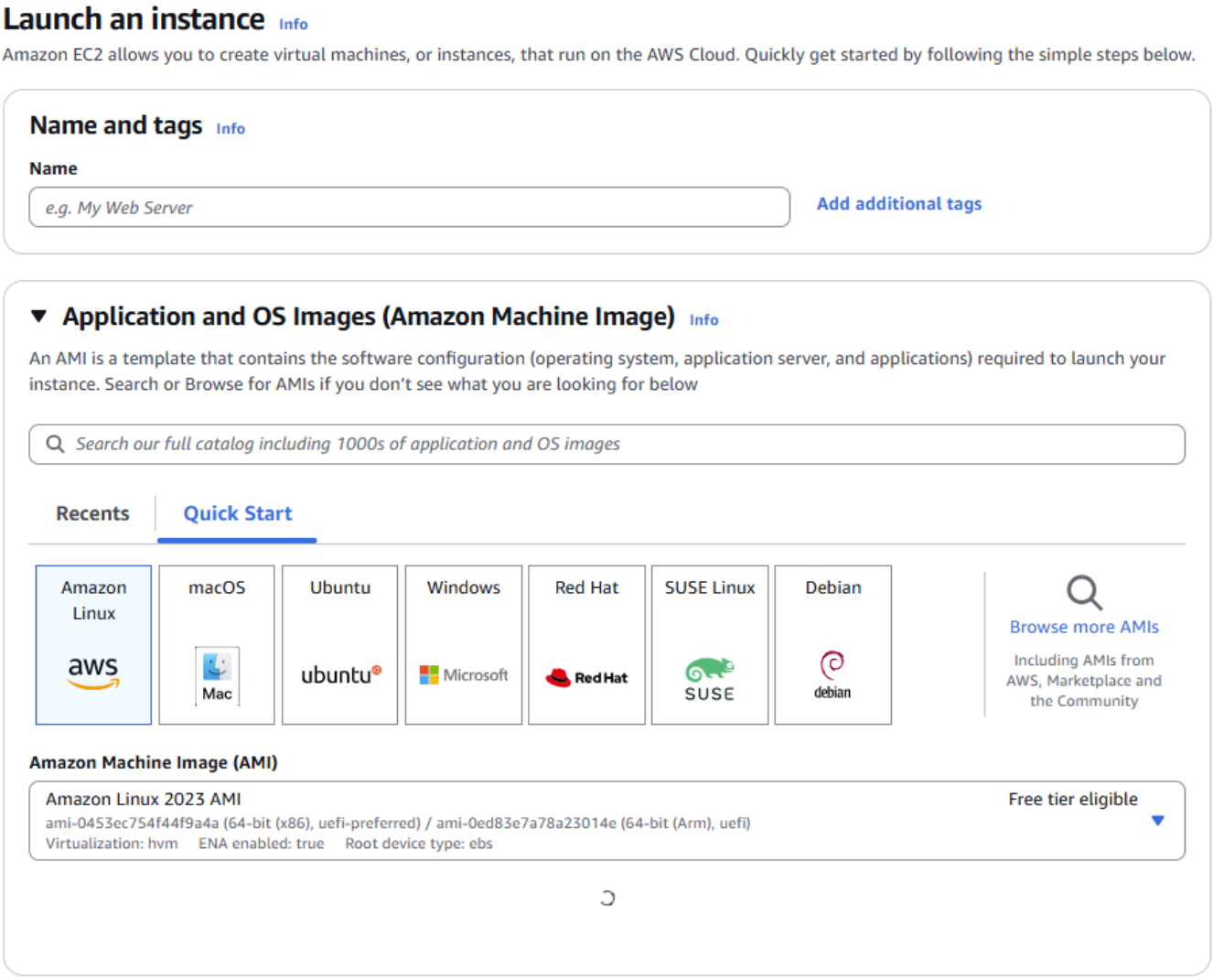
You can see that we have uploaded our project file from the local machine to the GitHub repository.

STEP 2: Create EC2 instance that will serve as Jenkins Server

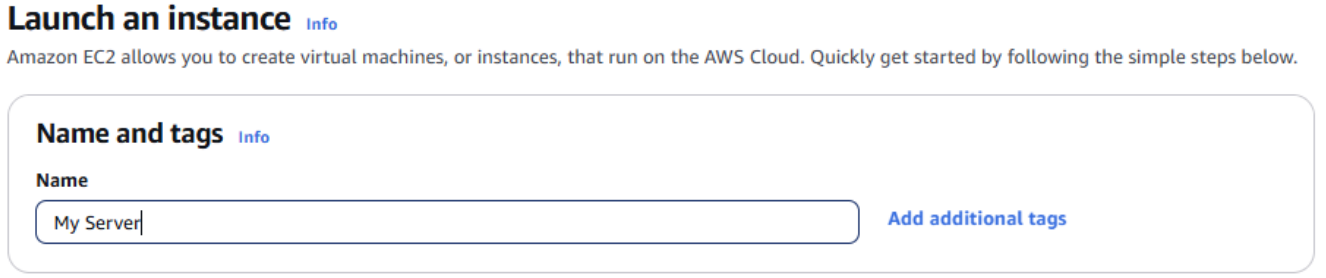
We will create an AWS Ubuntu Instance. Go to EC2 Dashboard



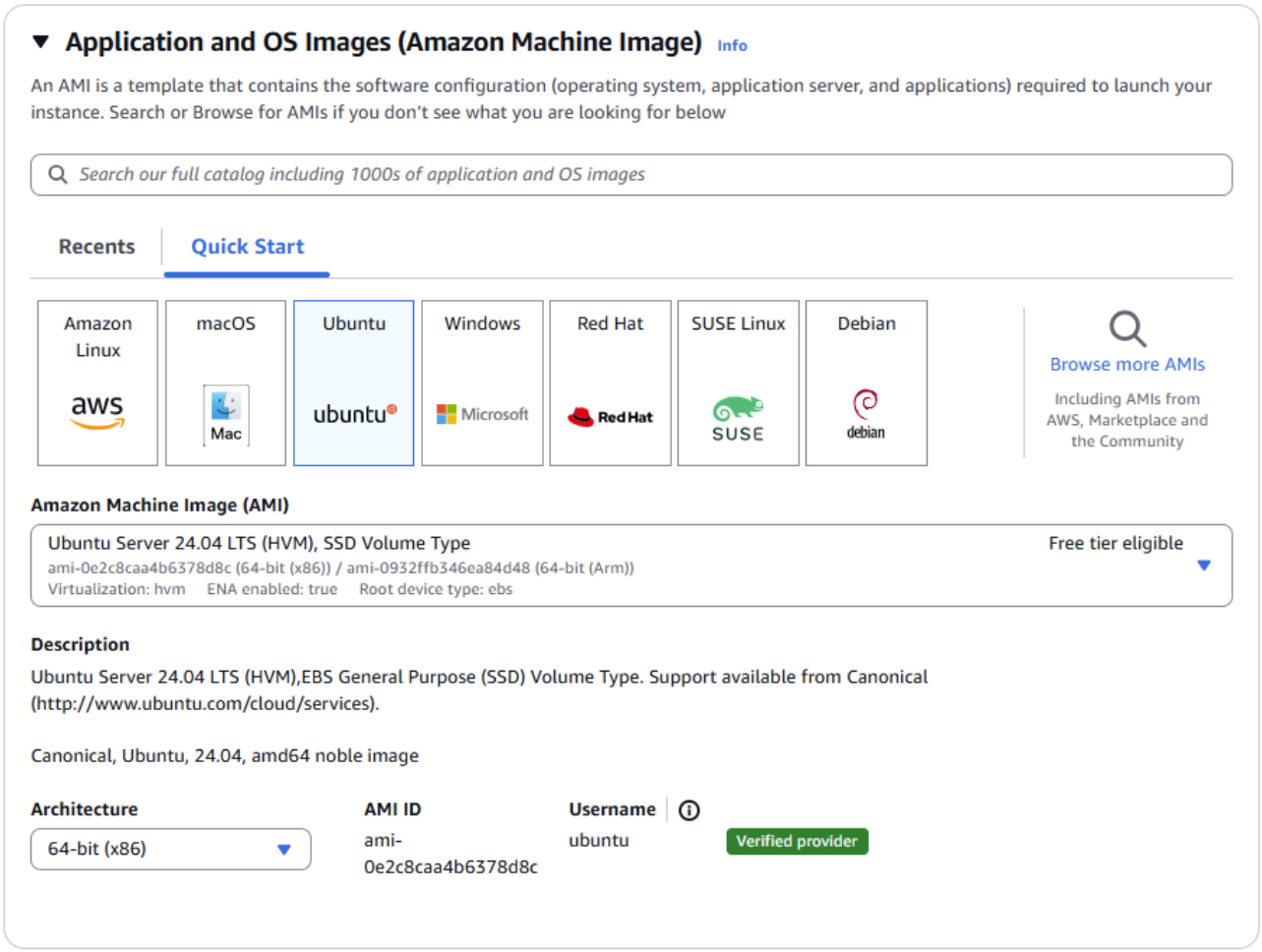
Click on “**Launch Instance**”



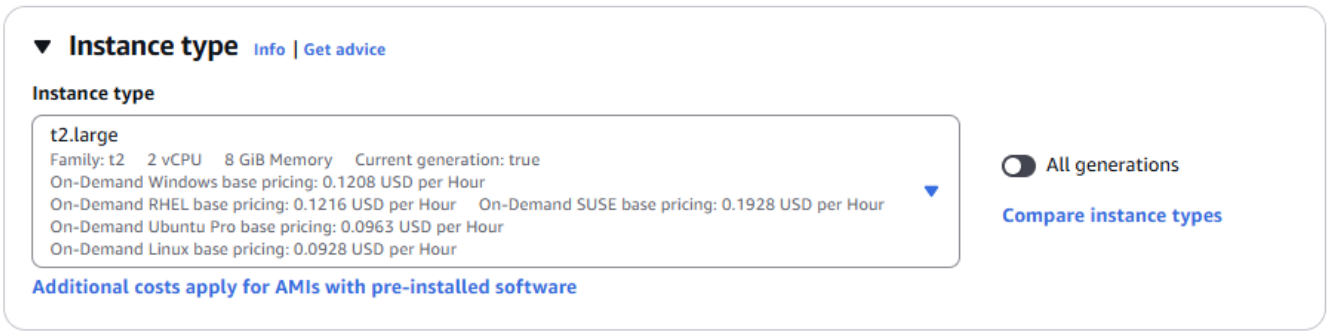
Let us give the instance the name “**My Server**”



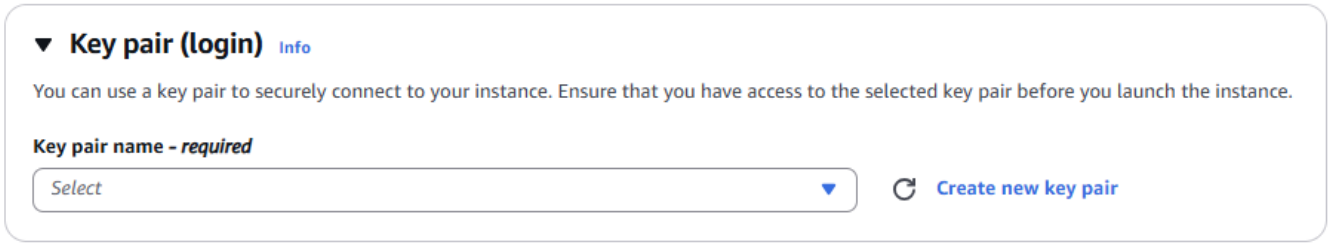
Scroll down to “Application and OS Images (Amazon Machine Images)” and select “ubuntu”



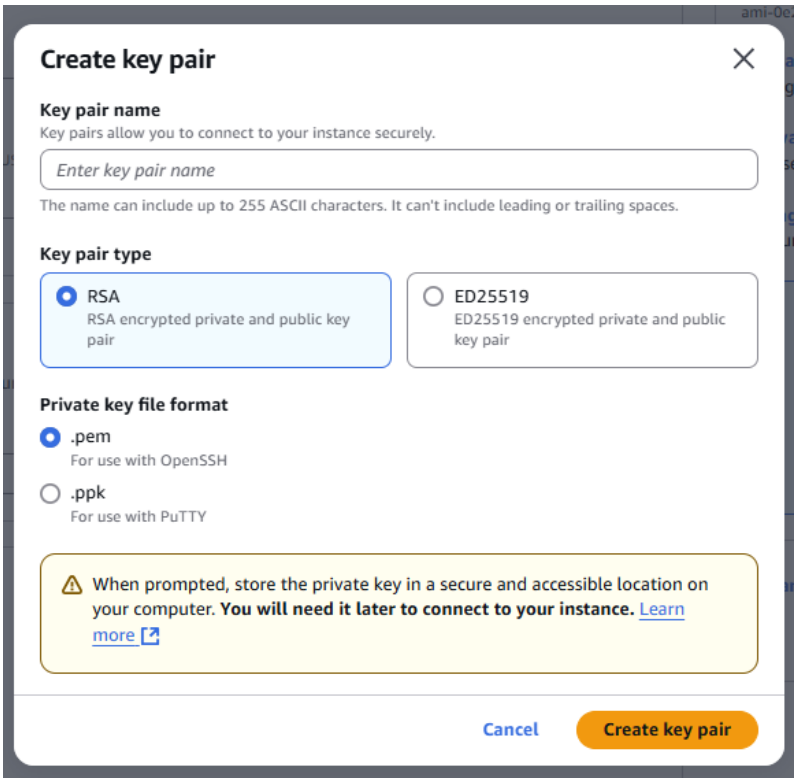
Scroll down to "**Instance Type**" and select "**t2.large**"



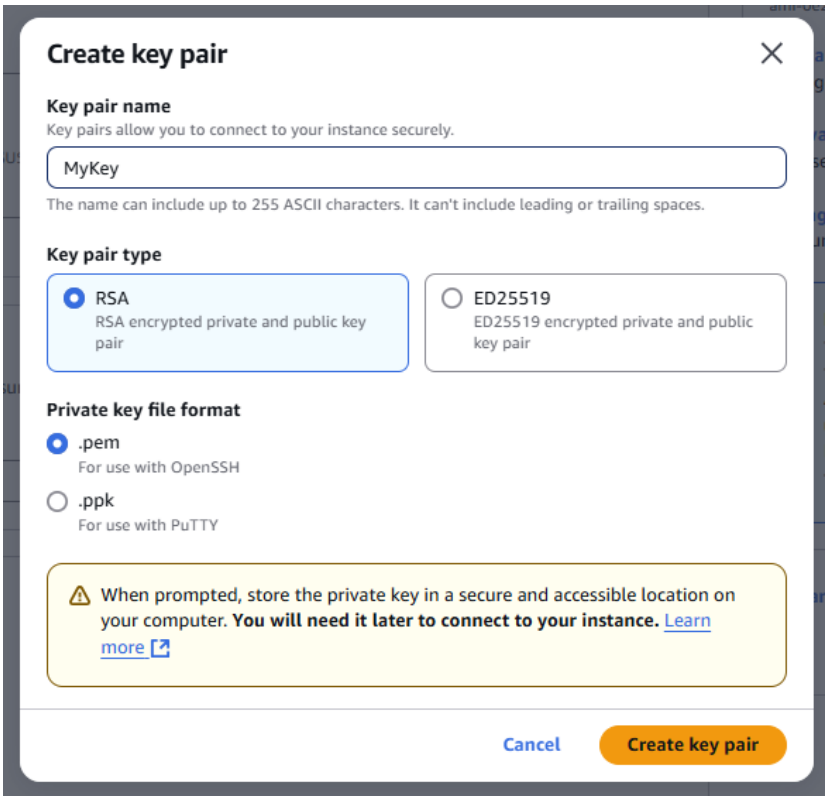
Scroll down to "Key Pair"



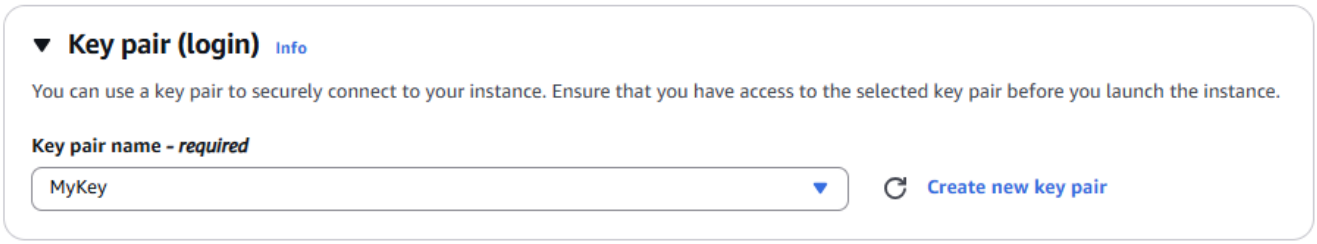
Click on “**Create new key Pair**”



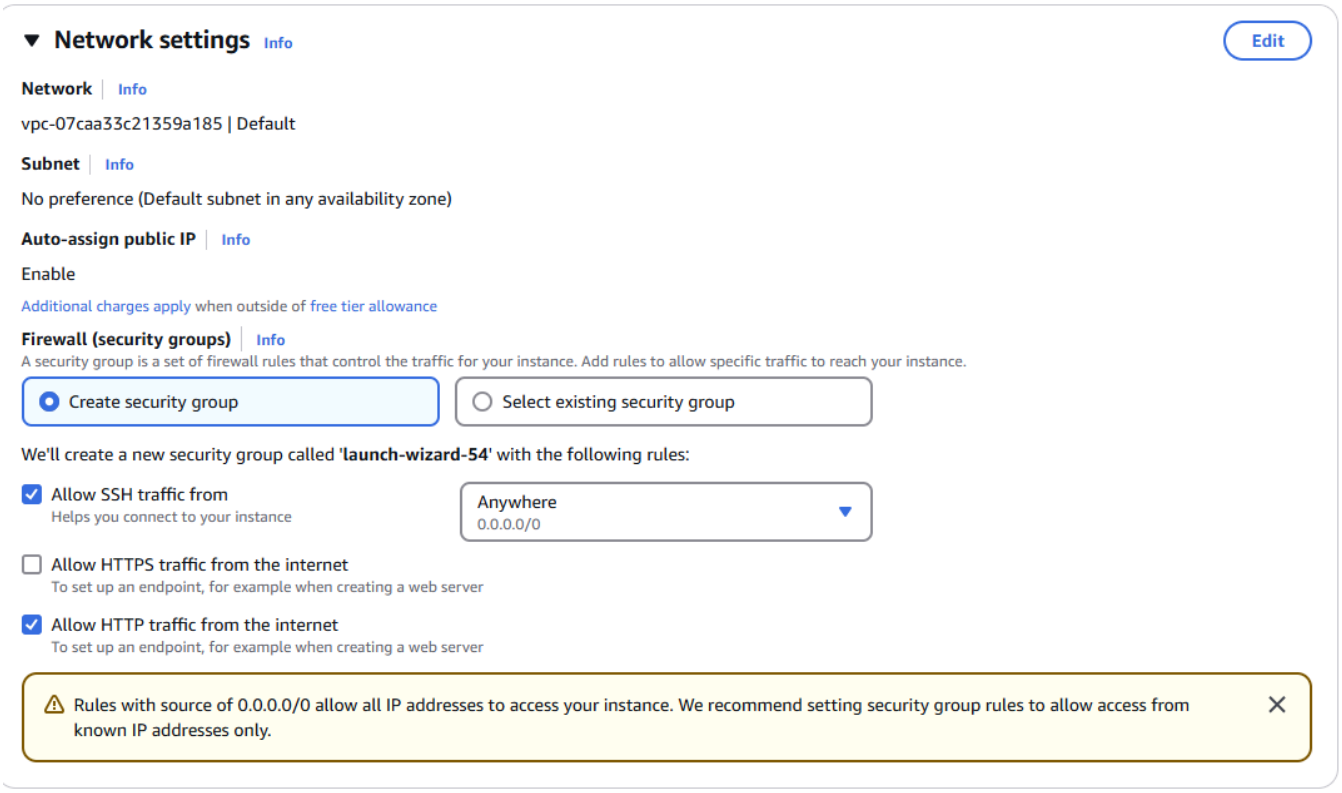
I will use “**MyKey**” for the Key Pair Name”



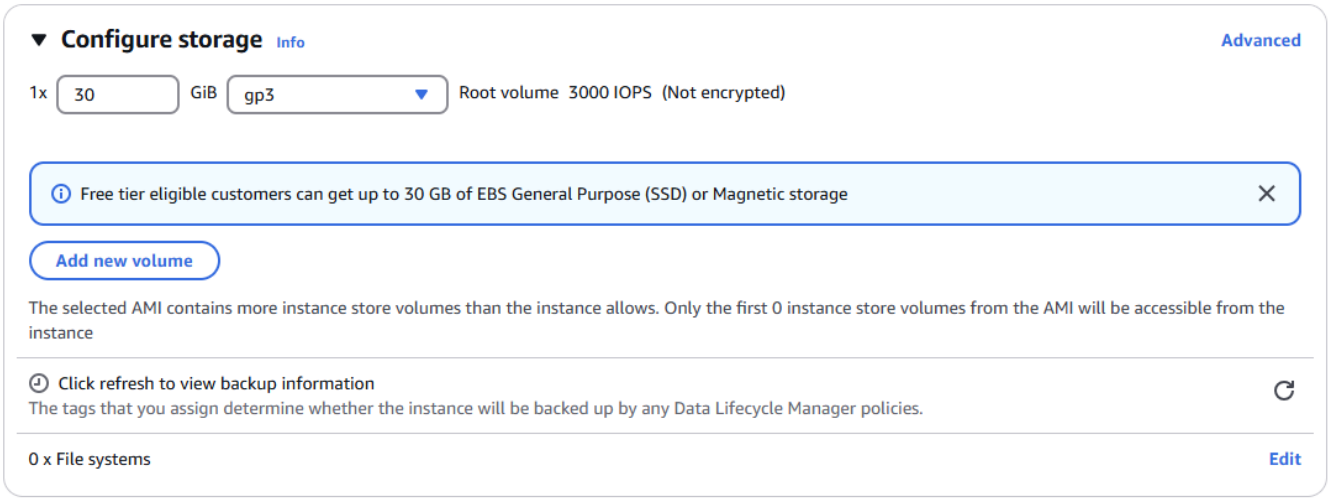
Click on “**Create Key Pair**”



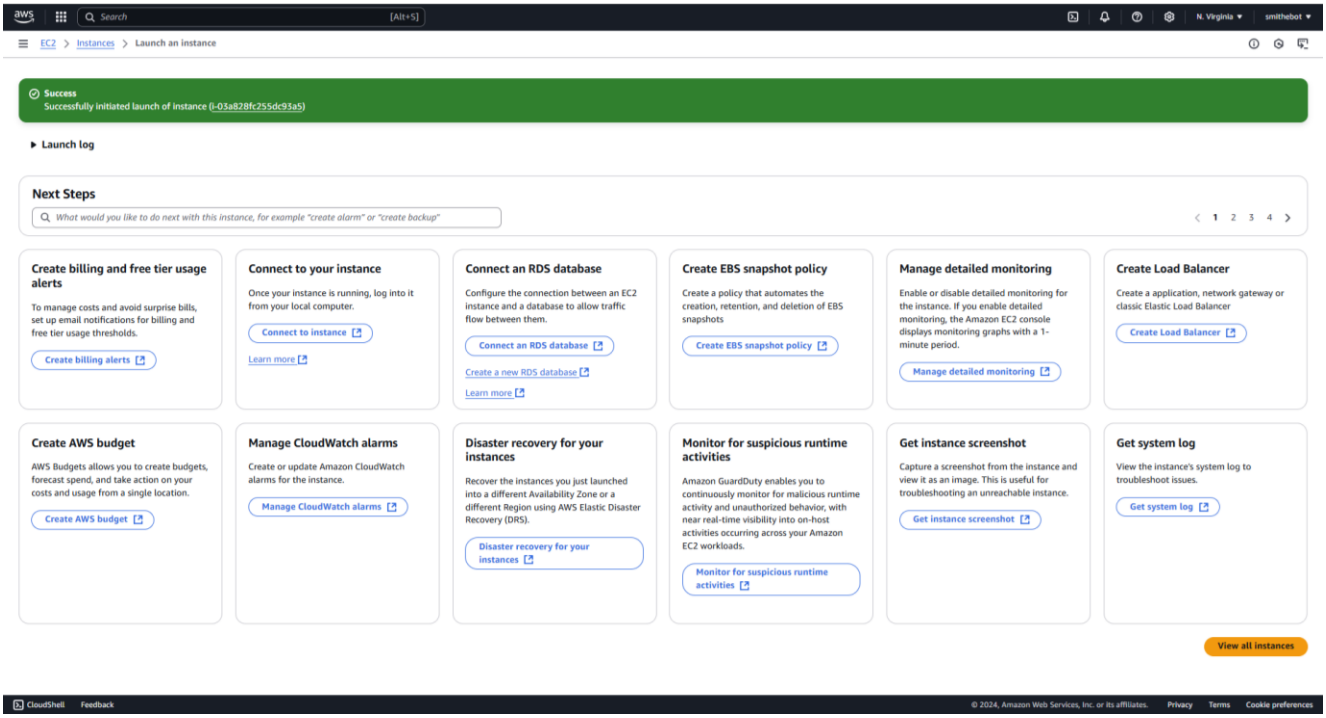
Scroll down to "**Network Settings**"



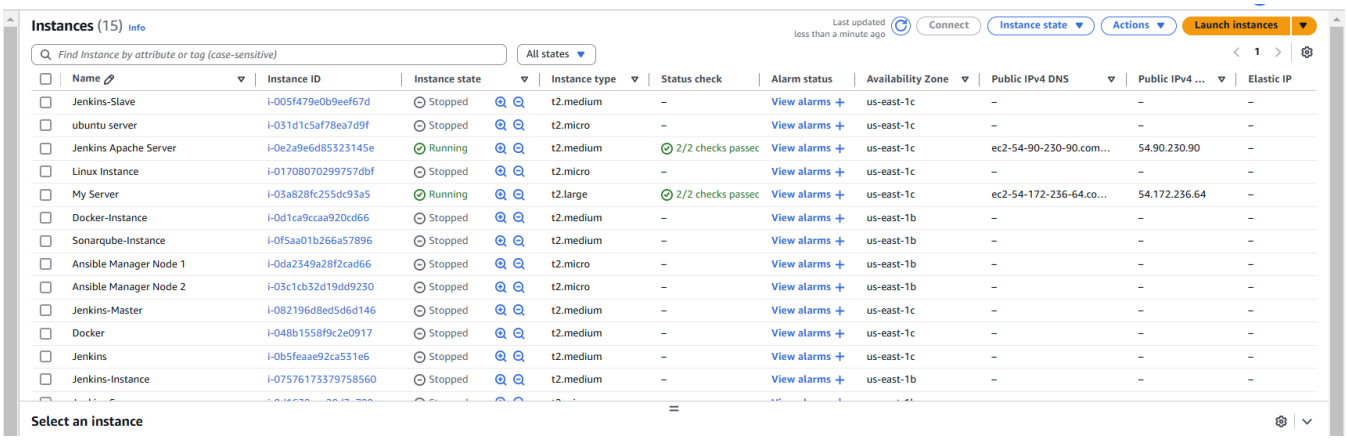
Scroll down to "**Configure Storage**" and make the size **30GiB**



Click on "**Launch Instance**"

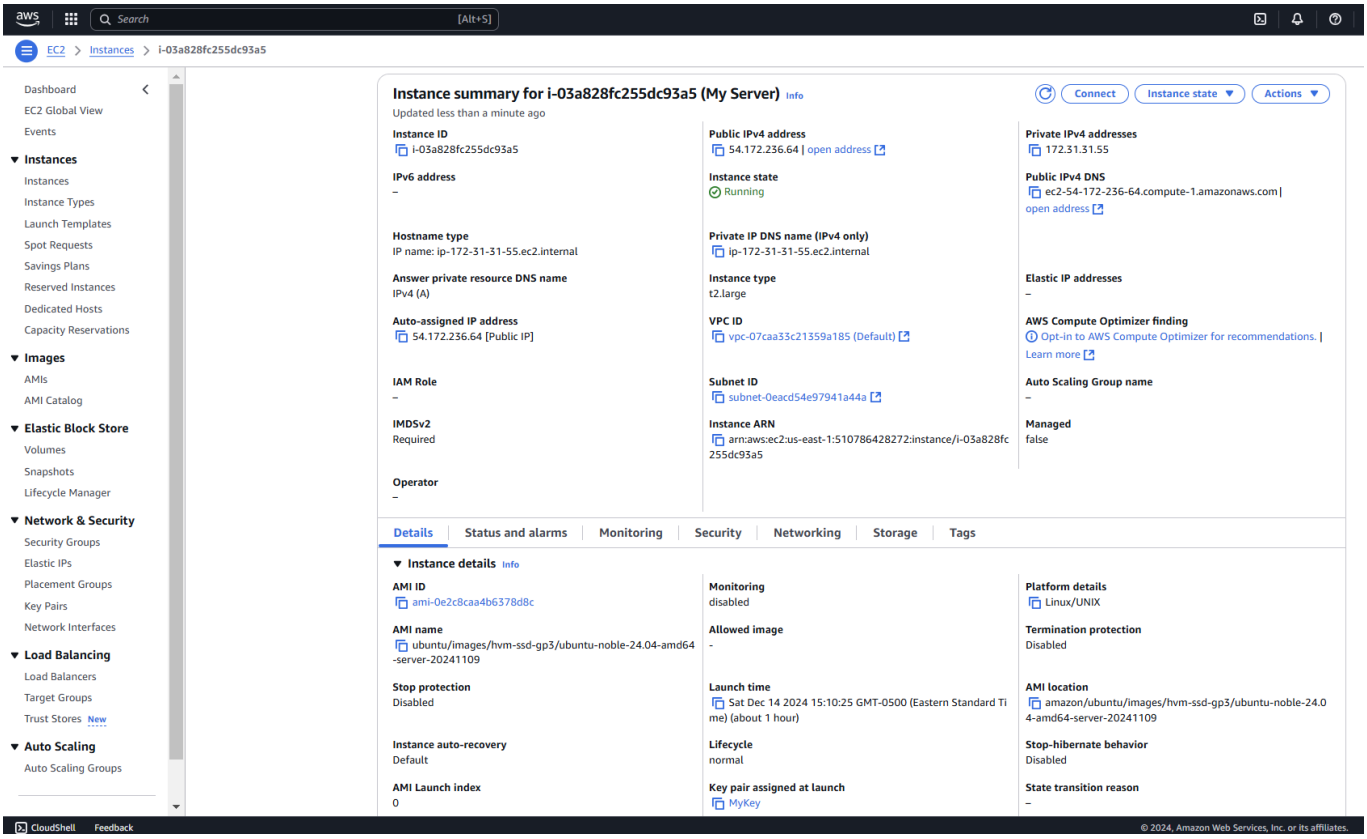


Click on "**View All Instances**"

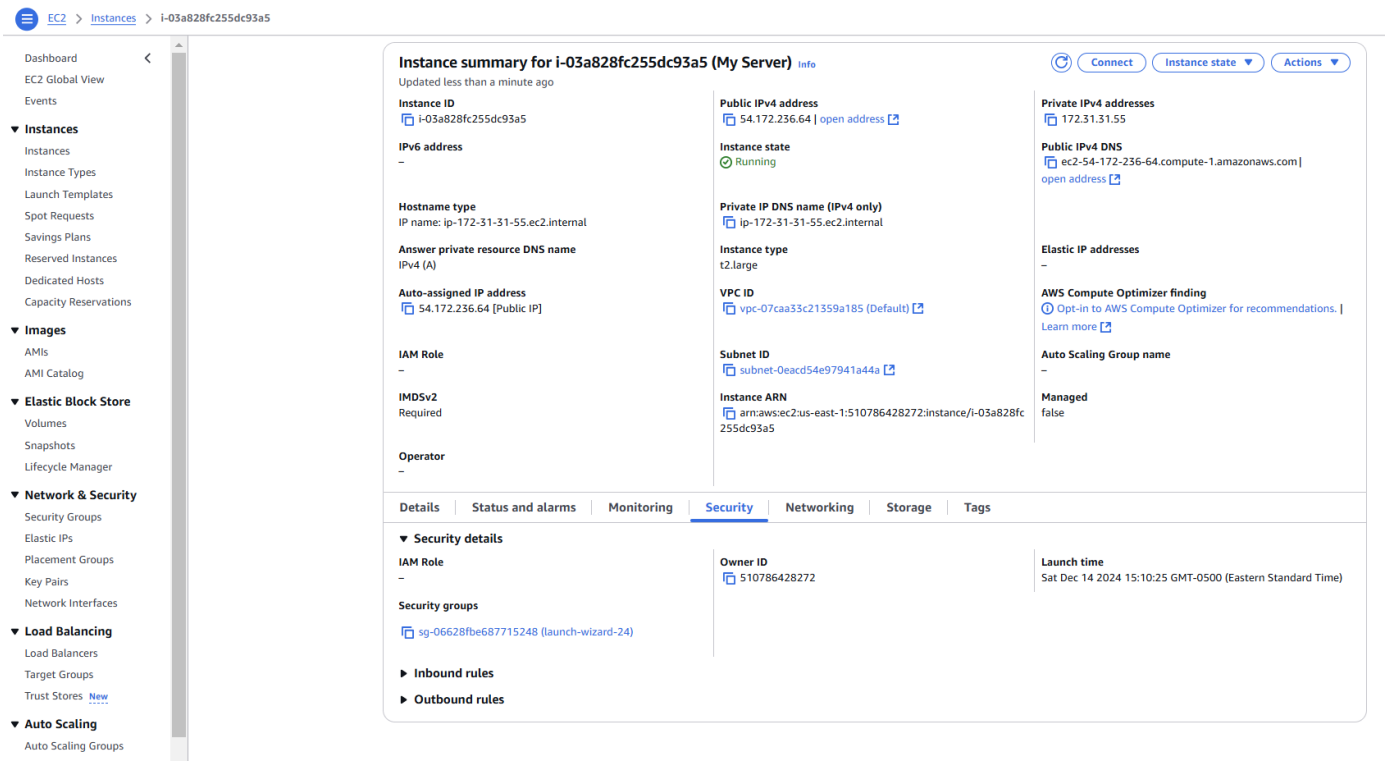


STEP 3: Add Port 8080 in the inbound rule of the Instance for Jenkins

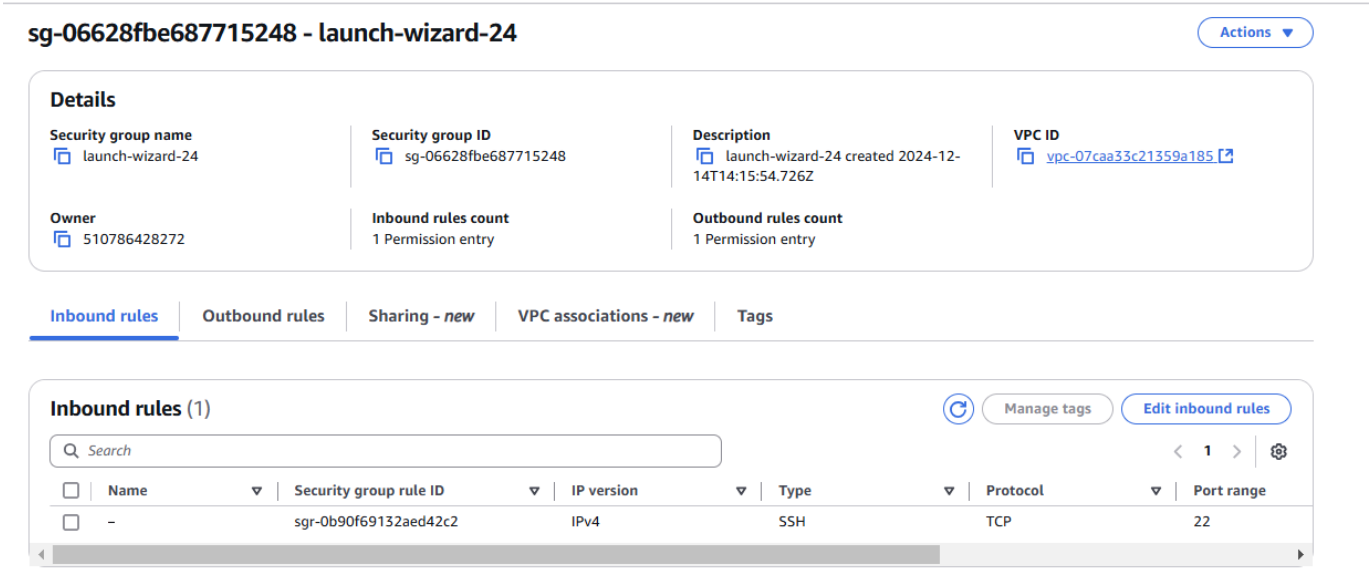
Click on the "**Instance ID**" of the instance we just created



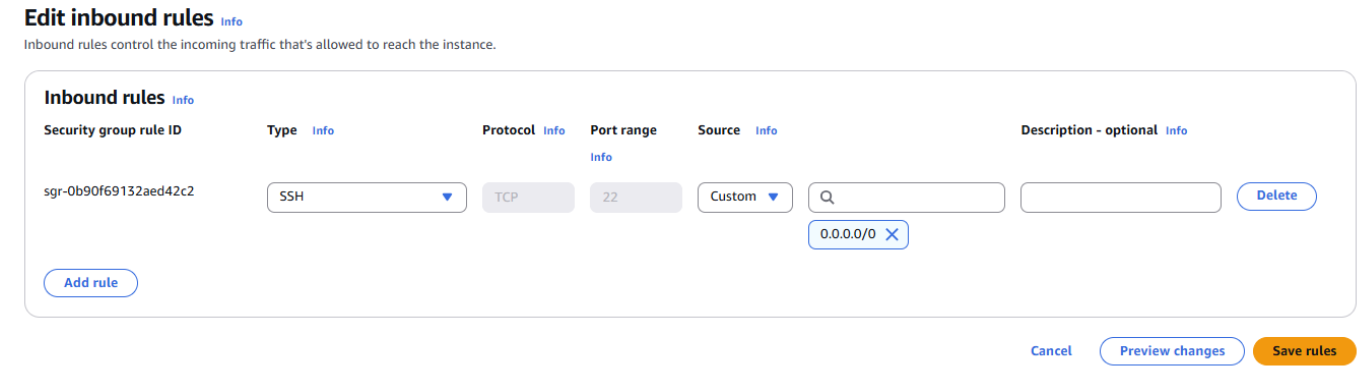
Click on "**Security**" tab



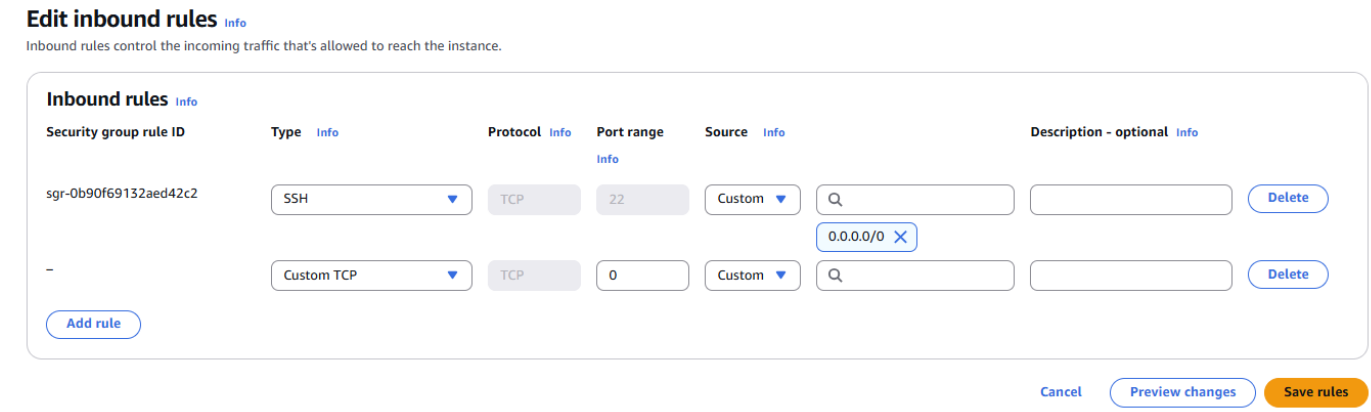
Click on “**Security Group**”



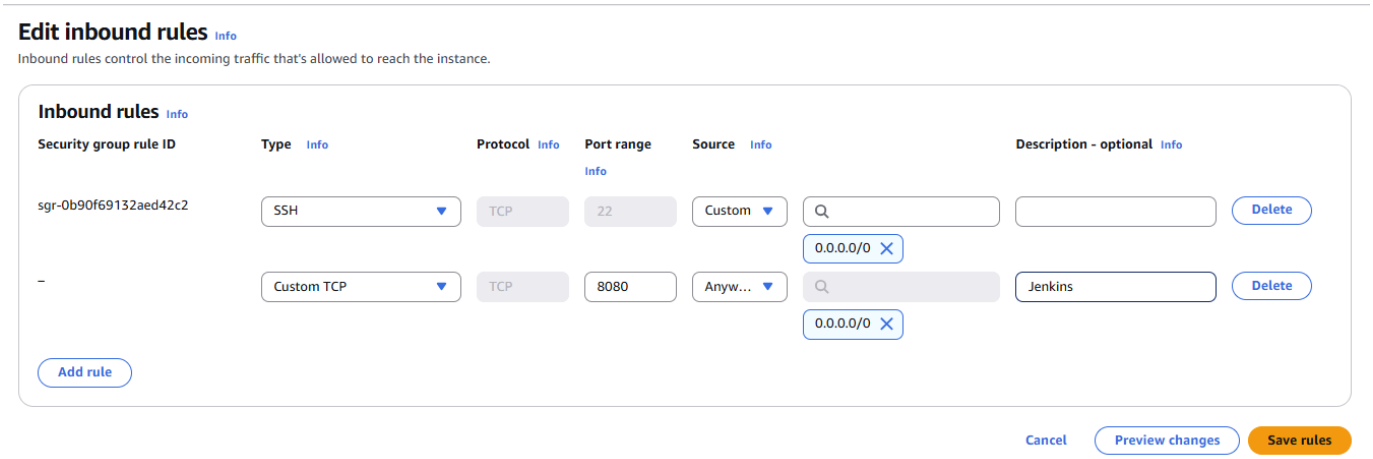
Click on “**Edit Inbound Rules**”



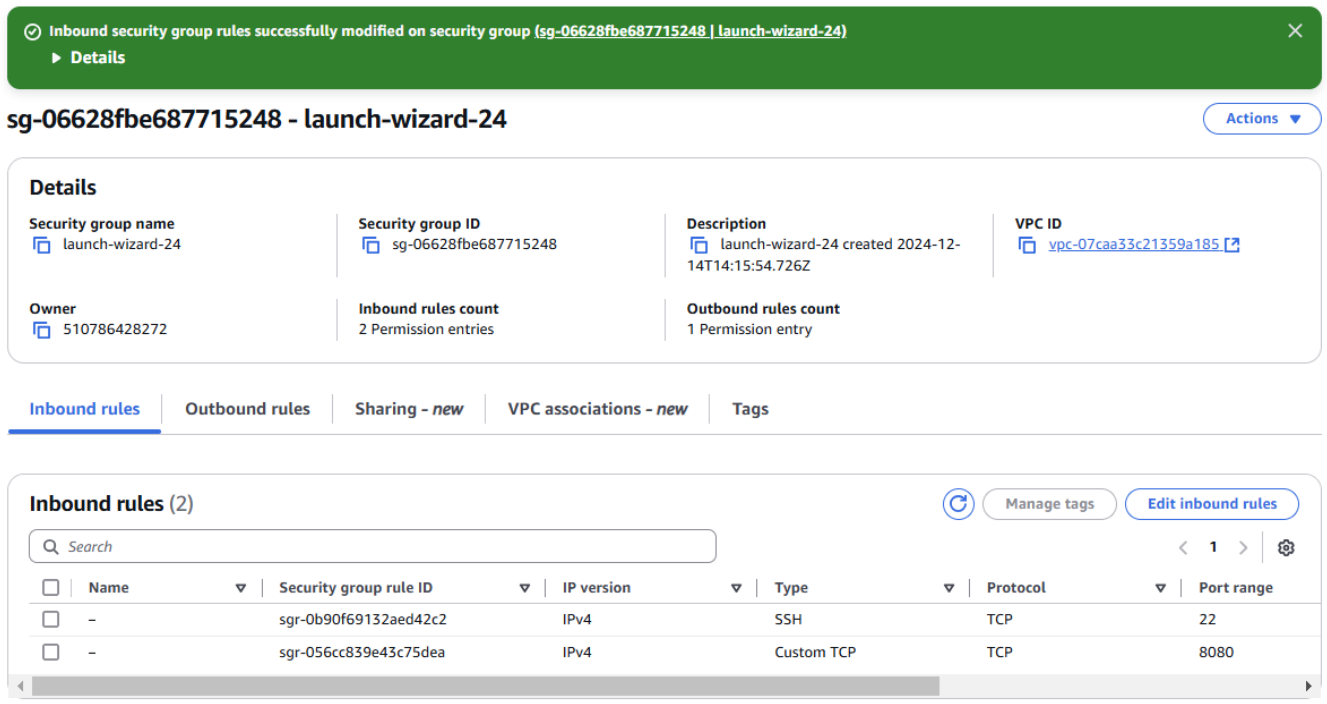
Click on “**Add Rule**”



Add Port 8080



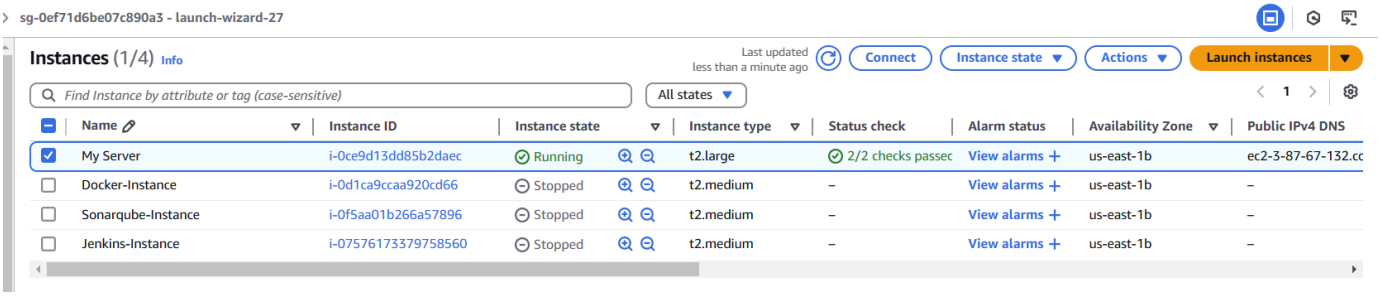
Click on "**Save Rules**"



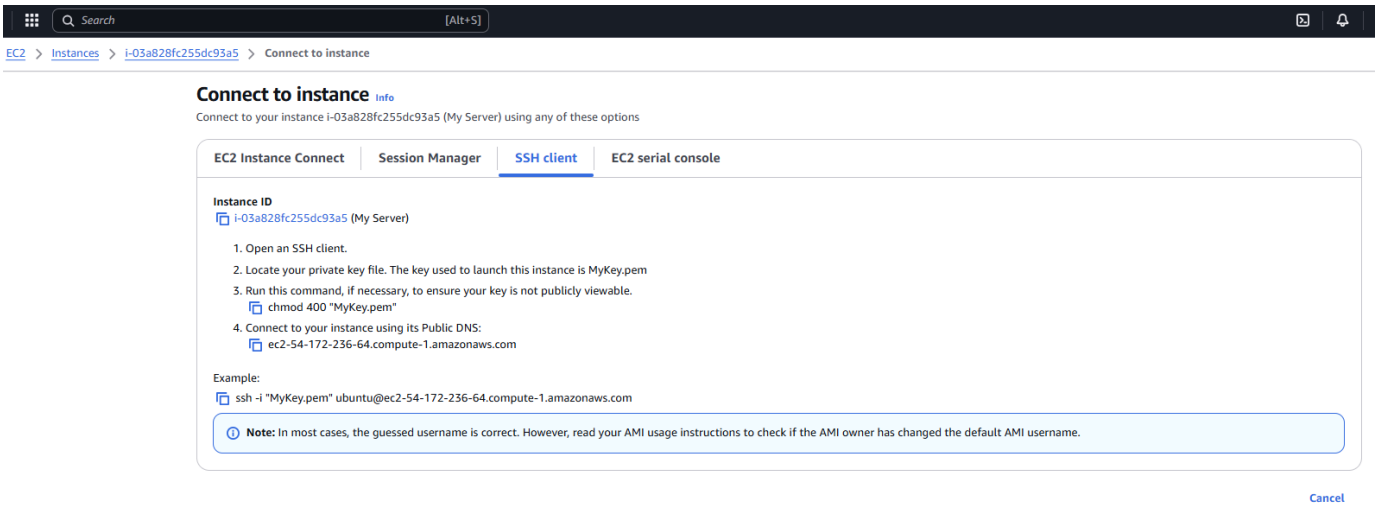
Port 8080 has been added for Jenkins

STEP 4: Connect to the server

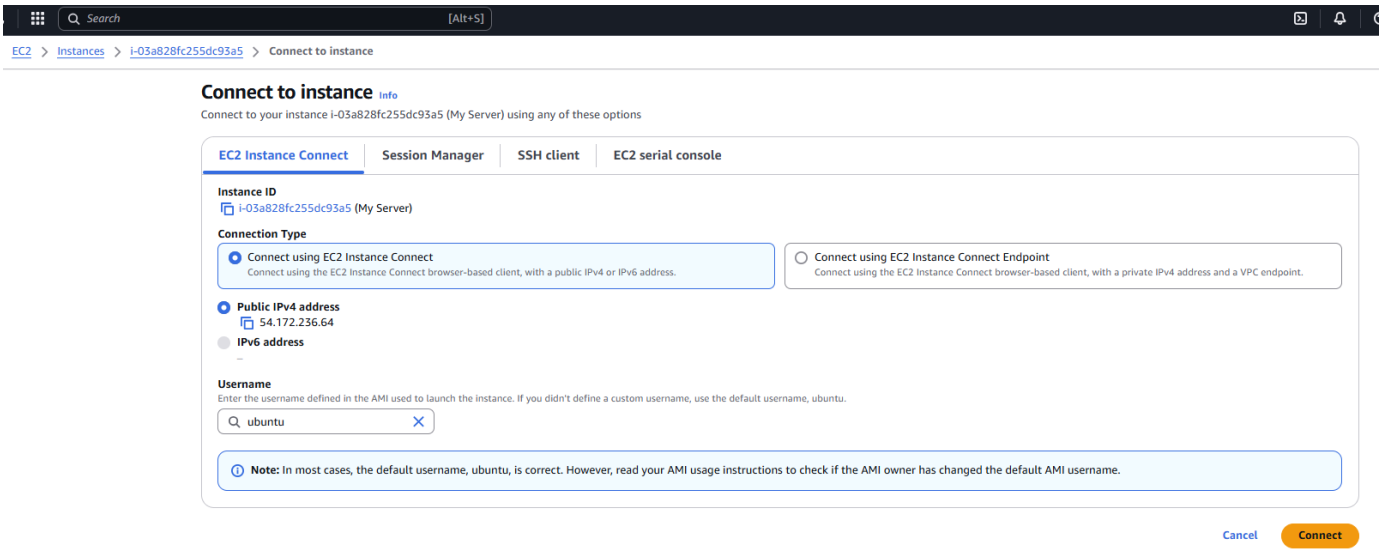
You can see the instance we just created. Select the instance



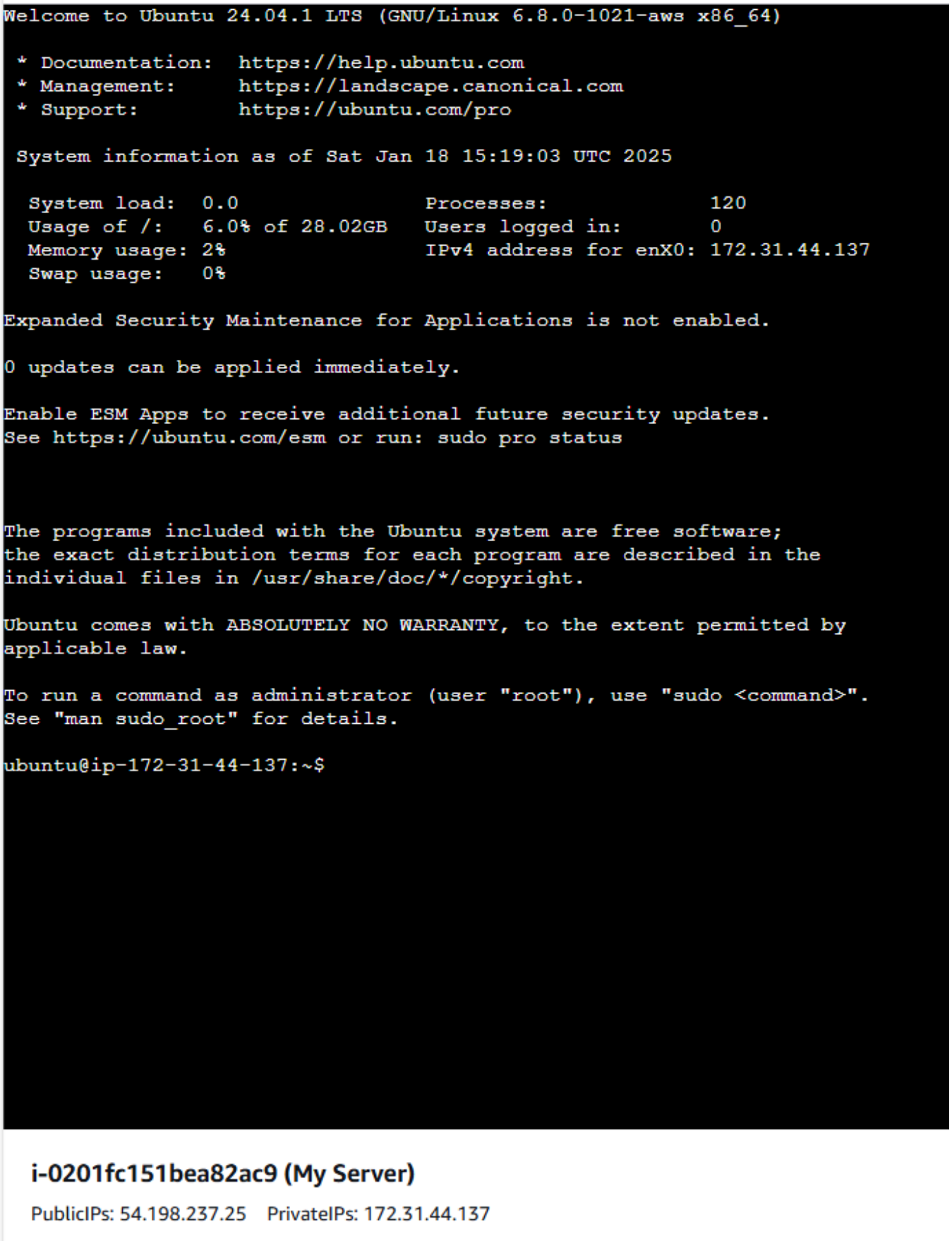
and click on “**Connect**”



Select the “**EC2 Instance Connect**” tab



Click on “**Connect**”



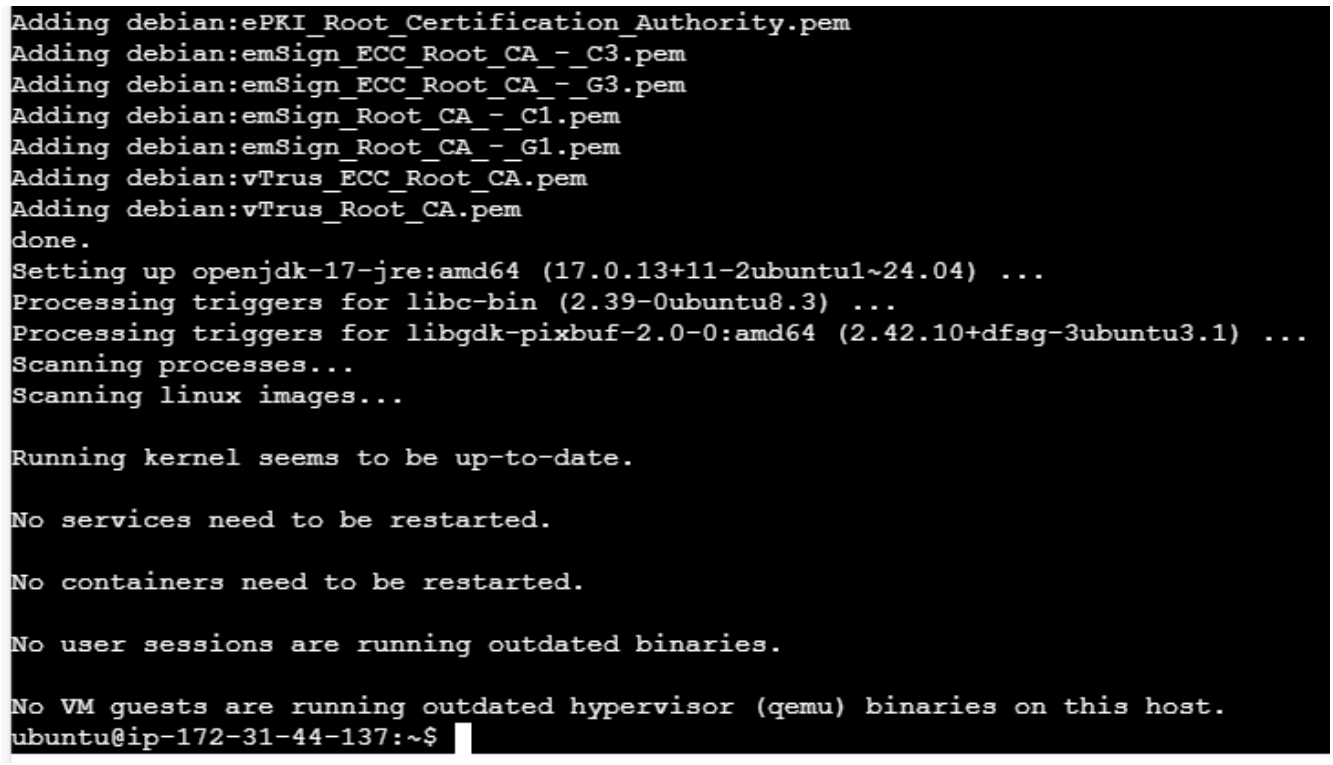
You can see that we are connected to the server.

STEP 5: Install Java JDK 17

We will run these commands:

```
sudo apt update
sudo apt install openjdk-17-jre
```

Type “Y” and press ENTER



Verify Java is Installed by running the command:

```
java -version
```

```
ubuntu@ip-172-31-44-137:~$ java -version
openjdk version "17.0.13" 2024-10-15
OpenJDK Runtime Environment (build 17.0.13+11-Ubuntu-2ubuntu124.04)
OpenJDK 64-Bit Server VM (build 17.0.13+11-Ubuntu-2ubuntu124.04, mixed mode, sharing)
ubuntu@ip-172-31-44-137:~$
```

STEP 6: Install Jenkins

Now, let us proceed to install Jenkins, Run the command:

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

```
ubuntu@ip-172-31-44-137:~$ curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Ign:2 https://pkg.jenkins.io/debian binary/ InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Get:5 https://pkg.jenkins.io/debian binary/ Release [2044 B]
Get:6 https://pkg.jenkins.io/debian binary/ Release.gpg [833 B]
Hit:7 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:8 https://pkg.jenkins.io/debian binary/ Packages [67.1 kB]
Fetched 70.0 kB in 0s (170 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  net-tools
The following NEW packages will be installed:
  jenkins net-tools
0 upgraded, 2 newly installed, 0 to remove and 1 not upgraded.
Need to get 94.1 MB of archives.
After this operation, 96.5 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Type "**Y**" and press ENTER

```
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Ign:2 https://pkg.jenkins.io/debian binary/ InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Get:5 https://pkg.jenkins.io/debian binary/ Release [2044 B]
Get:6 https://pkg.jenkins.io/debian binary/ Release.gpg [833 B]
Hit:7 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:8 https://pkg.jenkins.io/debian binary/ Packages [67.1 kB]
Fetched 70.0 kB in 0s (170 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  net-tools
The following NEW packages will be installed:
  jenkins net-tools
0 upgraded, 2 newly installed, 0 to remove and 1 not upgraded.
Need to get 94.1 MB of archives.
After this operation, 96.5 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 net-tools amd64 2.10-0.1ubuntu4 [204 kB]
Get:2 https://pkg.jenkins.io/debian binary/ jenkins 2.493 [93.9 MB]
Fetched 94.1 MB in 1min 5s (1441 kB/s)
Selecting previously unselected package net-tools.
(Reading database ... 85305 files and directories currently installed.)
Preparing to unpack .../net-tools_2.10-0.1ubuntu4_amd64.deb ...
Unpacking net-tools (2.10-0.1ubuntu4) ...
Selecting previously unselected package jenkins.
Preparing to unpack .../archives/jenkins_2.493_all.deb ...
Unpacking jenkins (2.493) ...
Setting up net-tools (2.10-0.1ubuntu4) ...
Setting up jenkins (2.493) ...
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.
ubuntu@ip-172-31-44-137:~$
```

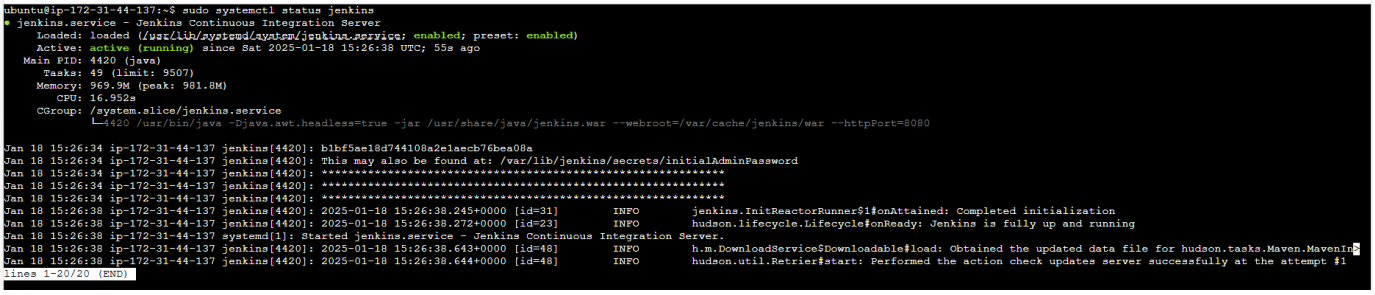
i-0201fc151bea82ac9 (My Server)

PublicIPs: 54.198.237.25 PrivateIPs: 172.31.44.137

CloudShell Feedback

Let us verify if Jenkins is installed or not by using the command:

`sudo systemctl status jenkins`



You can see that Jenkins is active and running. So, we have successfully installed Jenkins.

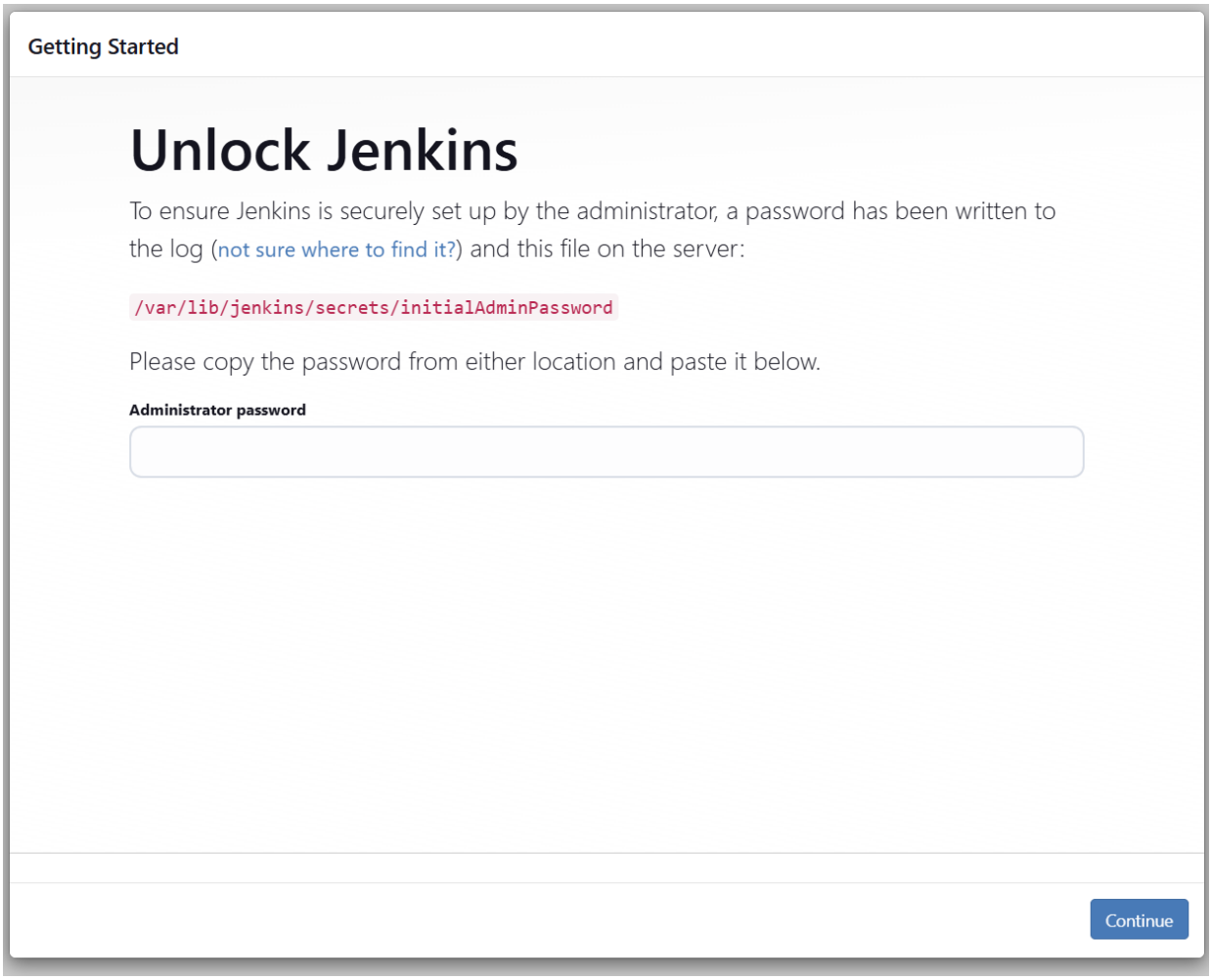
Return to the command mode by typing CTRL+C

STEP 7: Access Jenkins GUI on Browser

Now, let us access Jenkins GUI by using this one the browser:

`http://<Public IPv4 Address>:8080`

That is `http://54.198.237.25:8080`

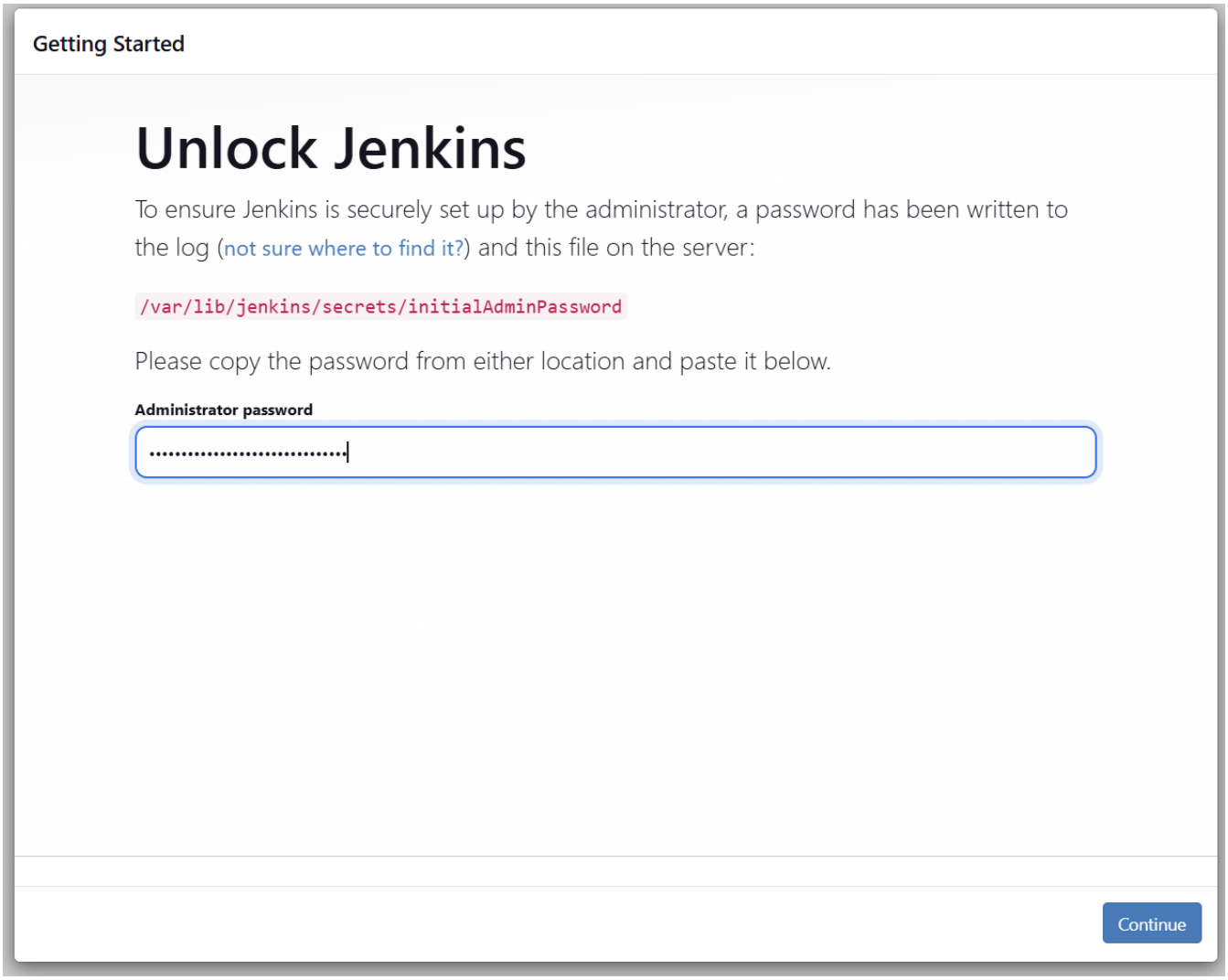


To get the password you have to run the command:

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

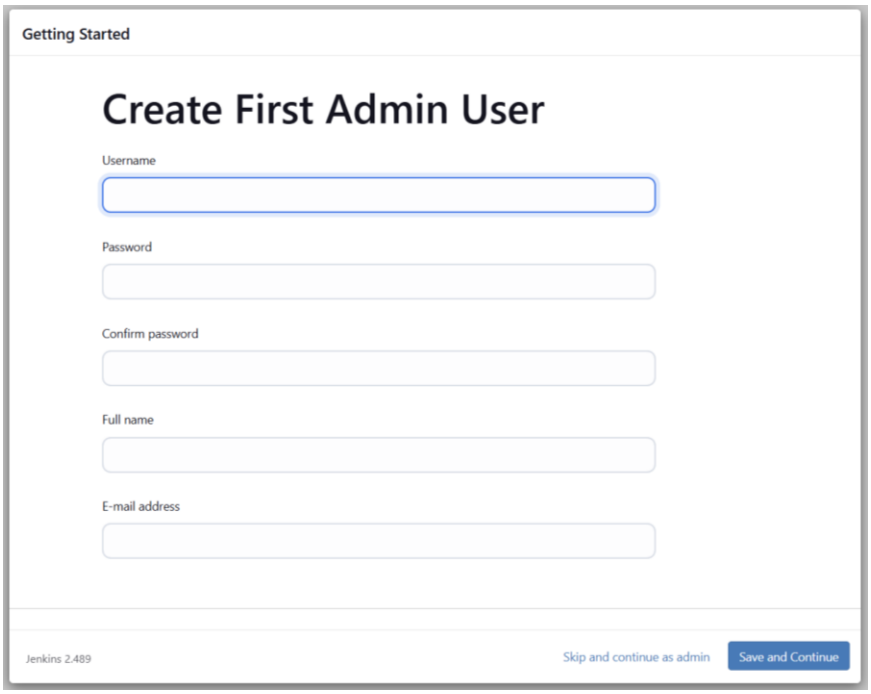
```
ubuntu@ip-172-31-31-55:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
fda0aaa5049a49e6aa453c6645e3c2e8
ubuntu@ip-172-31-31-55:~$
```

The password is `fda0aaa5049a49e6aa453c6645e3c2e8`

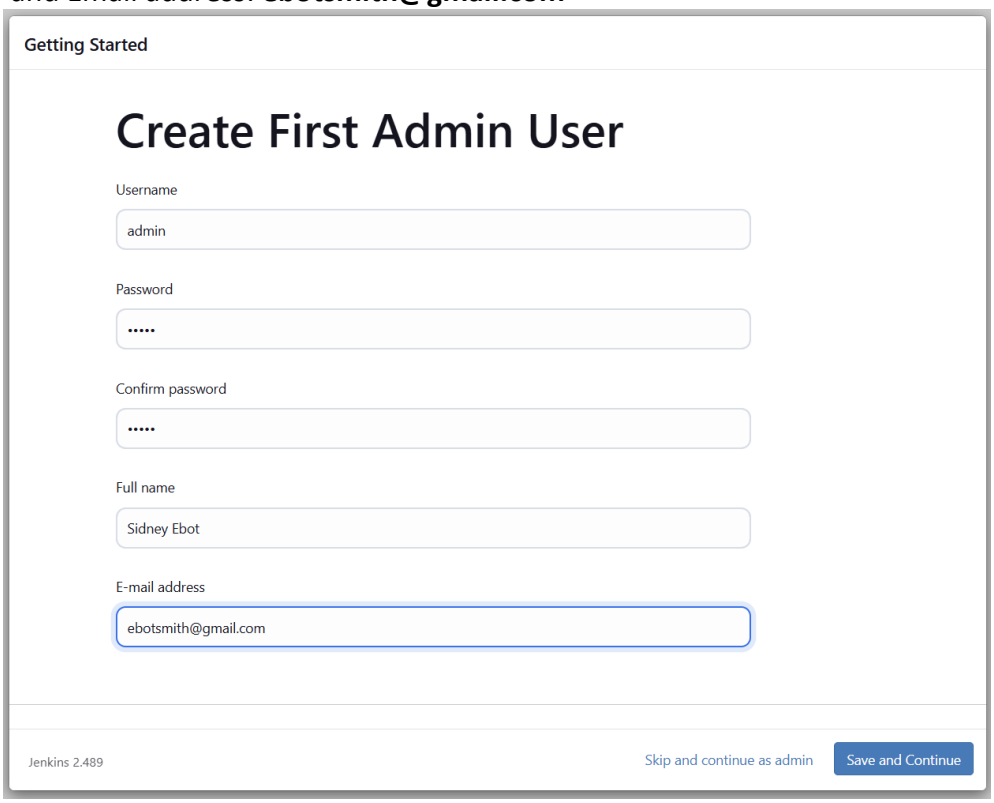


Click on “Continue”

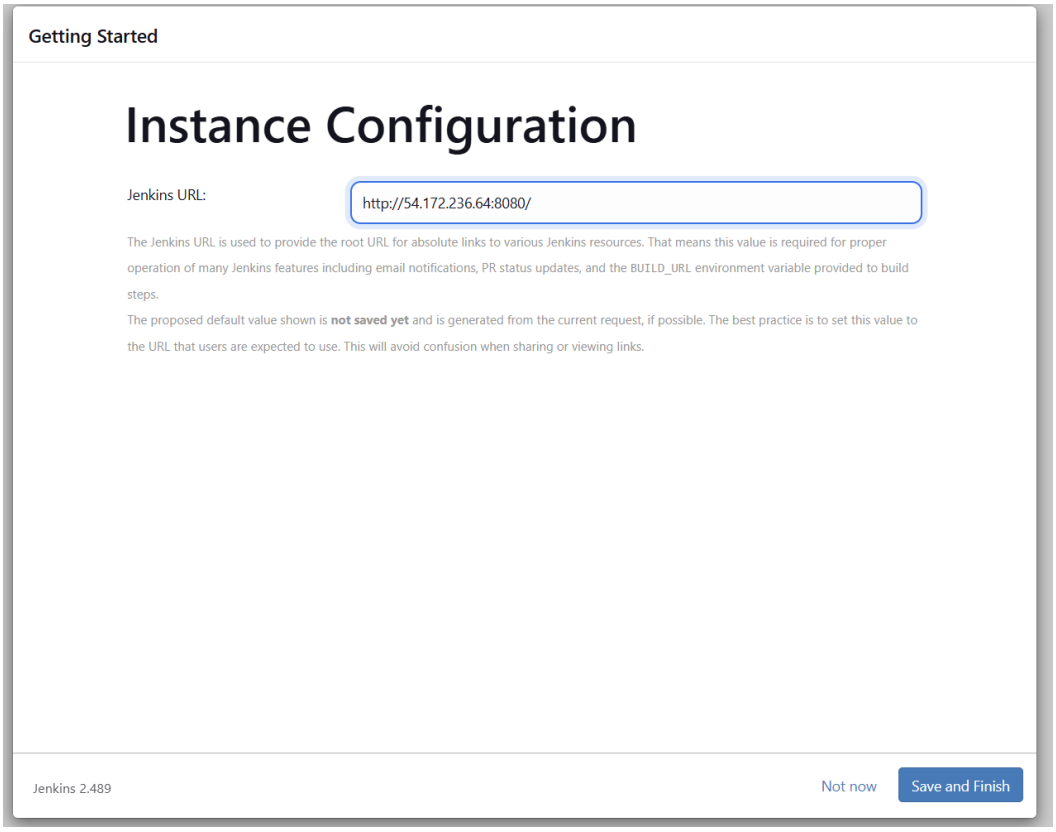
Click on “Install Suggested Plugins”



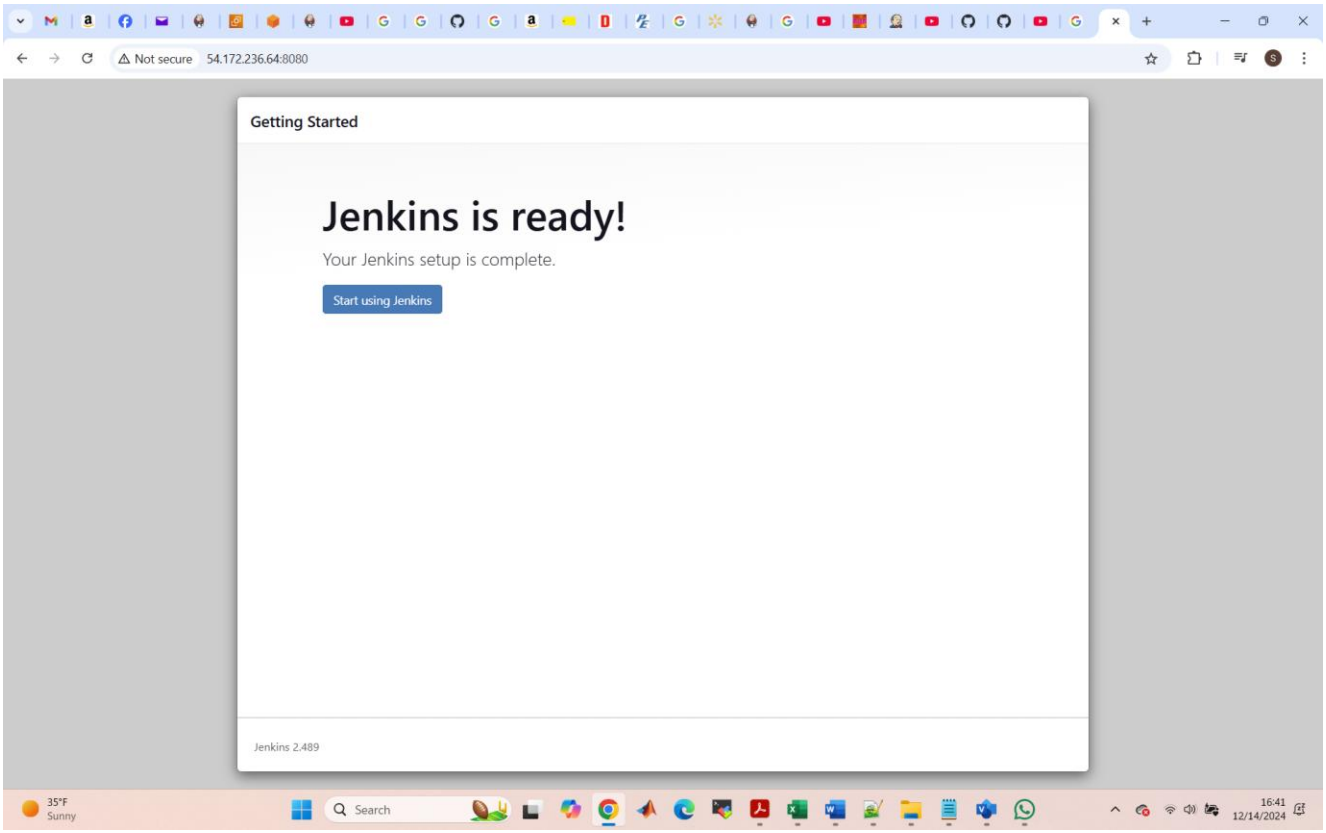
Enter the following:
Username: **admin**
Password: **admin**
Full name: **Sidney Ebot**
and Email address: **ebotsmith@gmail.com**



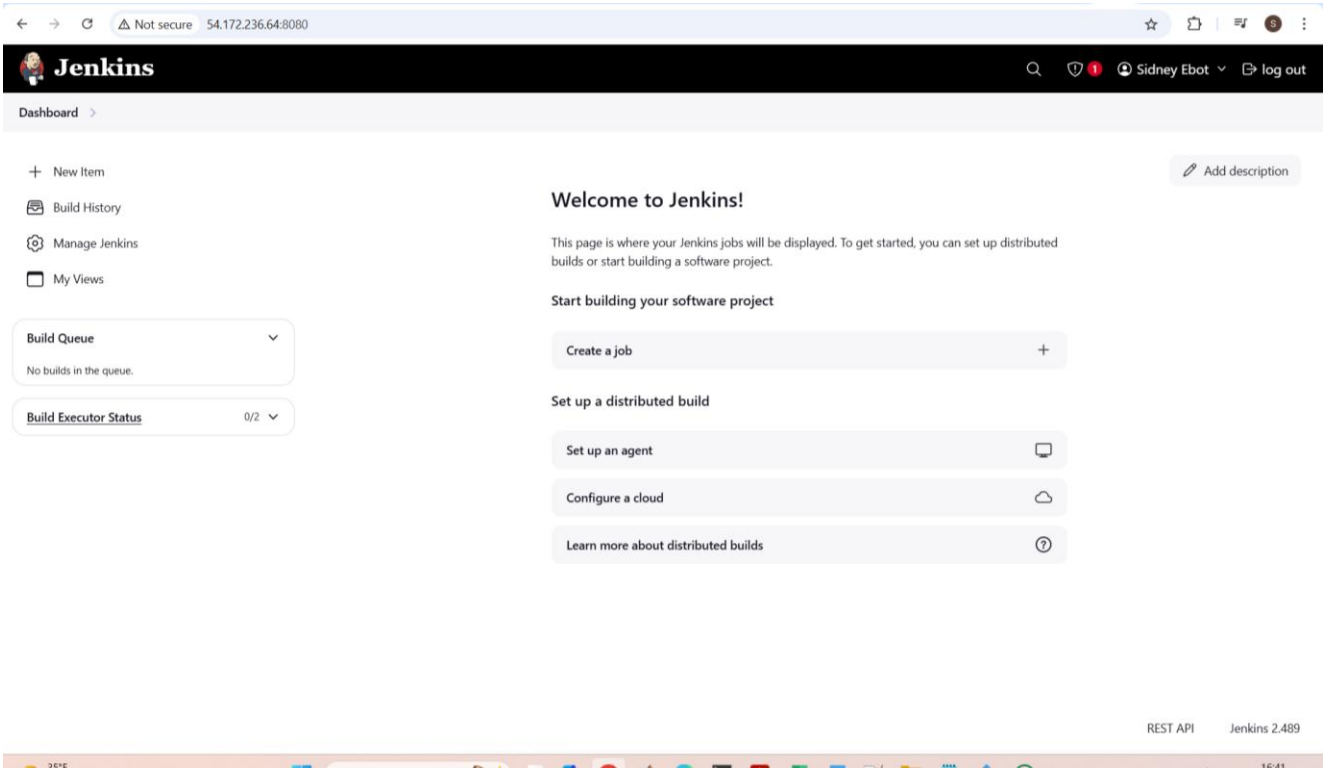
Click on “**Save and Continue**”



Then click on "**Save and Finish**"



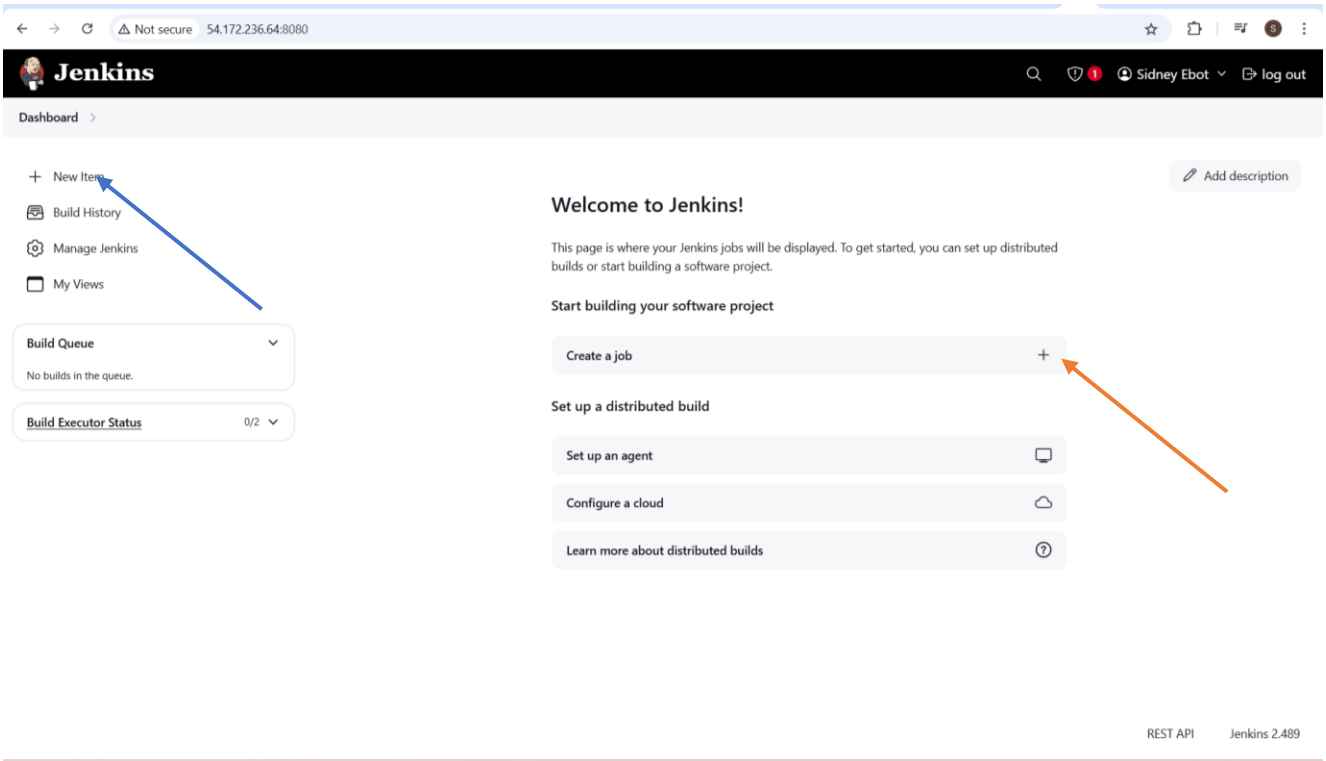
Then start Jenkins by clicking on "**Start using Jenkins**"



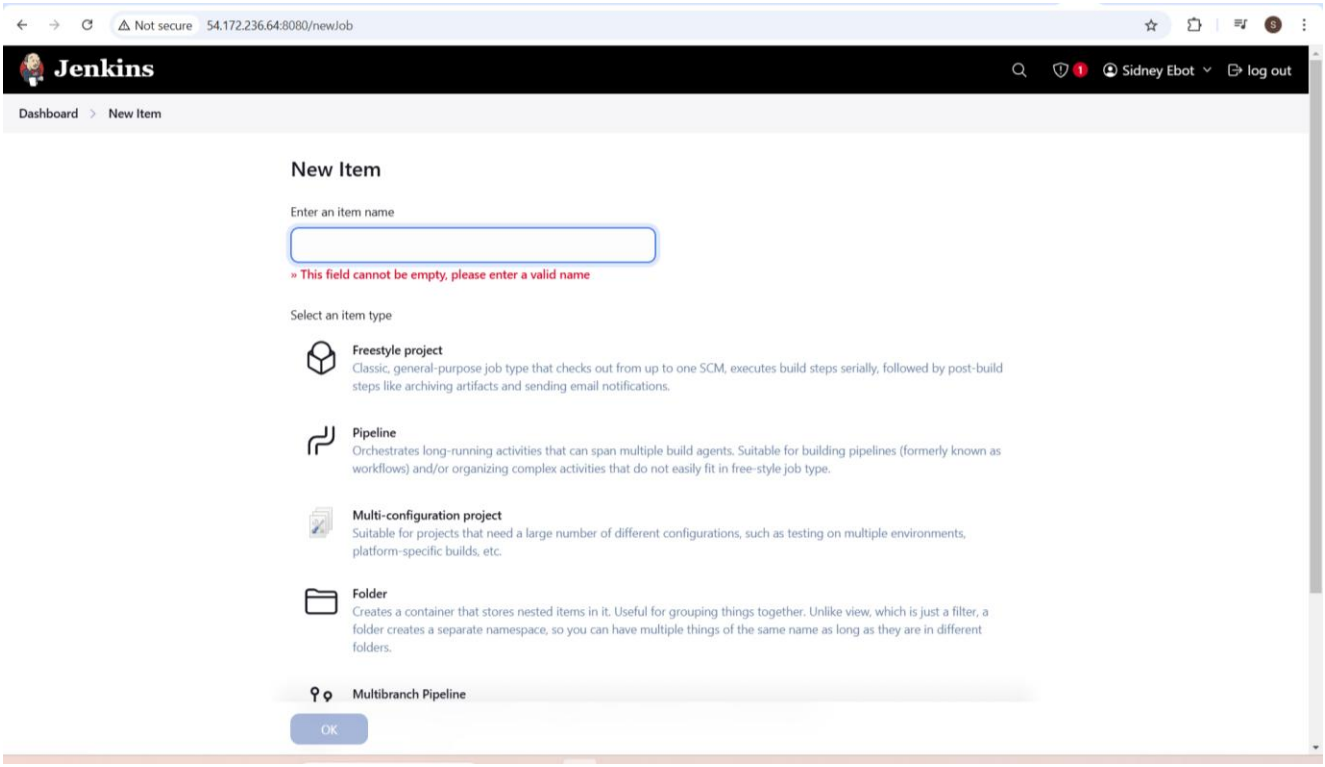
Jenkins is ready.

STEP 8: Create and Configure Pipeline on Jenkins

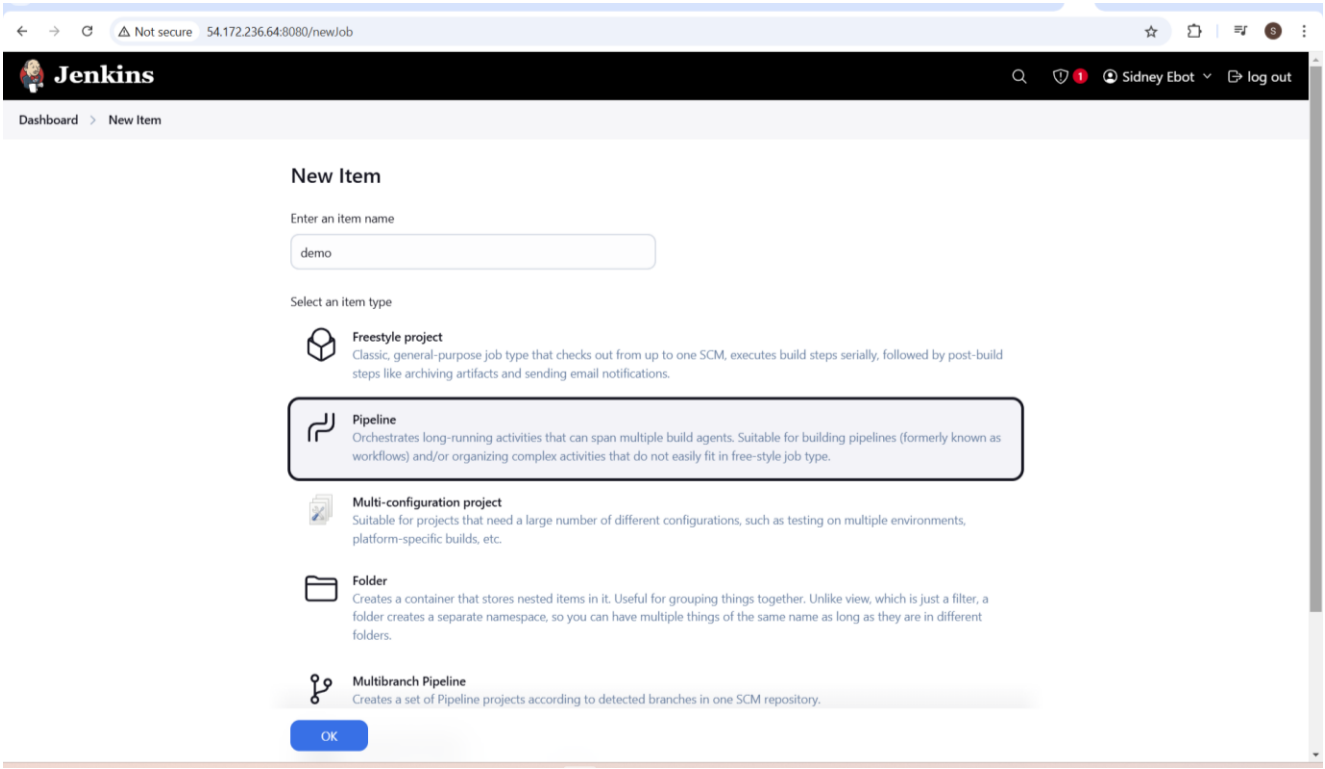
Now for our code compile we need maven. On the Jenkins GUI



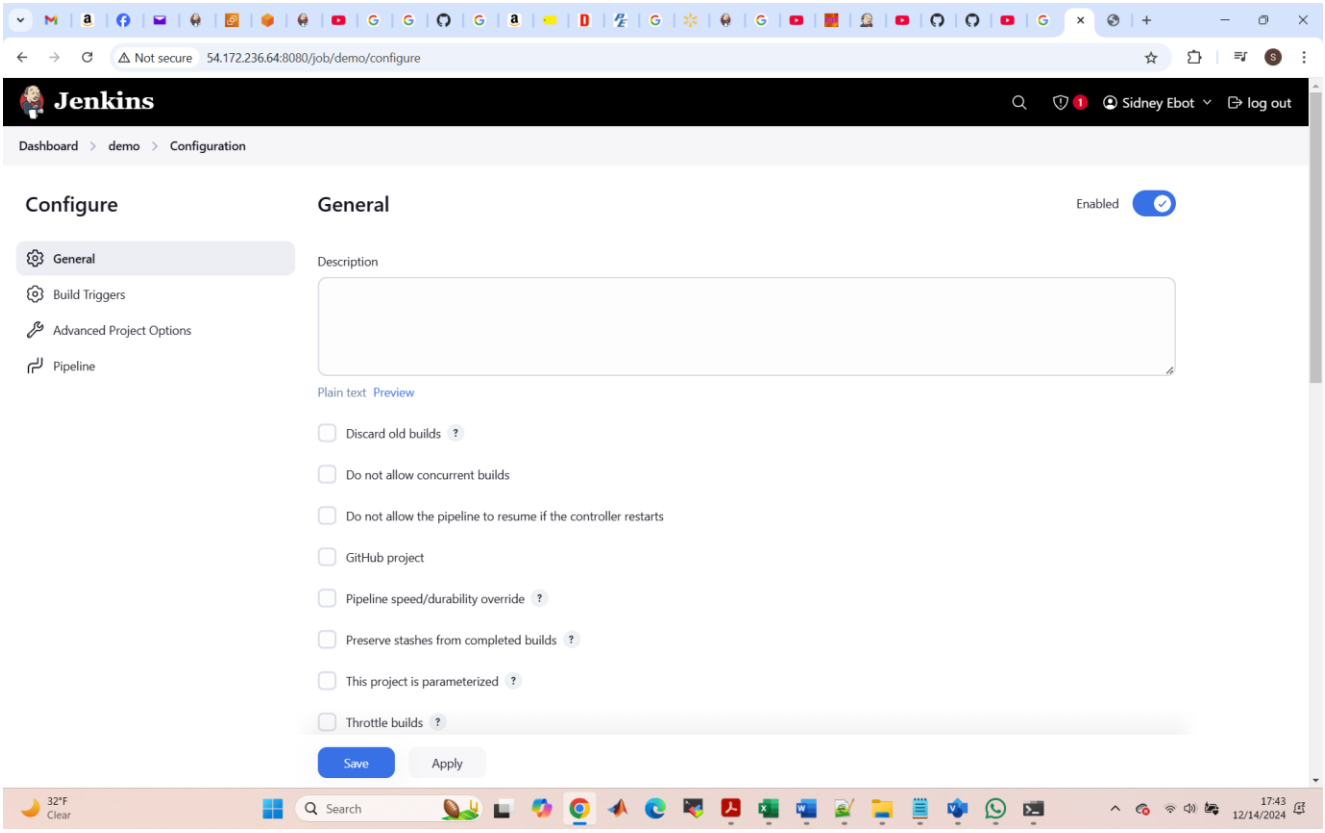
Click on "**Create a Job**" or "**New Item**"



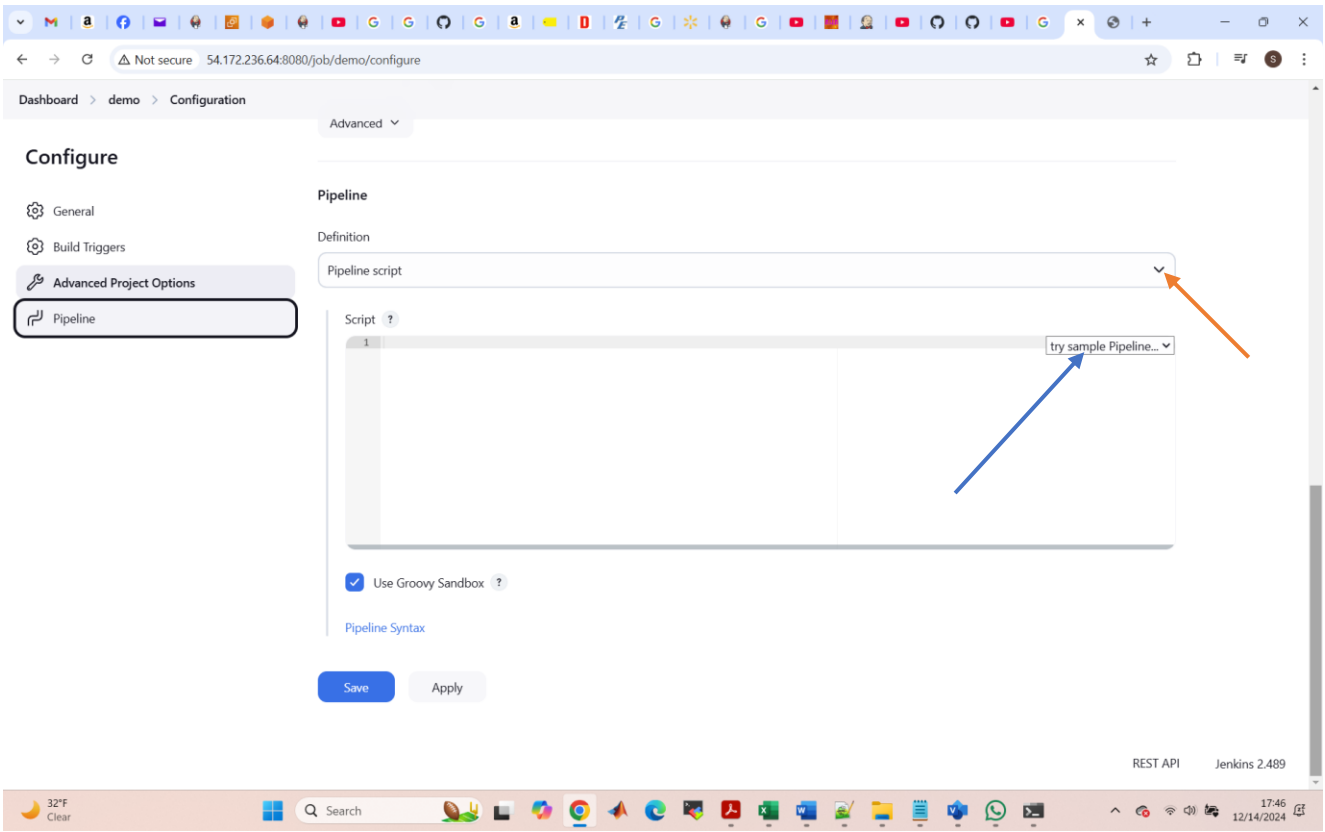
We will give the item the name "**demo**" and select "**Pipeline**"



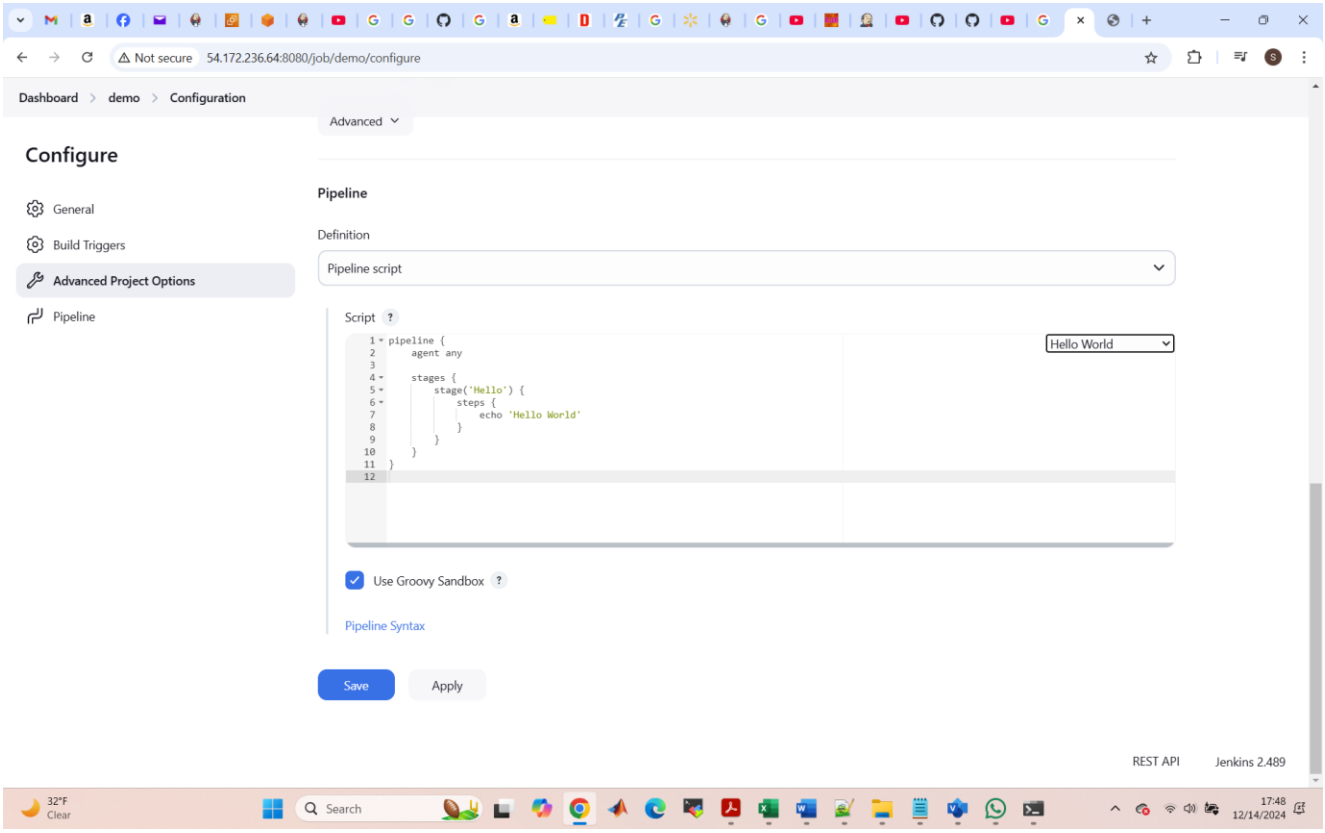
Click on "**OK**"



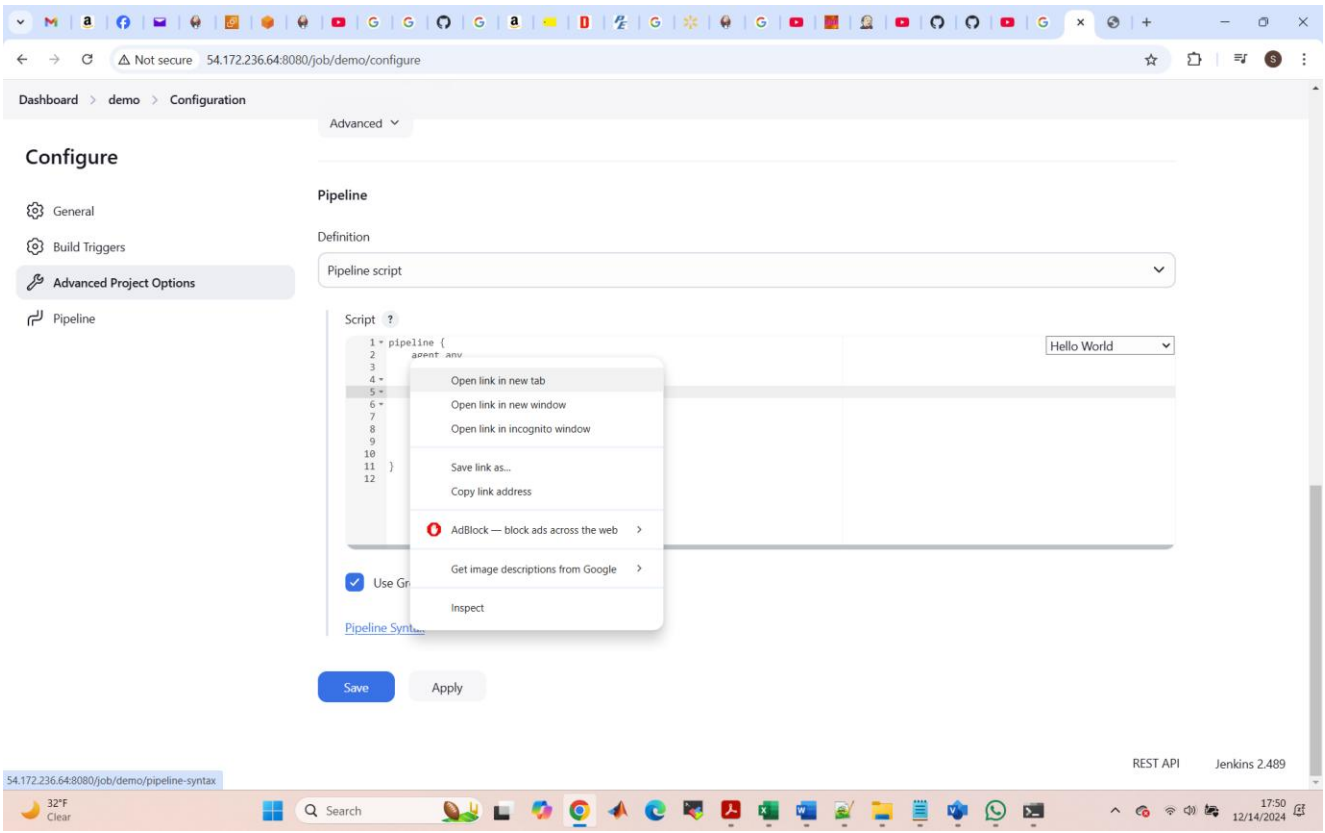
Click on “**Pipeline**” on the left-hand side



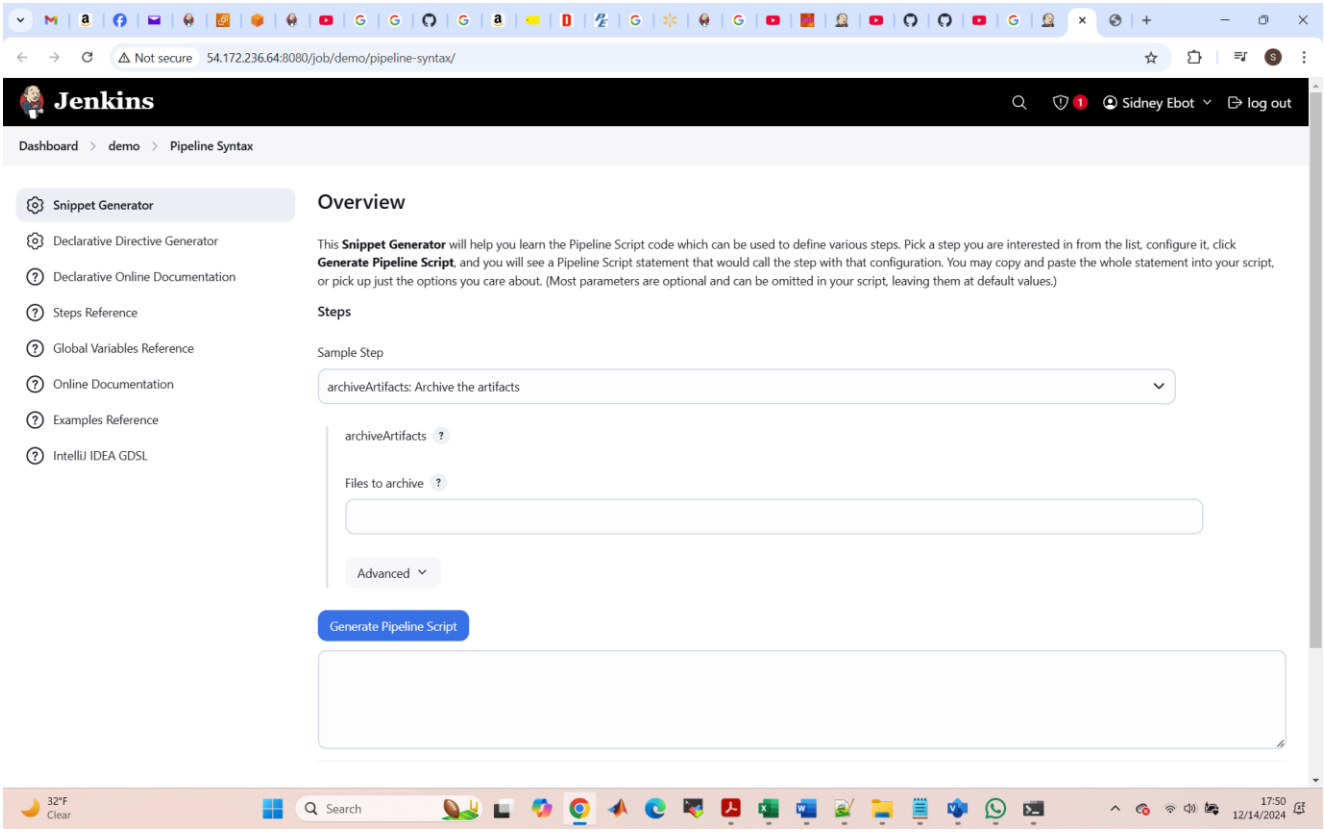
On “**Definition**”, select “**Pipeline Script**” and on “**Try Sample Pipeline**”, select “**HelloWorld**”



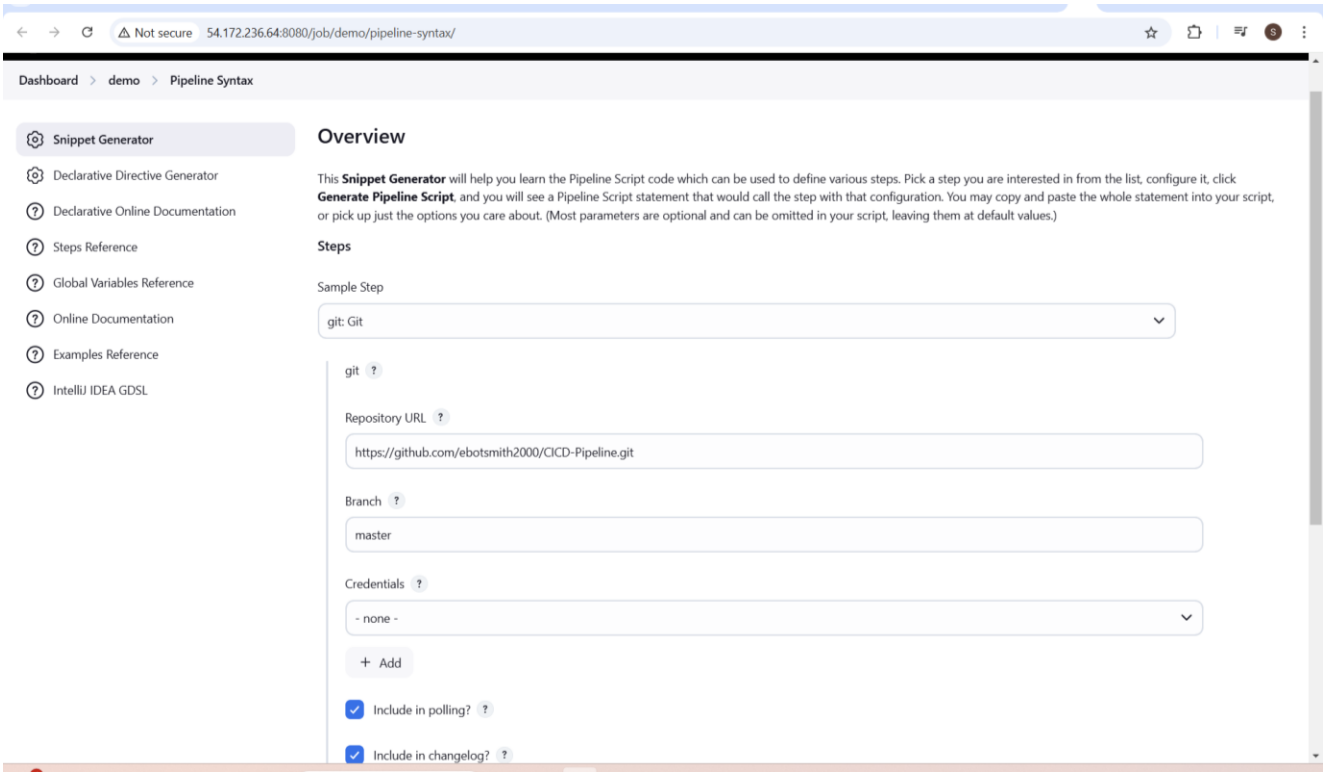
Right-click on "**Pipeline Syntax**"



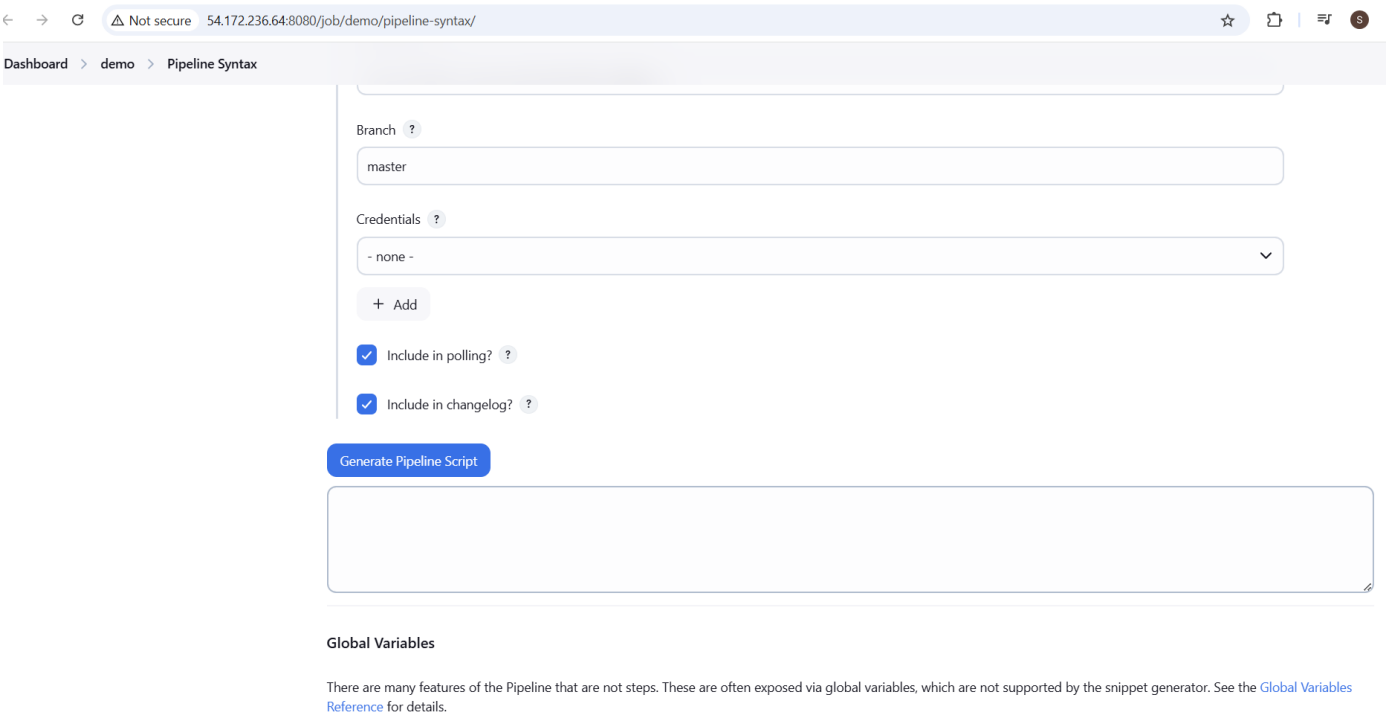
Select "**Open Link in new Tab**"



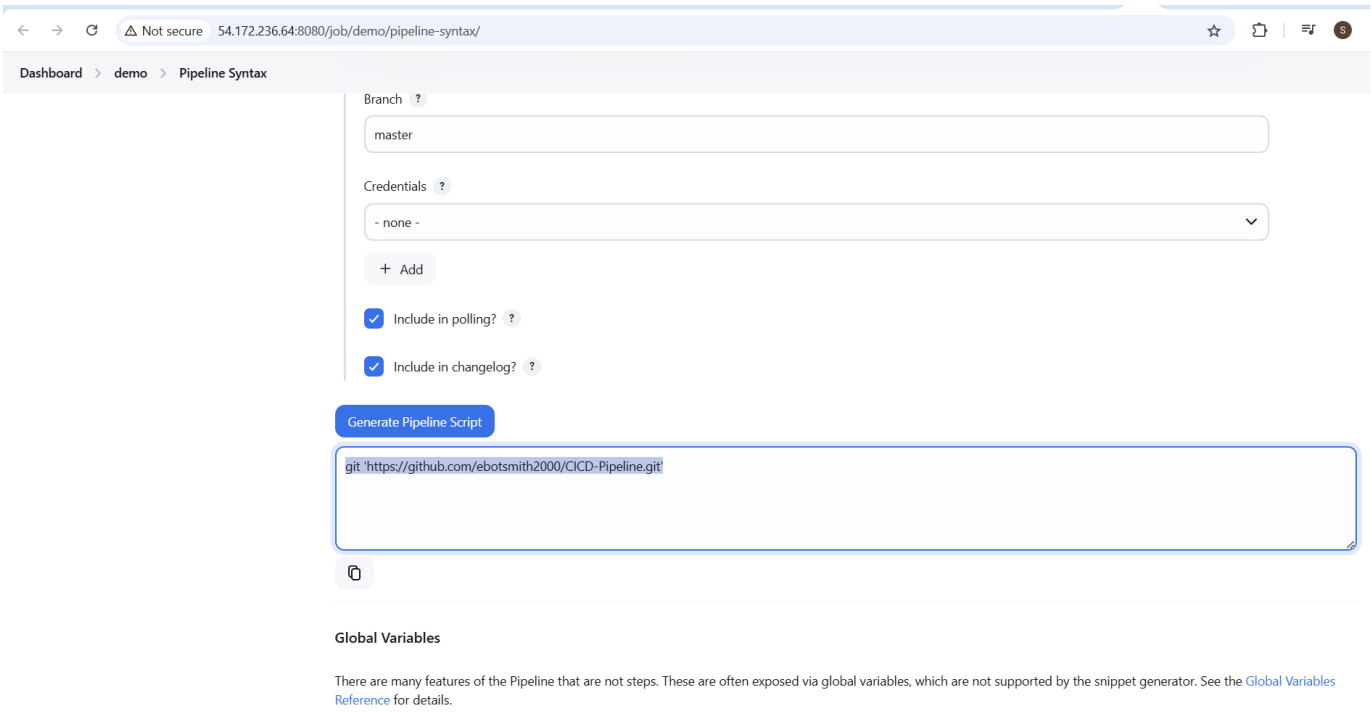
Under “**Sample Step**” in “**Steps**”, select “**git:Git**”. On “**Repository URL**” **copy the HTTP URL of your GitHub repository**.



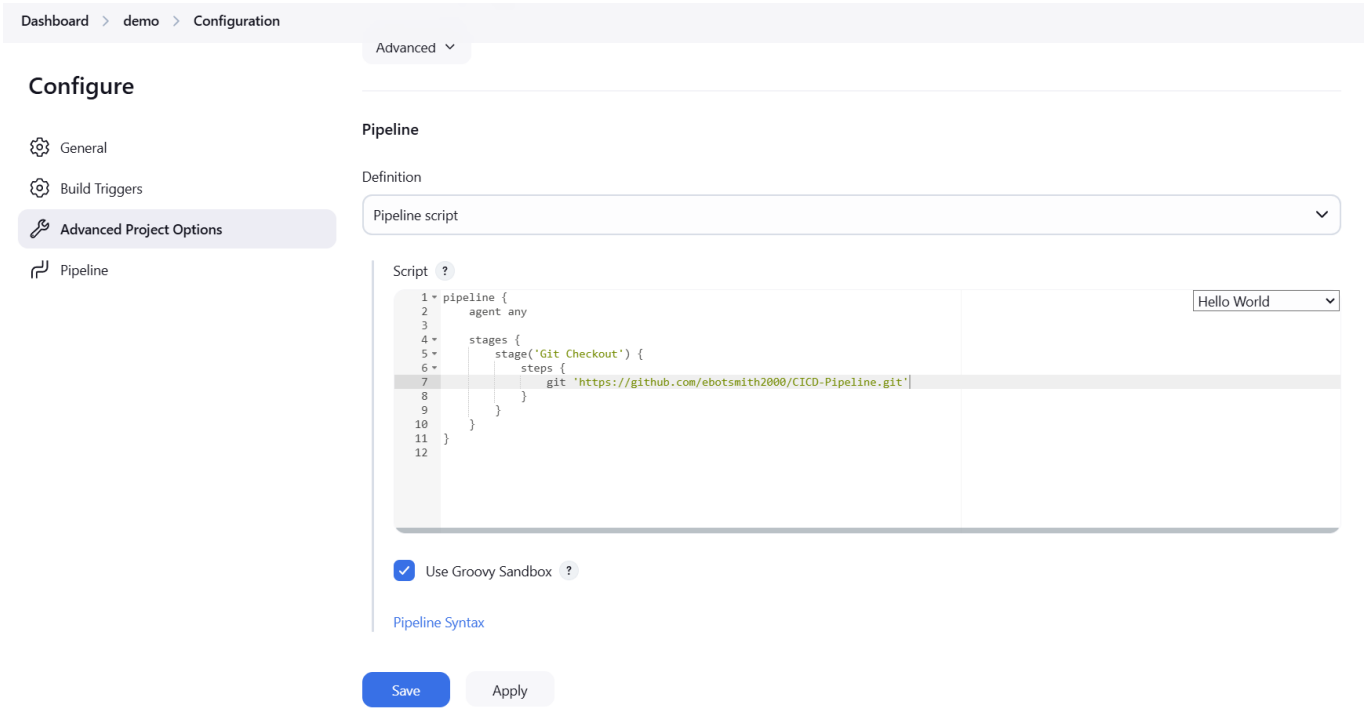
Scroll down to “**Generate Pipeline Script**”



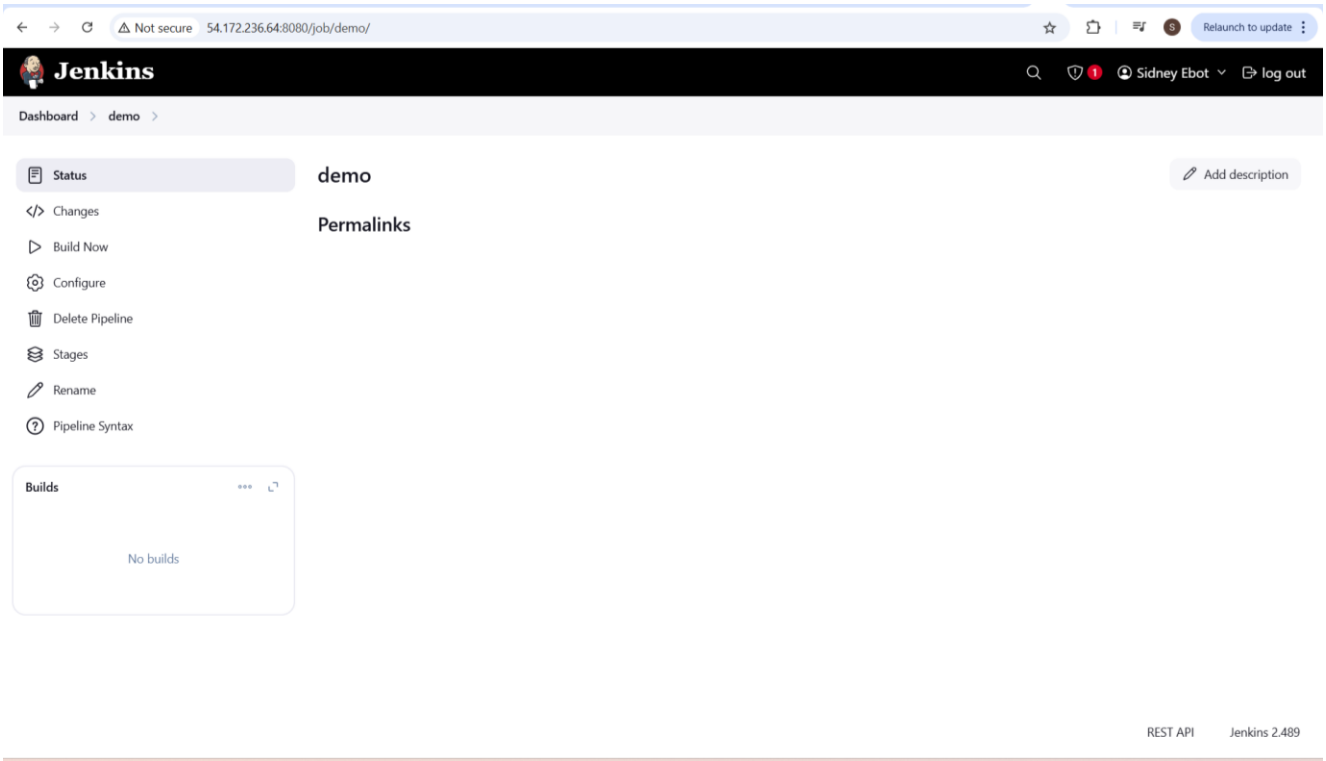
Click on “**Generate Pipeline Script**”



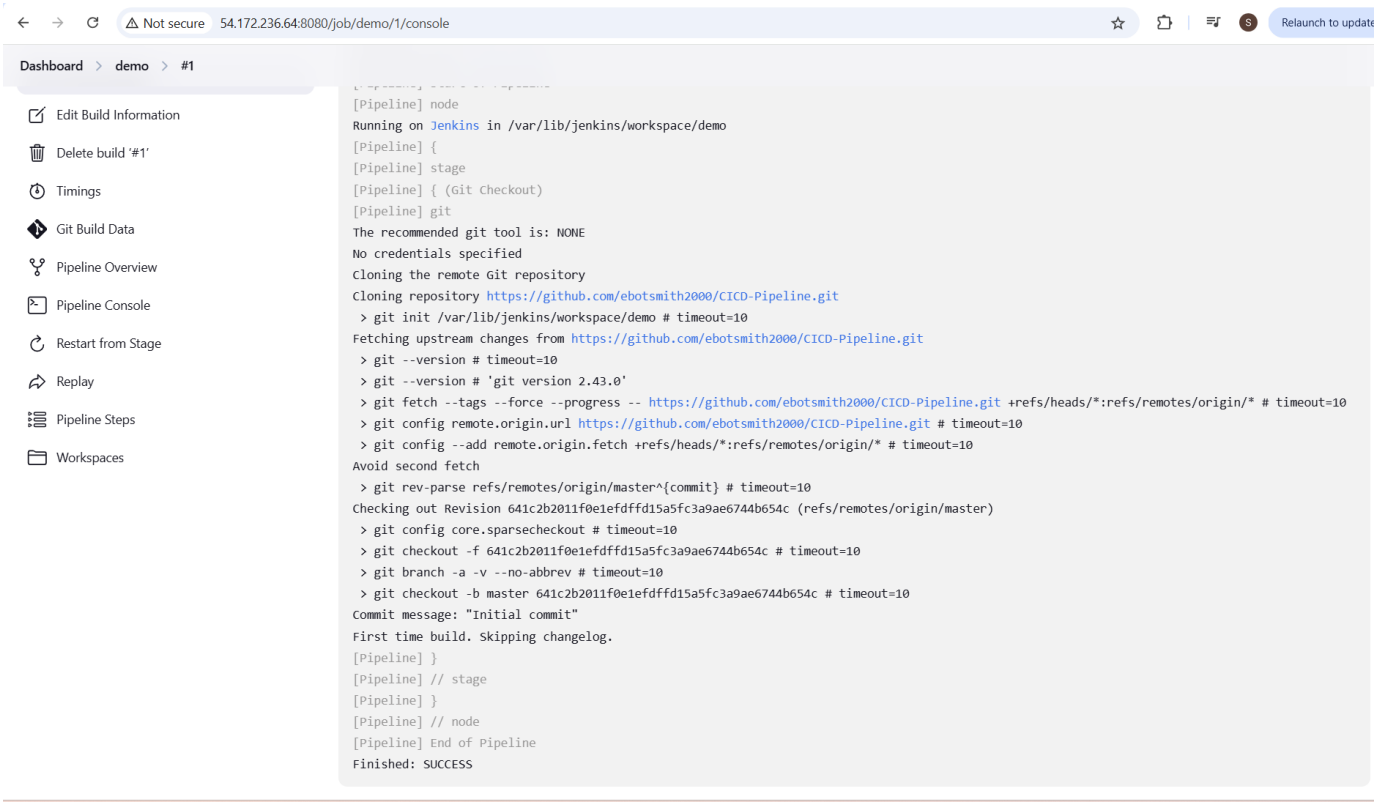
Copy the generated code and paste in the line for “**step**” in “**Script**” under “**Pipeline**”



Click on "**Save**"



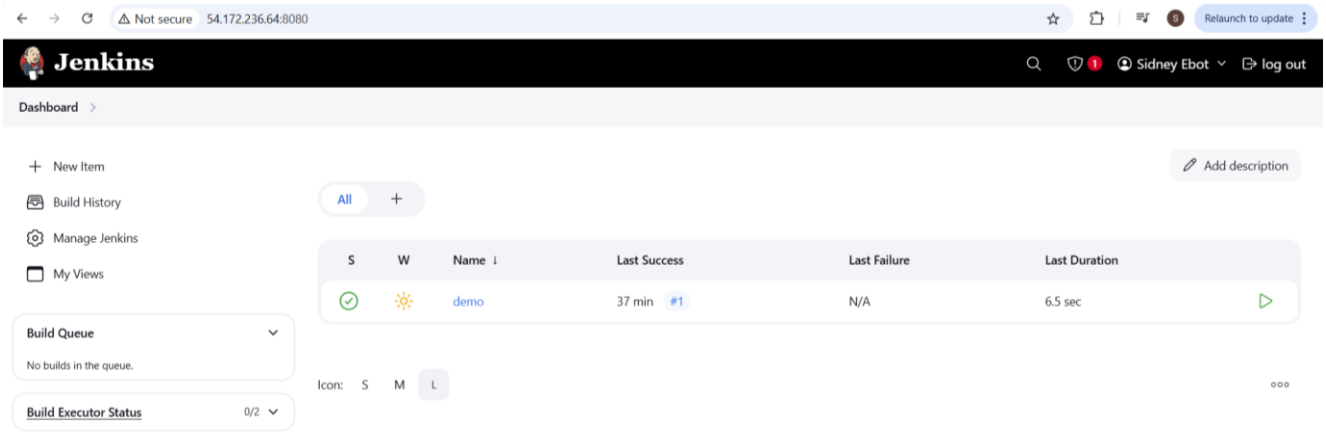
Click on "**Build Now**"



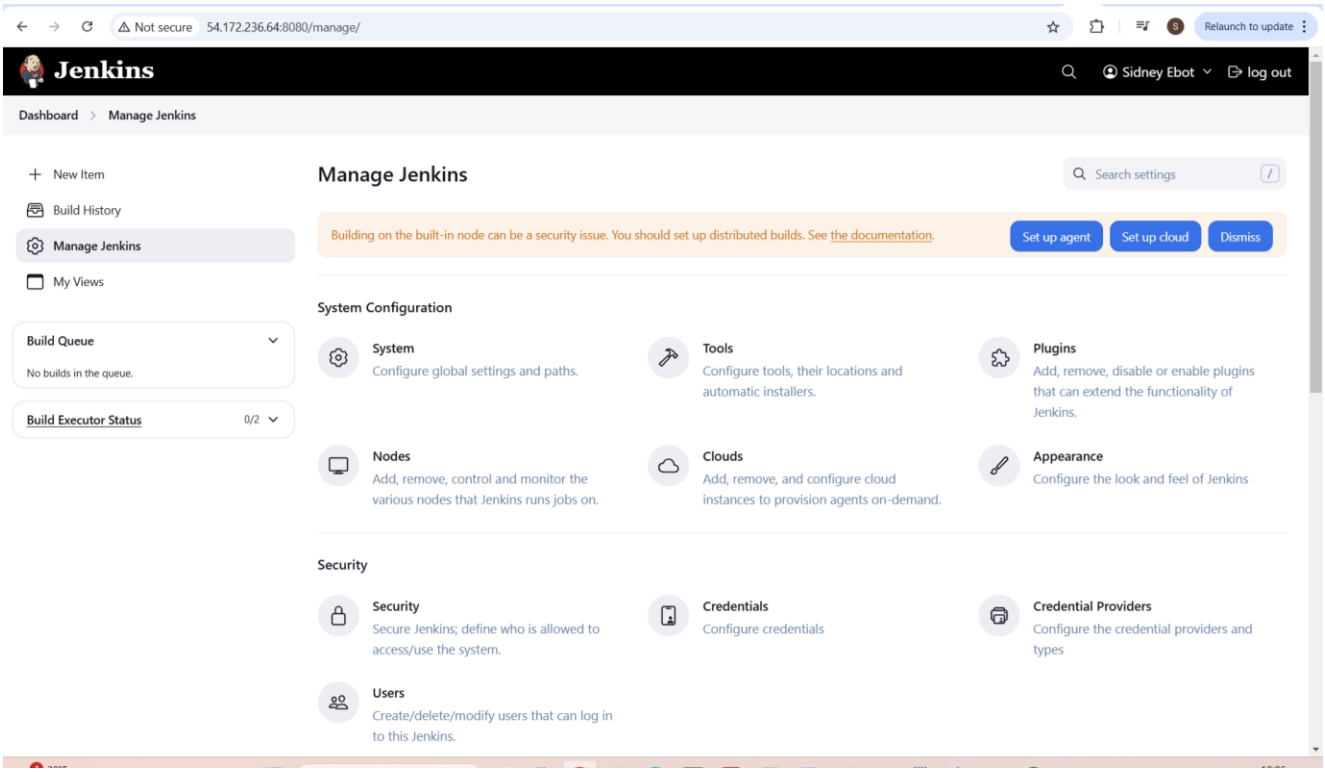
You can see that the build is successful

STEP 9: Install Pipeline Stage View Plugin

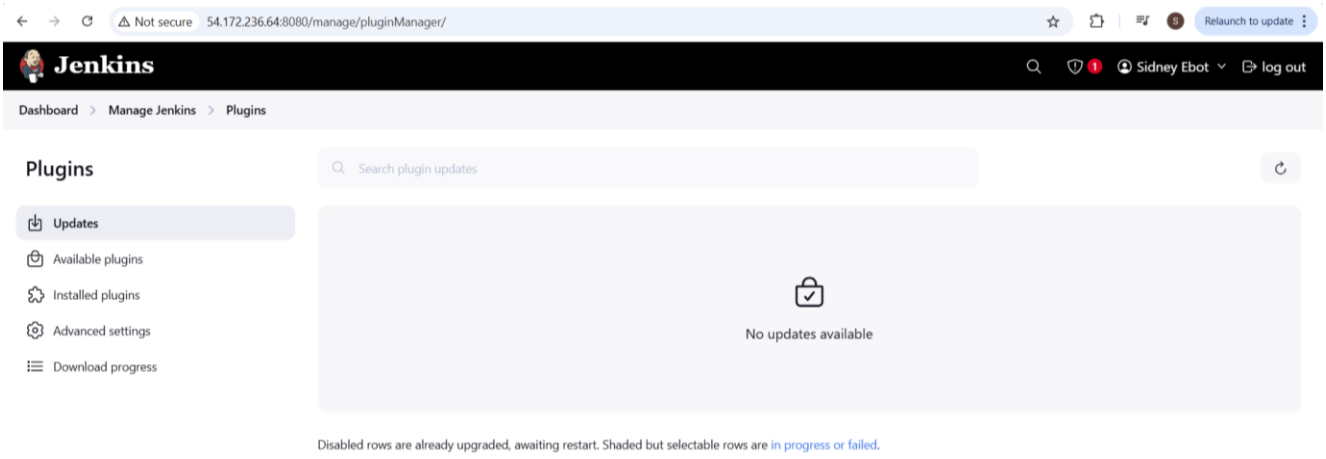
Now, click on "**Dashboard**"



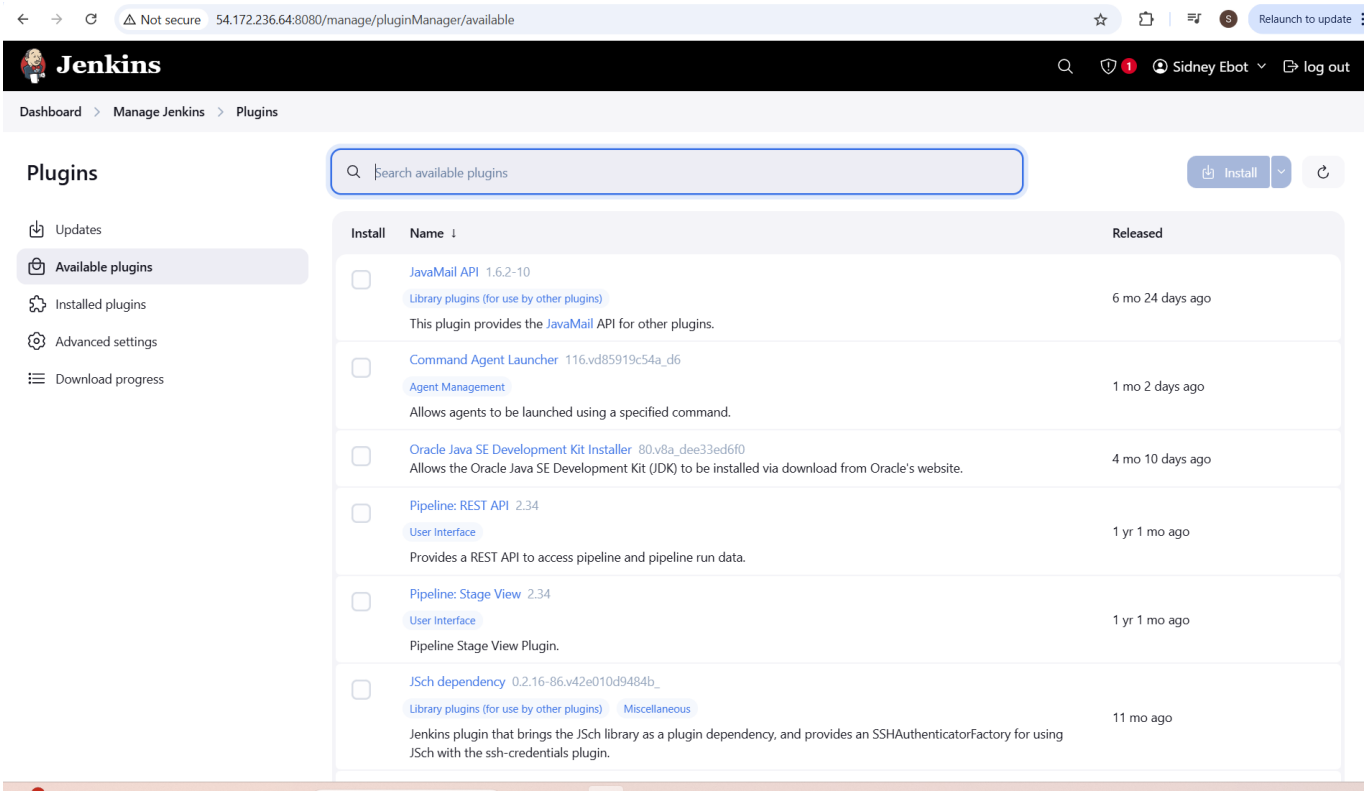
Click on "**Manage Jenkins**"



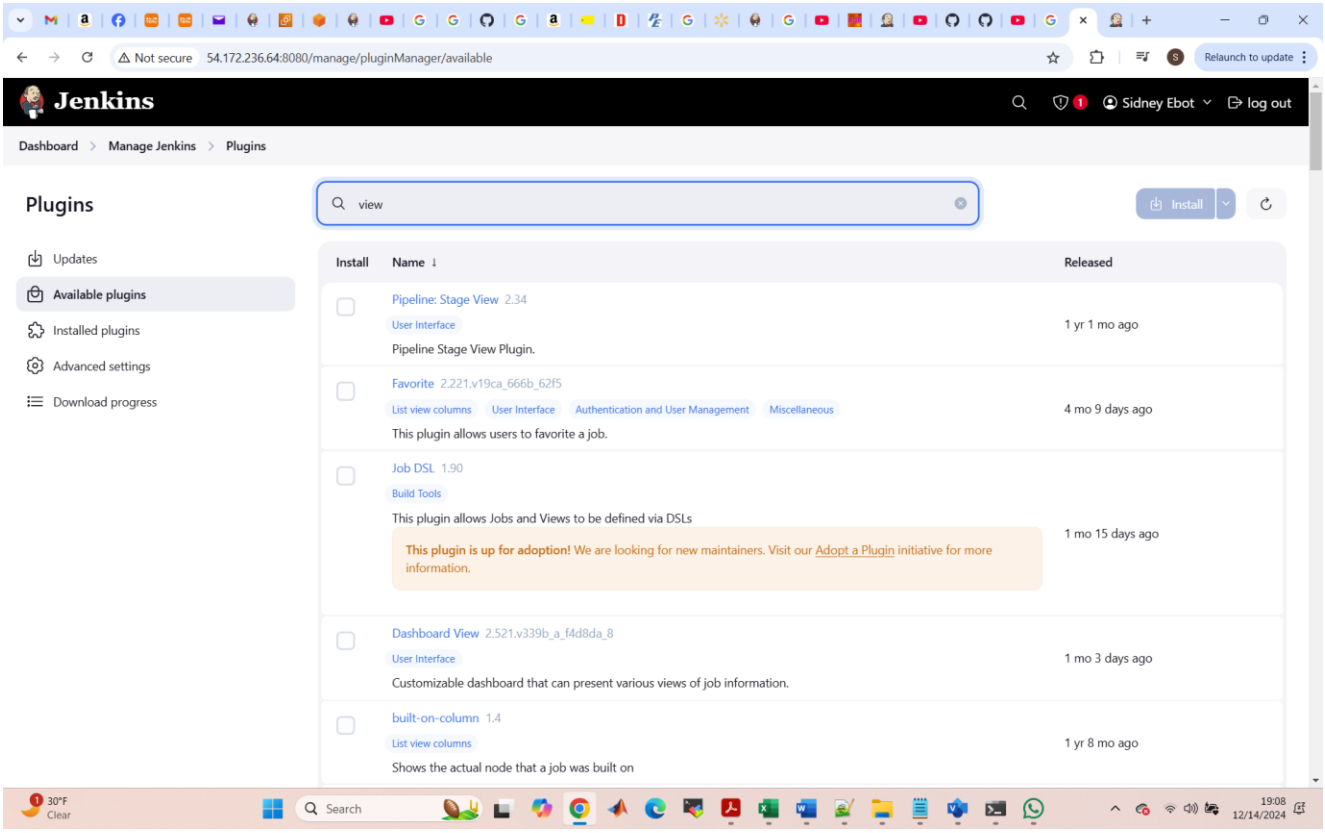
Click on "**Plugins**"



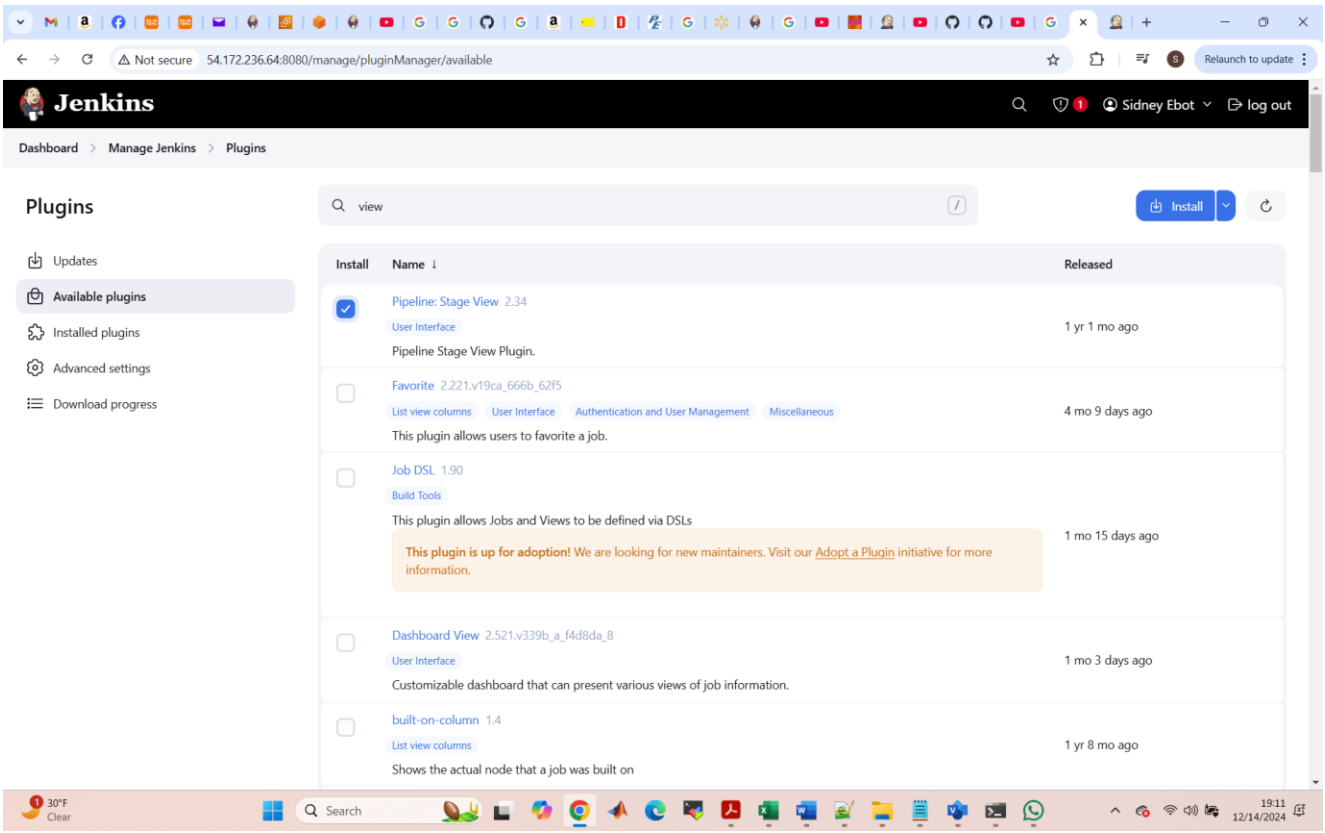
Click on “**Available Plugins**”



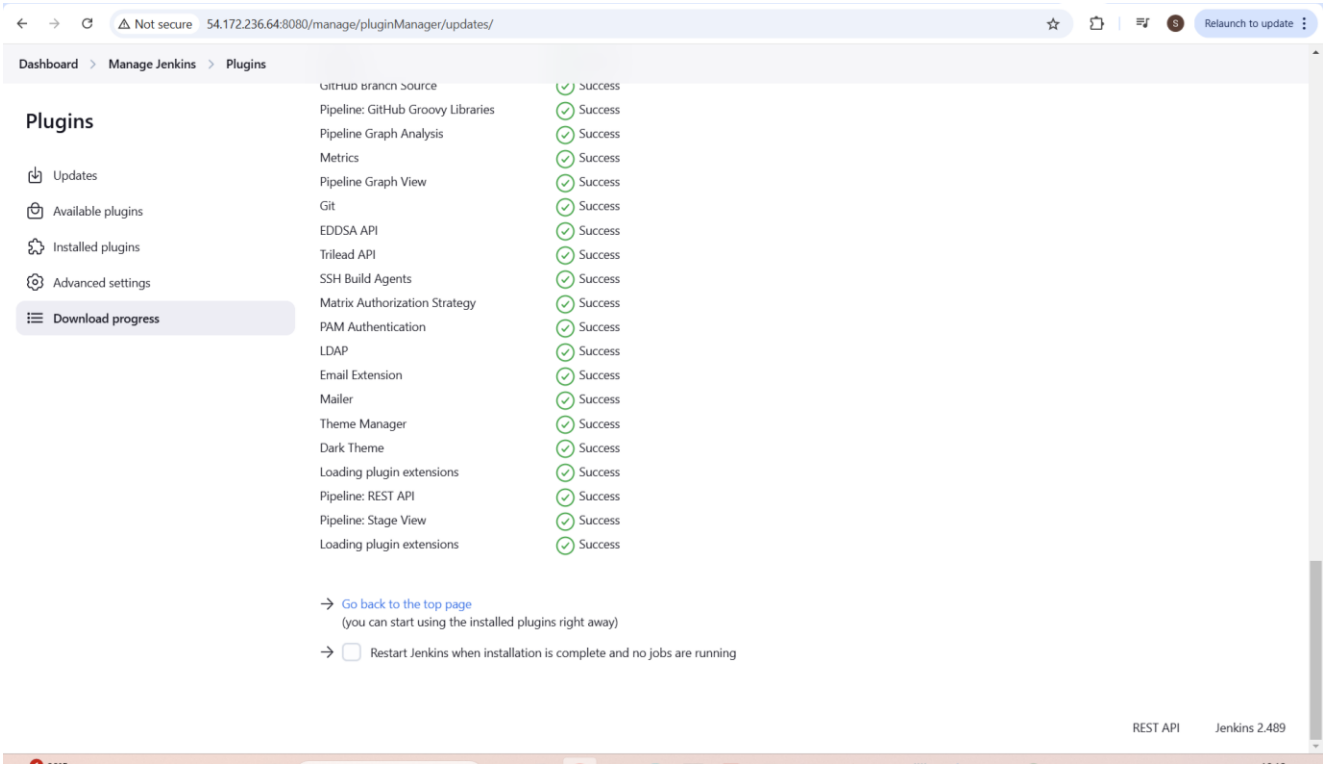
Search for “**View**”



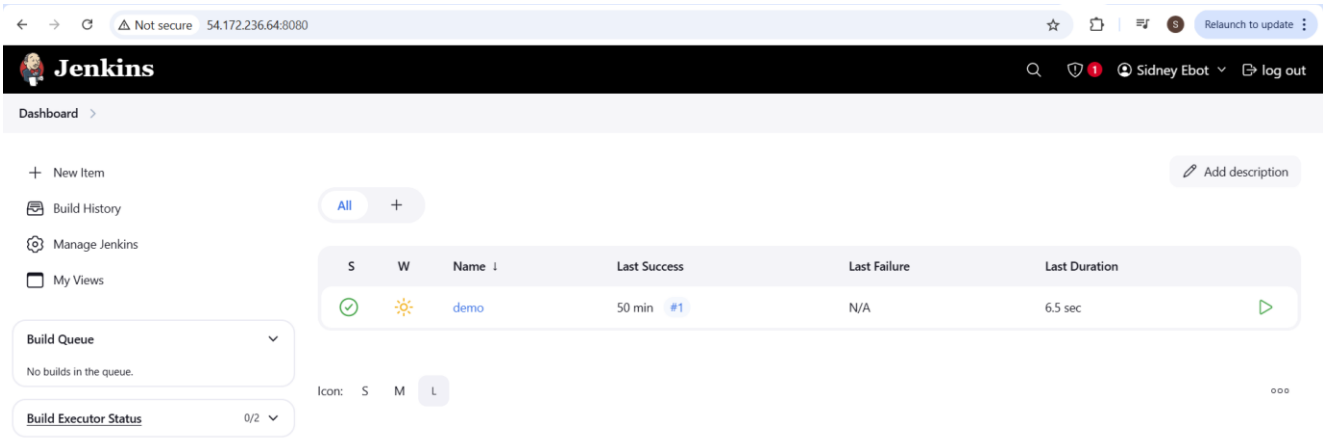
Select "**Pipeline Stage View**"



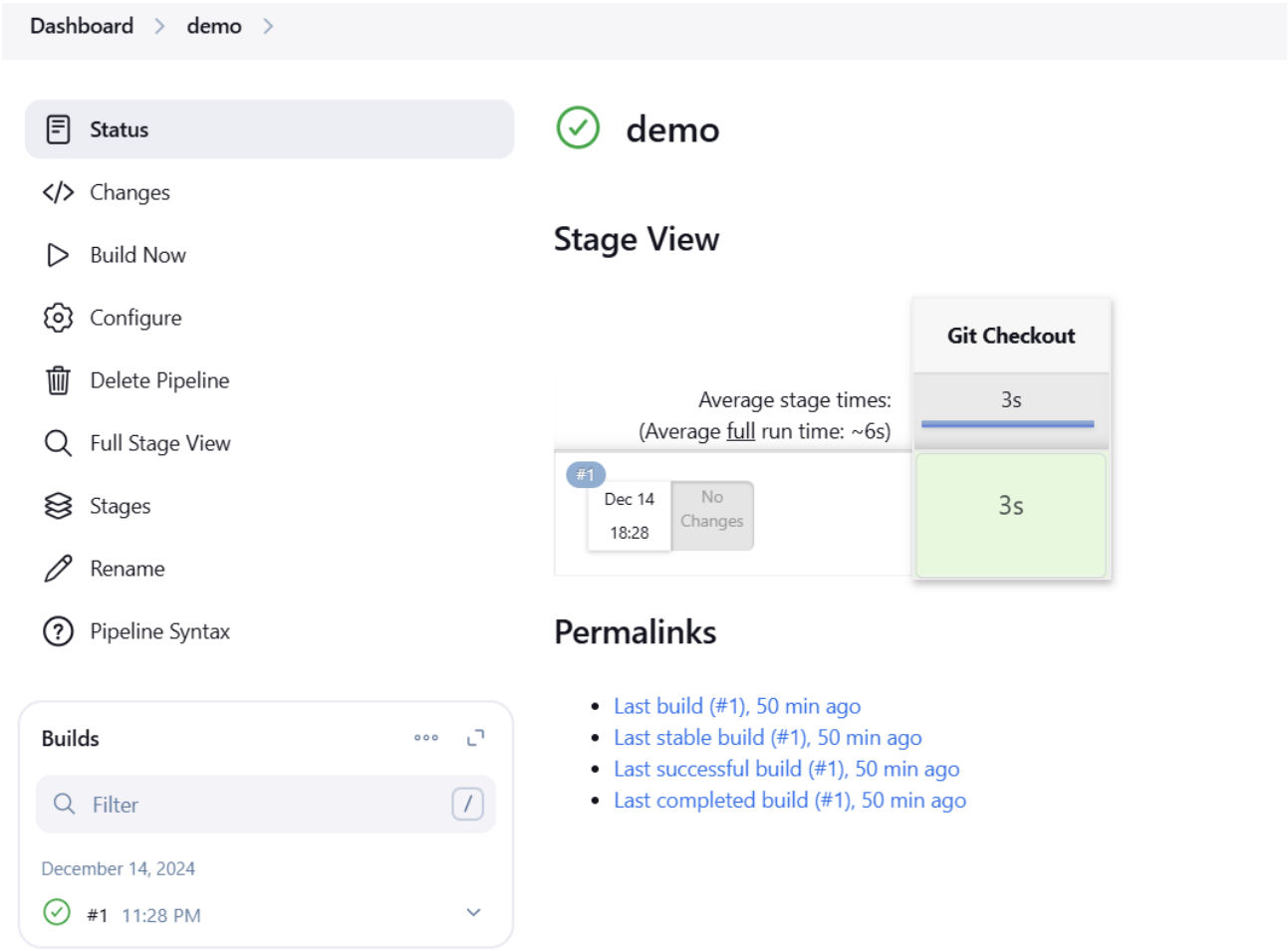
and click on "**Install**"



Click on “**Dashboard**”



Click on the pipeline “**demo**”



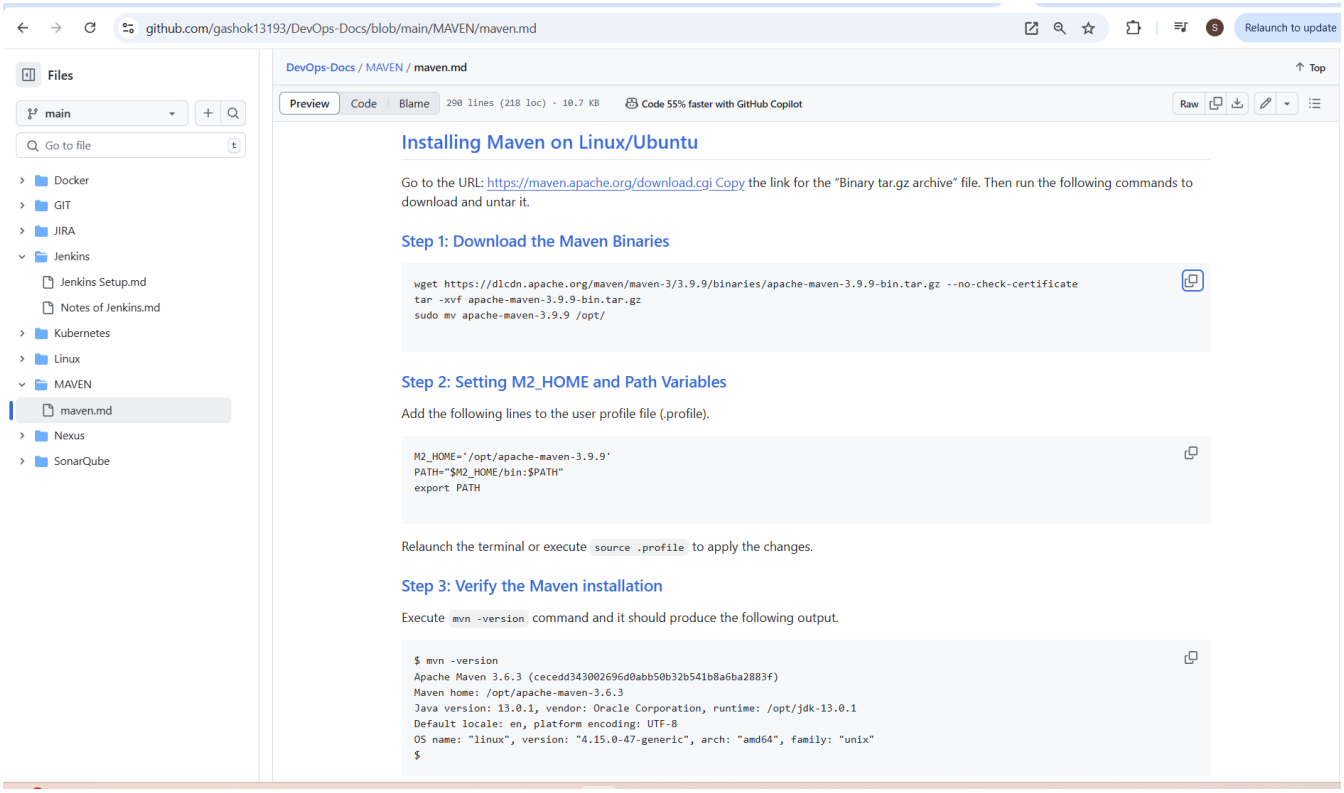
If you see, you are getting a clean view. First stage is to get checkout, so that is done.

STEP 10: Install and Configure Maven with Jenkins for Code Compile

The next step is Code compile using Maven. Let us install Maven. Go to this link:

https://github.com/gashok13193/DevOps-Docs

Click on “maven.md” under “Maven”



Run this command to install Maven:

```
wget https://dlcdn.apache.org/maven/maven-3/3.9.9/binaries/apache-maven-3.9.9-bin.tar.gz --no-check-certificate
tar -xvf apache-maven-3.9.9-bin.tar.gz
sudo mv apache-maven-3.9.9 /opt/
```

```
apache-maven-3.9.9/lib/org.eclipse.sisu.plexus-0.9.0.M3.jar
apache-maven-3.9.9/lib/jansi-2.4.1.jar
apache-maven-3.9.9/lib/maven-resolver-named-locks-1.9.22.jar
apache-maven-3.9.9/lib/maven-plugin-api-3.9.9.jar
apache-maven-3.9.9/lib/wagon-http-shared-3.5.3.jar
apache-maven-3.9.9/lib/httpclient-4.5.14.jar
apache-maven-3.9.9/lib/maven-resolver-transport-file-1.9.22.jar
apache-maven-3.9.9/lib/plexus-component-annotations-2.1.0.jar
apache-maven-3.9.9/lib/maven-resolver-connector-basic-1.9.22.jar
apache-maven-3.9.9/lib/plexus-utils-3.5.1.jar
ubuntu@ip-172-31-31-55:~$
```

Then run the next part of the code to set the maven path:

```
M2_HOME='/opt/apache-maven-3.9.9'
PATH="$M2_HOME/bin:$PATH"
export PATH
```

```
apache-maven-3.9.9/lib/maven-settings-builder-3.9.9.jar
apache-maven-3.9.9/lib/failureaccess-1.0.2.jar
apache-maven-3.9.9/lib/plexus-sec-dispatcher-2.0.jar
apache-maven-3.9.9/lib/maven-shared-utils-3.4.2.jar
apache-maven-3.9.9/lib/guava-33.2.1-jre.jar
apache-maven-3.9.9/lib/wagon-file-3.5.3.jar
apache-maven-3.9.9/lib/maven-model-builder-3.9.9.jar
apache-maven-3.9.9/lib/maven-slf4j-provider-3.9.9.jar
apache-maven-3.9.9/lib/plexus-interpolation-1.27.jar
apache-maven-3.9.9/lib/maven-resolver-transport-http-1.9.22.jar
apache-maven-3.9.9/lib/maven-model-3.9.9.jar
apache-maven-3.9.9/lib/plexus-xml-3.0.1.jar
apache-maven-3.9.9/lib/maven-resolver-spi-1.9.22.jar
apache-maven-3.9.9/lib/maven-compat-3.9.9.jar
apache-maven-3.9.9/lib/guice-5.1.0.jar
apache-maven-3.9.9/lib/commons-codec-1.17.1.jar
apache-maven-3.9.9/lib/javax.inject-1.jar
apache-maven-3.9.9/lib/org.eclipse.sisu.plexus-0.9.0.M3.jar
apache-maven-3.9.9/lib/jansi-2.4.1.jar
apache-maven-3.9.9/lib/maven-resolver-named-locks-1.9.22.jar
apache-maven-3.9.9/lib/maven-plugin-api-3.9.9.jar
apache-maven-3.9.9/lib/wagon-http-shared-3.5.3.jar
apache-maven-3.9.9/lib/httpclient-4.5.14.jar
apache-maven-3.9.9/lib/maven-resolver-transport-file-1.9.22.jar
apache-maven-3.9.9/lib/plexus-component-annotations-2.1.0.jar
apache-maven-3.9.9/lib/maven-resolver-connector-basic-1.9.22.jar
apache-maven-3.9.9/lib/plexus-utils-3.5.1.jar
ubuntu@ip-172-31-31-55:~$ M2_HOME='/opt/apache-maven-3.9.9'
PATH="$M2_HOME/bin:$PATH"
export PATH
ubuntu@ip-172-31-31-55:~$
```

Verify the Maven installation by using the command:

```
mvn -version
```

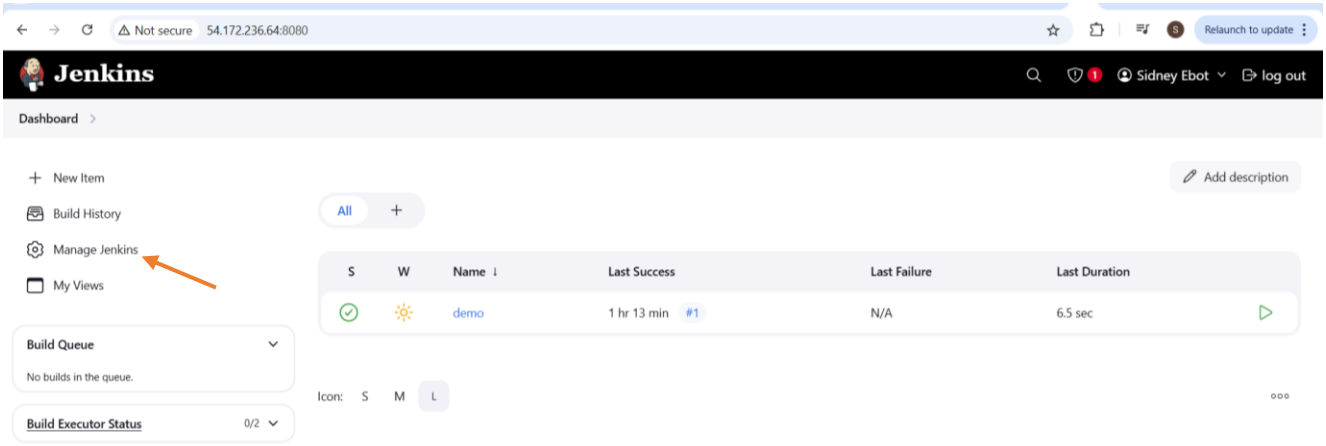
```
ubuntu@ip-172-31-31-55:~$ mvn -version
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: /opt/apache-maven-3.9.9
Java version: 17.0.13, vendor: Ubuntu, runtime: /usr/lib/jvm/java-17-openjdk-amd64
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "6.8.0-1018-aws", arch: "amd64", family: "unix"
ubuntu@ip-172-31-31-55:~$
```

Maven version 3.9.9 has been installed.

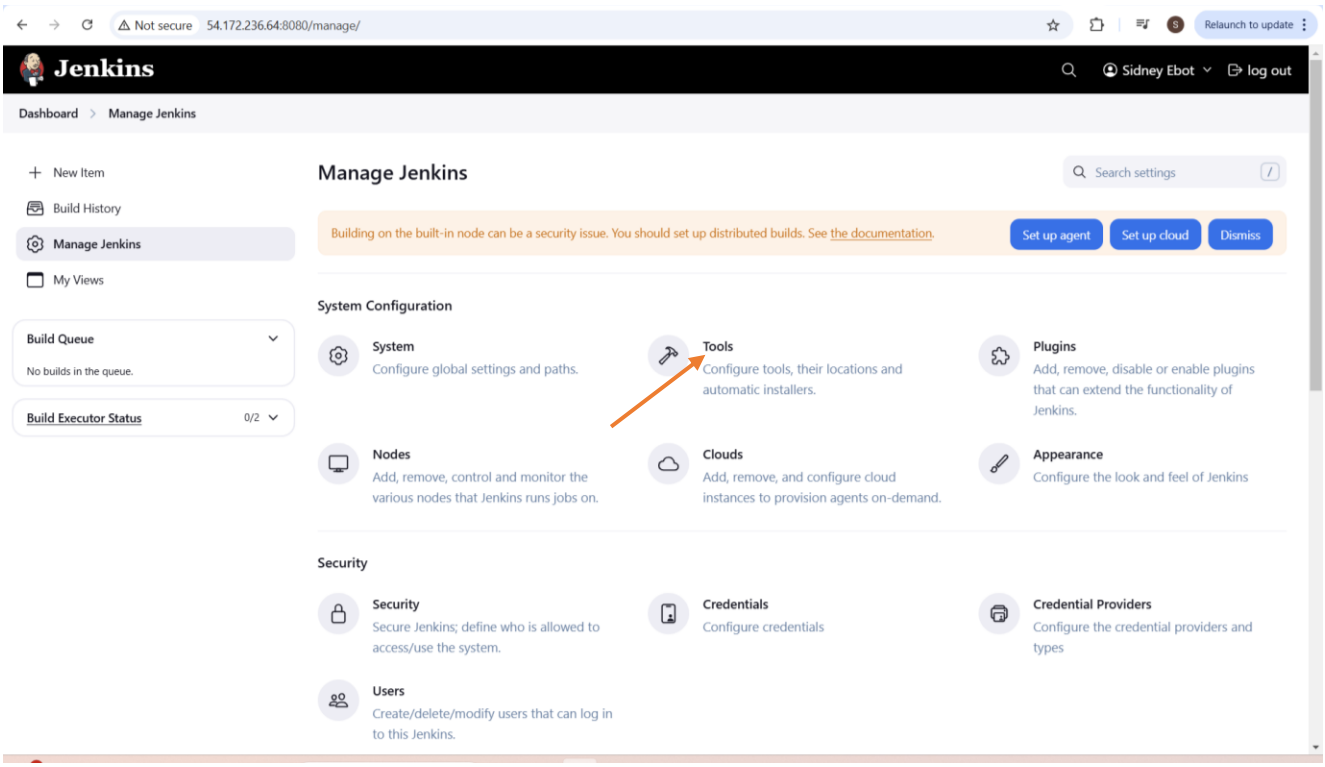
PART 1: Configure Maven with Jenkins

Now, let us configure Maven with Jenkins

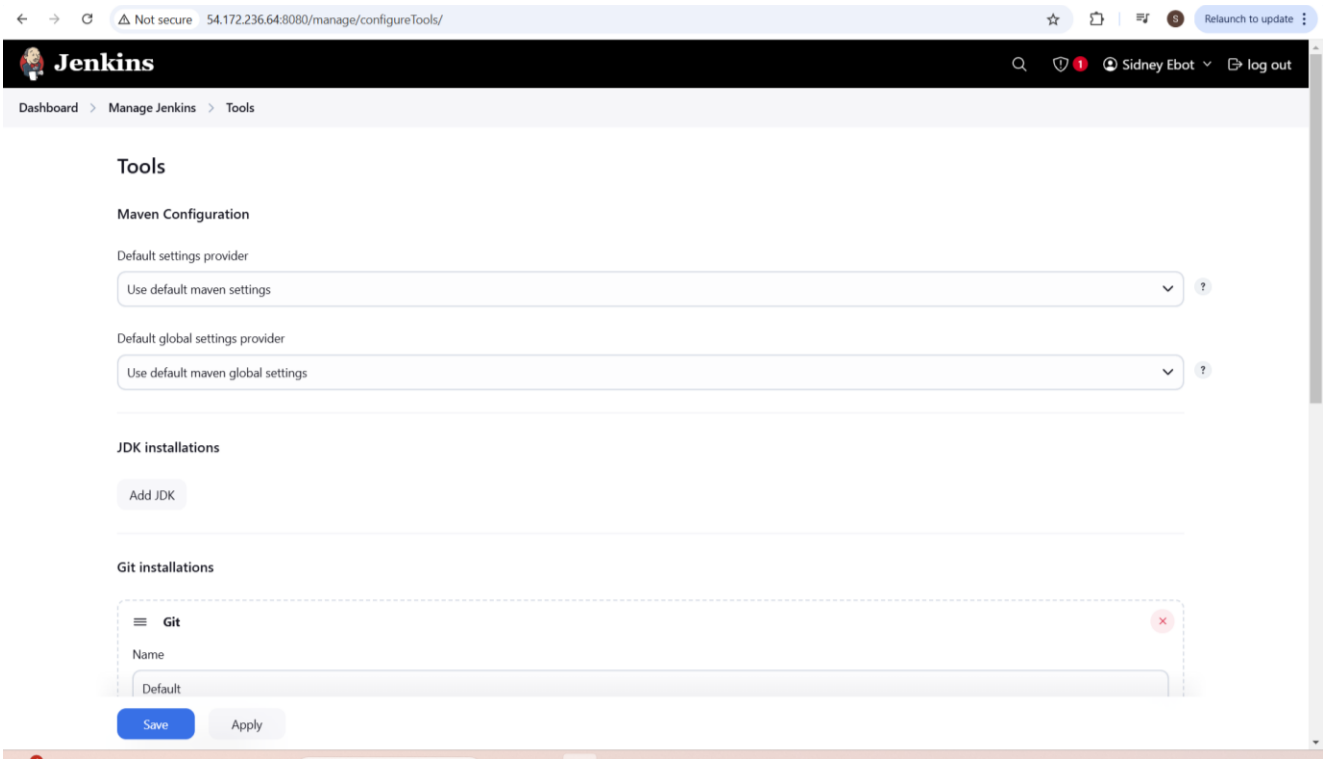
Go to "**Dashboard**"



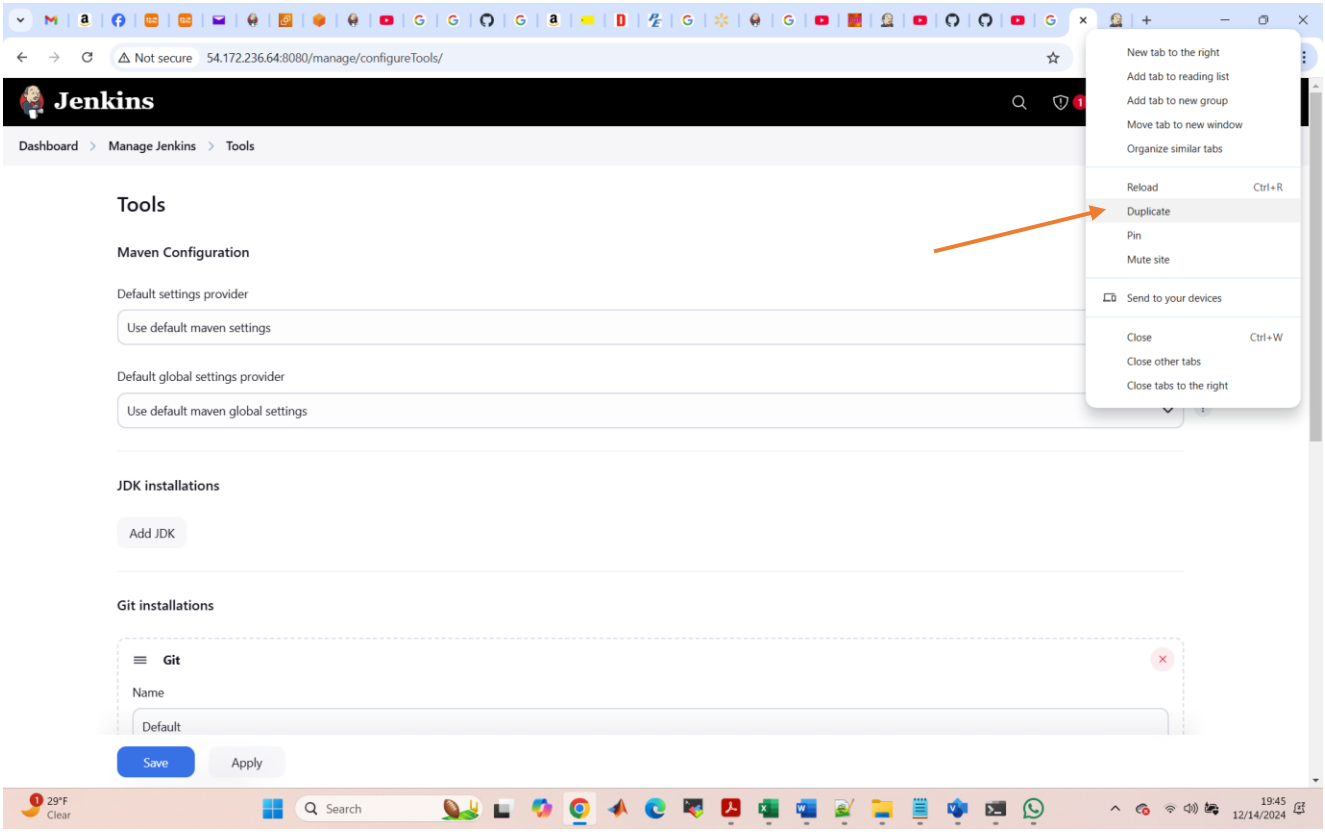
Click on "**Manage Jenkins**"



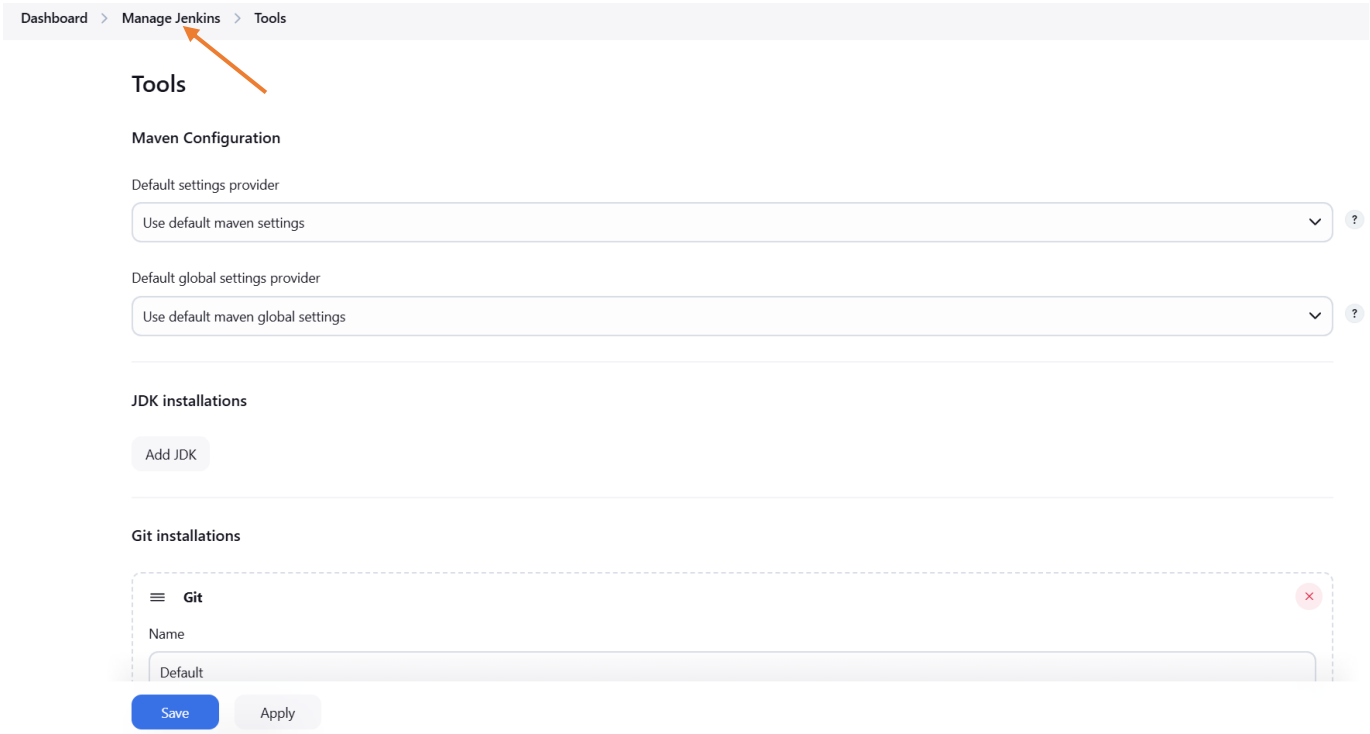
Click on "**Tools**"



Duplicate the Tab

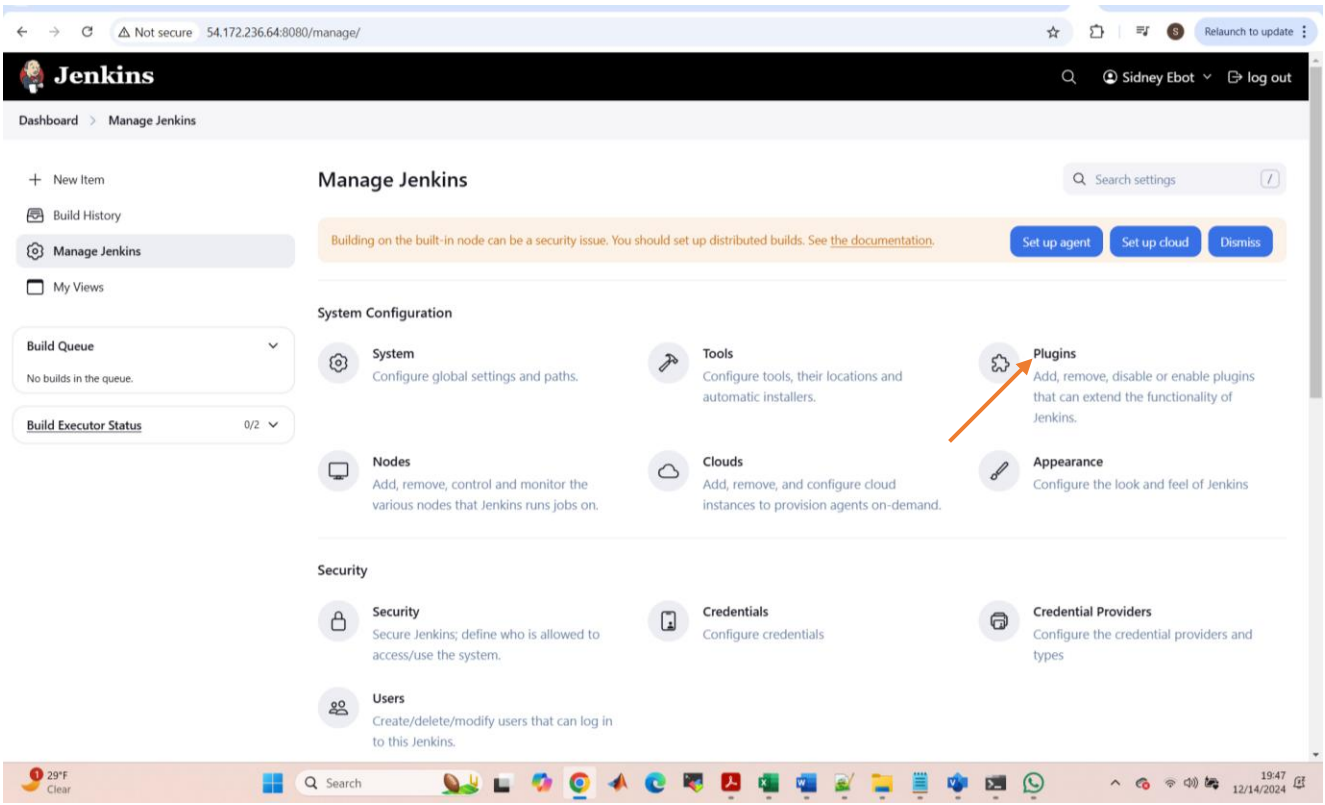


Select "**Duplicate**"

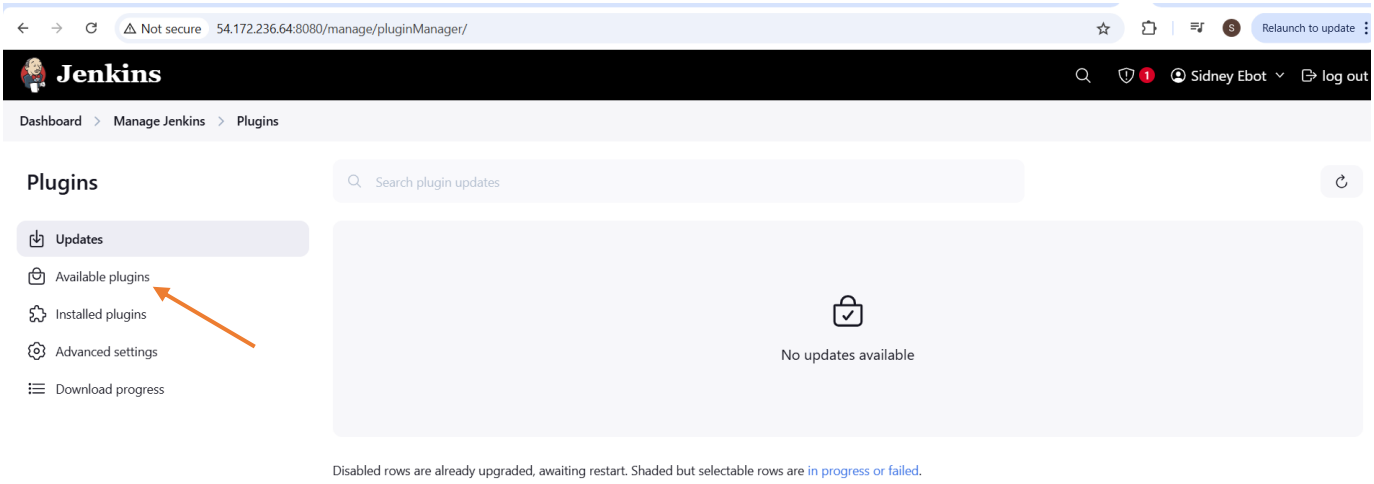


PART 2: Install Plugins

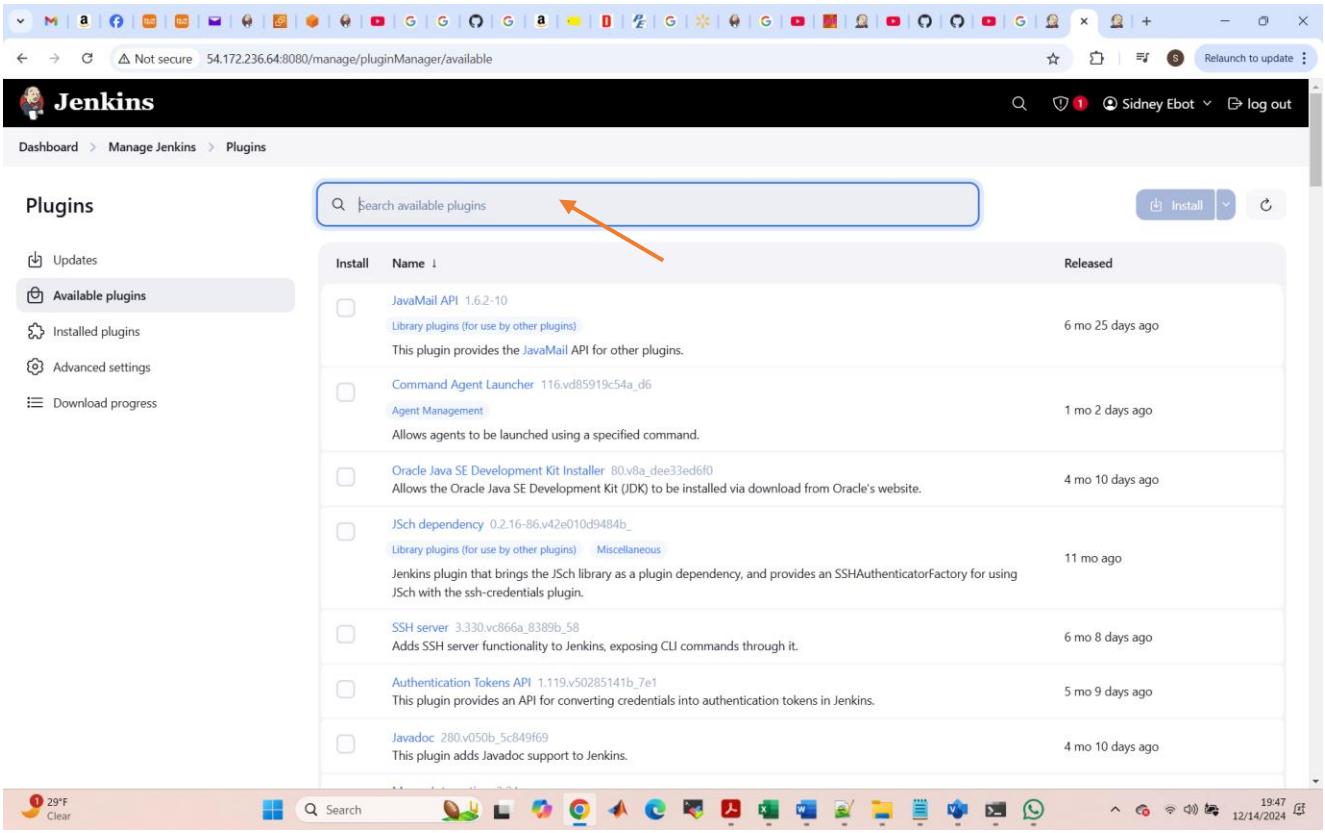
Click on "**Manage Jenkins**"



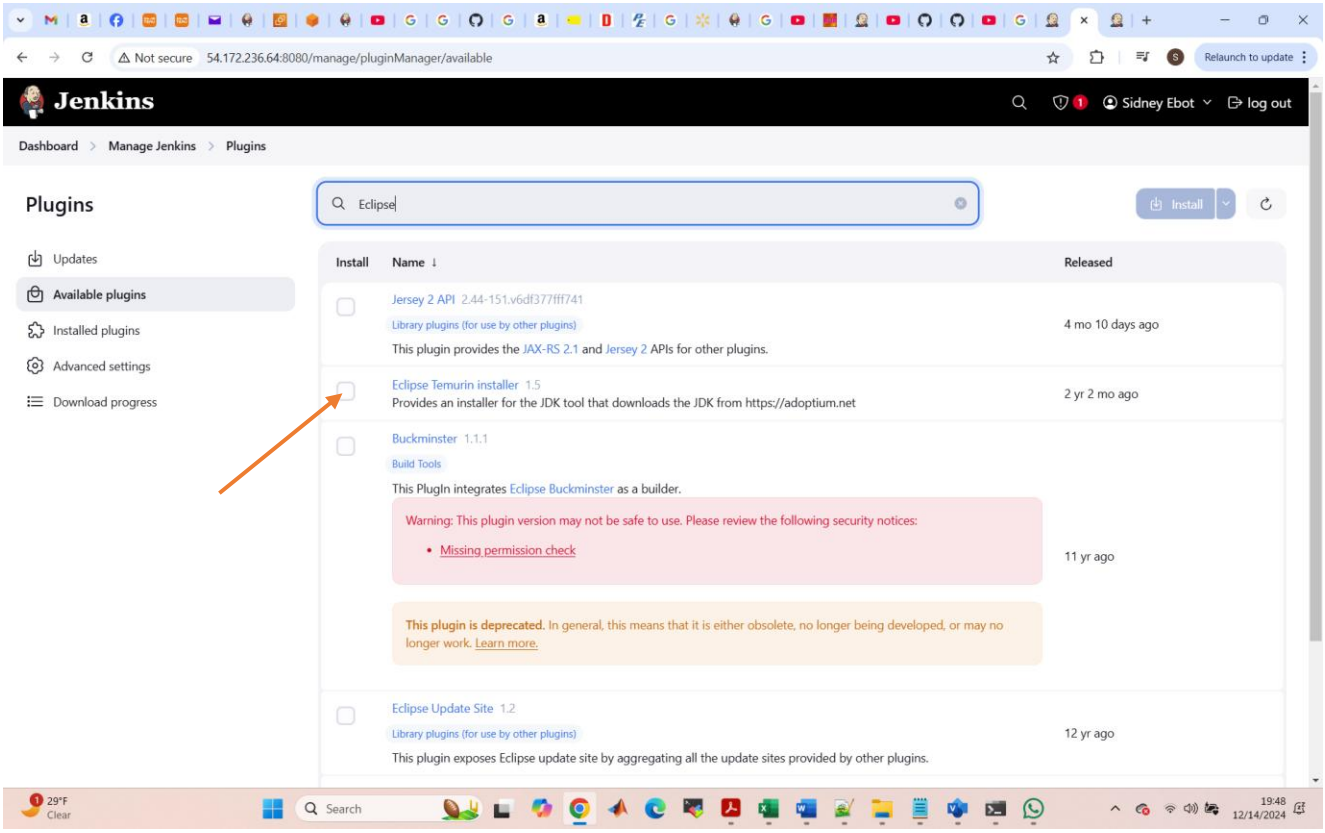
Click on "**Plugins**"



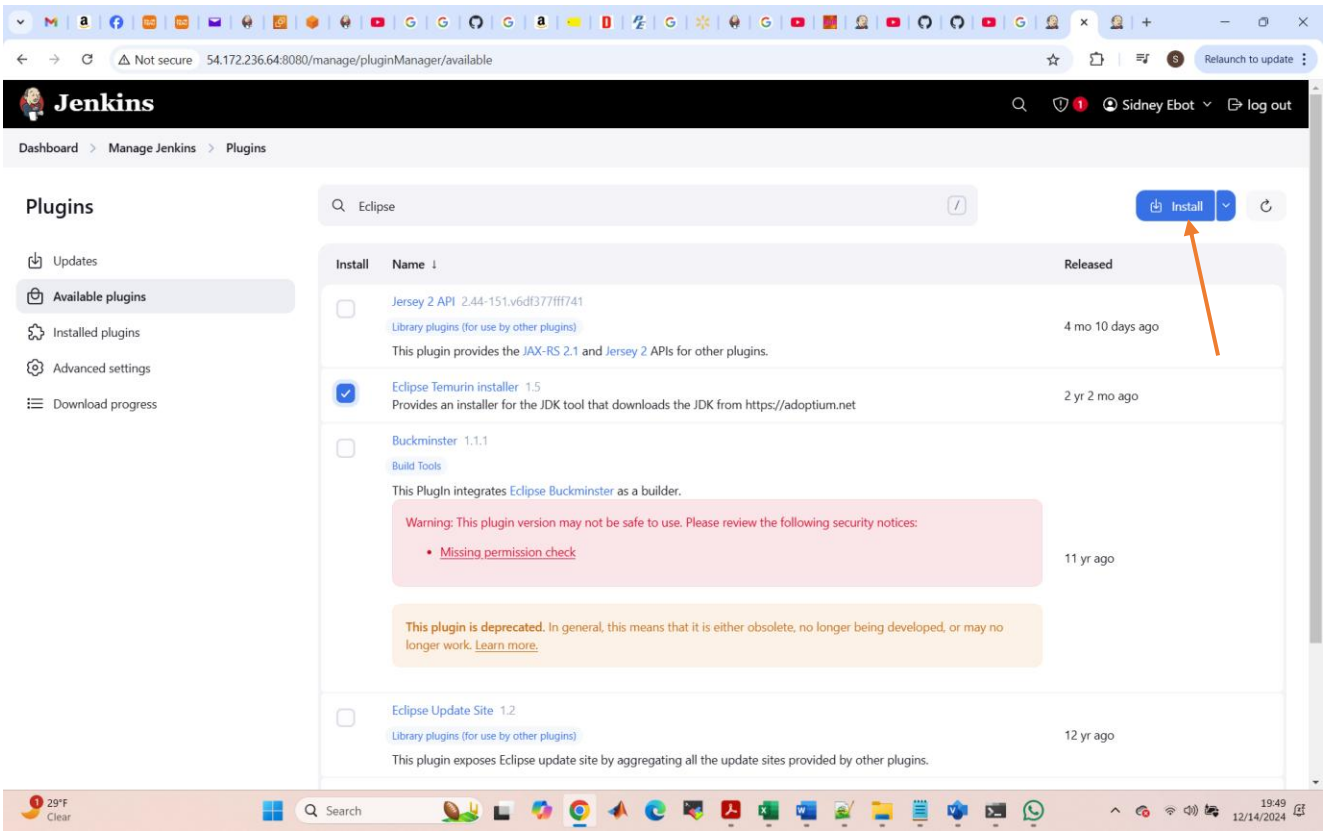
Click on "**Available Plugins**"



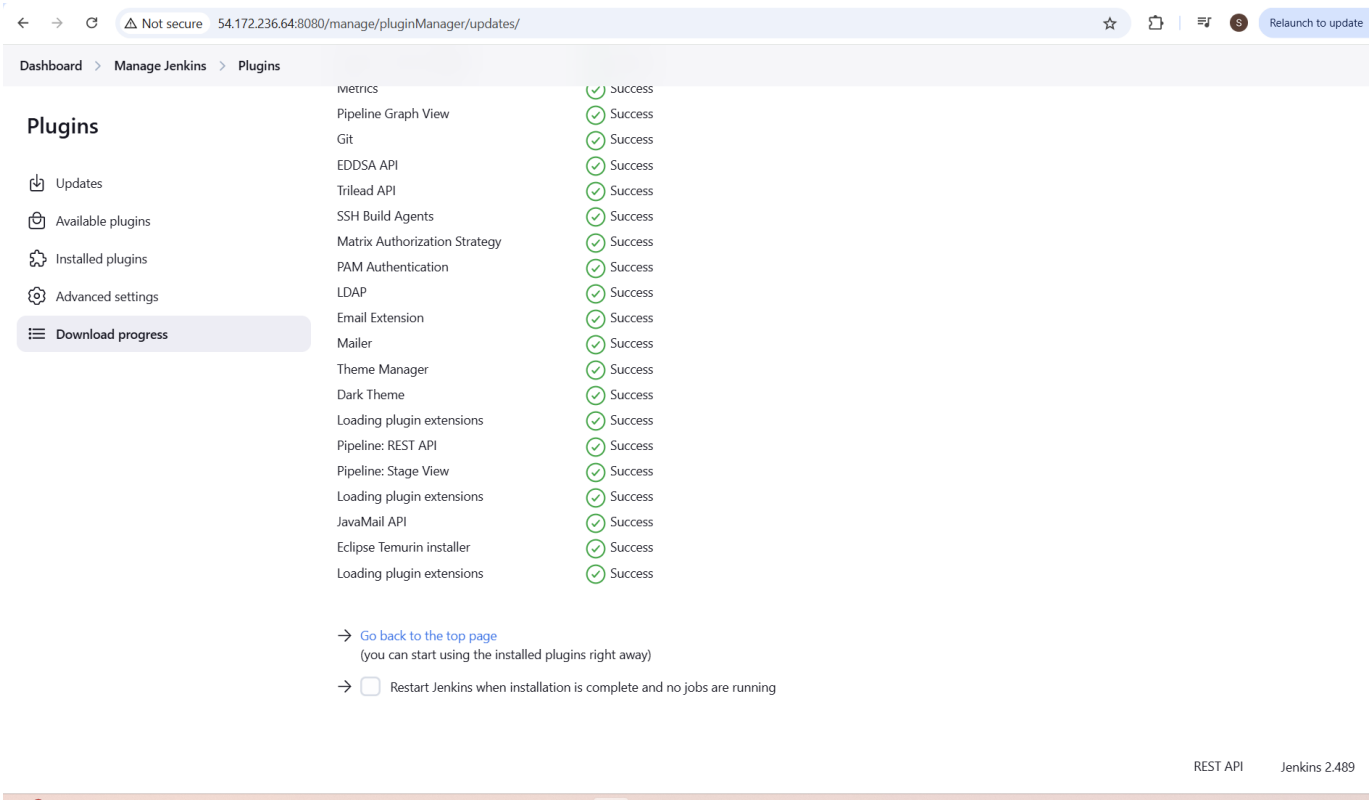
Search for "**Eclipse**"



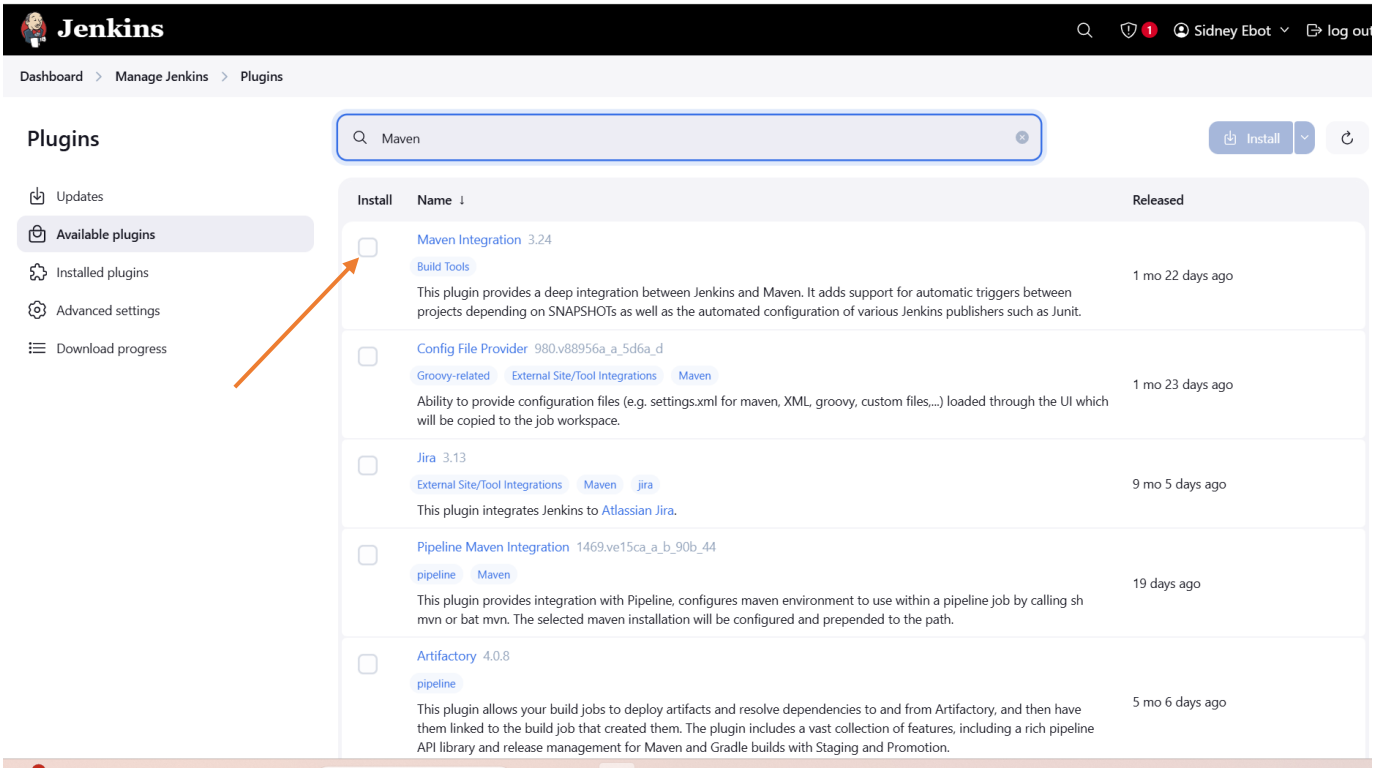
Select "**Eclipse Temurin Installer**"



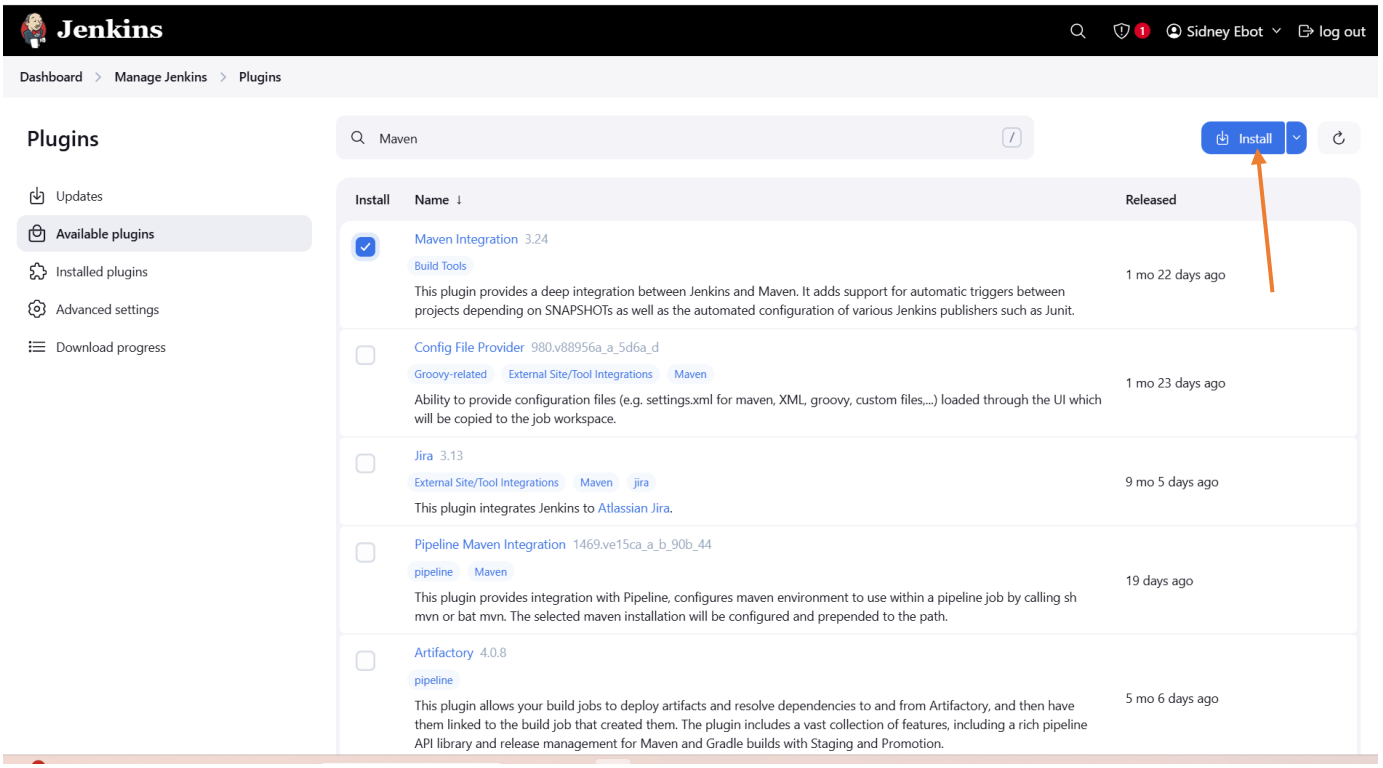
Click on "**Install**"



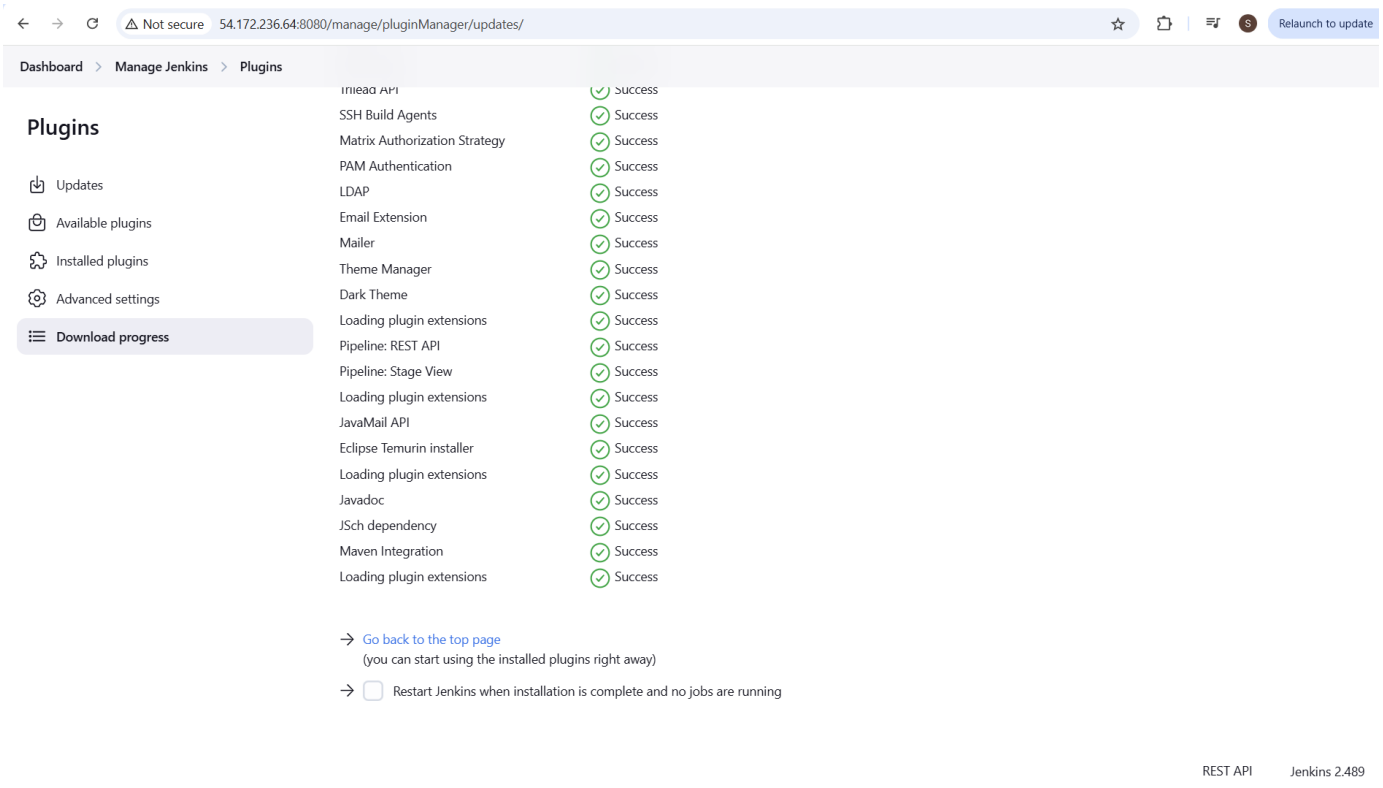
It has been installed. Also search for "**Maven**"



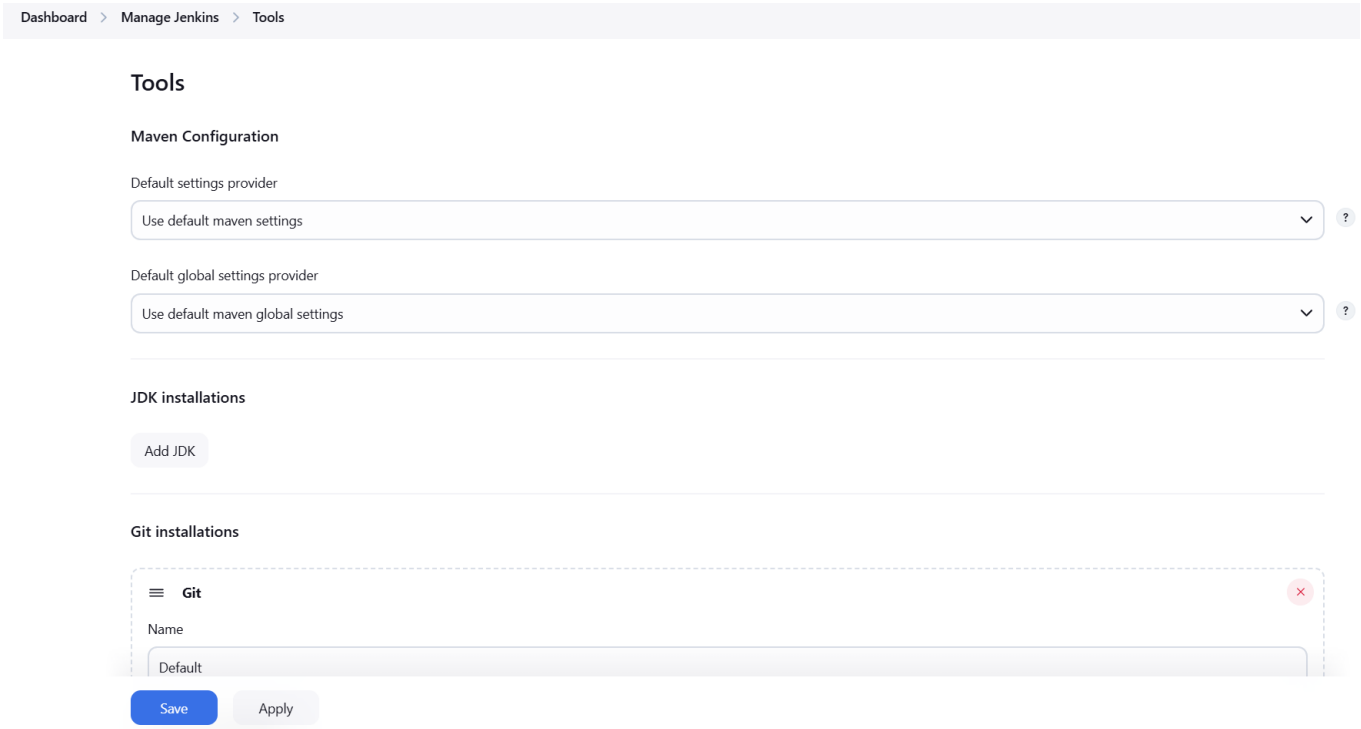
Select "**Maven Integration**"



Click on “**Install**”

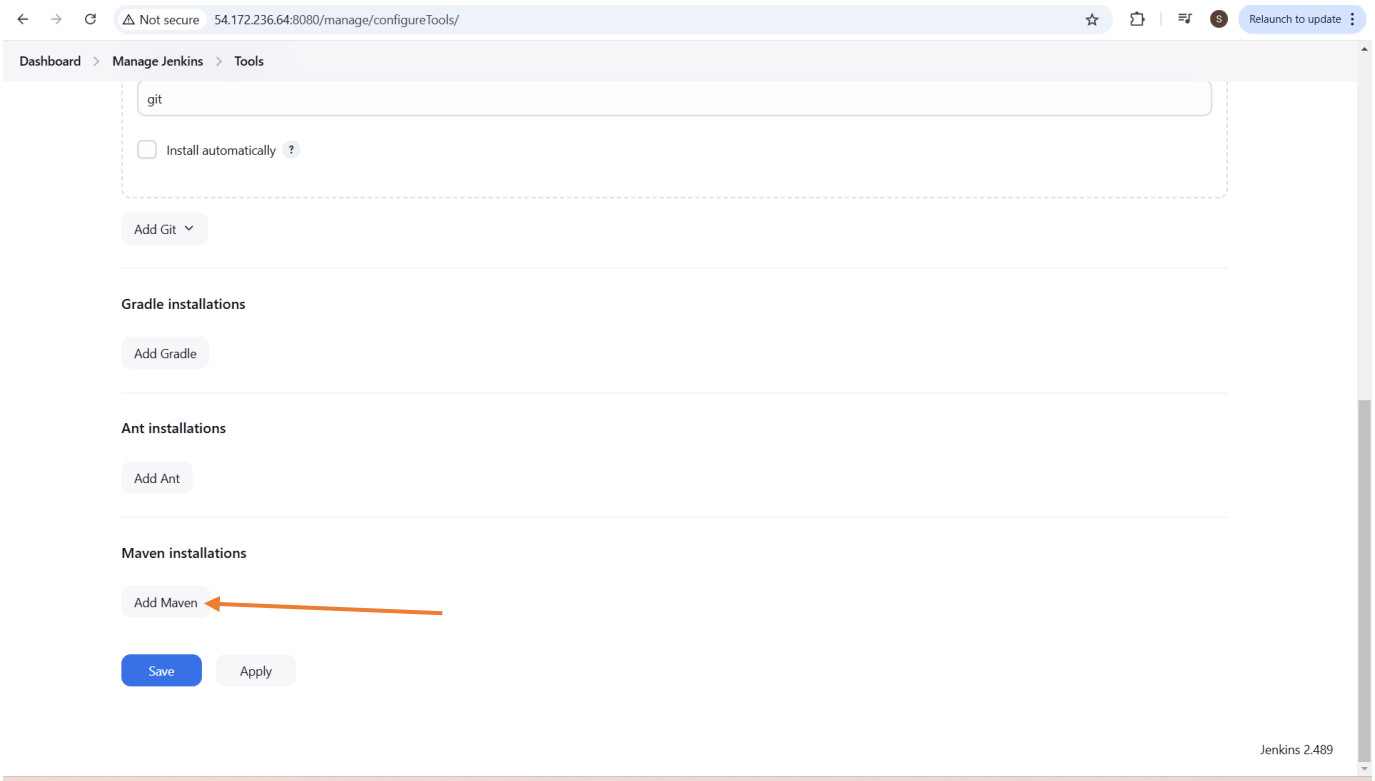


The installation is done, refresh the previous page of “**tools**”

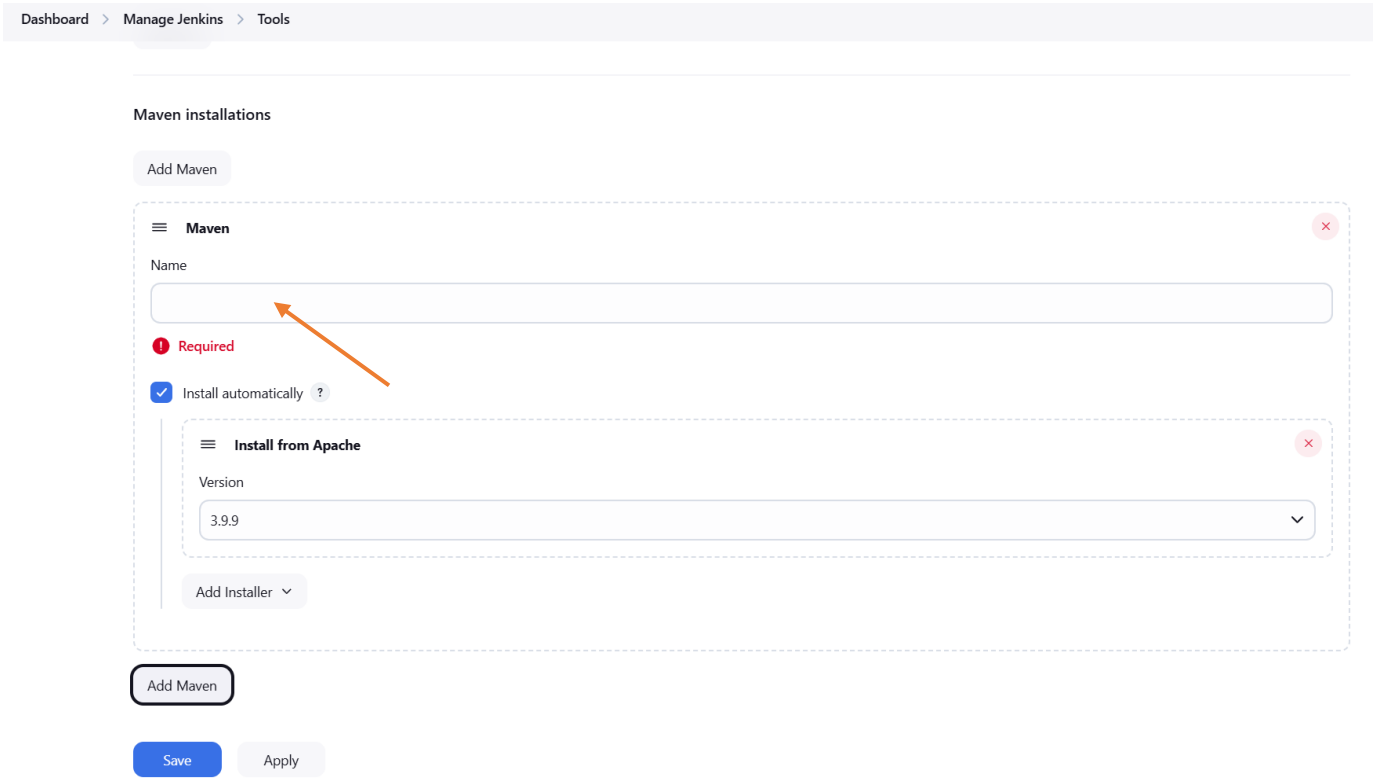


PART 3: Configure Maven Installation on Jenkins

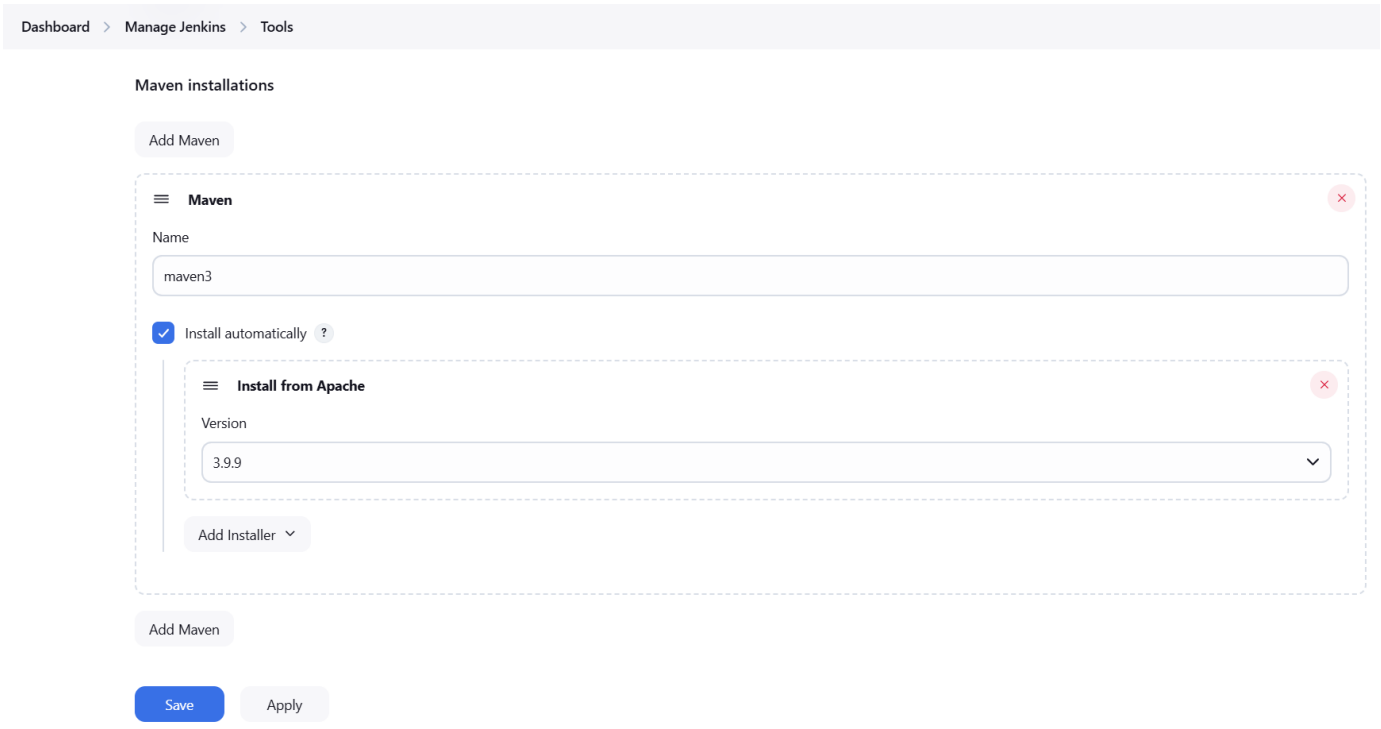
Scroll down to "**Maven Installation**"



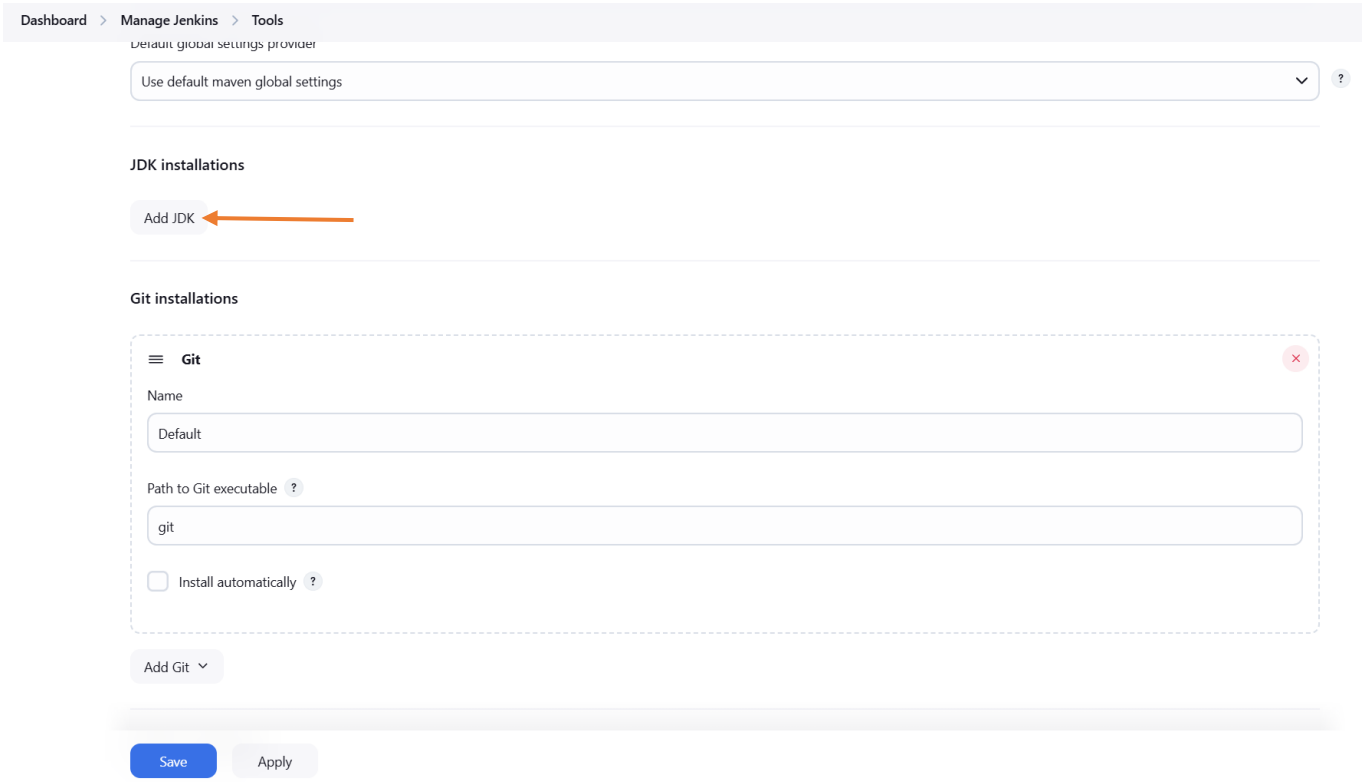
Under "Maven Installations", click on "**Add Maven**"



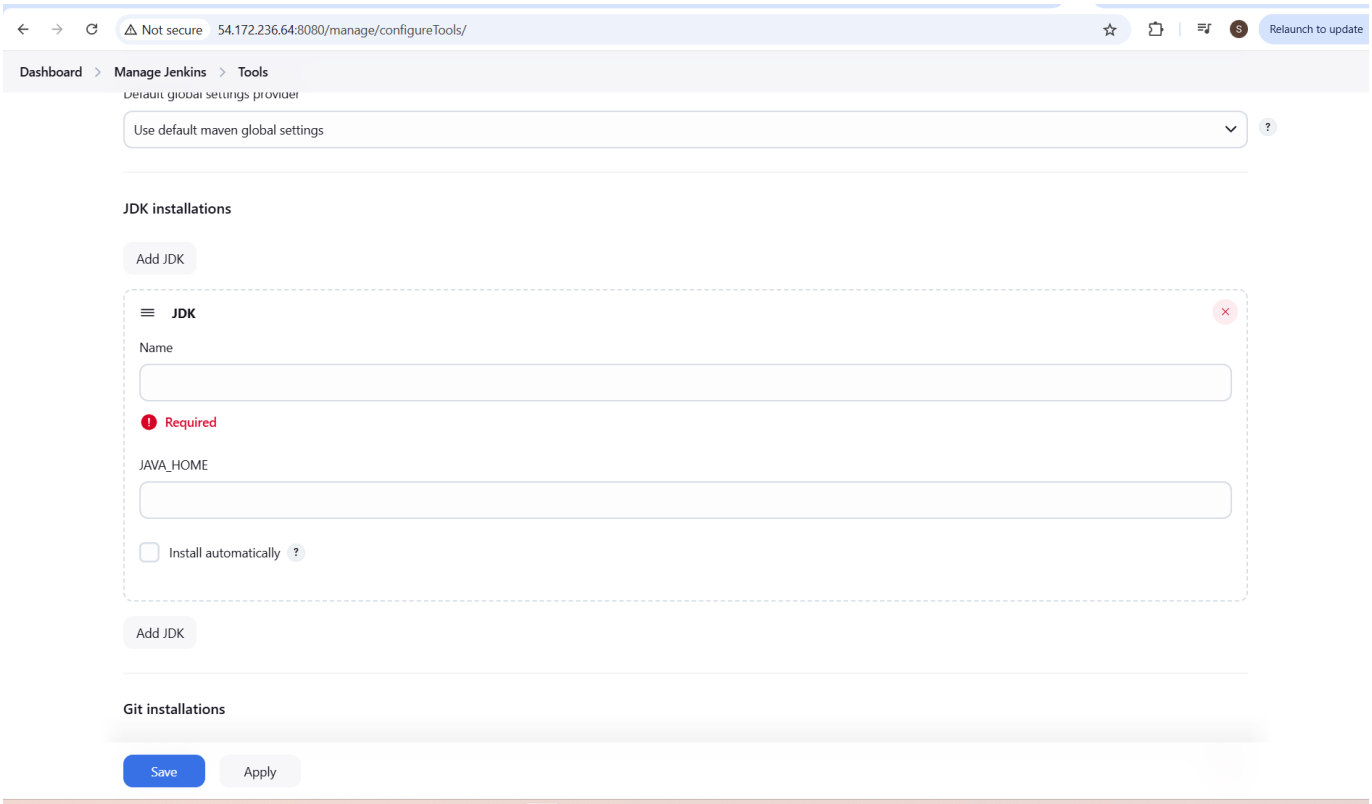
Give it name as “**maven3**”



Then scroll up to “**JDK Installations**”

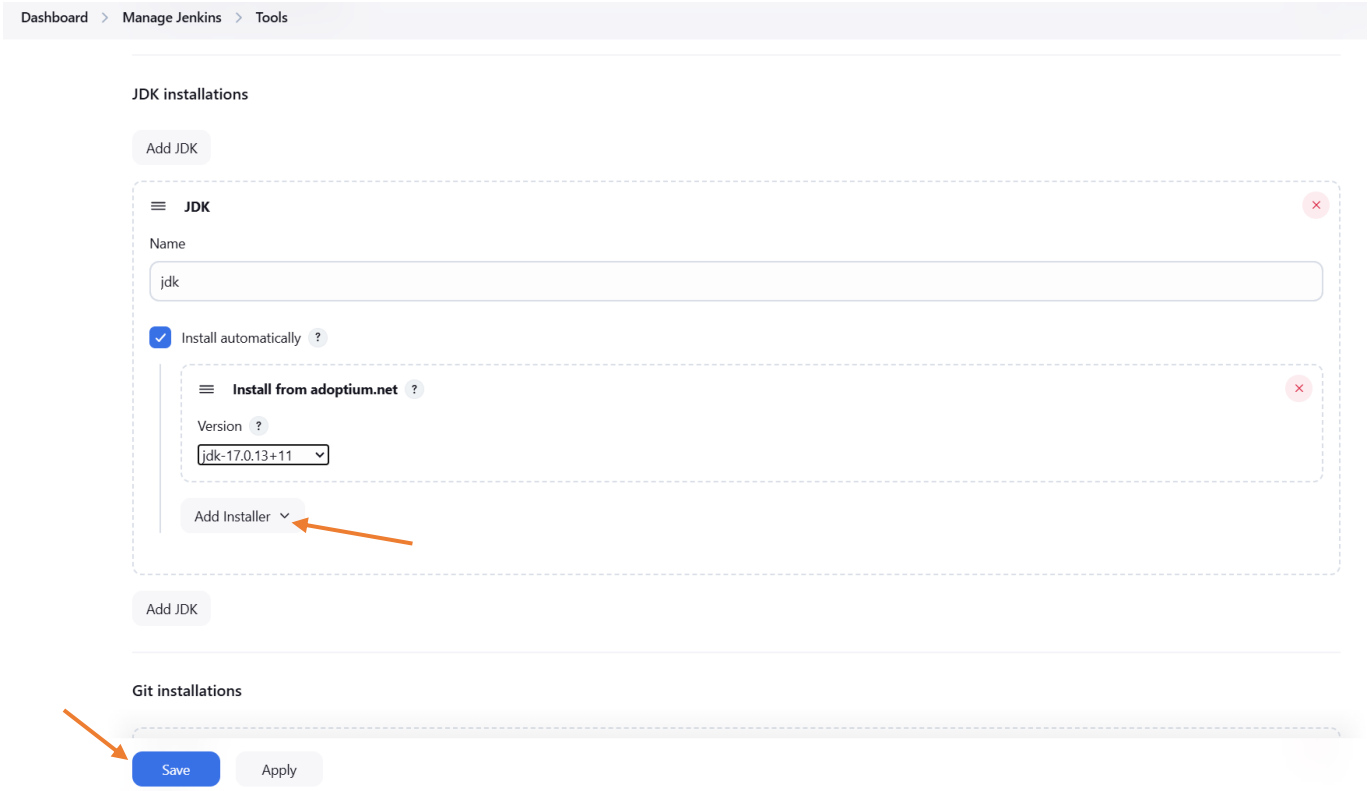


Click on “**Add JDK**”

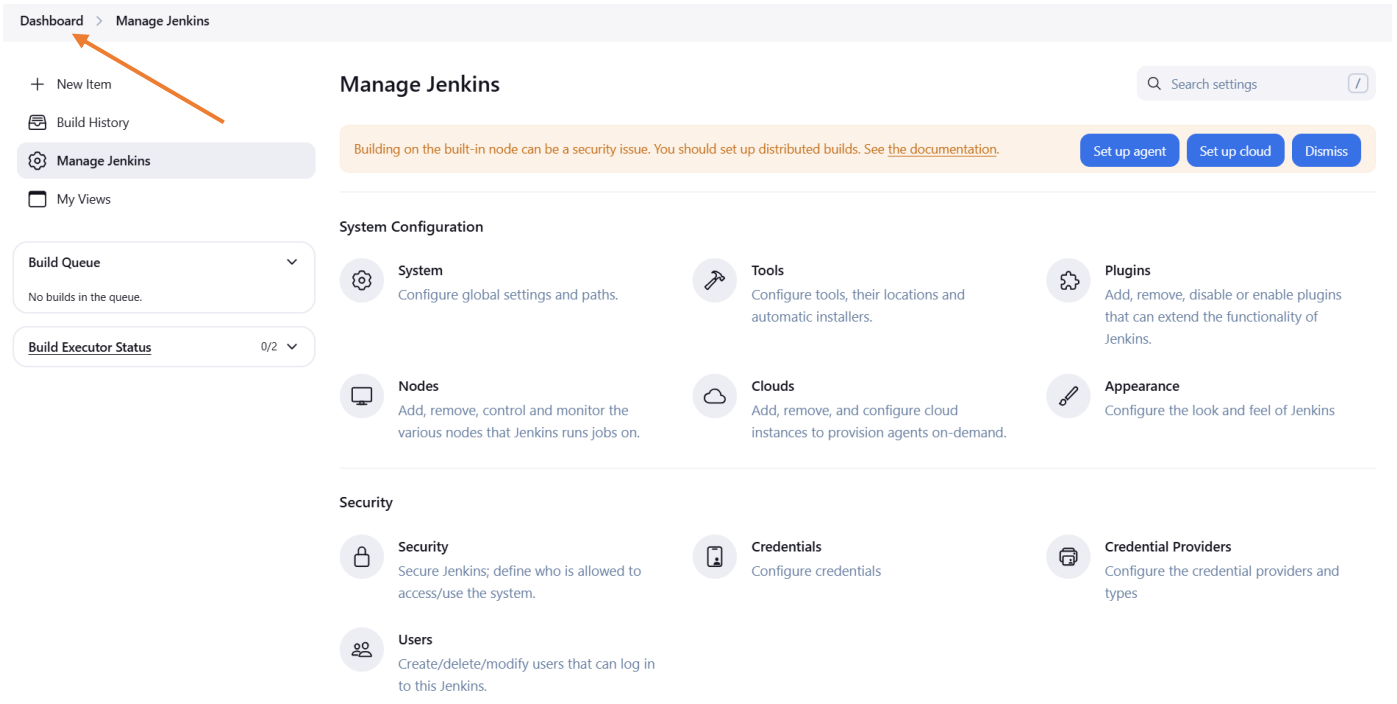


Make the naming and selection as shown below:

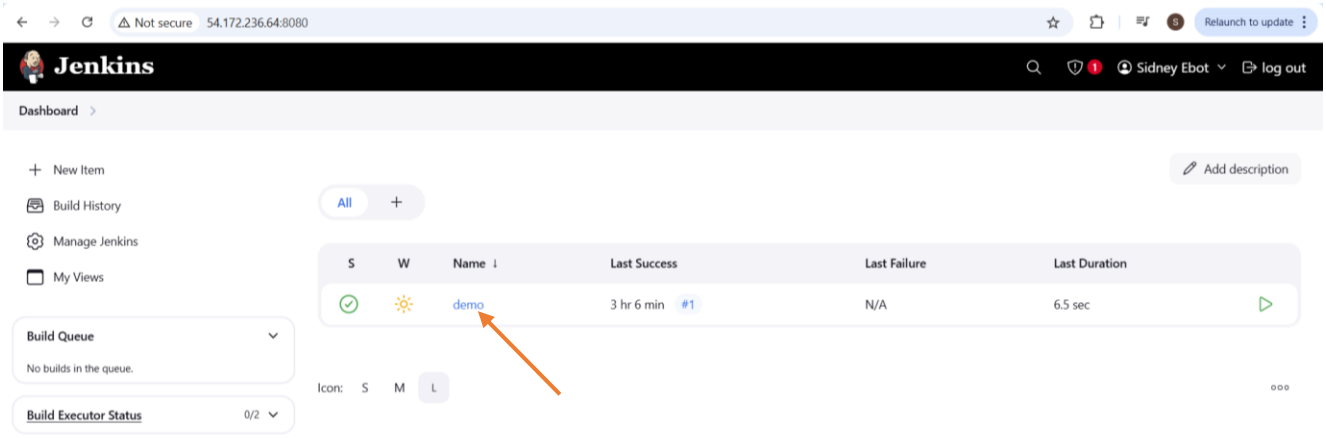
Use the name “**jdk**”, check the “**Install Automatically**” and click on the drop down on “**Add Installer**” and select “**Install from Adoptium.net**” then select “**Jdk version 17**”



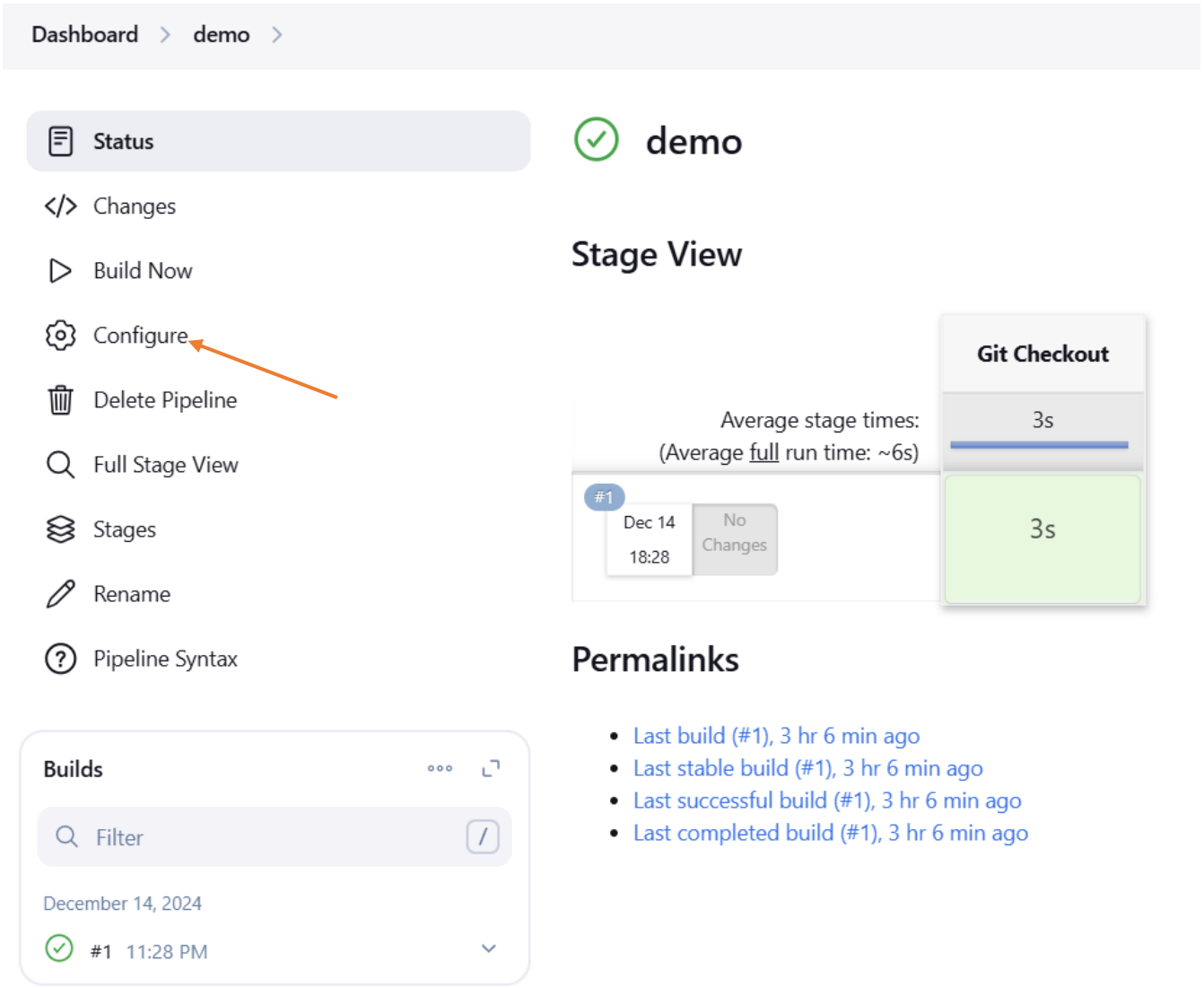
Then click on “**Save**”



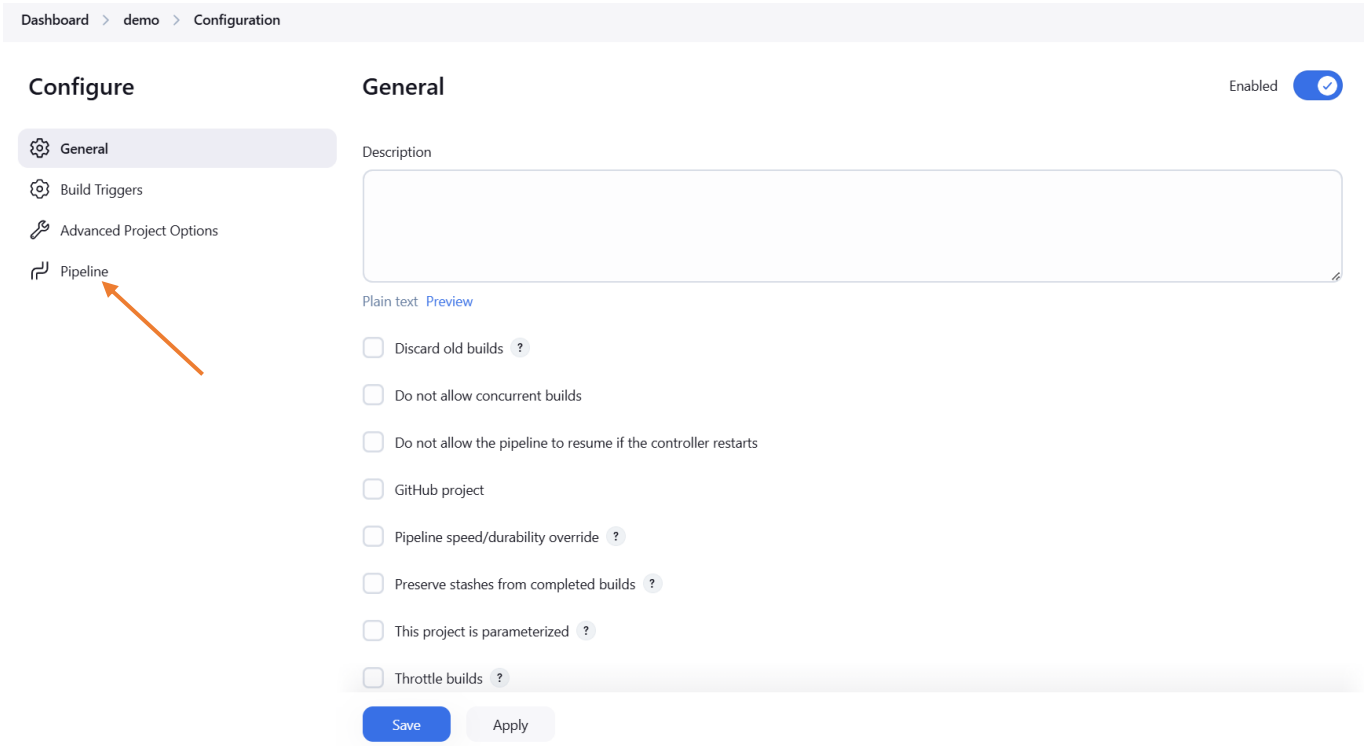
Then, go to “**Dashboard**” again



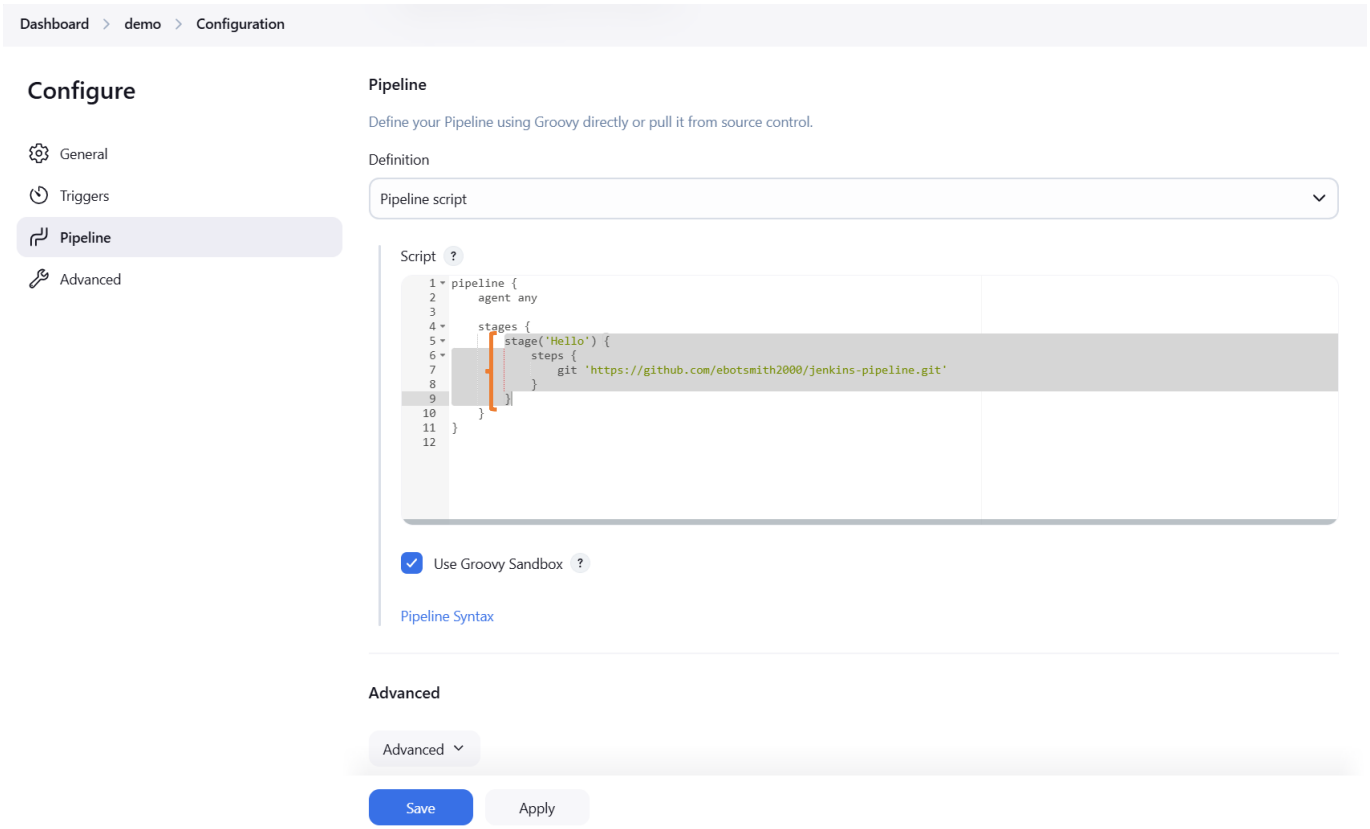
Click on the pipeline "**demo**"



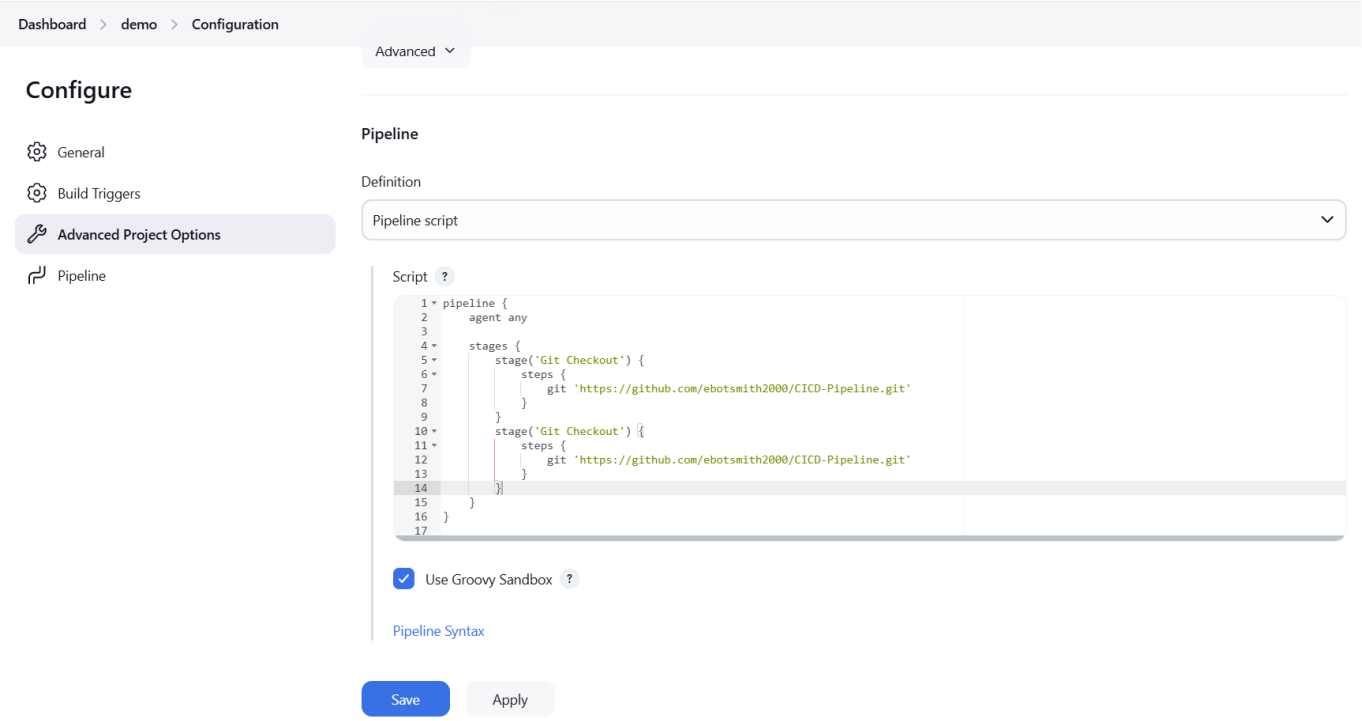
Click on "**Configure**"



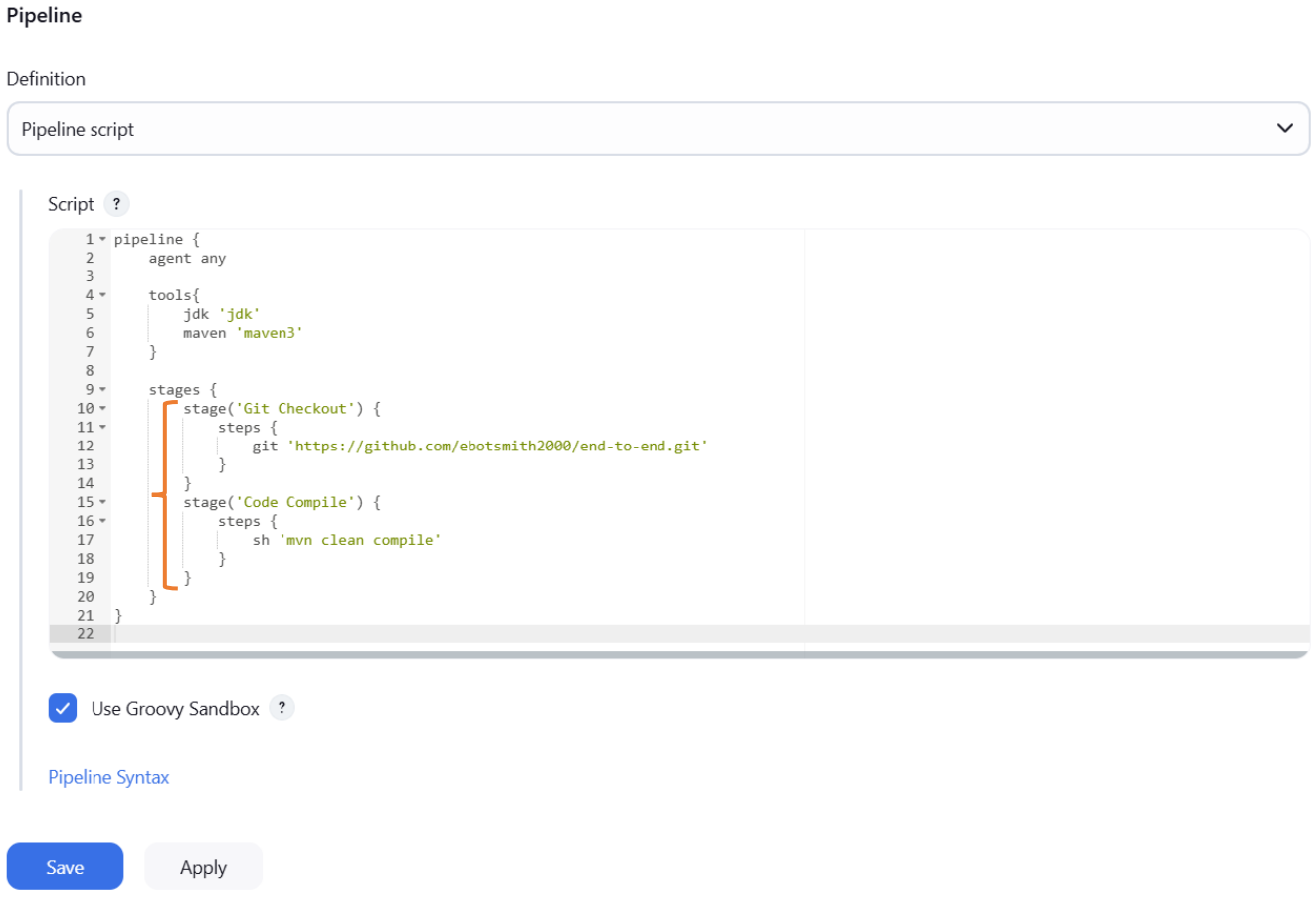
Click on “**Pipeline”** on the left-hand side to go to the code section of the pipeline or scroll down to “**Pipeline**”. Copy the stage



and paste it to have a second stage.



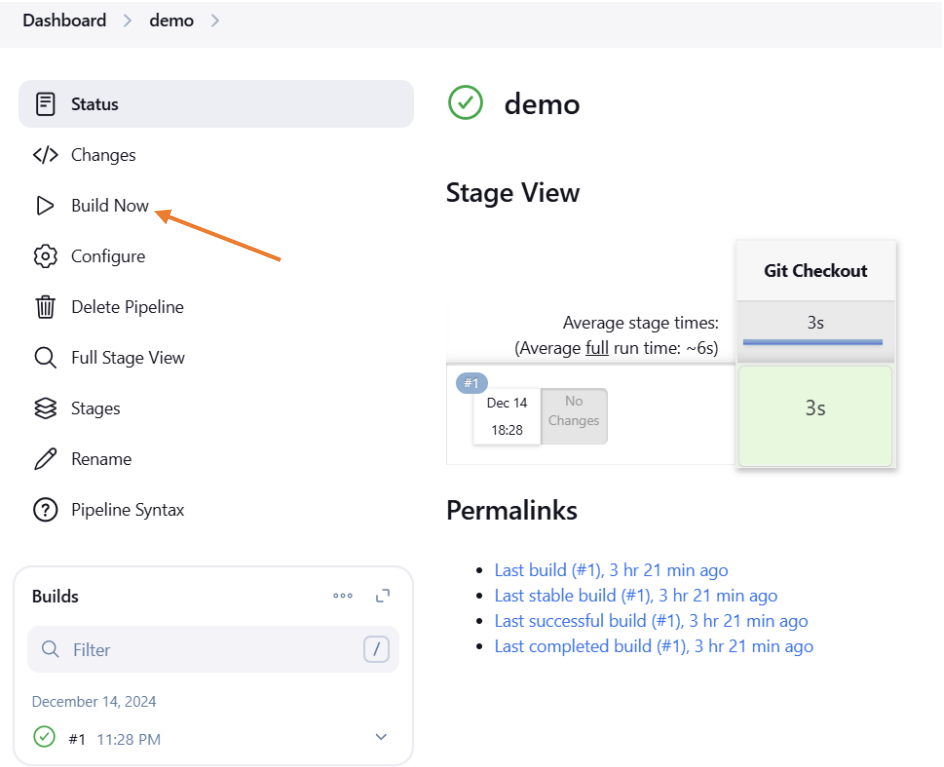
We have to configure the Maven. Before that we have installed two tools (**Maven and Java**), the tools will be configured with names as stated in tools part of the code in the "**Pipeline**"



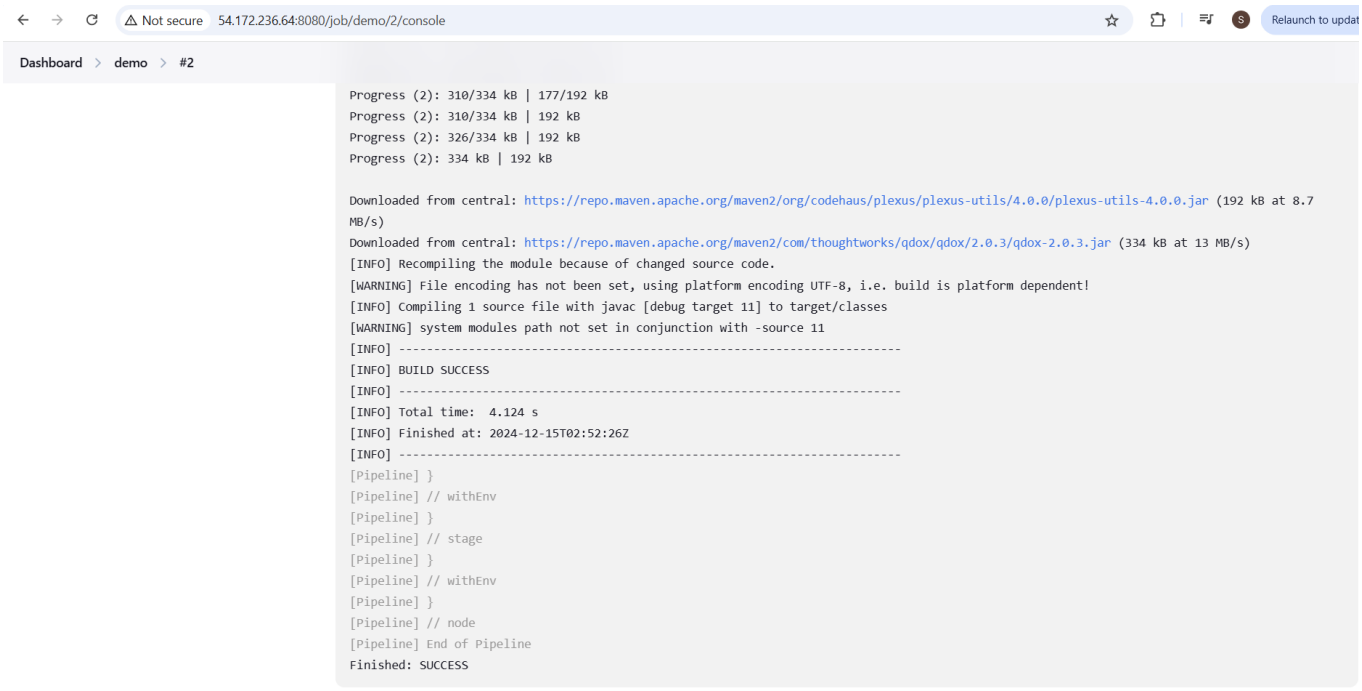
The code on the pipeline will be like below

```
pipeline {
    agent any

    tools{
        jdk 'jdk'
        maven 'maven3'
    }

    stages {
        stage('Git Checkout') {
            steps {
                git 'https://github.com/ebotsmith2000/jenkins-pipeline.git'
            }
        }
        stage('Code Compile') {
            steps {
                sh 'mvn clean compile'
            }
        }
    }
}
```

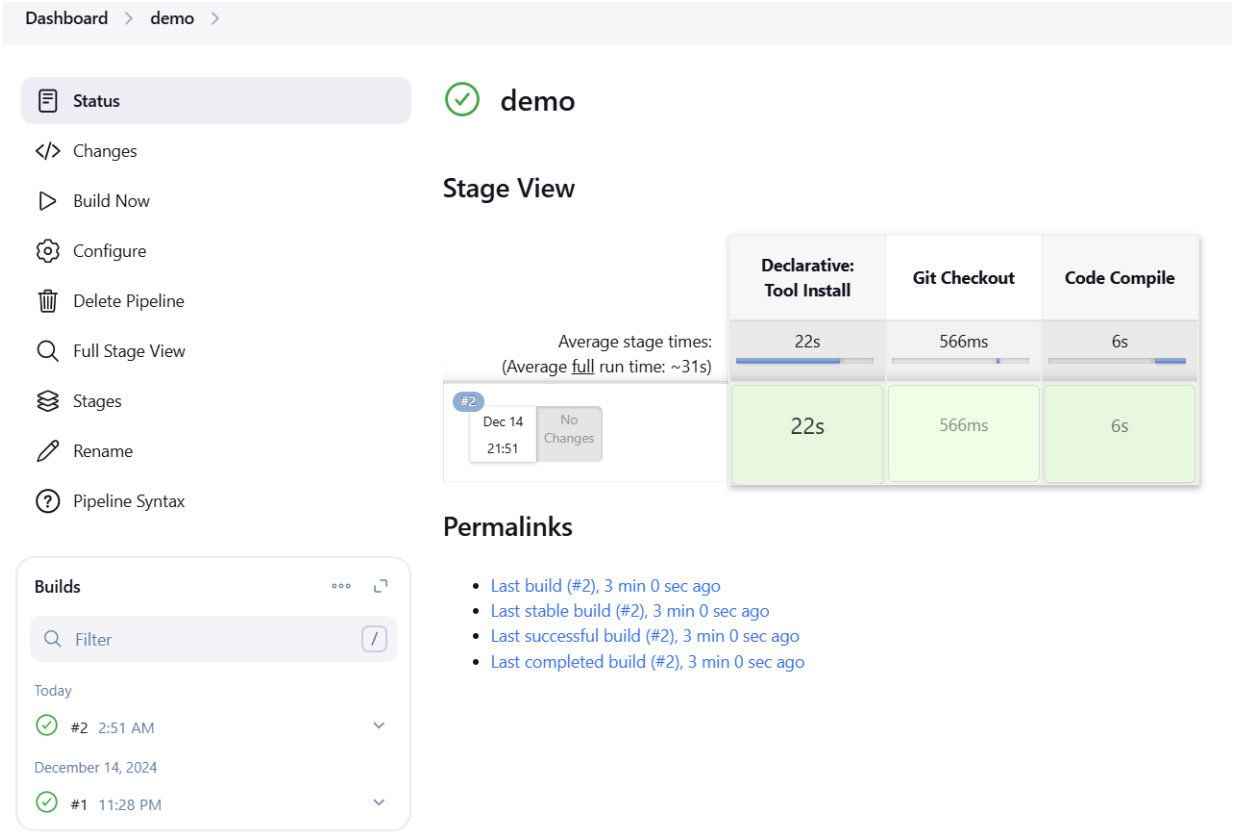
Click on "**Save**"



Let us try to build the code again by clicking on "**Build Now**"



The build is successful, it has compiled the code and everything is ready. Now, let us click on "**demo**"



Here you can see the stages "**Git Checkout**", "**Code Compile**" and an additional stage called "**Declarative Tool Install**" for the installed tools, Now, we are done with Code Compile.

STEP 11: Install and Configure SonarQube with Jenkins for Static Code Analysis

Now, we have to do “Static Code Analysis” using SonarQube.

PART 1: Install SonarQube on Server

Go to the instance terminal and run this command to download the package:

```
wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-10.2.1.78527.zip
```

```
ubuntu@ip-172-31-31-55:~$ wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.9.0.65466.zip
--2024-12-15 03:01:46--  https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.9.0.65466.zip
Resolving binaries.sonarsource.com (binaries.sonarsource.com)... 99.84.188.45, 99.84.188.106, 99.84.188.21, ...
Connecting to binaries.sonarsource.com (binaries.sonarsource.com)|99.84.188.45|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 293899334 (280M) [binary/octet-stream]
Saving to: 'sonarqube-9.9.0.65466.zip'

sonarqube-9.9.0.65466.zip                                   100%[=========================================================

2024-12-15 03:01:49 (83.4 MB/s) - 'sonarqube-9.9.0.65466.zip' saved [293899334/293899334]

ubuntu@ip-172-31-31-55:~$
```

The package has been downloaded.

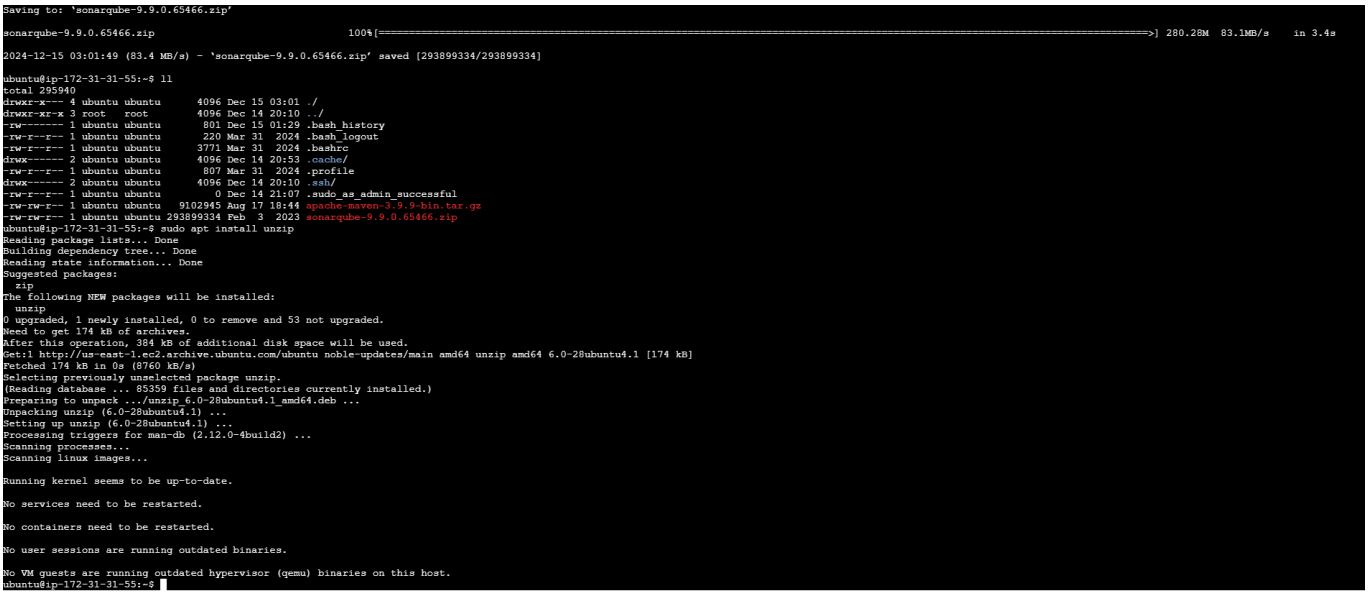
Run the command:

11

```
ubuntu@ip-172-31-31-55:~$ wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.9.0.65466.zip
--2024-12-15 03:01:46--  https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.9.0.65466.zip
Resolving binaries.sonarsource.com (binaries.sonarsource.com)... 99.84.188.45, 99.84.188.106, 99.84.188.21, ...
Connecting to binaries.sonarsource.com (binaries.sonarsource.com)|99.84.188.45|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 293899334 (280M) [binary/octet-stream]
Saving to: 'sonarqube-9.9.0.65466.zip'

sonarqube-9.9.0.65466.zip                   100%[=========================================================================>] 280.28M  83.1MB/s    in 3.4s

2024-12-15 03:01:49 (83.4 MB/s) - 'sonarqube-9.9.0.65466.zip' saved [293899334/293899334]

ubuntu@ip-172-31-31-55:~$ ll
total 295940
drwxr-x--- 4 ubuntu ubuntu      4096 Dec 15 03:01 ./
drwxr-xr-x 3 root   root        4096 Dec 14 20:10 ../
-rw------- 1 ubuntu ubuntu       801 Dec 15 01:29 .bash_history
-rw-r--r-- 1 ubuntu ubuntu       220 Mar 31  2024 .bash_logout
-rw-r--r-- 1 ubuntu ubuntu      3771 Mar 31  2024 .bashrc
drwx------ 2 ubuntu ubuntu      4096 Dec 14 20:53 .cache/
-rw-r--r-- 1 ubuntu ubuntu       807 Mar 31  2024 .profile
drwx------ 2 ubuntu ubuntu      4096 Dec 14 20:10 .ssh/
-rw-r--r-- 1 ubuntu ubuntu         0 Dec 14 21:07 .sudo_as_admin_successful
-rw-rw-r-- 1 ubuntu ubuntu   9102945 Aug 17 18:44 apache-maven-3.9.9-bin.tar.gz
-rw-rw-r-- 1 ubuntu ubuntu 293899334 Feb  3  2023 sonarqube-9.9.0.65466.zip
ubuntu@ip-172-31-31-55:~$
```

You can see the SonarQube package, we have to unzip the zipped folders. So, we have to run a command to install unzip package.

`sudo apt install unzip`

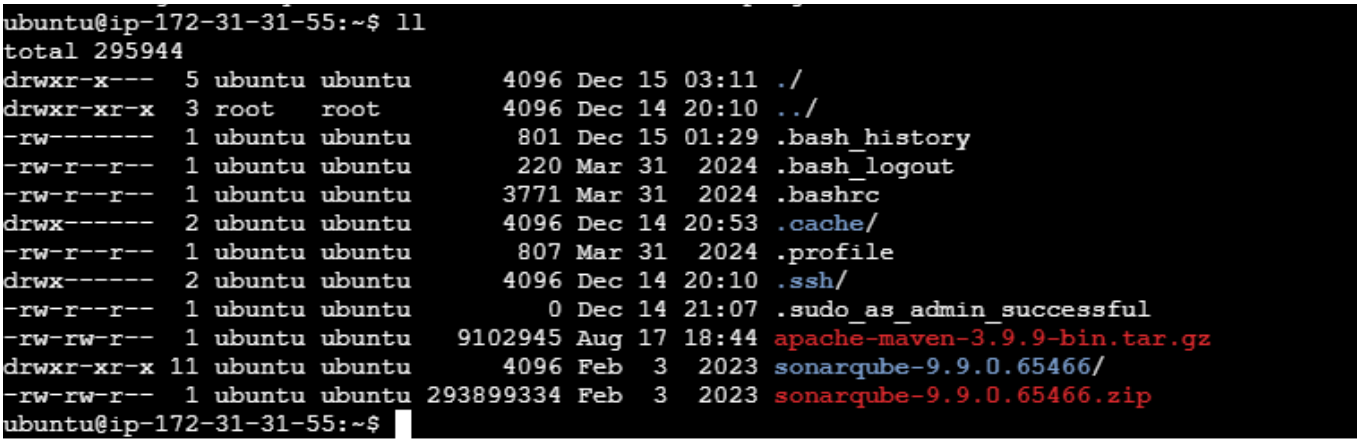


The installation is done, let us now unzip by using the command:

unzip sonarqube-10.2.1.78527.zip

It has extracted. Now, if you run the command:

ll



You can see that we have the package SonarQube. Navigate to the package SonarQube by using the command:

cd sonarqube-10.2.1.78527/

```
ubuntu@ip-172-31-31-55:~$ ll
total 295944
drwxr-x---  5 ubuntu ubuntu      4096 Dec 15 03:11 ./
drwxr-xr-x  3 root   root        4096 Dec 14 20:10 ../
-rw-------  1 ubuntu ubuntu       801 Dec 15 01:29 .bash_history
-rw-r--r--  1 ubuntu ubuntu       220 Mar 31  2024 .bash_logout
-rw-r--r--  1 ubuntu ubuntu      3771 Mar 31  2024 .bashrc
drwx------  2 ubuntu ubuntu      4096 Dec 14 20:53 .cache/
-rw-r--r--  1 ubuntu ubuntu       807 Mar 31  2024 .profile
drwx------  2 ubuntu ubuntu      4096 Dec 14 20:10 .ssh/
-rw-r--r--  1 ubuntu ubuntu         0 Dec 14 21:07 .sudo_as_admin_successful
-rw-rw-r--  1 ubuntu ubuntu   9102945 Aug 17 18:44 apache-maven-3.9.9-bin.tar.gz
drwxr-xr-x 11 ubuntu ubuntu      4096 Feb  3  2023 sonarqube-9.9.0.65466/
-rw-rw-r--  1 ubuntu ubuntu 293899334 Feb  3  2023 sonarqube-9.9.0.65466.zip
ubuntu@ip-172-31-31-55:~$ cd sonarqube-9.9.0.65466/
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466$
```

Run the command:

ll

```
ubuntu@ip-172-31-31-55:~$ cd sonarqube-9.9.0.65466/
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466$ ll
total 128
drwxr-xr-x 11 ubuntu ubuntu  4096 Feb  3  2023 ./
drwxr-x---  5 ubuntu ubuntu  4096 Dec 15 03:11 ../
-rw-r--r--  1 ubuntu ubuntu  7651 Feb  3  2023 COPYING
drwxr-xr-x  6 ubuntu ubuntu  4096 Feb  3  2023 bin/
drwxr-xr-x  2 ubuntu ubuntu  4096 Feb  3  2023 conf/
drwxr-xr-x  2 ubuntu ubuntu  4096 Feb  3  2023 data/
-rw-r--r--  1 ubuntu ubuntu 77588 Feb  3  2023 dependency-license.json
drwxr-xr-x  7 ubuntu ubuntu  4096 Feb  3  2023 elasticsearch/
drwxr-xr-x  4 ubuntu ubuntu  4096 Feb  3  2023 extensions/
drwxr-xr-x  5 ubuntu ubuntu  4096 Feb  3  2023 lib/
drwxr-xr-x  2 ubuntu ubuntu  4096 Feb  3  2023 logs/
drwxr-xr-x  2 ubuntu ubuntu  4096 Feb  3  2023 temp/
drwxr-xr-x  5 ubuntu ubuntu  4096 Feb  3  2023 web/
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466$
```

We have all the SonarQube files in a bin, no matter which application it is, it will always be in the bin. Navigate to the bin by using the command:

cd bin/

```
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466$ ll
total 128
drwxr-xr-x 11 ubuntu ubuntu  4096 Feb  3  2023 ./
drwxr-x---  5 ubuntu ubuntu  4096 Dec 15 03:11 ../
-rw-r--r--  1 ubuntu ubuntu  7651 Feb  3  2023 COPYING
drwxr-xr-x  6 ubuntu ubuntu  4096 Feb  3  2023 bin/
drwxr-xr-x  2 ubuntu ubuntu  4096 Feb  3  2023 conf/
drwxr-xr-x  2 ubuntu ubuntu  4096 Feb  3  2023 data/
-rw-r--r--  1 ubuntu ubuntu 77588 Feb  3  2023 dependency-license.json
drwxr-xr-x  7 ubuntu ubuntu  4096 Feb  3  2023 elasticsearch/
drwxr-xr-x  4 ubuntu ubuntu  4096 Feb  3  2023 extensions/
drwxr-xr-x  5 ubuntu ubuntu  4096 Feb  3  2023 lib/
drwxr-xr-x  2 ubuntu ubuntu  4096 Feb  3  2023 logs/
drwxr-xr-x  2 ubuntu ubuntu  4096 Feb  3  2023 temp/
drwxr-xr-x  5 ubuntu ubuntu  4096 Feb  3  2023 web/
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466$ cd bin/
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466/bin$
```

Then run the command:

ll

```
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466$ cd bin/
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466/bin$ ll
total 24
drwxr-xr-x  6 ubuntu ubuntu 4096 Feb  3  2023 ./
drwxr-xr-x 11 ubuntu ubuntu 4096 Feb  3  2023 ../
drwxr-xr-x  2 ubuntu ubuntu 4096 Feb  3  2023 linux-x86-64/
drwxr-xr-x  2 ubuntu ubuntu 4096 Feb  3  2023 macosx-universal-64/
drwxr-xr-x  3 ubuntu ubuntu 4096 Feb  3  2023 windows-x86-64/
drwxr-xr-x  2 ubuntu ubuntu 4096 Feb  3  2023 winsw-license/
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466/bin$
```

Then navigate to linux-x86-64, using the command:

cd linux-x86-64/

```
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466$ cd bin/
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466/bin$ ll
total 24
drwxr-xr-x  6 ubuntu ubuntu 4096 Feb  3  2023 ./
drwxr-xr-x 11 ubuntu ubuntu 4096 Feb  3  2023 ../
drwxr-xr-x  2 ubuntu ubuntu 4096 Feb  3  2023 linux-x86-64/
drwxr-xr-x  2 ubuntu ubuntu 4096 Feb  3  2023 macosx-universal-64/
drwxr-xr-x  3 ubuntu ubuntu 4096 Feb  3  2023 windows-x86-64/
drwxr-xr-x  2 ubuntu ubuntu 4096 Feb  3  2023 winsw-license/
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466/bin$ cd linux-x86-64/
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466/bin/linux-x86-64$
```

Then run the command:

ll

```
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466$ cd bin/
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466/bin$ ll
total 24
drwxr-xr-x  6 ubuntu ubuntu 4096 Feb  3  2023 ./
drwxr-xr-x 11 ubuntu ubuntu 4096 Feb  3  2023 ../
drwxr-xr-x  2 ubuntu ubuntu 4096 Feb  3  2023 linux-x86-64/
drwxr-xr-x  2 ubuntu ubuntu 4096 Feb  3  2023 macosx-universal-64/
drwxr-xr-x  3 ubuntu ubuntu 4096 Feb  3  2023 windows-x86-64/
drwxr-xr-x  2 ubuntu ubuntu 4096 Feb  3  2023 winsw-license/
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466/bin$ cd linux-x86-64/
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466/bin/linux-x86-64$ ll
total 16
drwxr-xr-x 2 ubuntu ubuntu 4096 Feb  3  2023 ./
drwxr-xr-x 6 ubuntu ubuntu 4096 Feb  3  2023 ../
-rwxr-xr-x 1 ubuntu ubuntu 7133 Feb  3  2023 sonar.sh*
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466/bin/linux-x86-64$
```

Run the command:

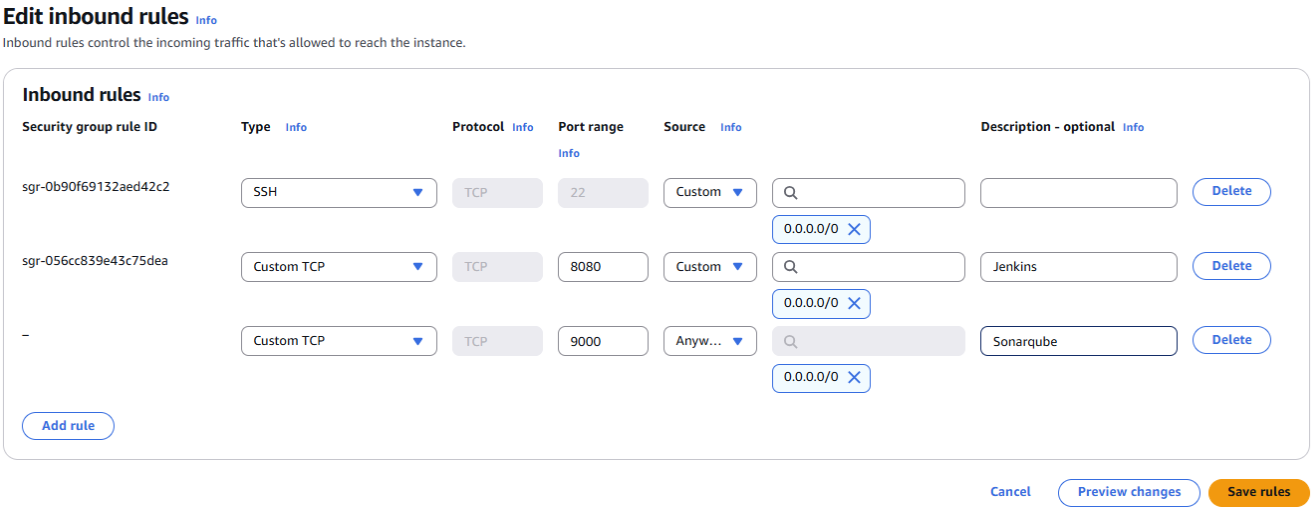
`./sonar.sh start`

```
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466/bin$ cd linux-x86-64/
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466/bin/linux-x86-64$ ll
total 16
drwxr-xr-x 2 ubuntu ubuntu 4096 Feb  3  2023 ./
drwxr-xr-x 6 ubuntu ubuntu 4096 Feb  3  2023 ../
-rwxr-xr-x 1 ubuntu ubuntu 7133 Feb  3  2023 sonar.sh*
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466/bin/linux-x86-64$ ./sonar.sh start
/usr/bin/java
Starting SonarQube...
Started SonarQube.
ubuntu@ip-172-31-31-55:~/sonarqube-9.9.0.65466/bin/linux-x86-64$
```

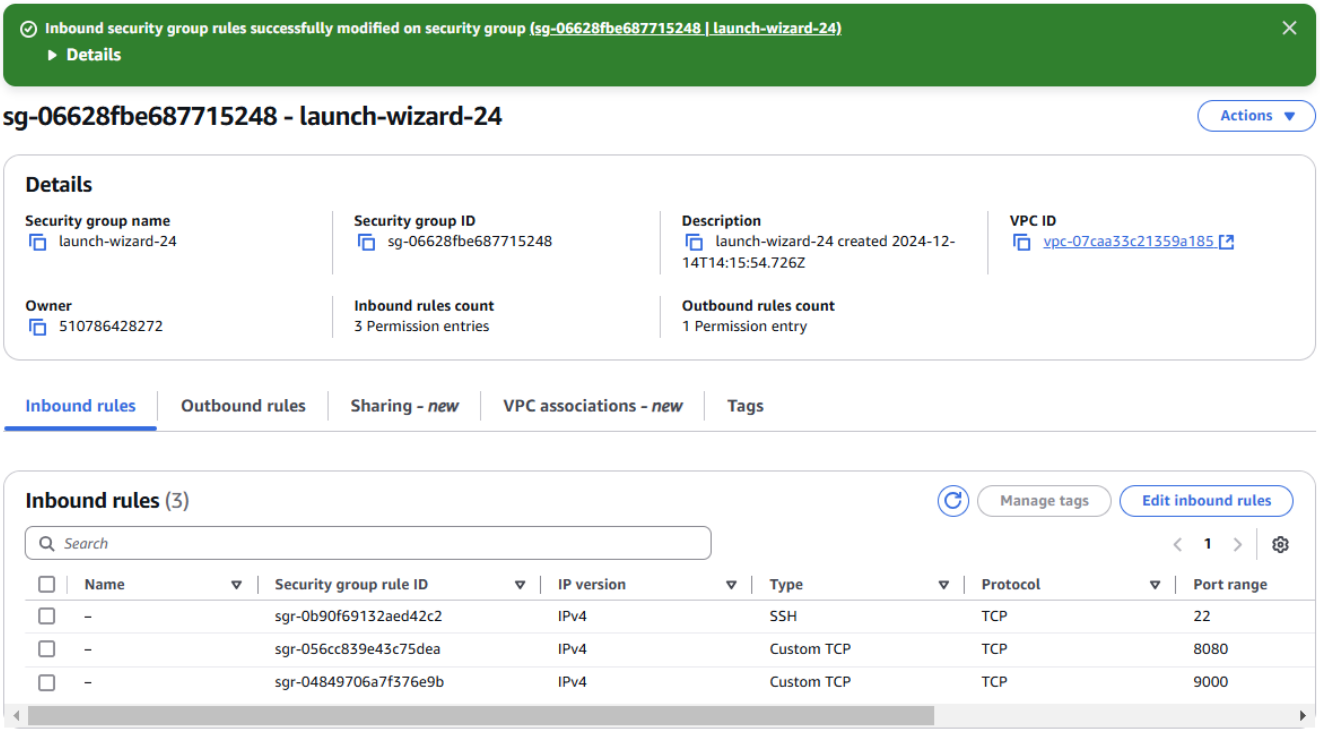
So, SonarQube has started.

PART 2: Add Port 9000 to the Instance

By default, SonarQube will be working on **PORT 9000**. We have to add this PORT to the inbound rule of out instance.



Click on "**Save Rules**"

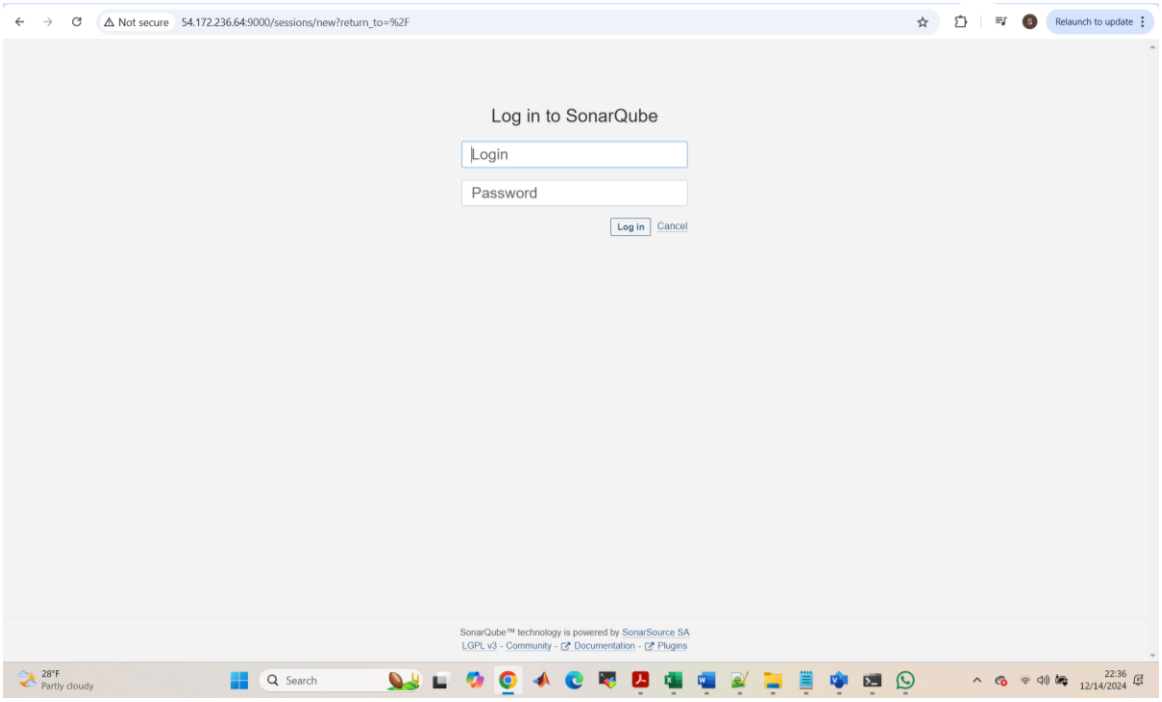


PART 3: Access SonarQube on Browser

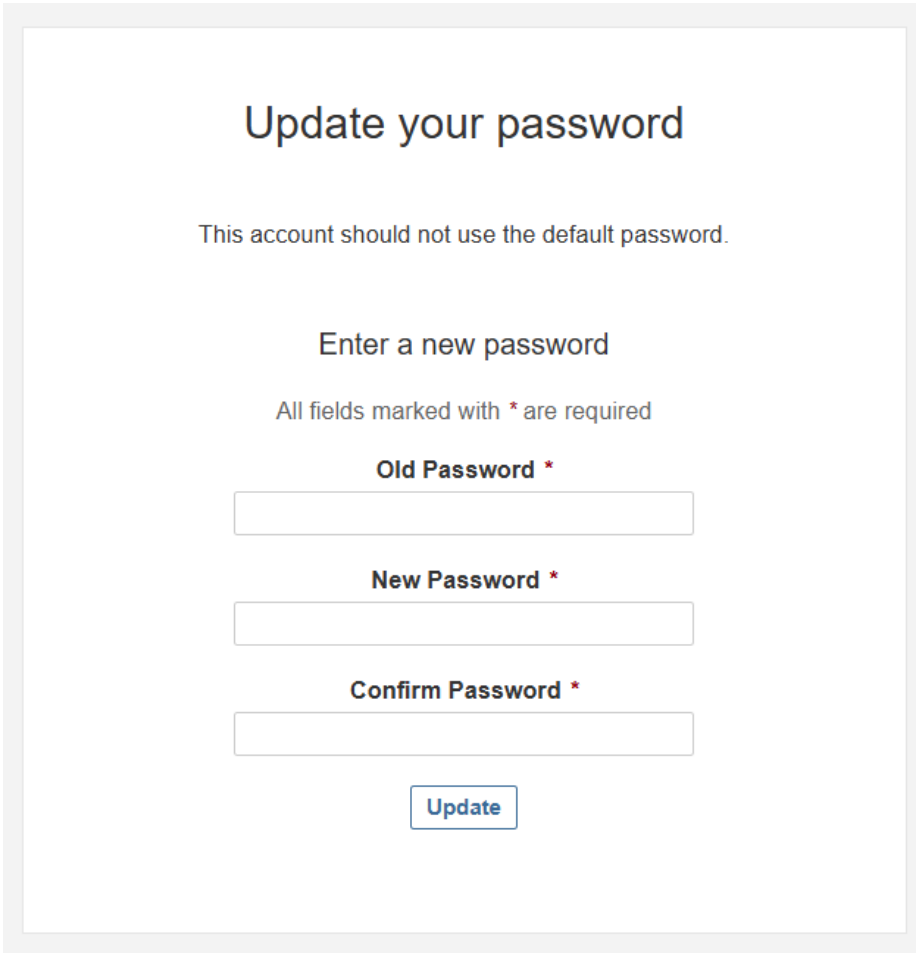
So, go to the browser and paste

`http://<Public IPv4 Address>:9000`

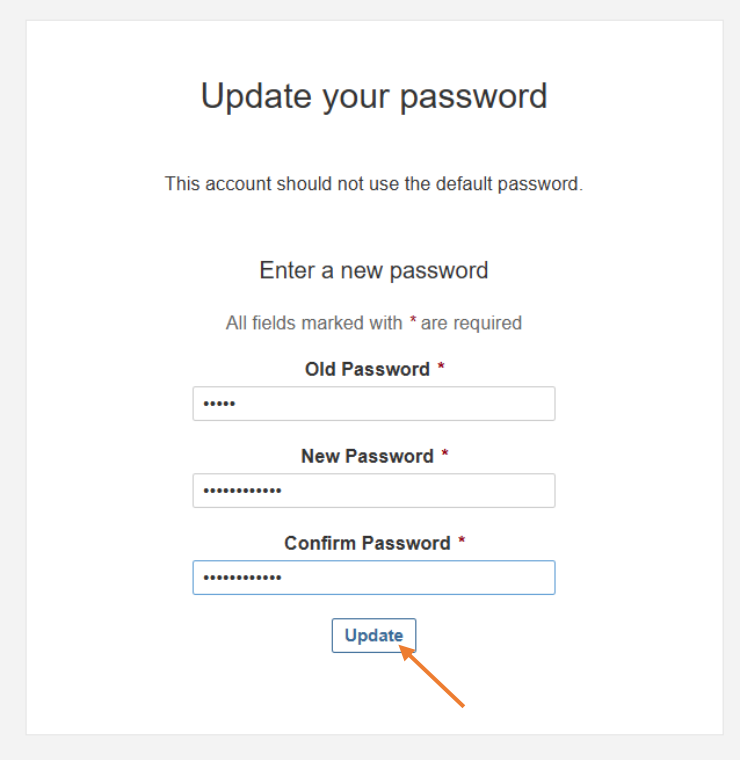
That is http://54.198.237.25:9000



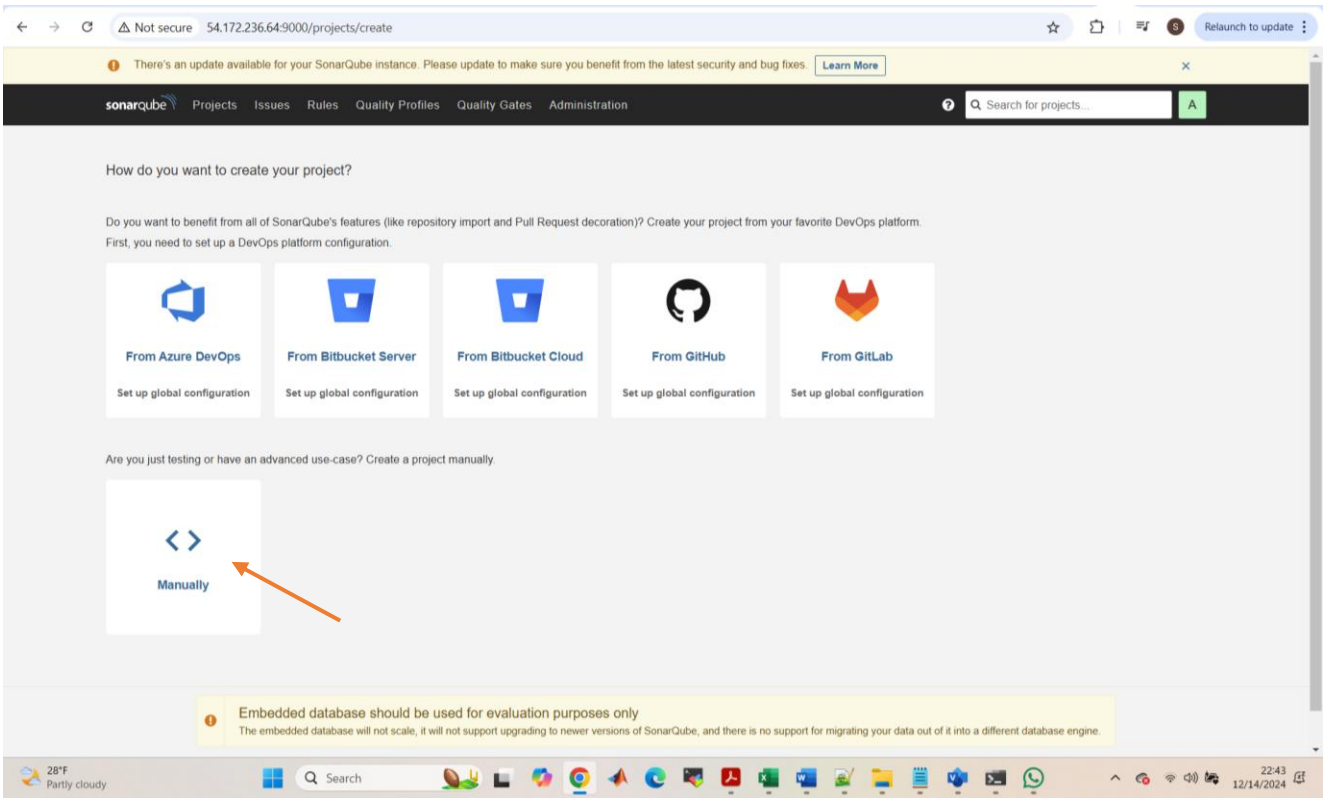
The default username is “**admin**” and password is “**admin**”. Use this to login



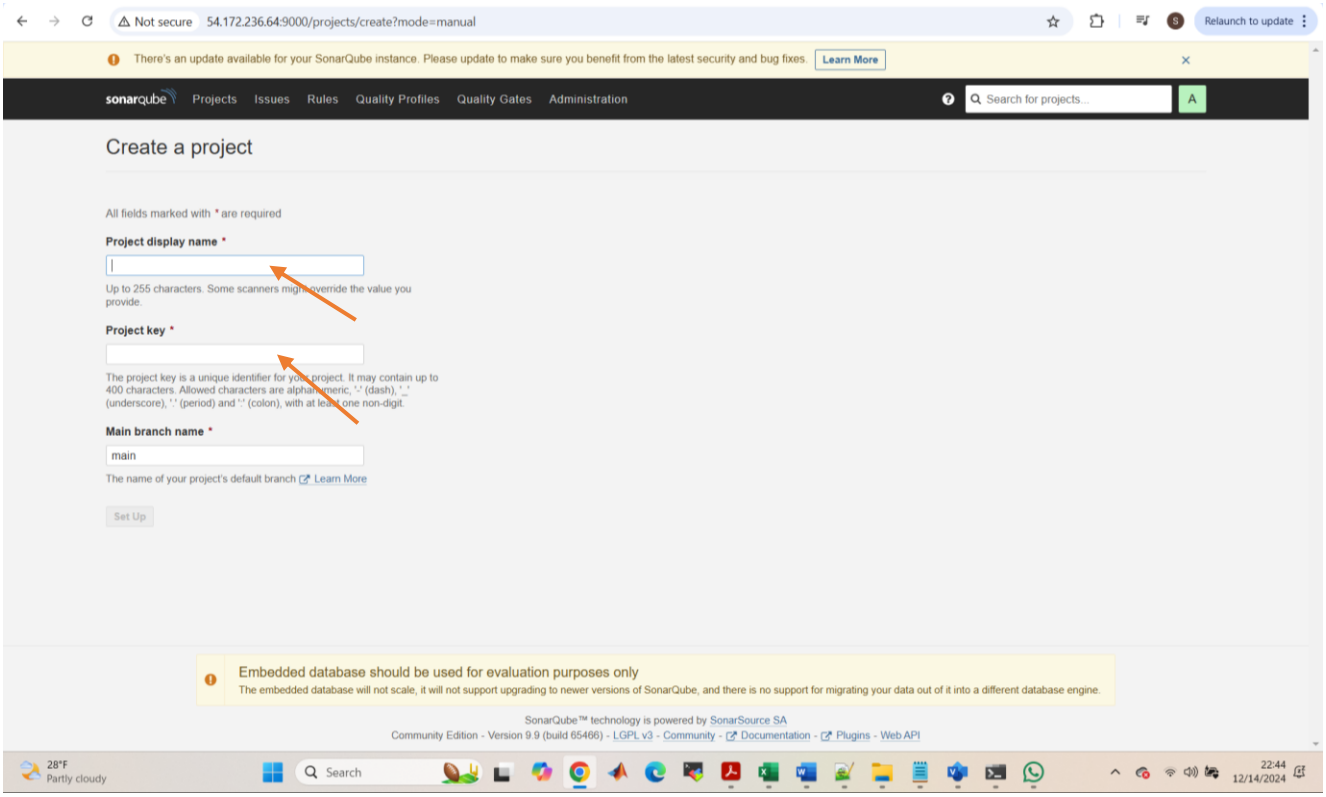
We are requested to change the password. I will use “**sosoebot**” as new password



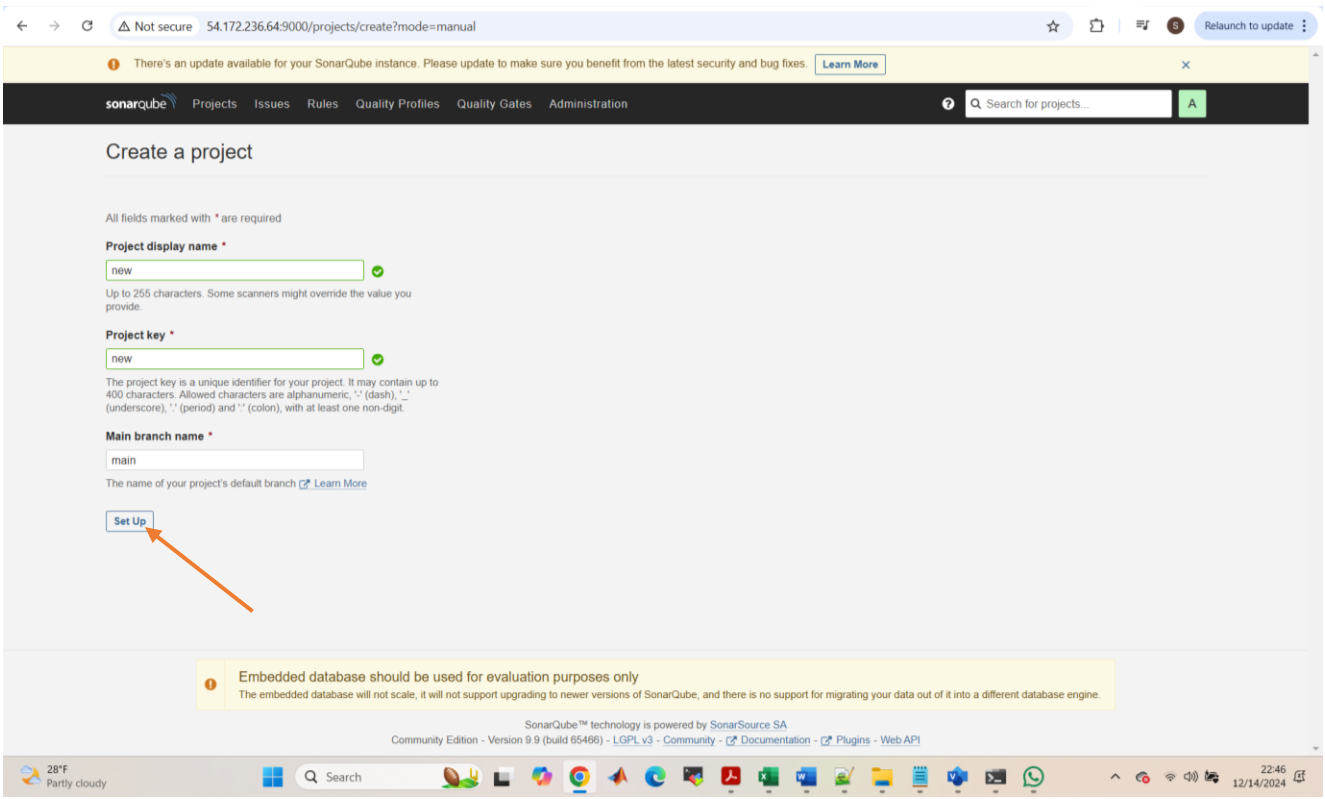
Click on “**Update**”



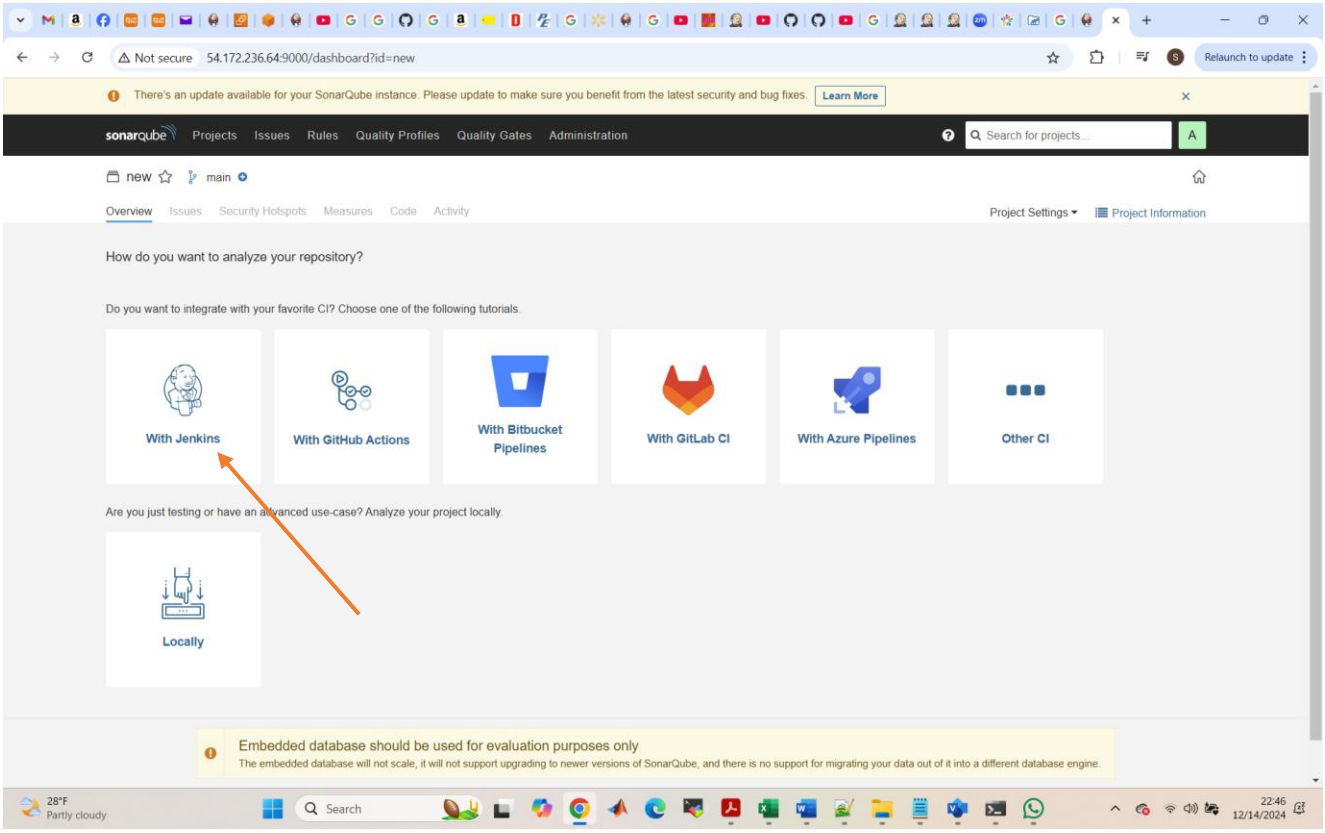
It is ready, click on “**Manually**”



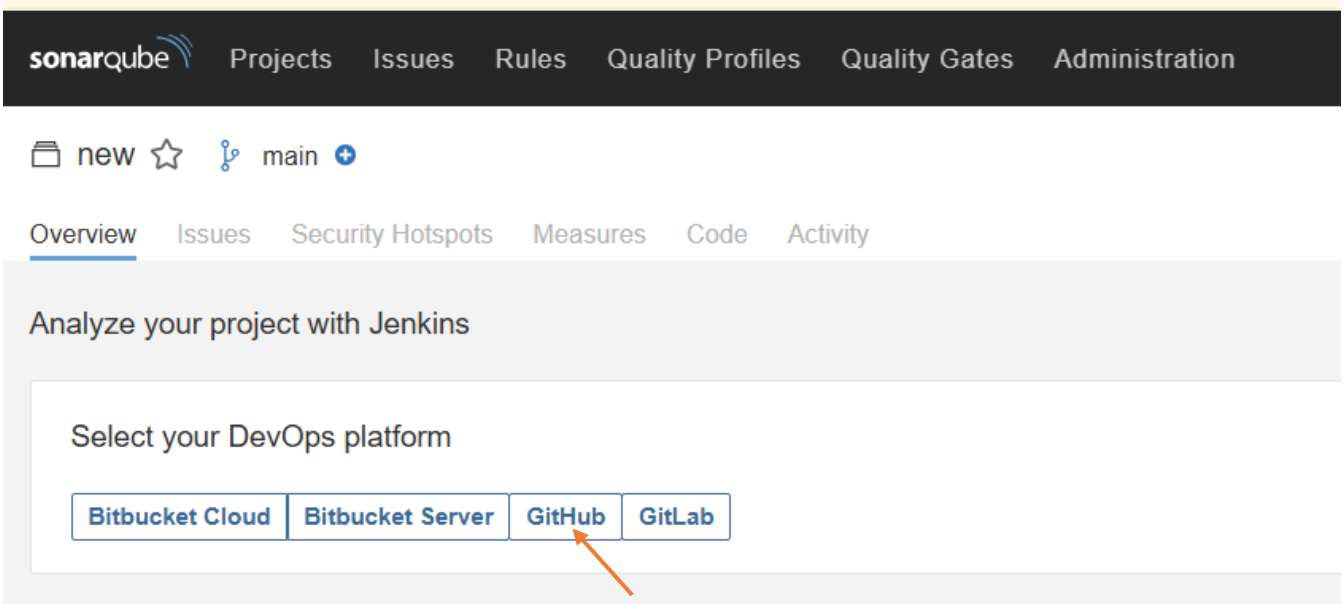
We will give it the name "**new**", key as "**new**" and branch as "**main**"



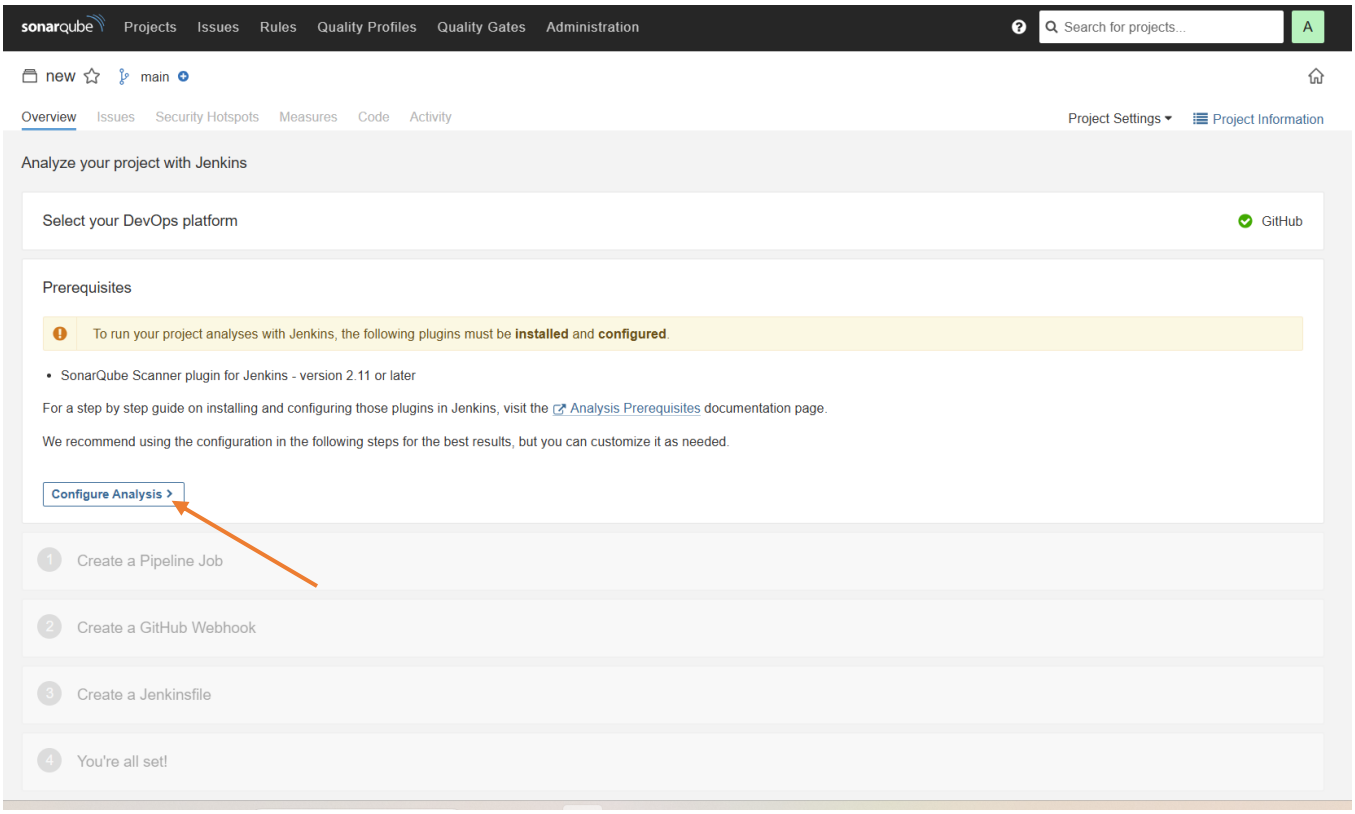
Click on "**Set up**"



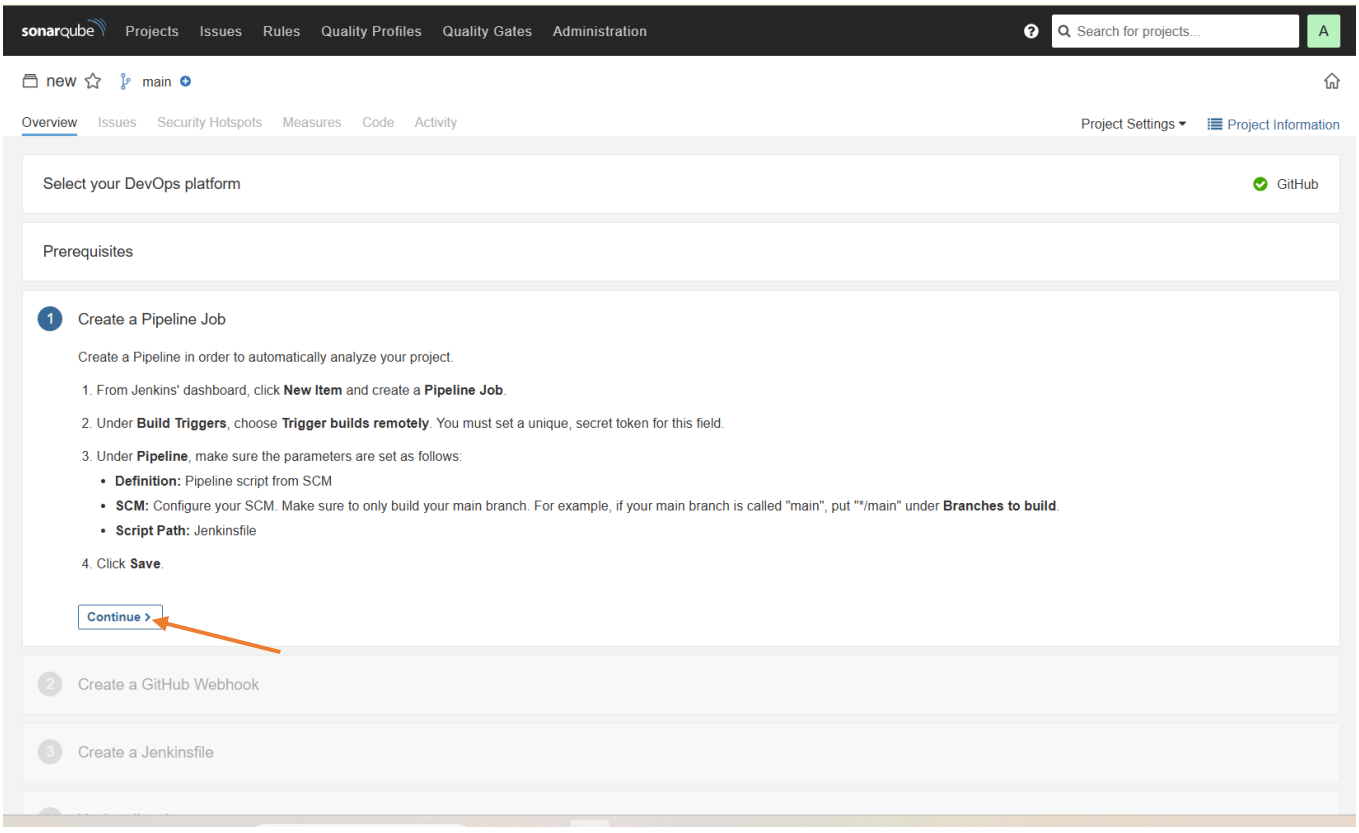
Click on "**With Jenkins**" because we are using Jenkins for CI/CD



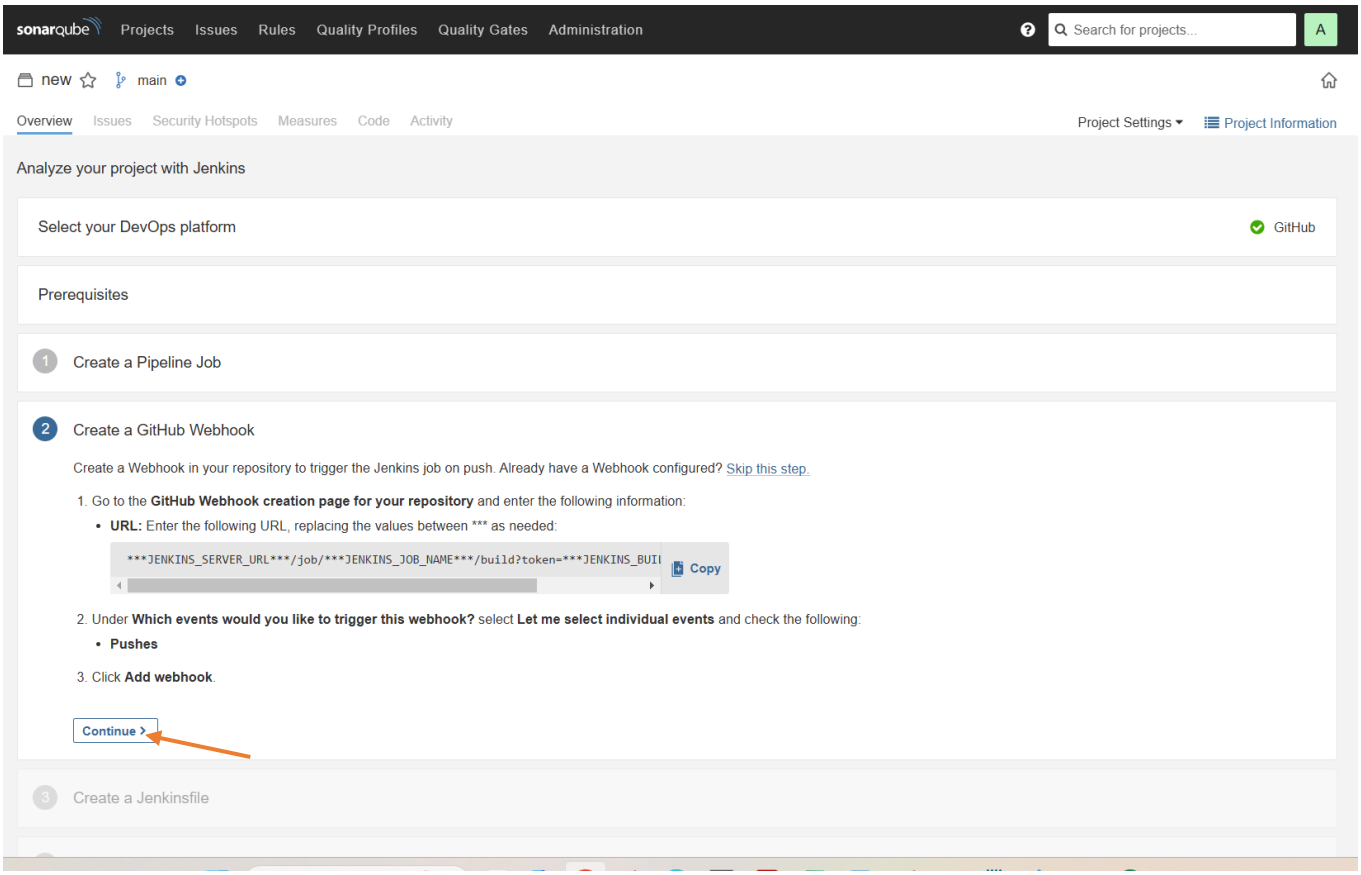
We are using GitHub, so select "**GitHub**"



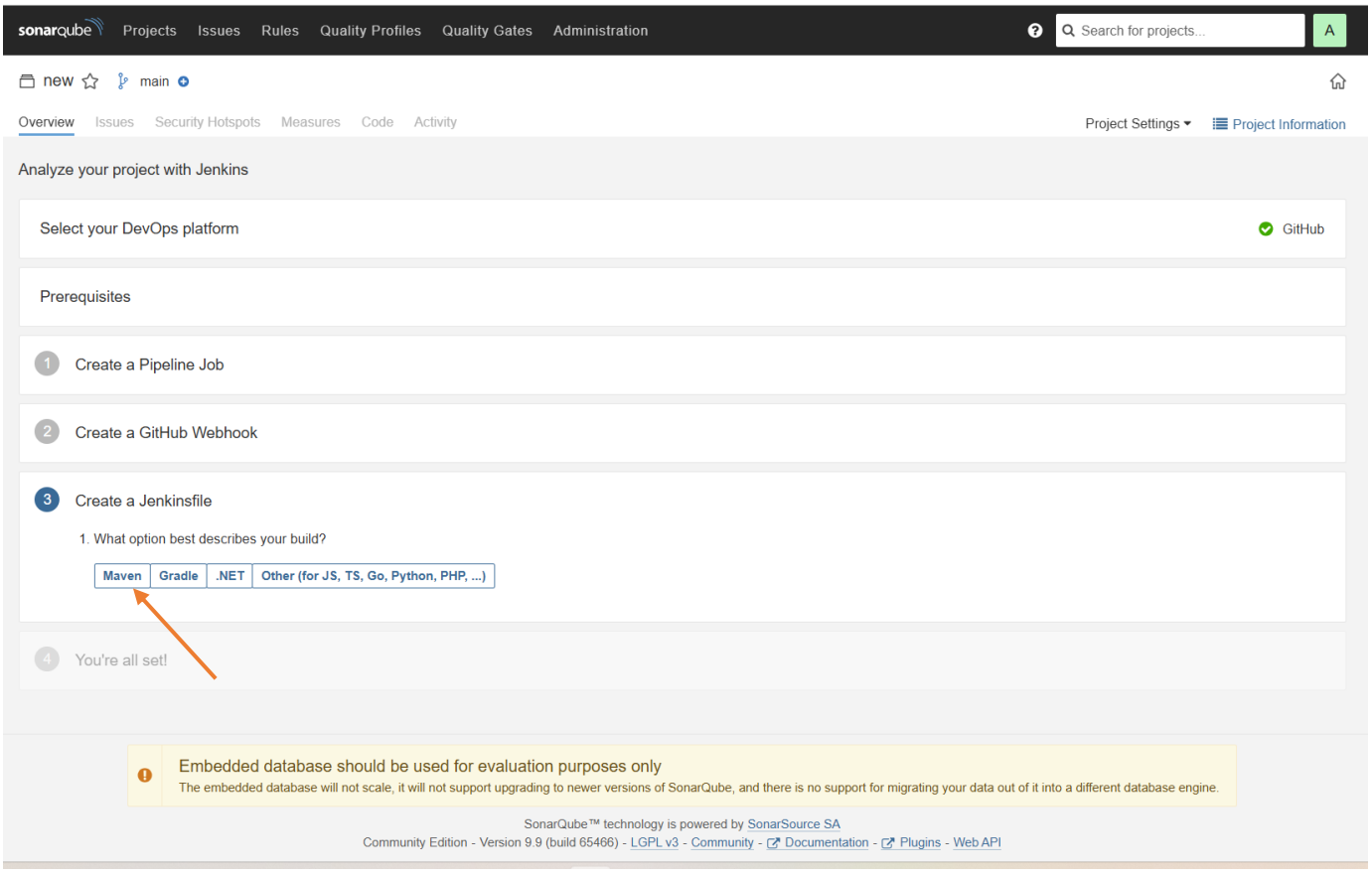
Click on "**Configure Analysis**"



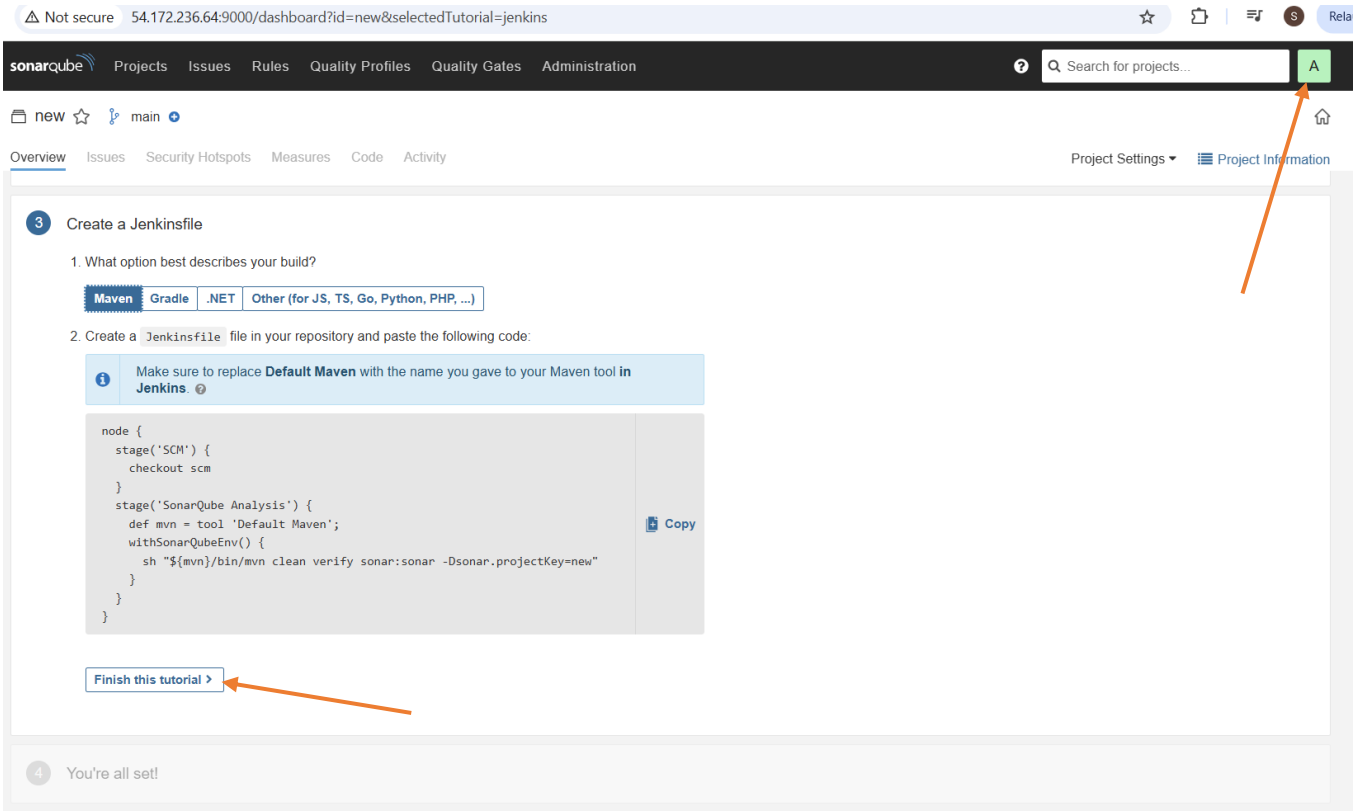
Click on "**Continue**"



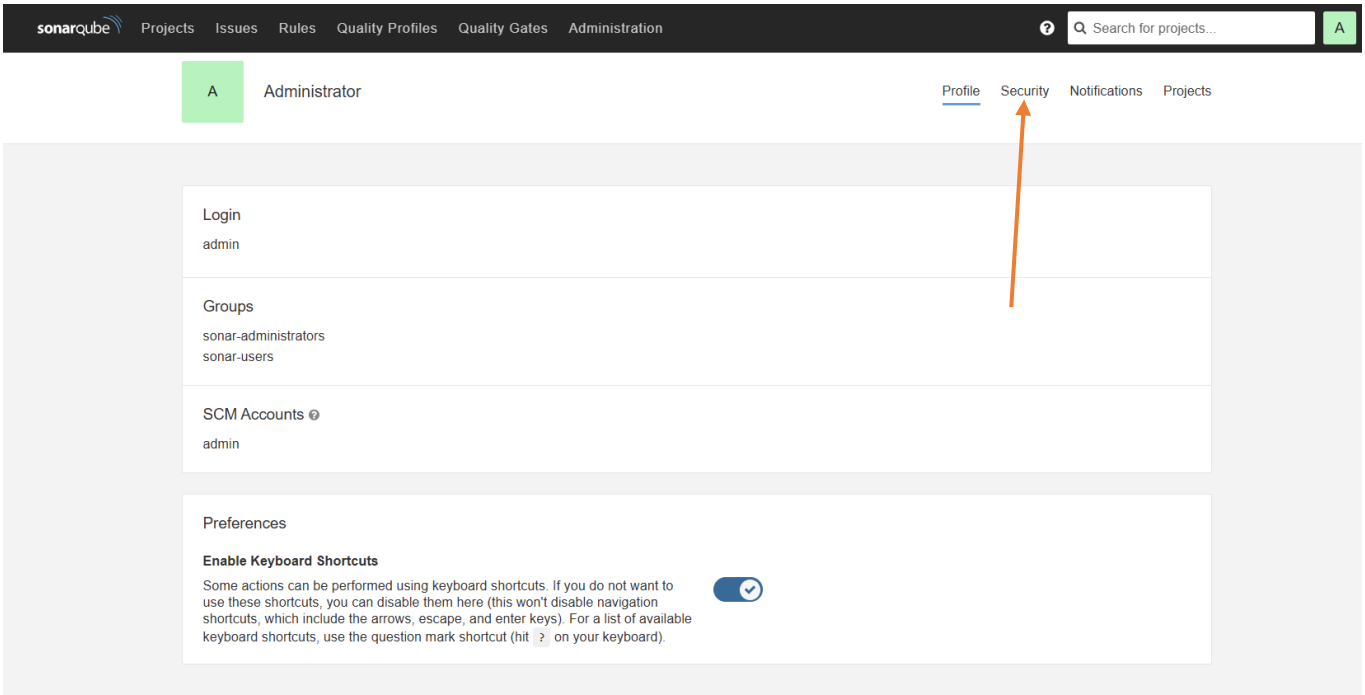
Click on "**Continue**" again



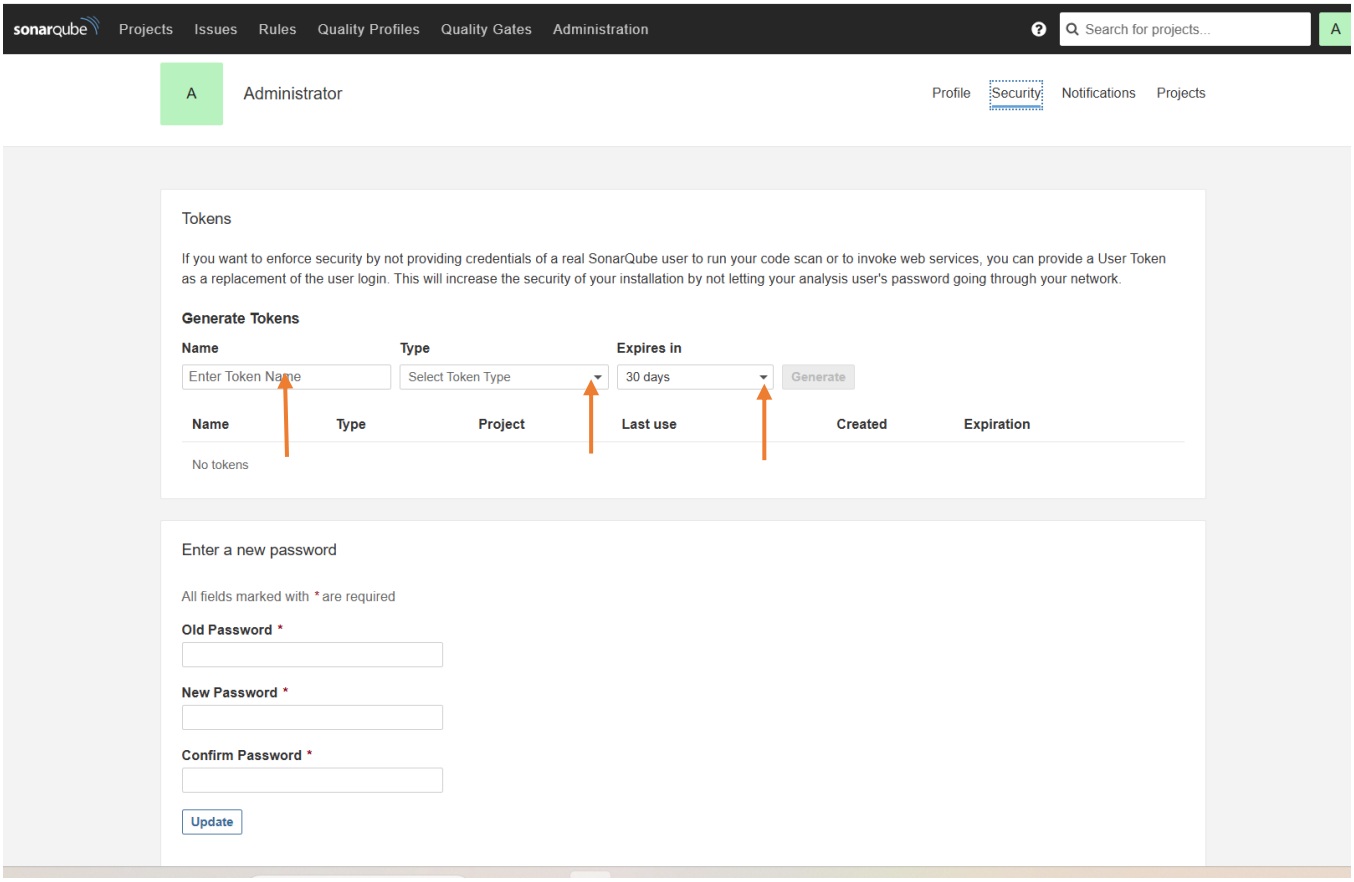
Select “**Maven**”



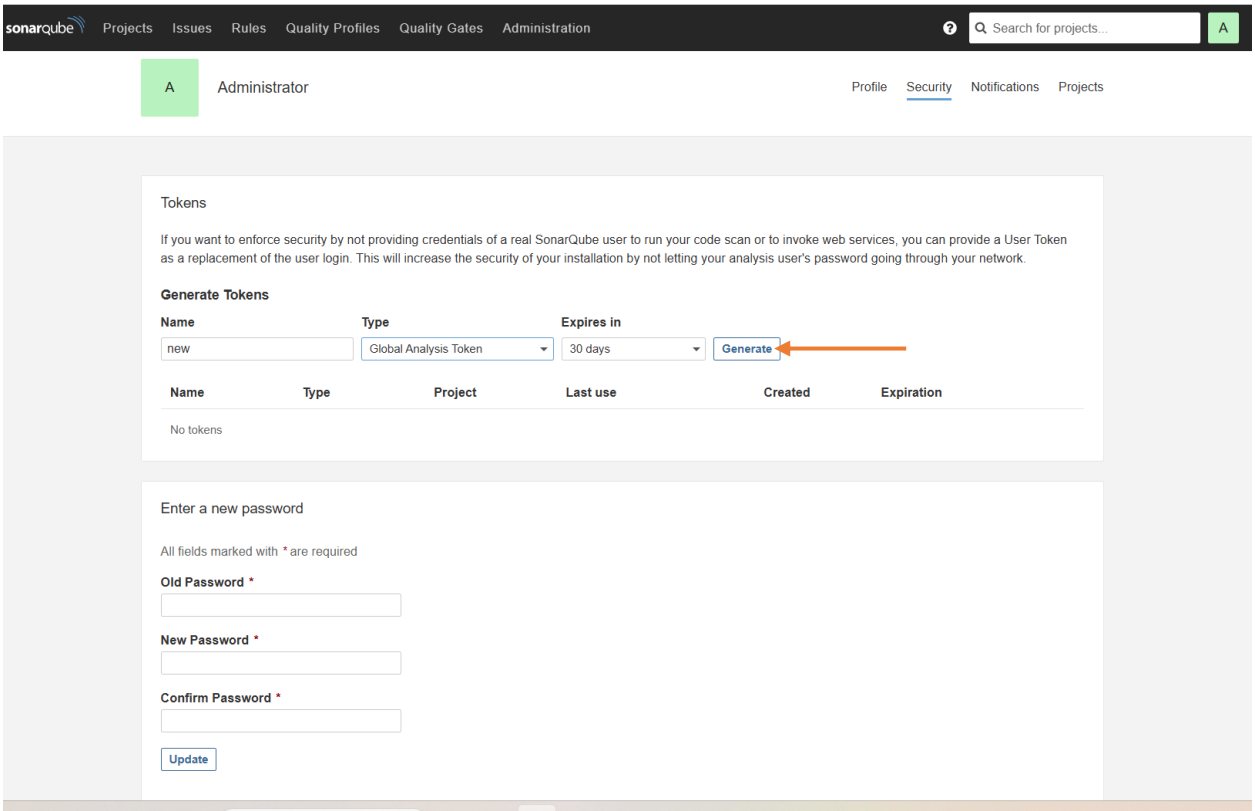
Click on “**Finish this Tutorial**”, And then click on the “**A**” and select “**My Account**”



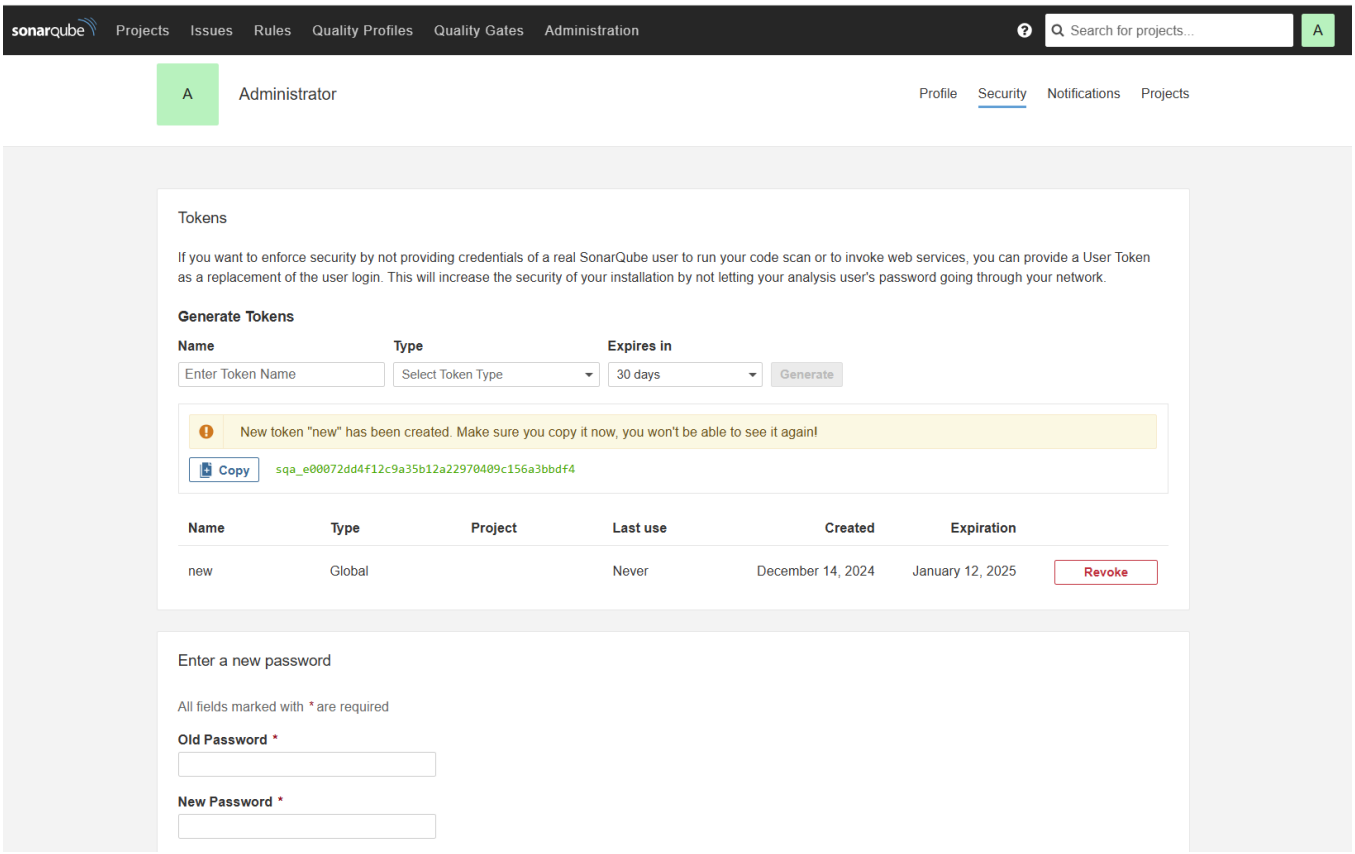
Click on “**Security**”



Give a name for the token, I will give it the name "**new**" and for type, select "**Global Analysis token**" and for Expires in, select "**30 days**".



Click on “**Generate**”



Copy the token and save it somewhere: sqa_d2fe69fec824a0b1e5dc5cb671c6389e1bb11488

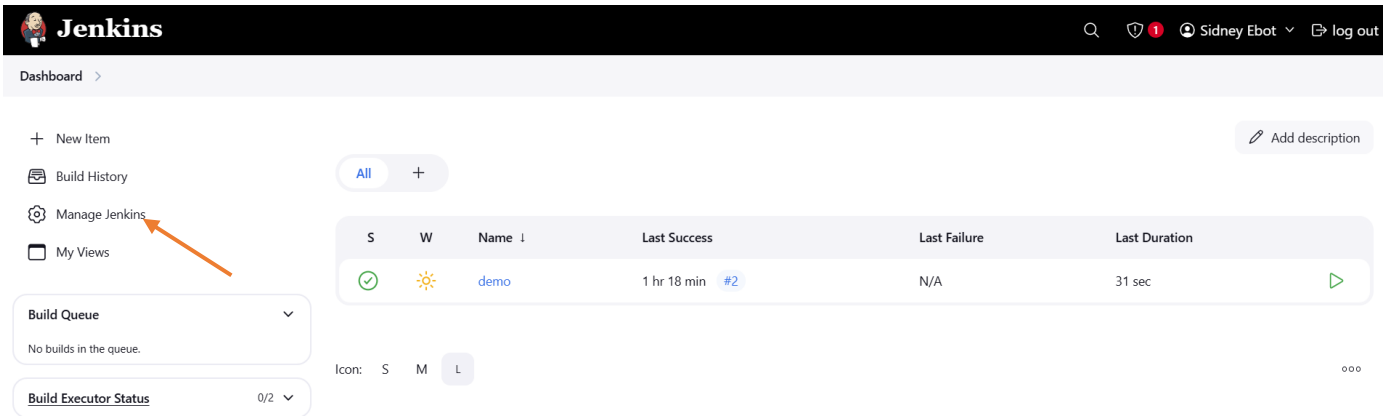
This will be used for authenticating Jenkins.

PART 4: Install SonarQube Plugin

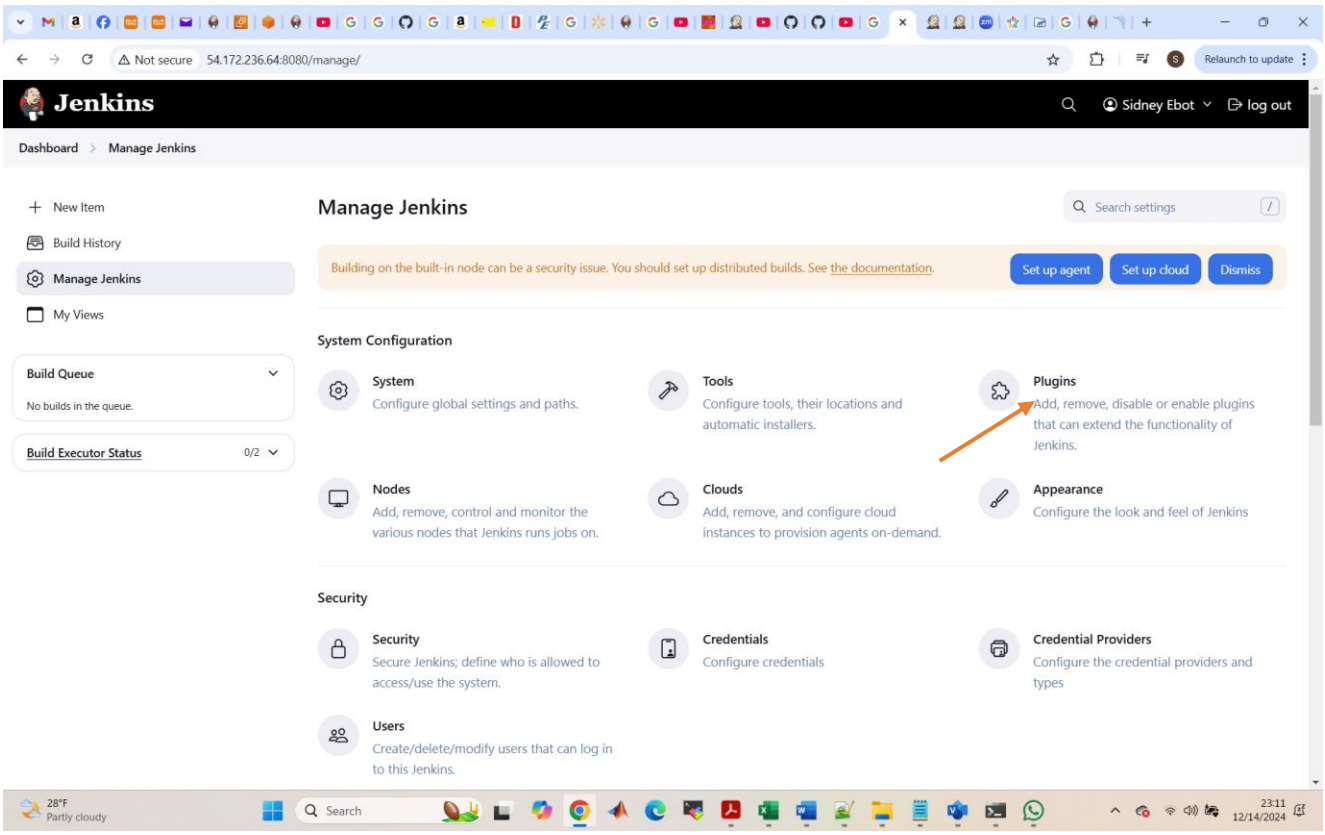
Now, go back to Jenkins GUI and click on Dashboard

Before you can start using SonarQube in your Jenkins pipeline, you need to install the SonarQube plugin.

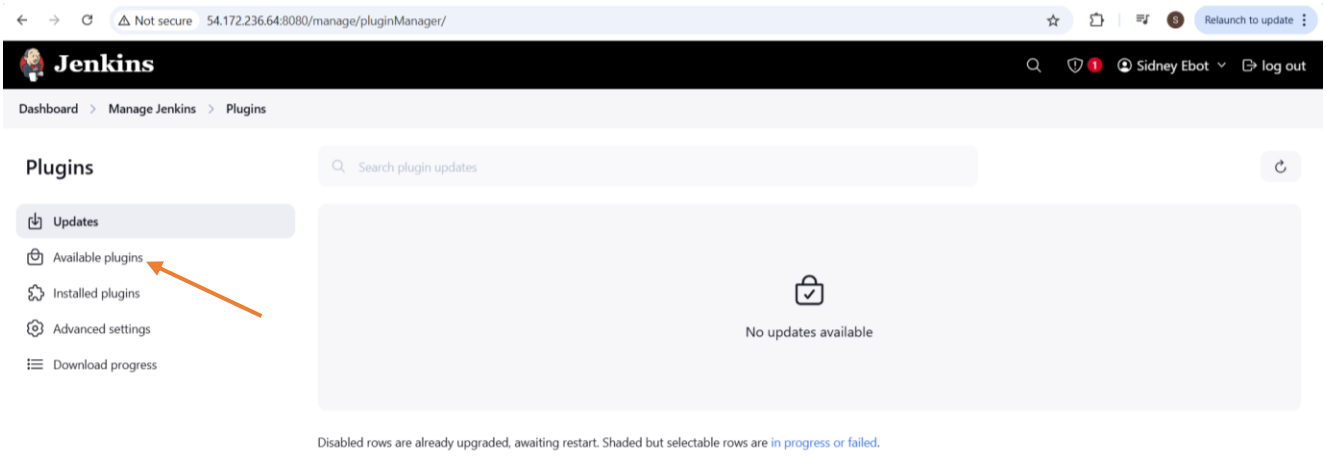
- Go to **Manage Jenkins** > **Manage Plugins** > **Available**.
- Search for “**SonarQube Scanner**” and install the plugin.



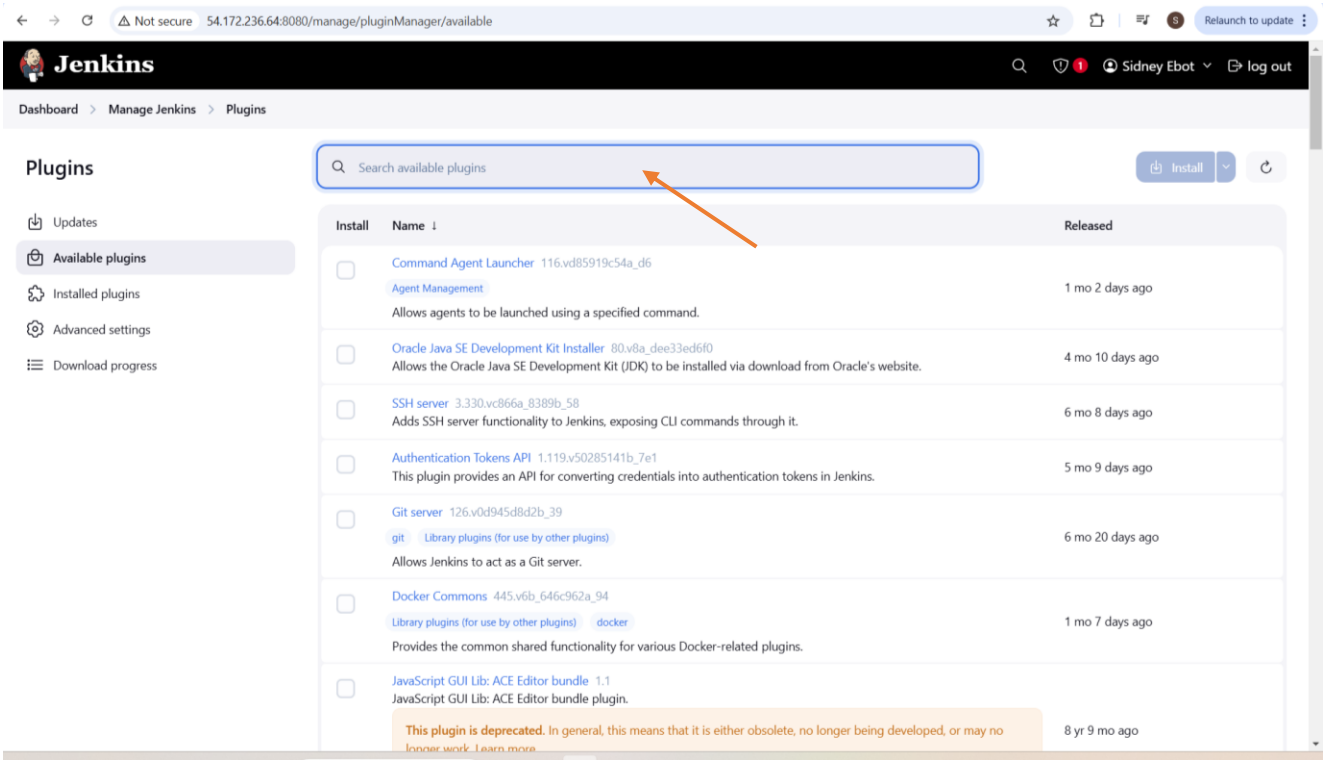
Click on “**Manage Jenkins**”



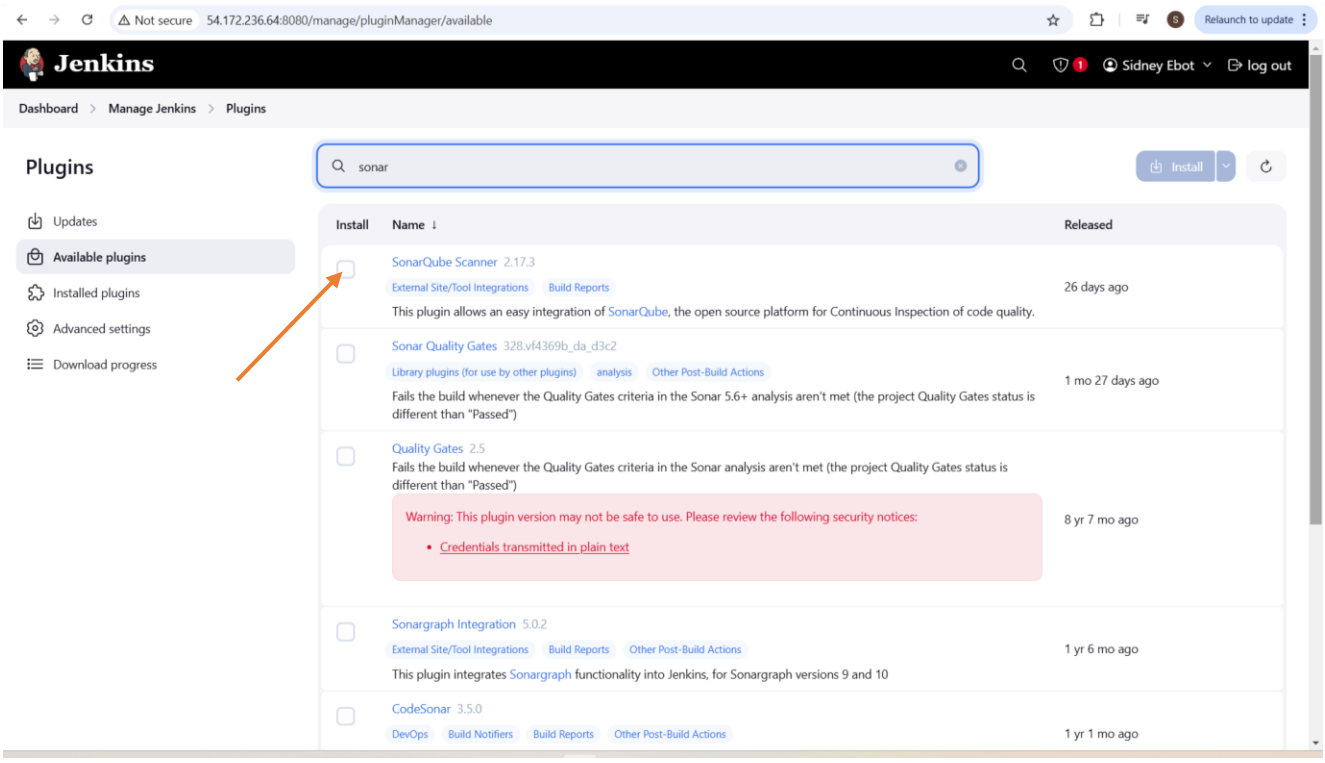
Click on “**Plugins**”



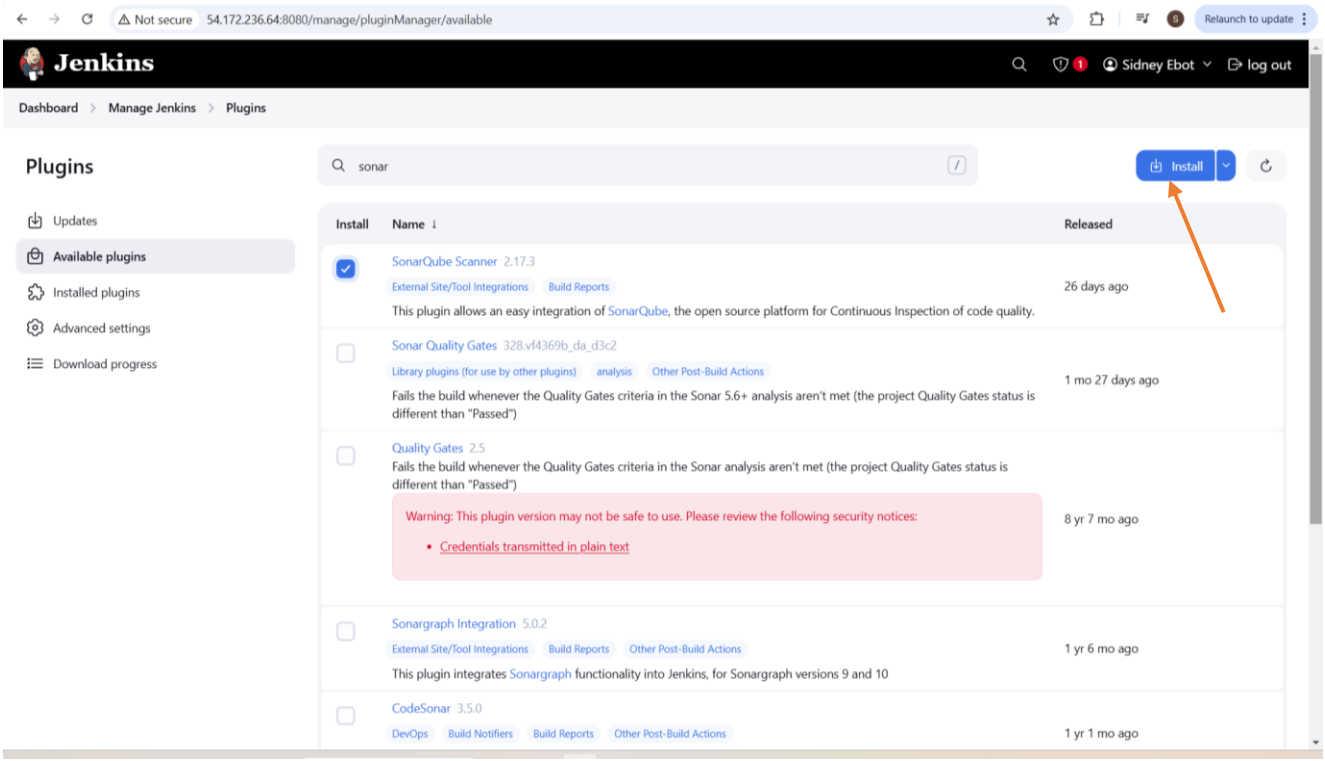
Click on “**Available Plugins**”



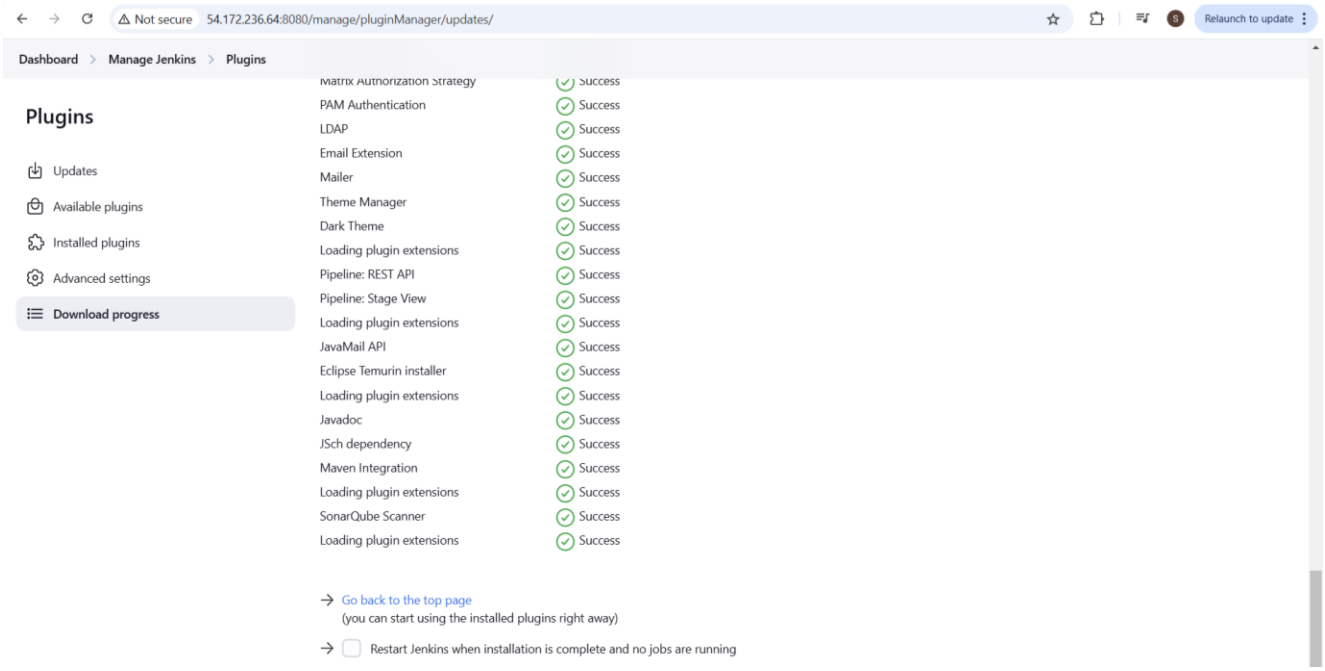
Search for “**sonar**”



Check “**SonarQube Scanner**”



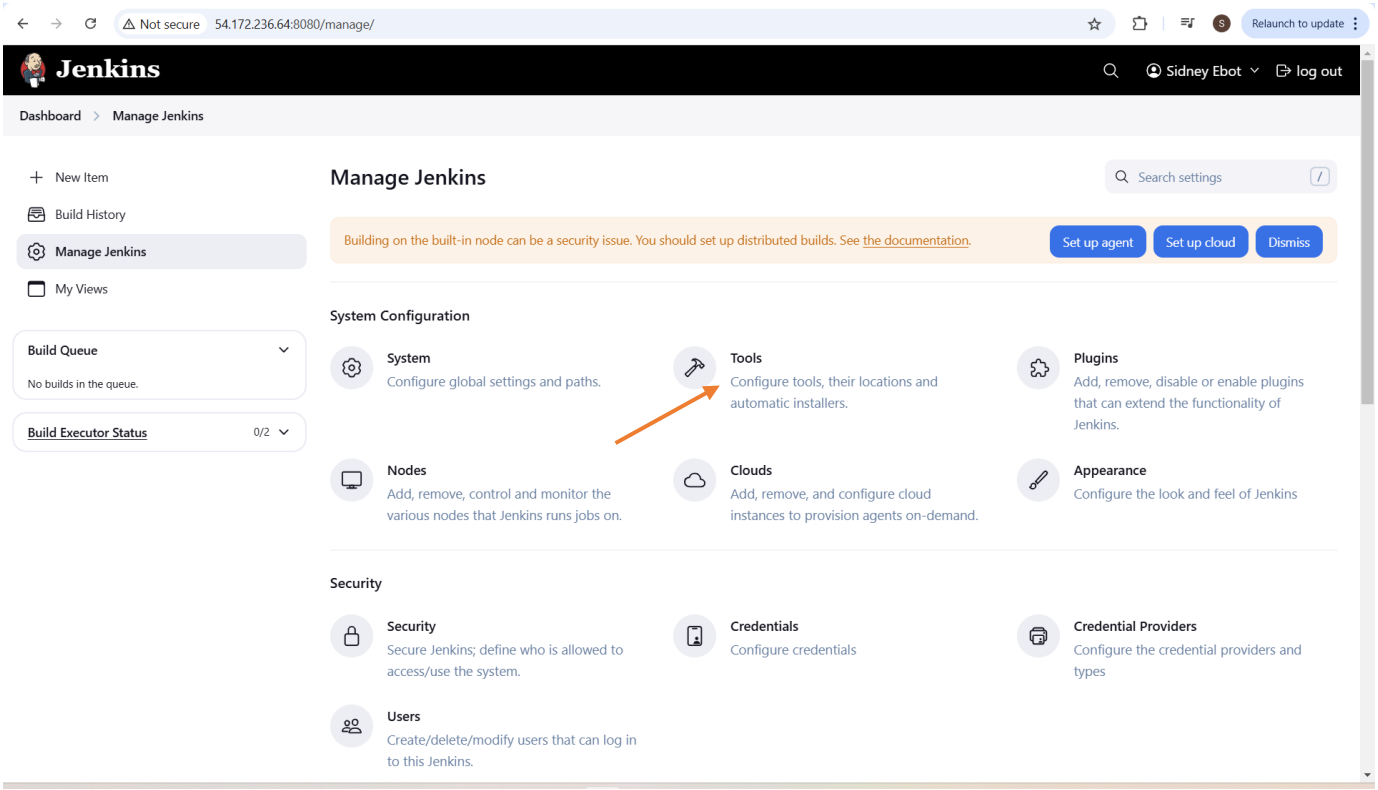
Click on "**Install**"



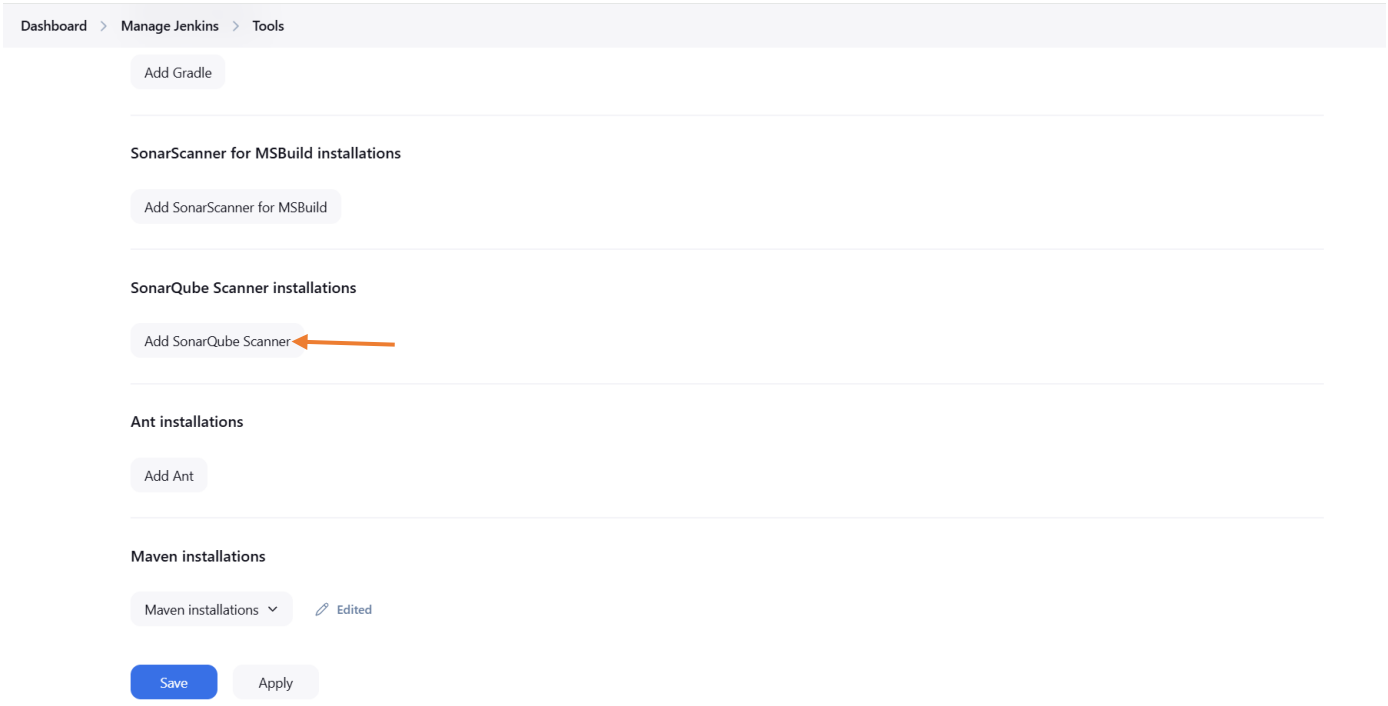
PART 5: Add SonarQube Scanner to Jenkins on GUI

- Go to **Manage Jenkins** > **Tools**.
- Scroll to **SonarQube Scanner Installation** and click **Add SonarQube Scanner**.
- Provide a **name** and **installation method**.

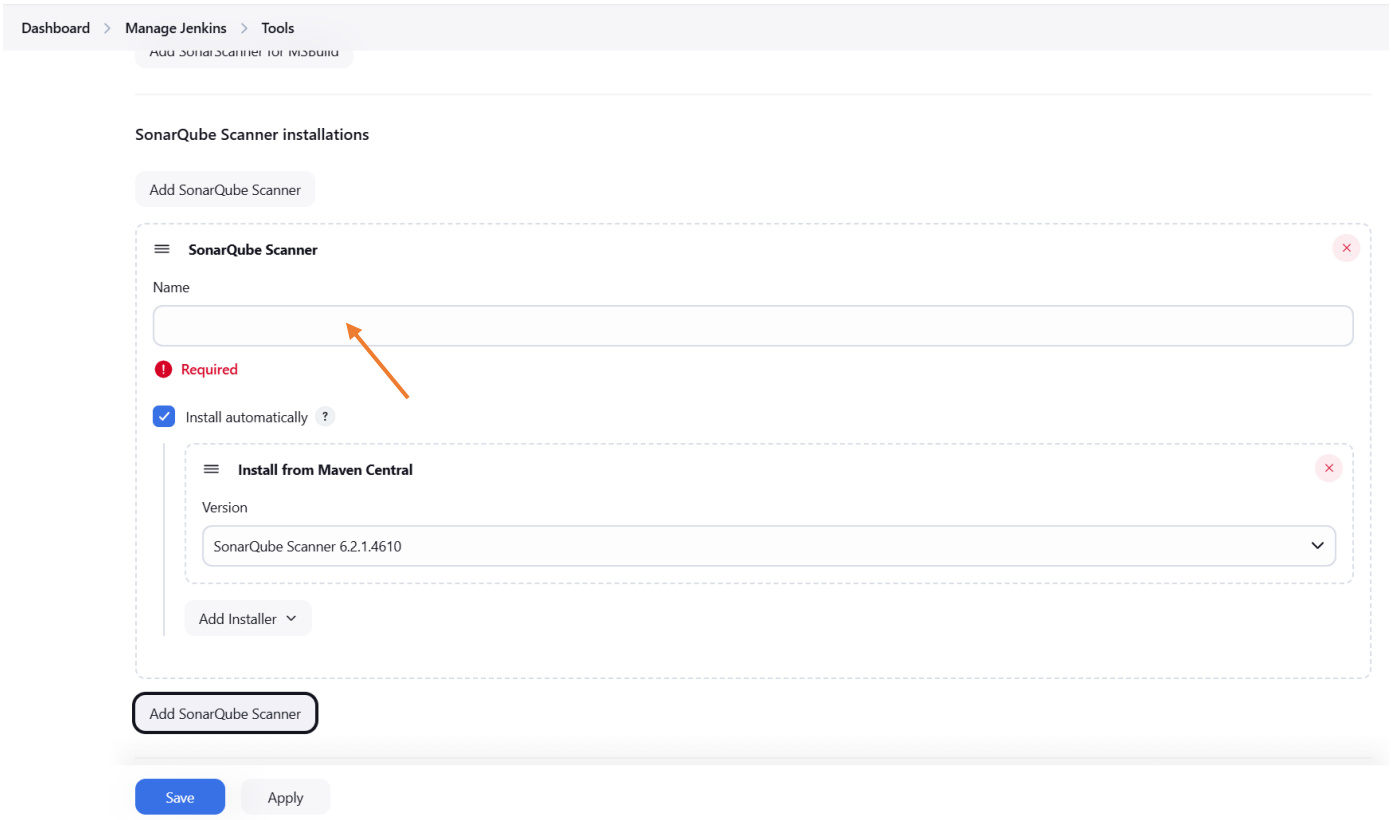
Click on "**Manage Jenkins**" again



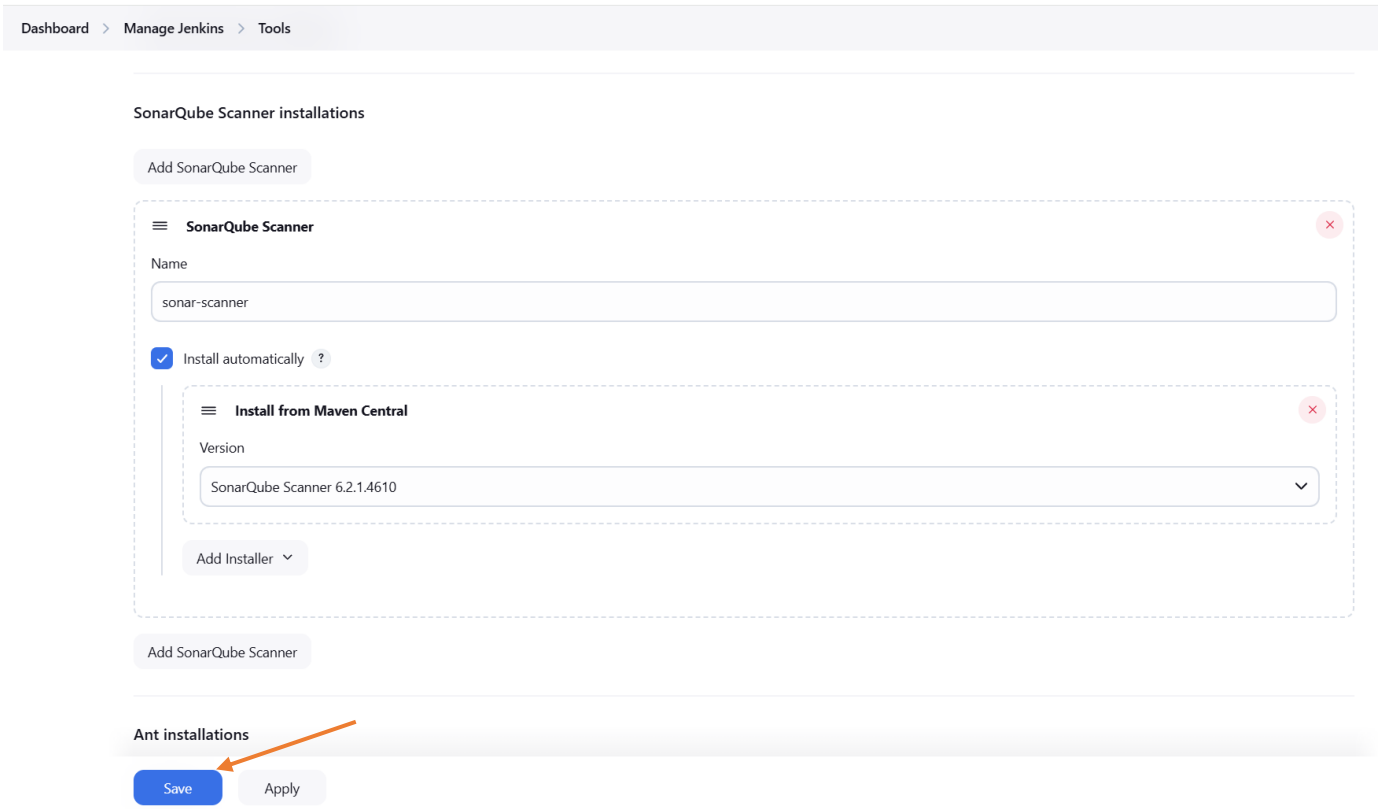
Click on "**Tools**" and scroll down to "**SonarQube Scanner Installation**"



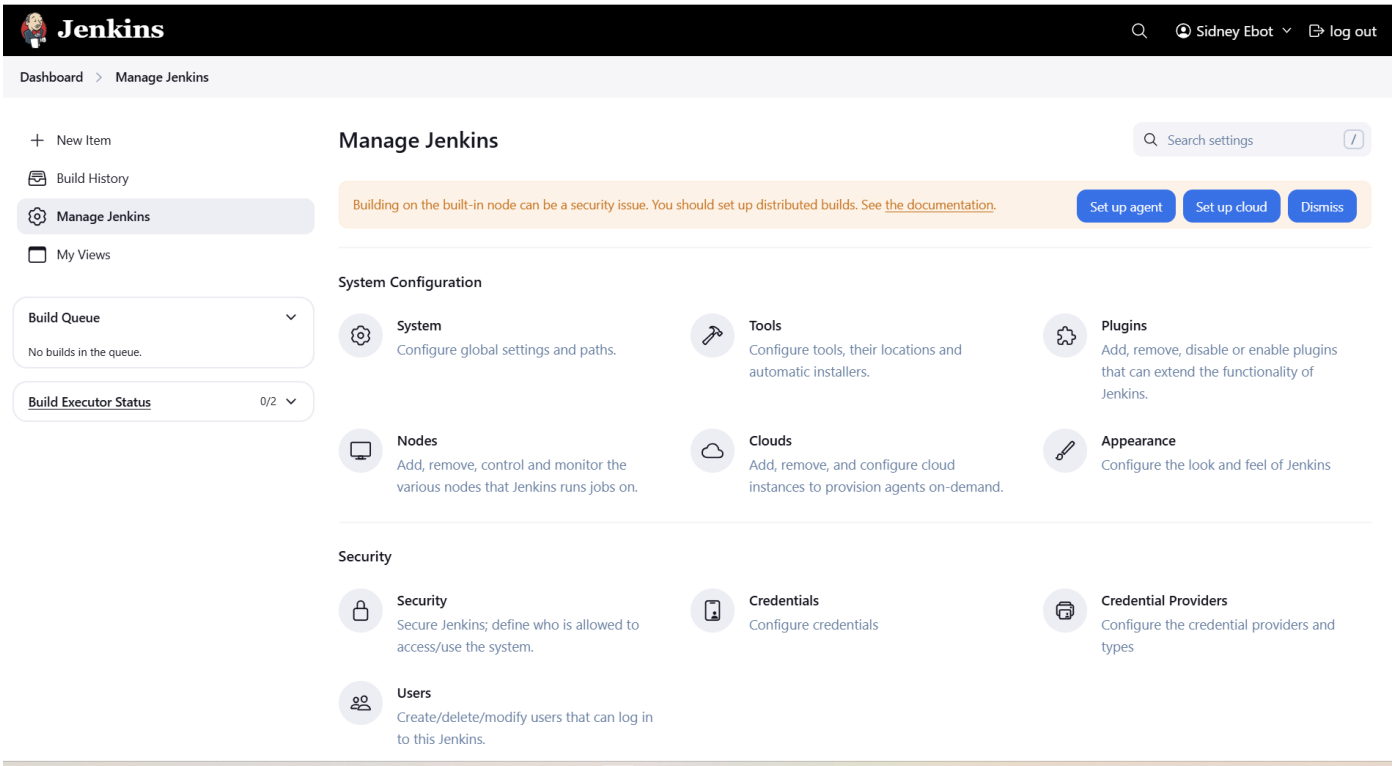
Under "**SonarQube Scanner Installations**", click on "**Add SonarQube Scanner**"



We will give it the name "**sonar-scanner**"



Click on "**Save**"

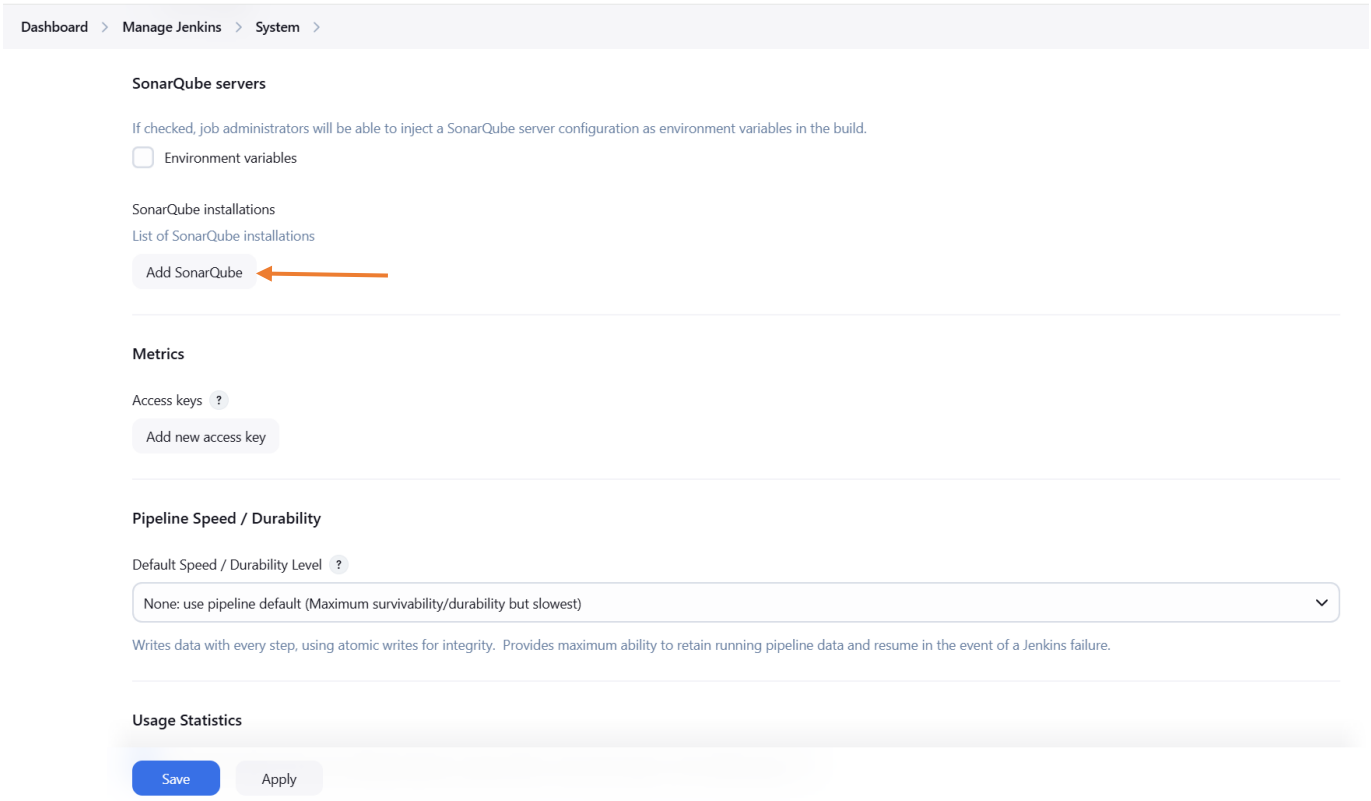


PART 6: Configure SonarQube Server on Jenkins

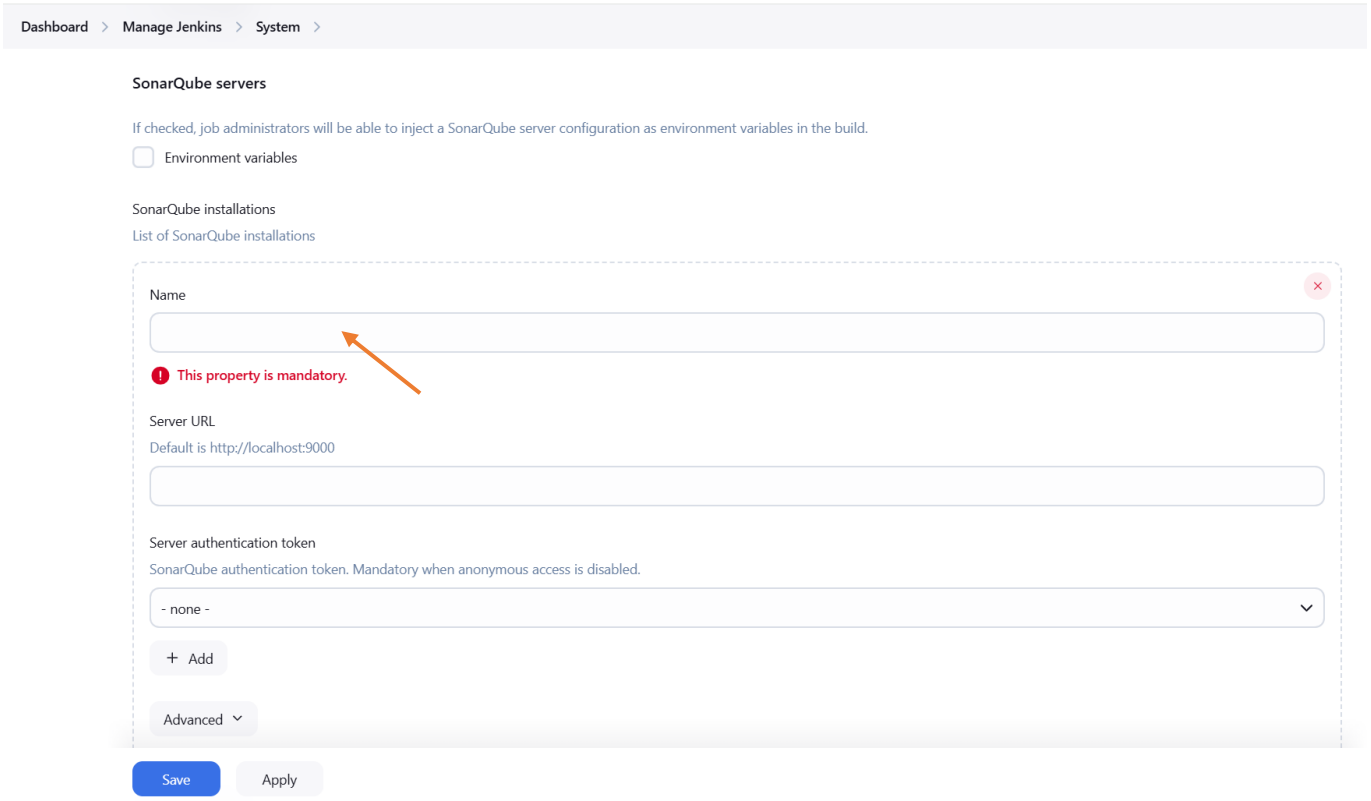
After installing the plugin, you need to configure your SonarQube server in Jenkins.

- Go to ***Manage Jenkins*** -> ***System***.
- Scroll down to ***SonarQube Servers*** and click ***Add SonarQube*.**
- Provide your ***SonarQube server URL*** and ***authentication token***.

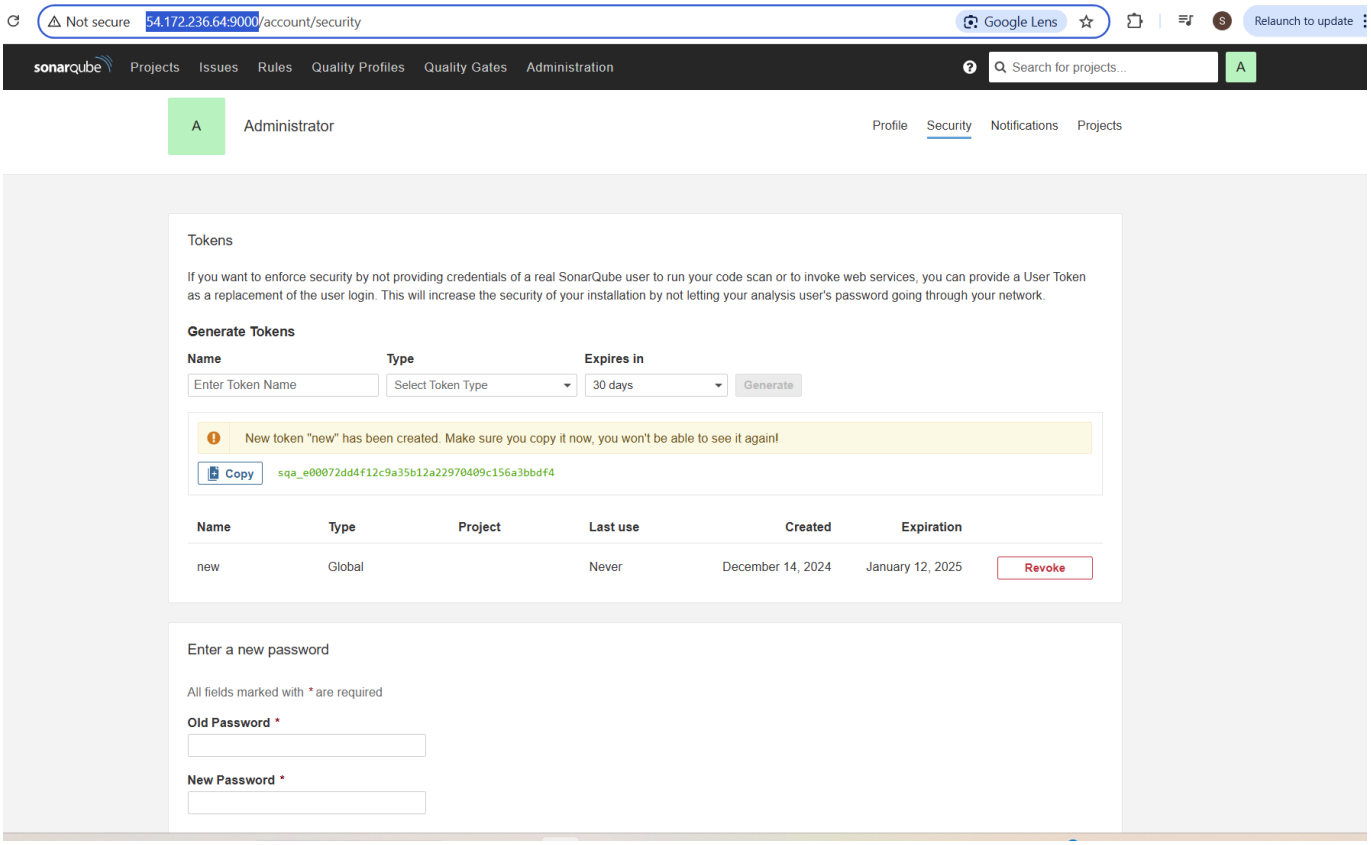
Click on "**System**" and scroll down to "**SonarQube Server**"



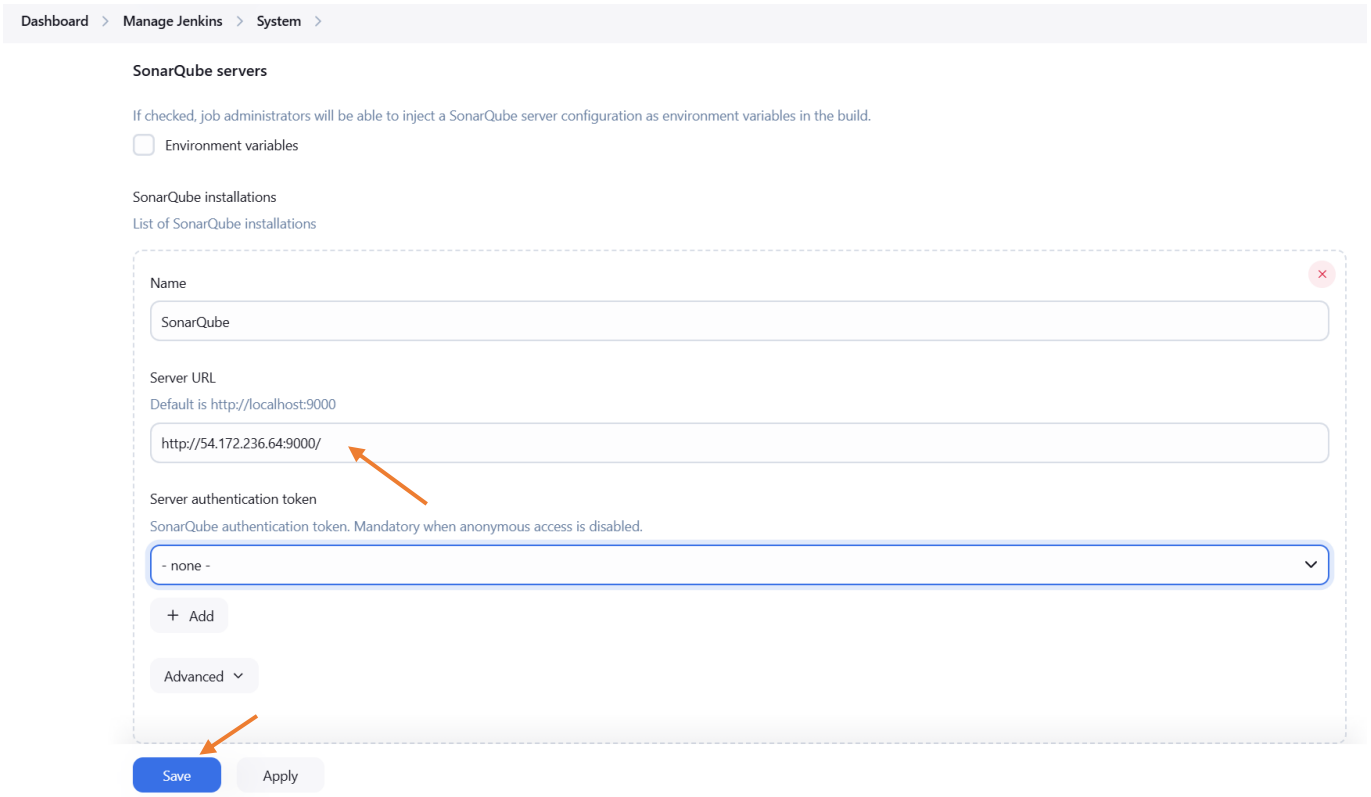
Click on "**Add SonarQube**"



We will give it the name "**SonarQube**", then copy the IP address from the SonarQube browser



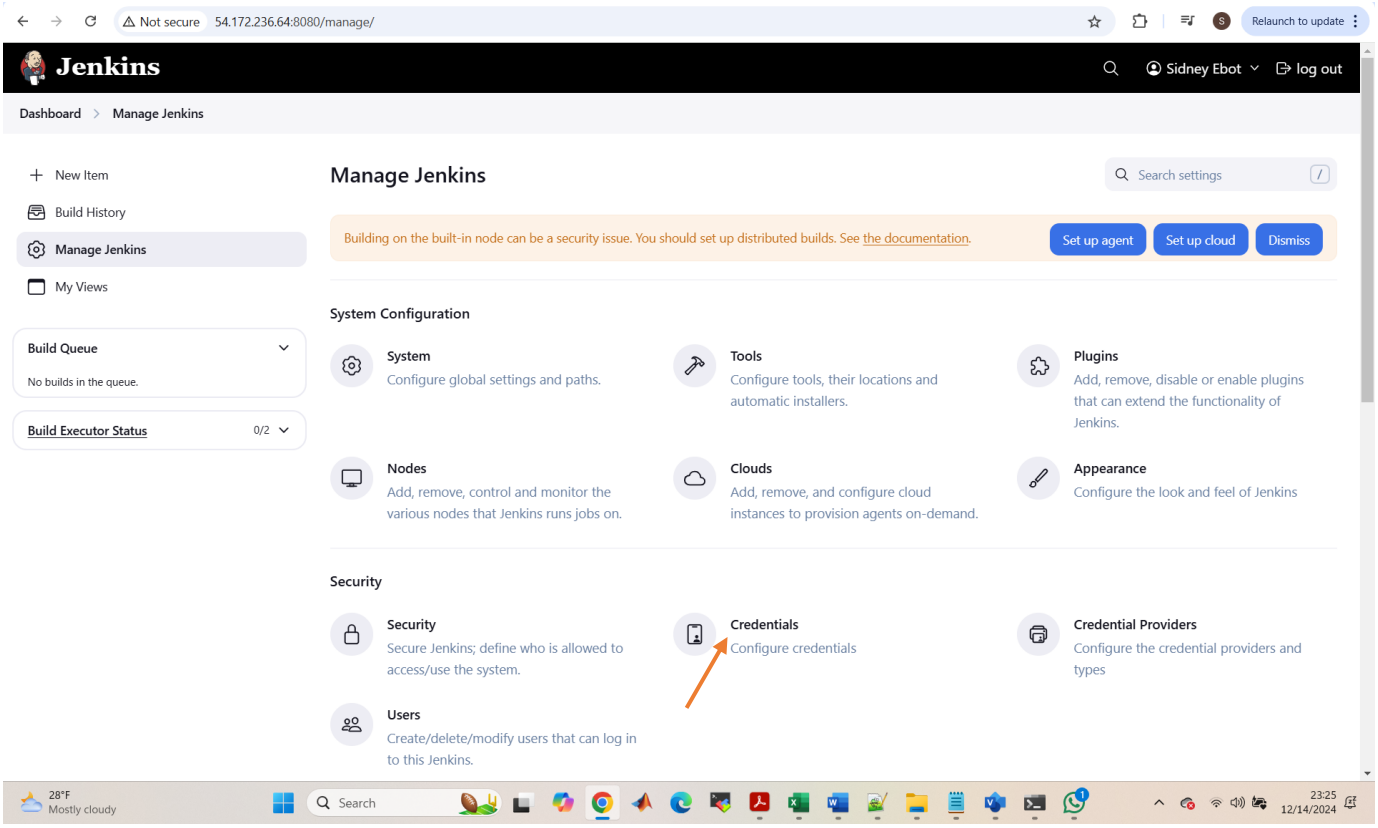
Copy the IP address and



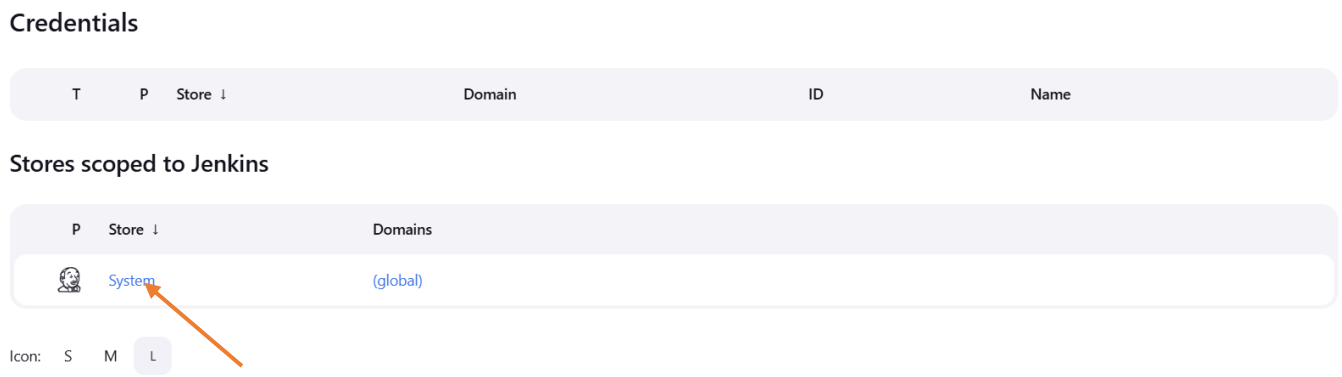
Paste on the ***authentication token*** Jenkins GUI and click on "**Save**"

PART 7: Create Server Authentication token

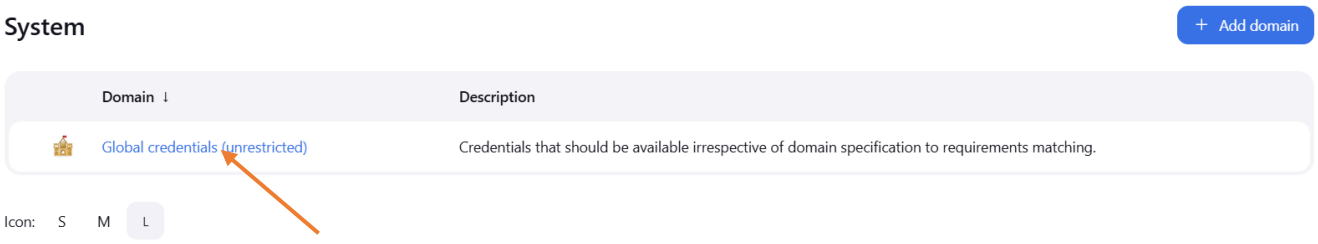
For the "**server Authentication Token**", Go to "**Manage Jenkins**" on another window



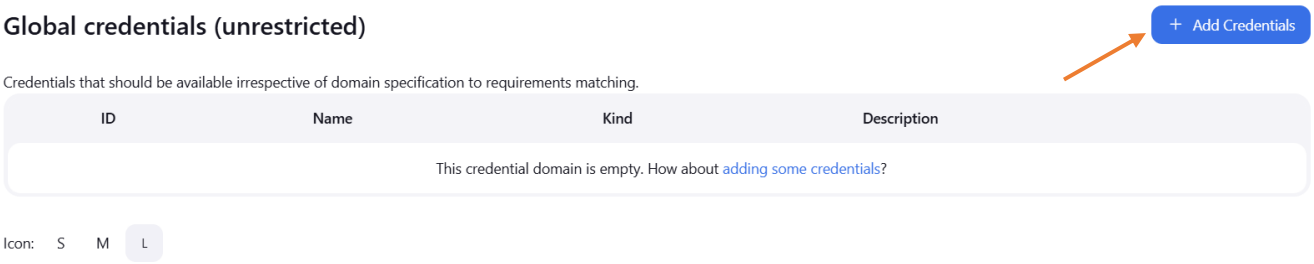
Click on "**Credentials**"



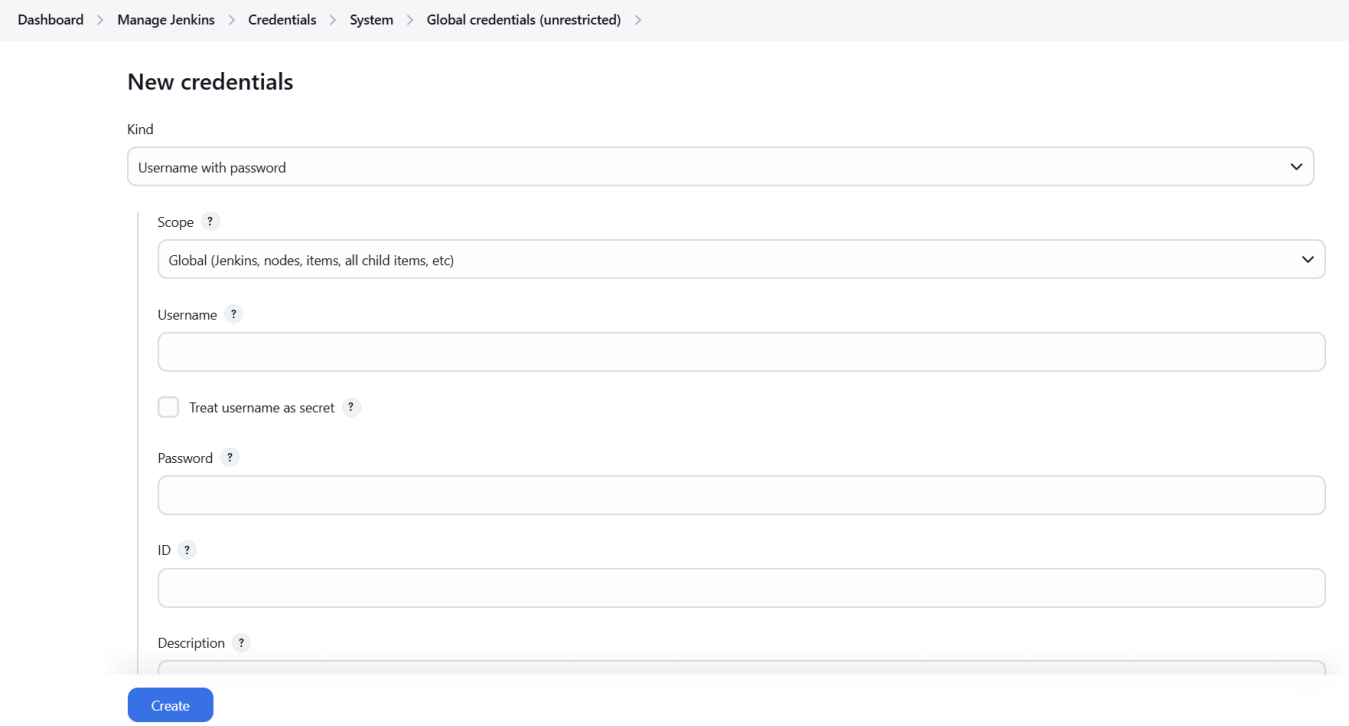
Click on "**System**"



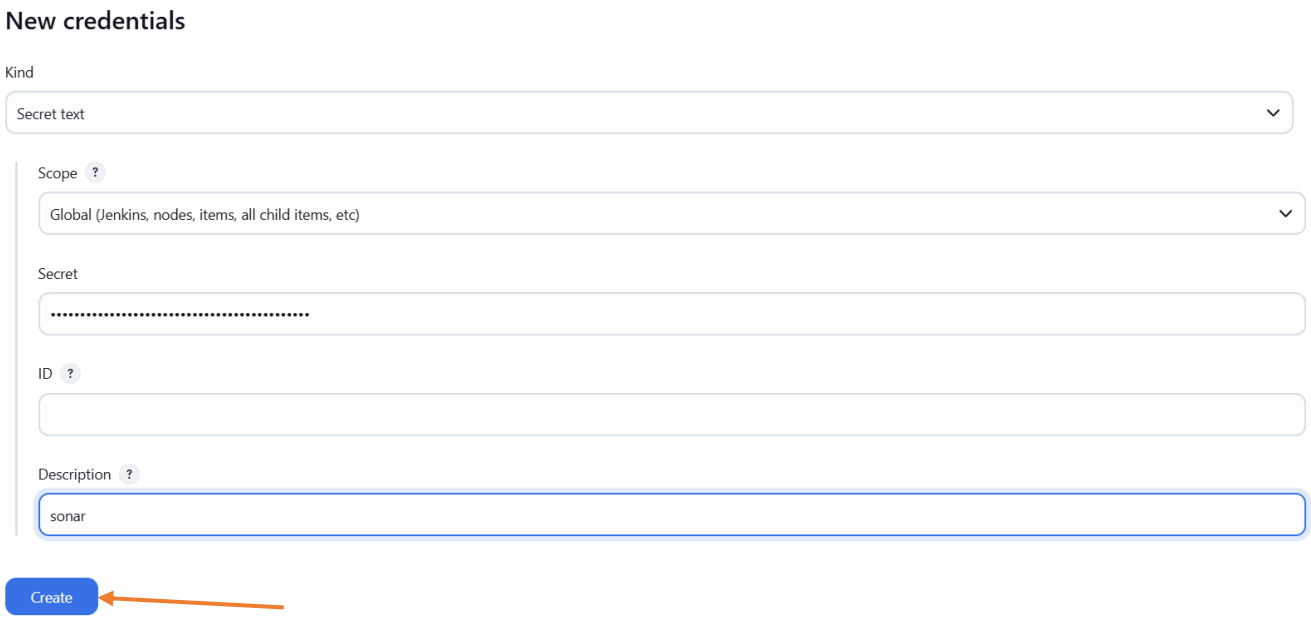
Click on "**Global Credential**"



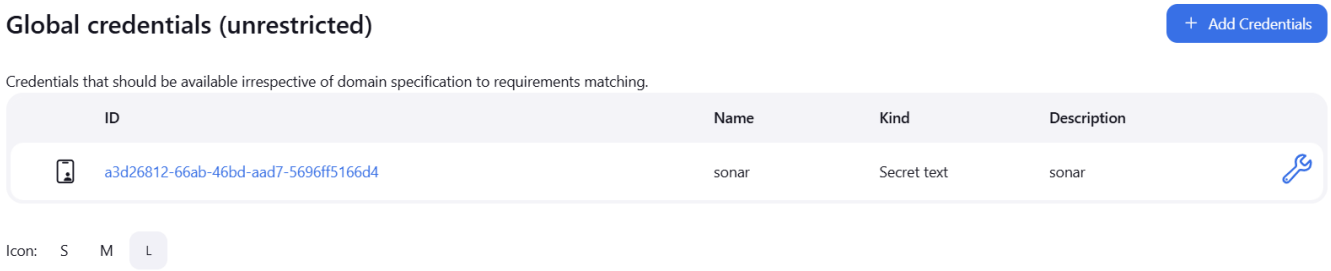
Click on “**Add Credentials**”



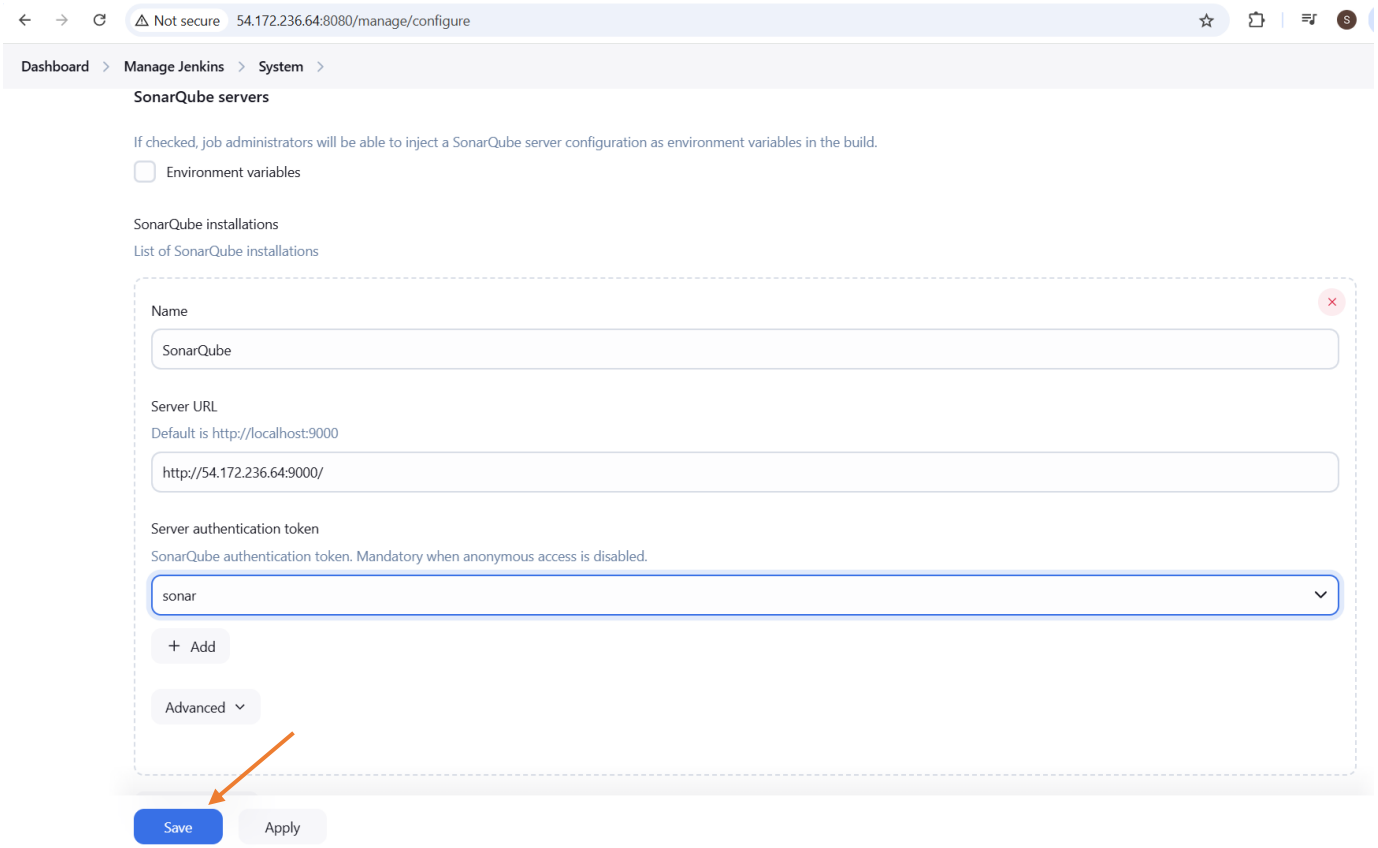
On “**Kind**”, select “**Secret Text**”, on “**Secret**” paste the key we copied from SonarQube, for “**Description**” enter “**sonar**”



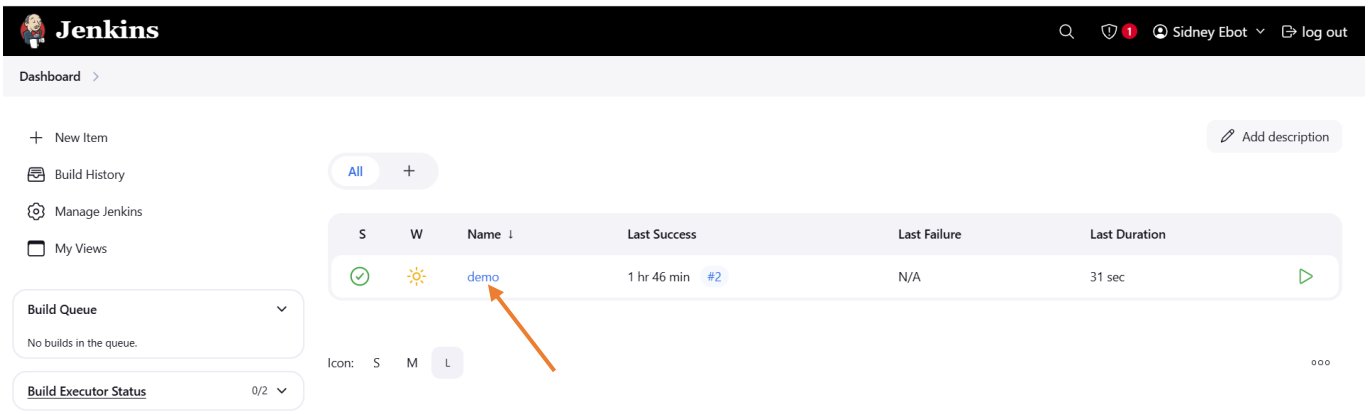
Click on “**Create**”



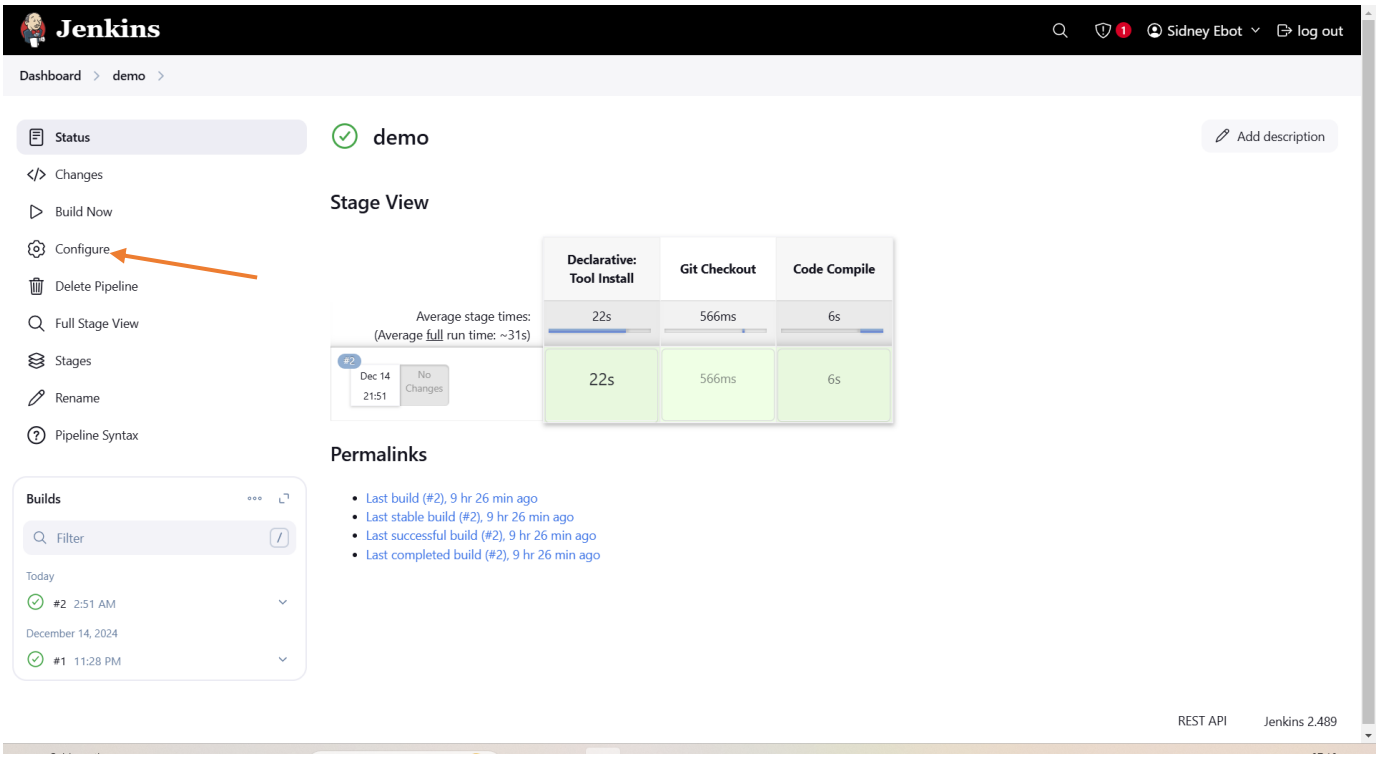
Go back to the **Jenkins** -> **System page and refresh it**. On “Server Authentication Token” you will now select “**sonar**”



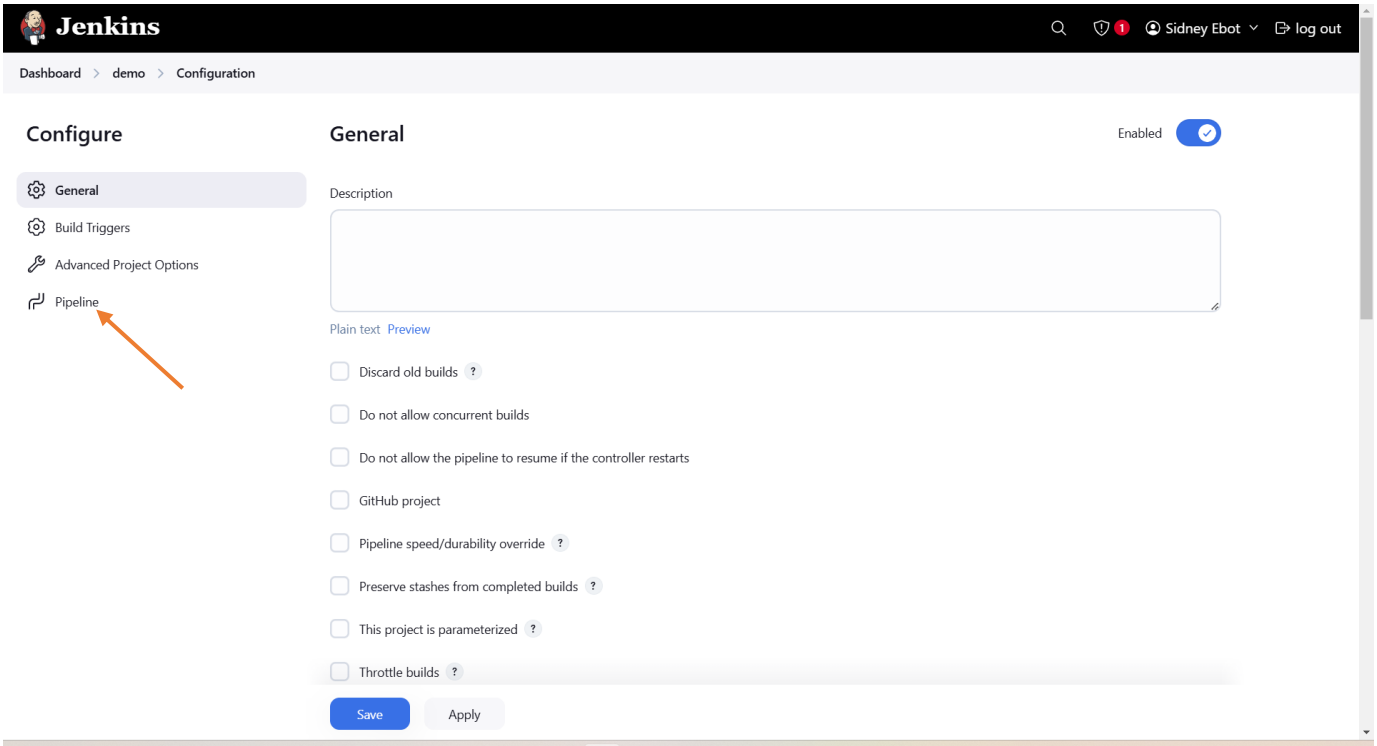
Click on “**Save**”



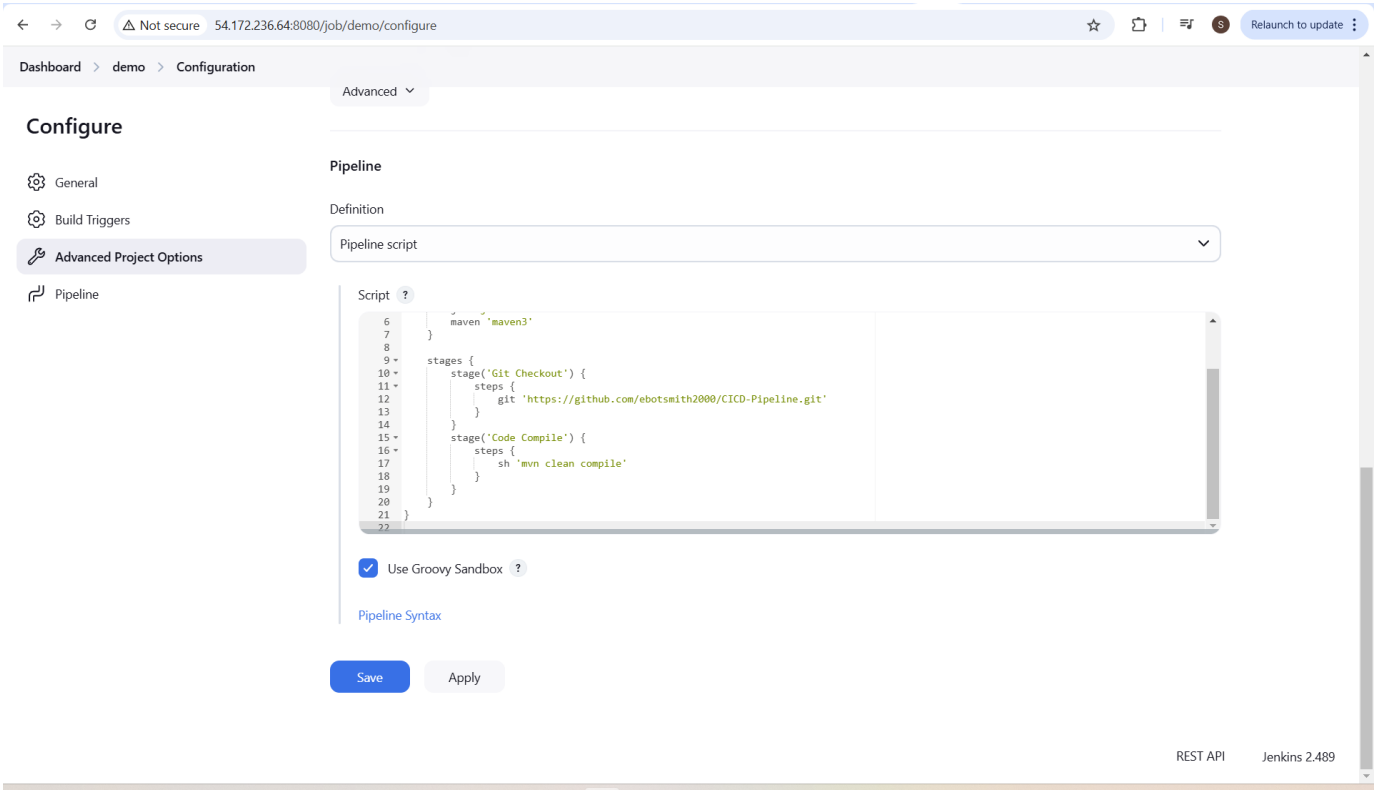
Click on the Pipeline “**demo**”



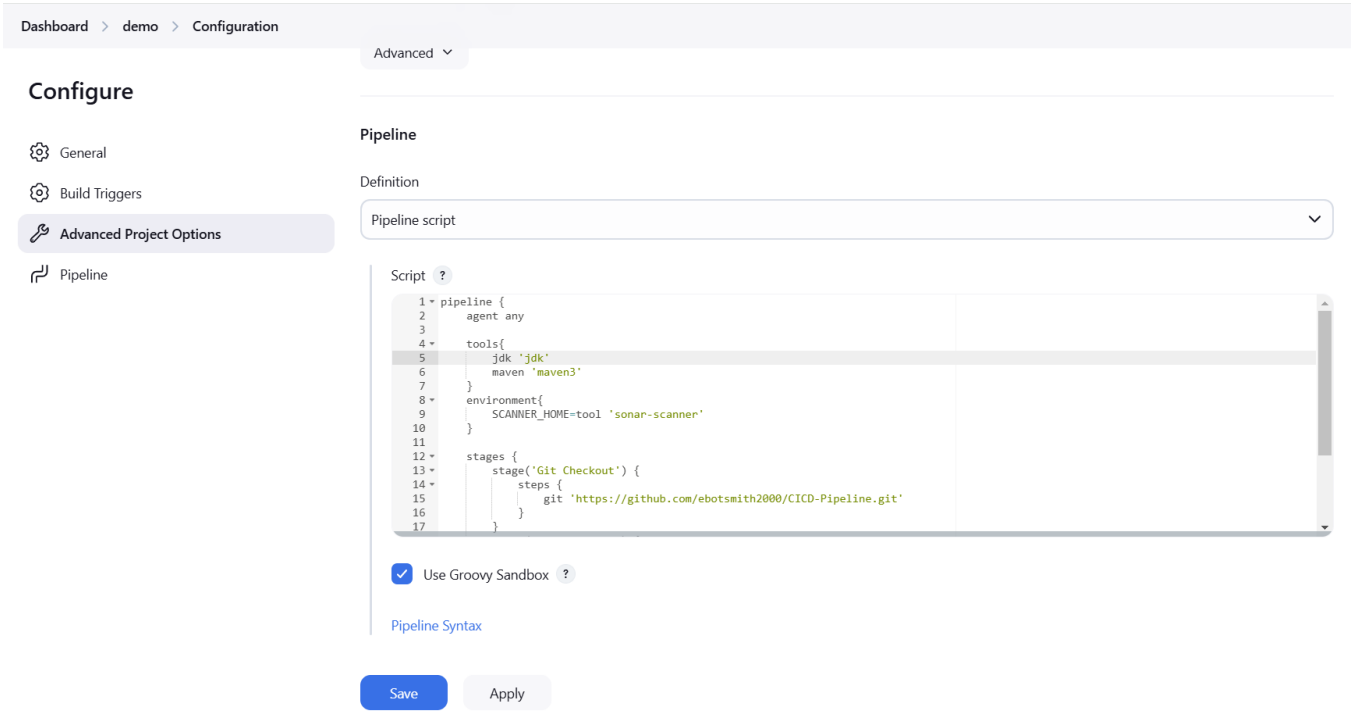
Then click on “**Configure**”



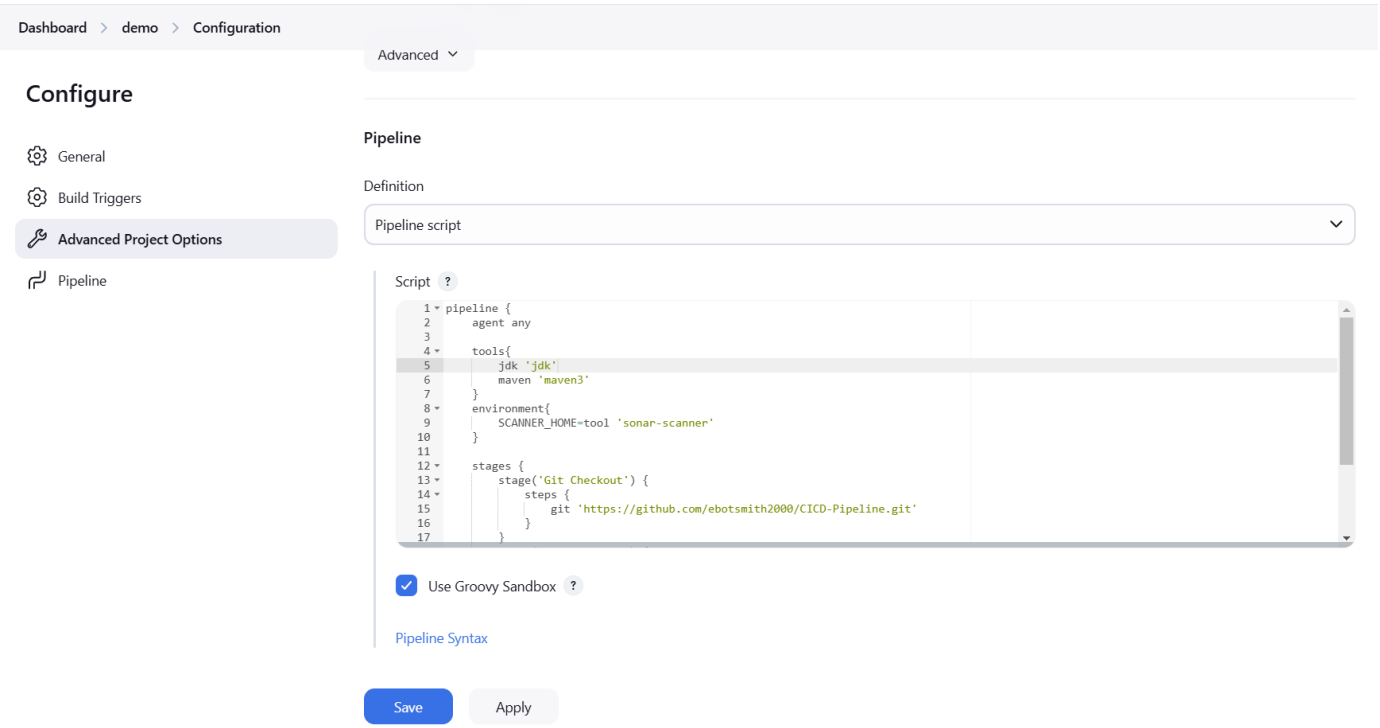
Scroll down to “**Pipeline**”



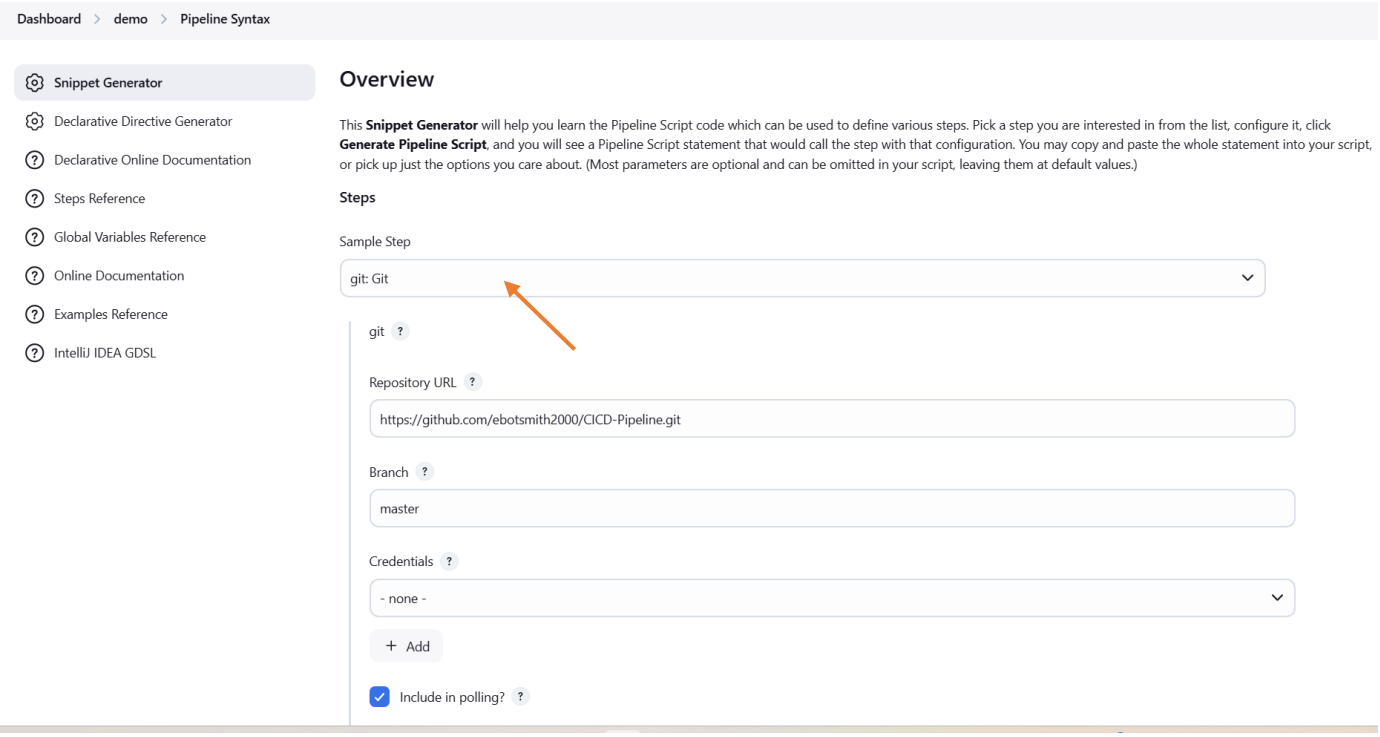
We have configured another tool called "**Sonar-Scanner**" for SonarQube. We have to define it as well but we cannot define it under tools. So, we have to define it under "**environment variable**". We will give the variable the name "**SCANNER_HOME**", it will be the HOME path of scanner. There we have to define the tool using the name we gave to the SonarQube Scanner, that is "**sonar-scanner**" under "**tools**".



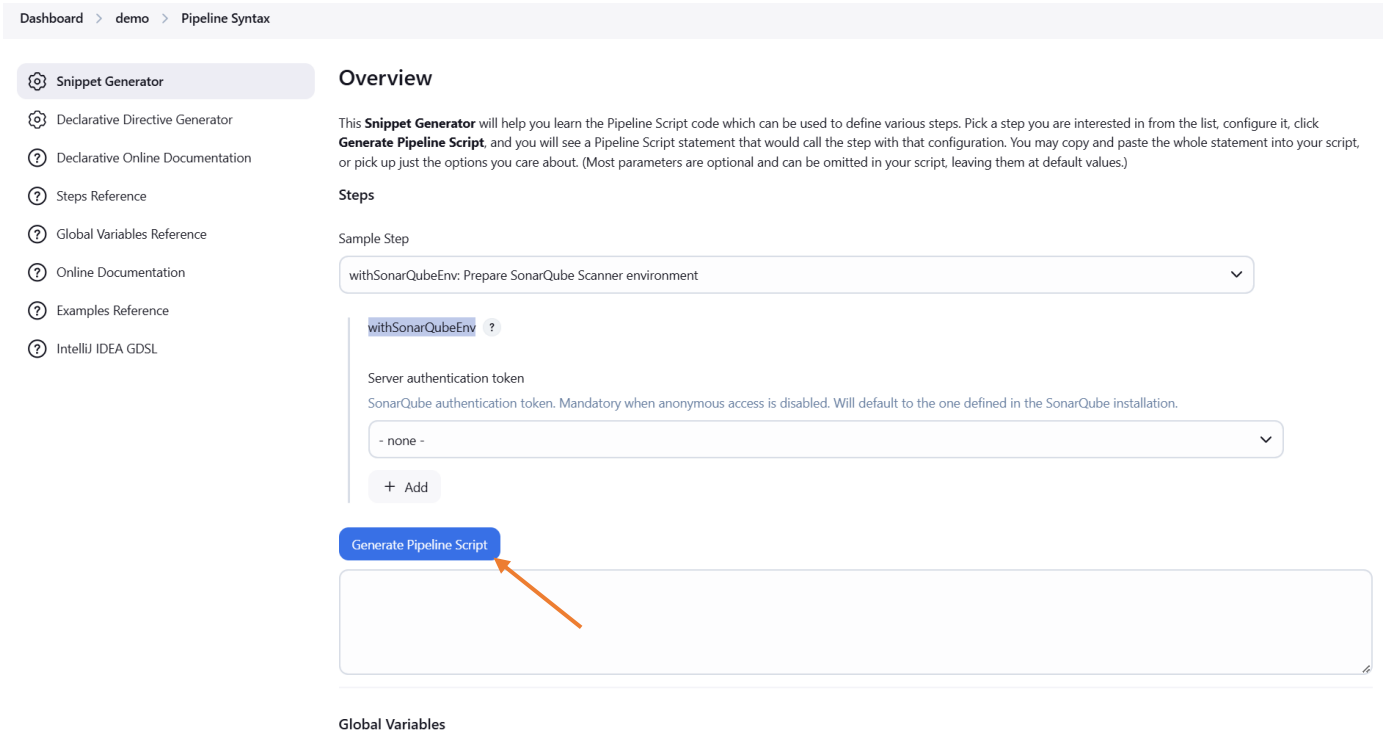
SonarQube has been defined as an environment variable.



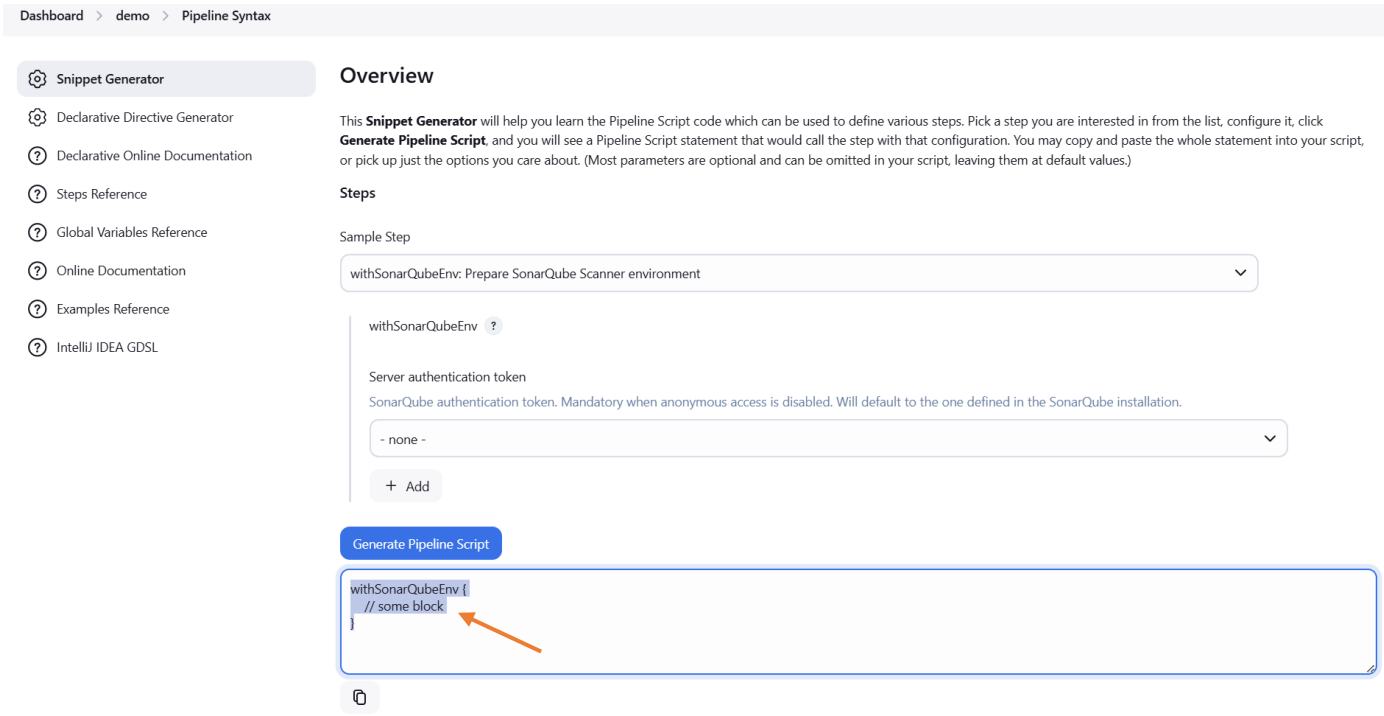
Now, let us add the stage for the SonarQube scanner. First, go to the page for "**Pipeline Syntax**"



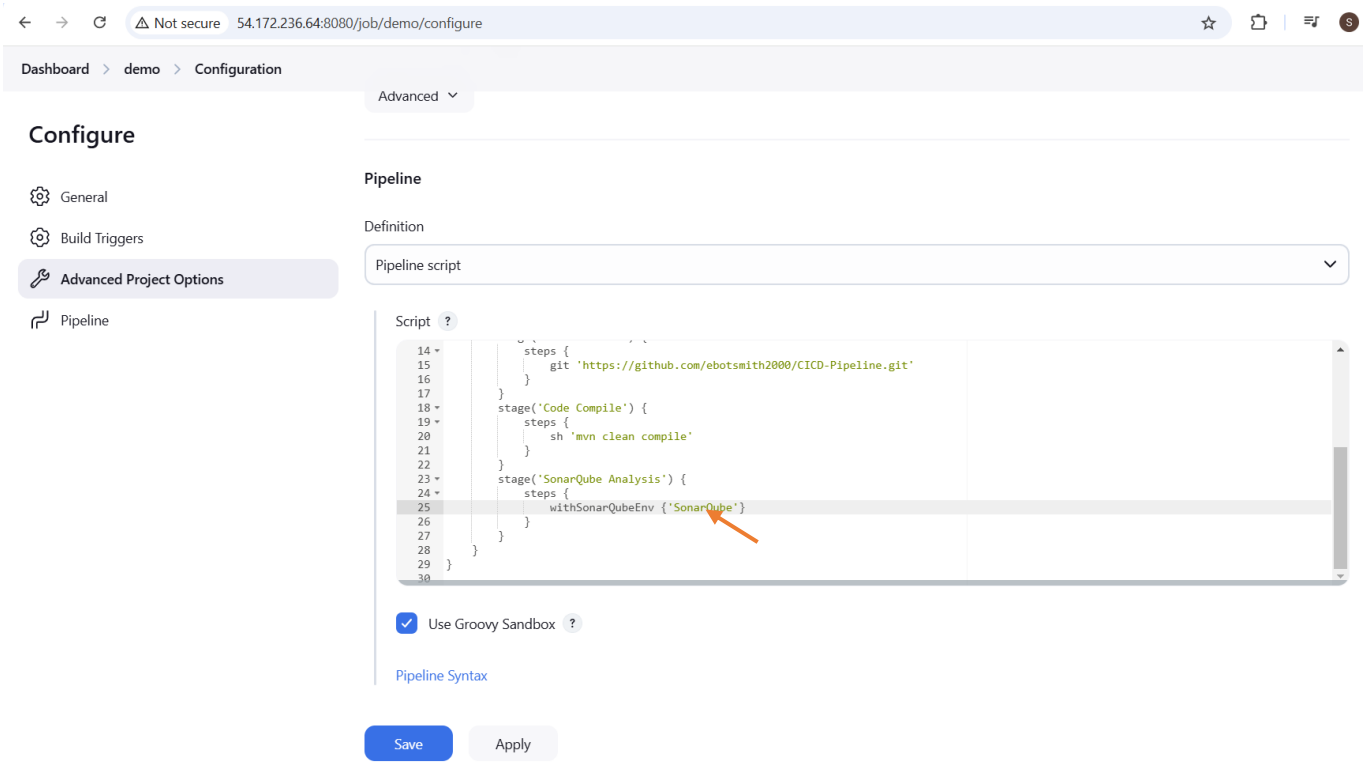
Refresh the page and under "**sample step**", select the item related to SonarQube. In this case it is "**withSonarQubeEnv**"



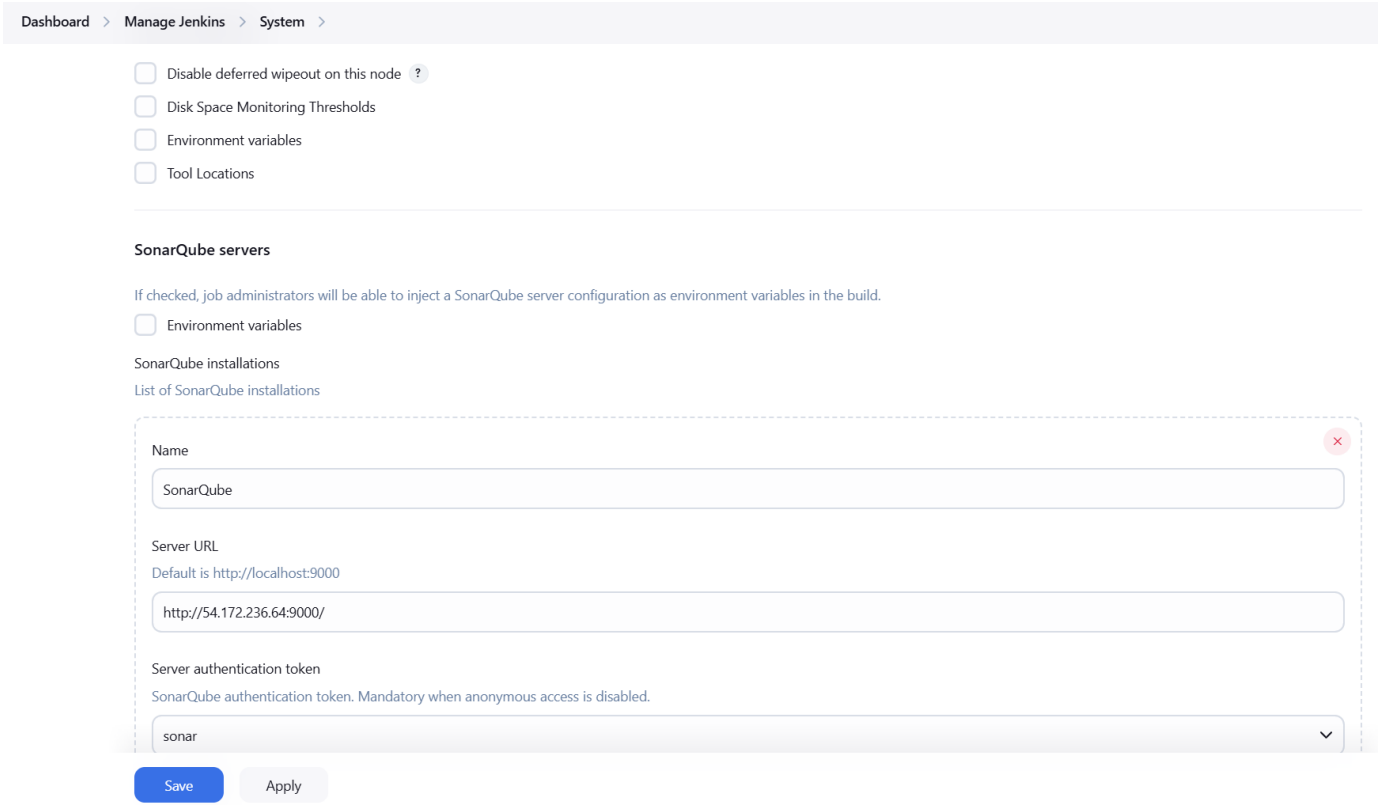
Click on “**Generate Pipeline Script**”



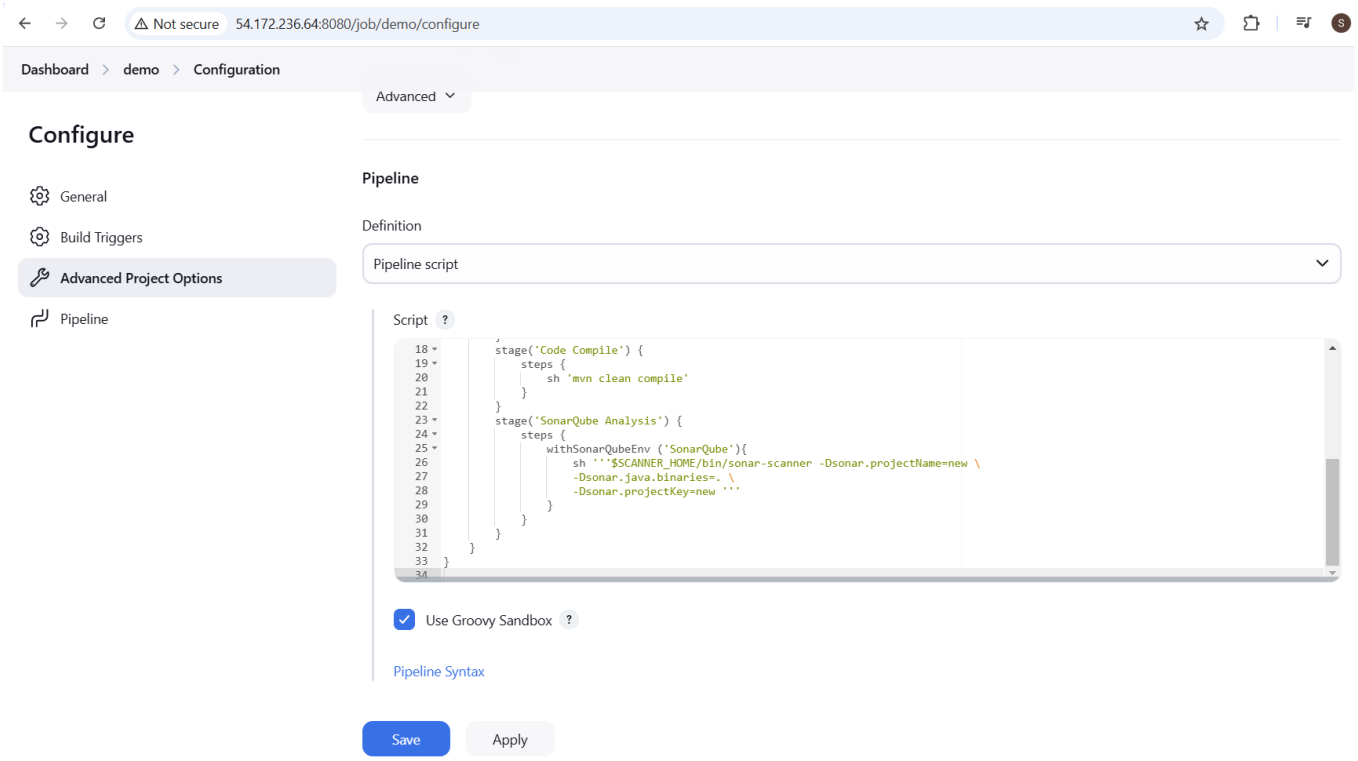
We have to copy this and use on the definition of the stage in our configuration of SonarQube

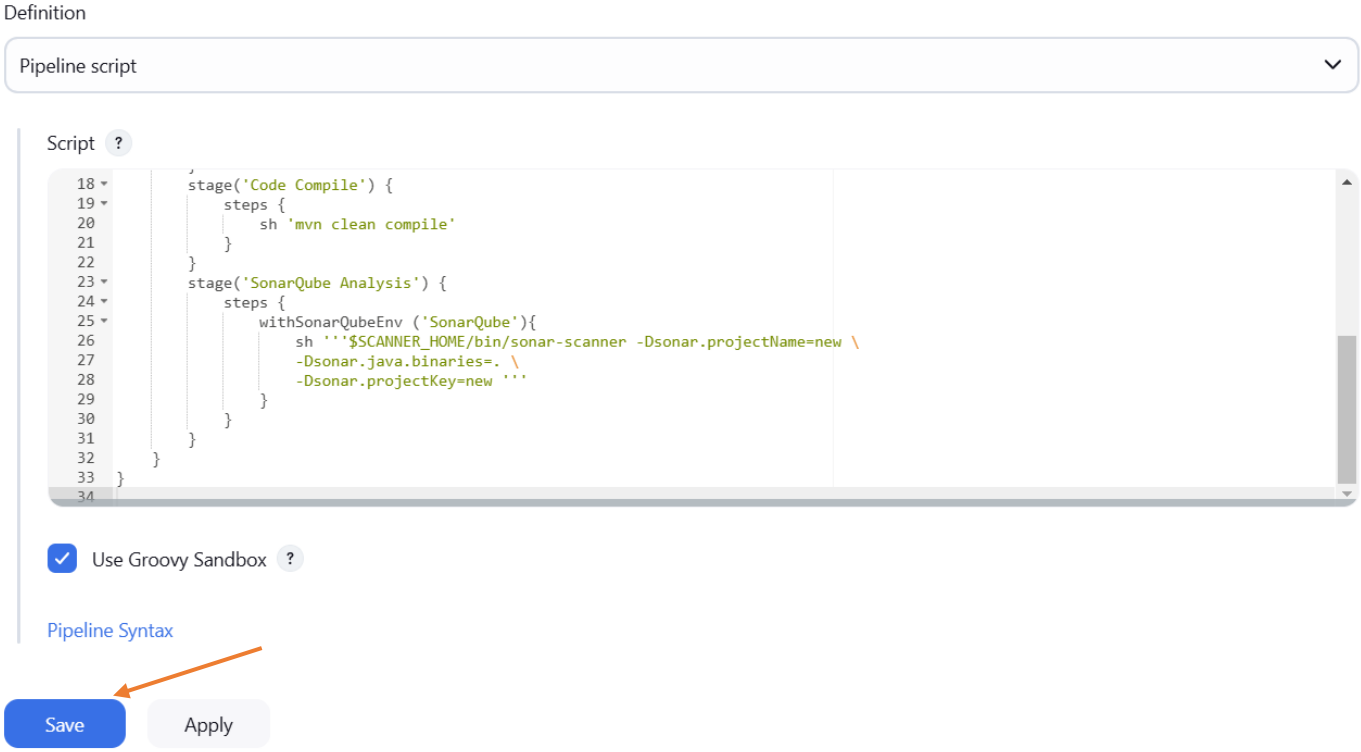


We got the name "**SonarQube**" by going to "**Manage Jenkins**" -> "**Systems**" and scroll down to "**SonarQube Servers**" as shown below.



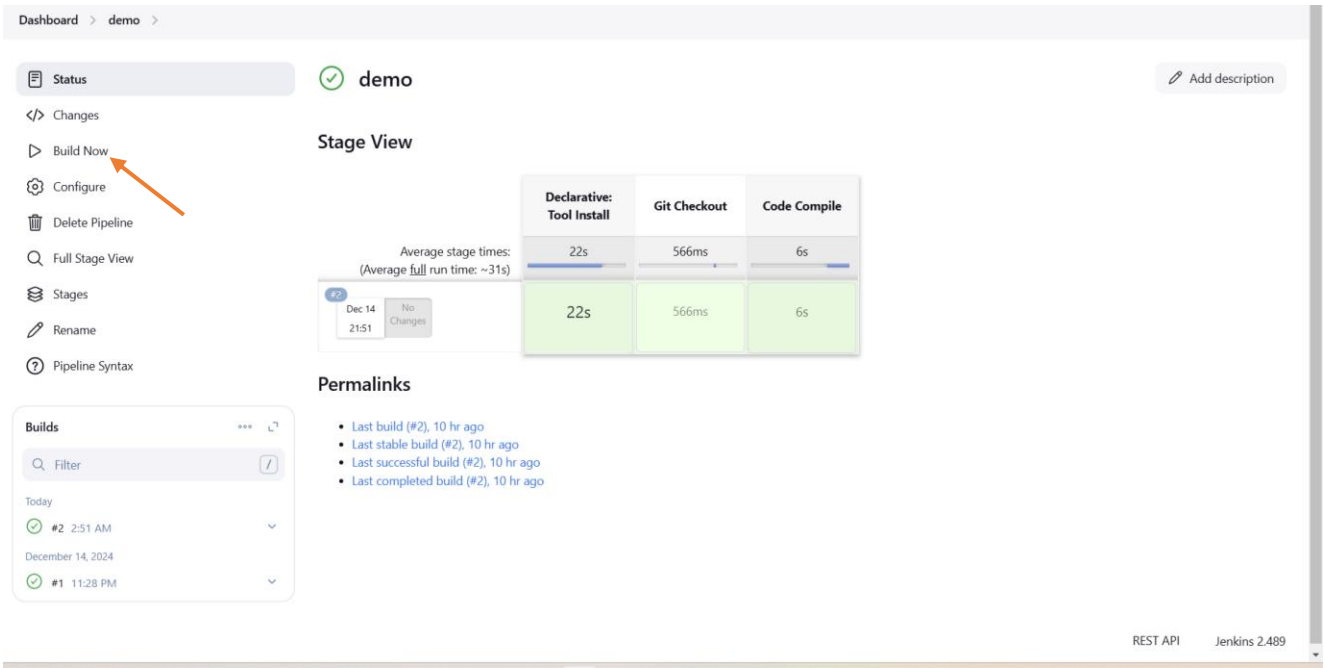
On the "**Configuration**" page and continue


Not secure 54.172.236.64:8080/job/demo/configure
Dashboard > demo > Configuration
Configure
General
Build Triggers
Advanced Project Options
Pipeline
Advanced
Pipeline
Definition
Pipeline script
Script
stage('Code Compile') {
steps {
sh 'mvn clean compile'
}
}
stage('SonarQube Analysis') {
steps {
withSonarQubeEnv ('SonarQube'){
sh '''$SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=new \
-Dsonar.java.binaries=. \
-Dsonar.projectKey=new '''
}
}
}
}
Use Groovy Sandbox
Pipeline Syntax
Save
Apply

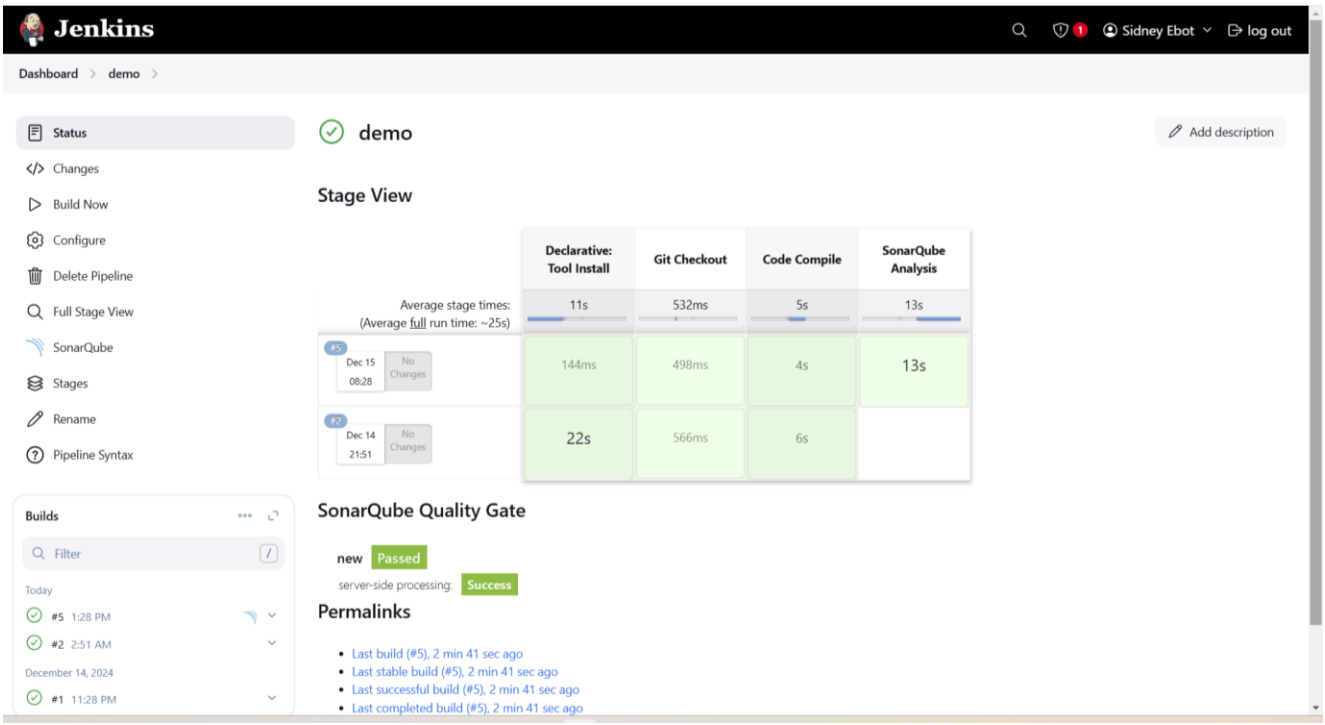

Pipeline
Definition
Pipeline script
Script
stage('Code Compile') {
steps {
sh 'mvn clean compile'
}
}
stage('SonarQube Analysis') {
steps {
withSonarQubeEnv ('SonarQube'){
sh '''$SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=new \
-Dsonar.java.binaries=. \
-Dsonar.projectKey=new '''
}
}
}
}
Use Groovy Sandbox
Pipeline Syntax
Save
Apply

```
pipeline {
    agent any

    tools{
        jdk 'jdk'
        maven 'maven3'
    }
    environment{
        SCANNER_HOME= tool 'sonar-scanner'
    }

    stages {
        stage('Git Checkout') {
            steps {
                git 'https://github.com/ebotsmith2000/jenkins-pipeline.git'
            }
        }
        stage('Code Compile') {
            steps {
                sh 'mvn clean compile'
            }
        }
         stage('SonarQube Analysis') {
            steps {
                withSonarQubeEnv('SonarQube'){
                    sh '''$SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=new \
                    -Dsonar.java.binaries=. \
                    -Dsonar.projectKey=new '''
                }
            }
        }
    }
}
```

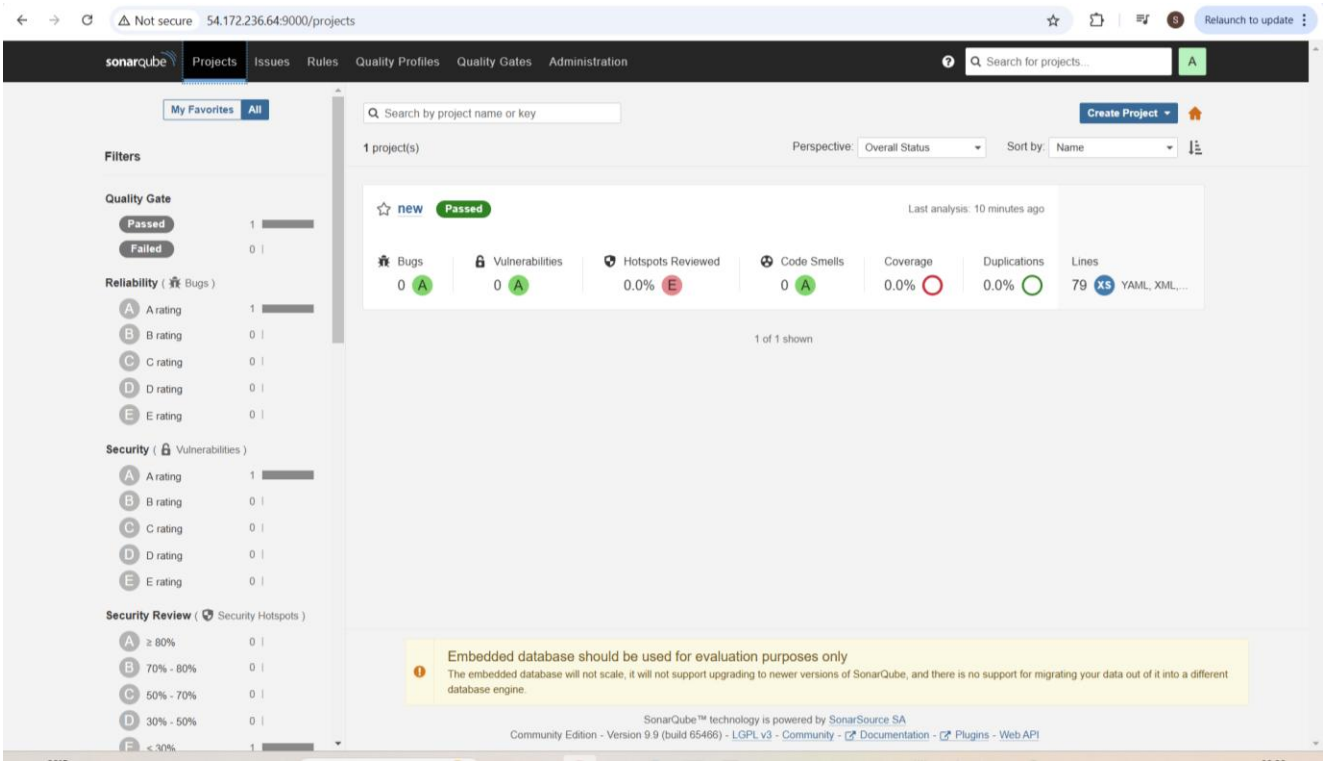
Click on "**Save**"



Now, let us build the pipeline by clicking on “**Build Now**”



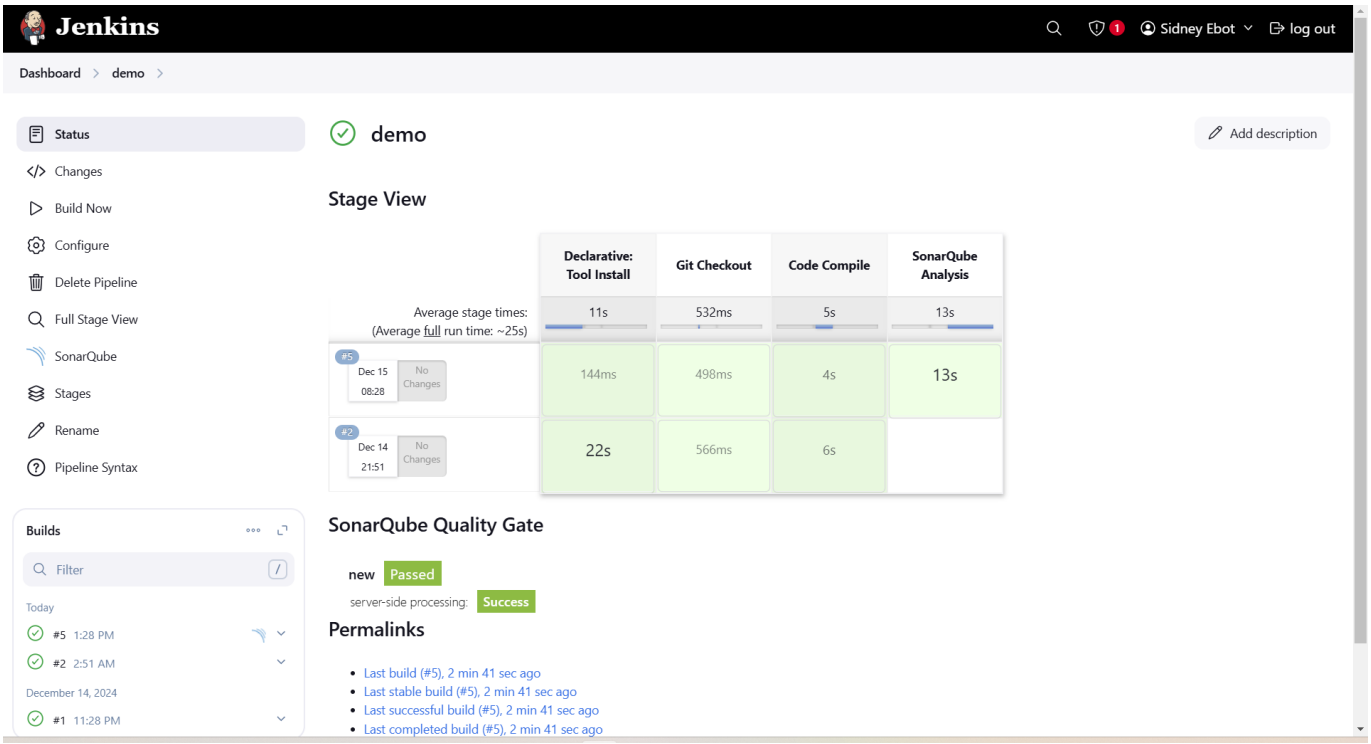
The build is successful. Go now to SonarQube browser and refresh it. Then click on the “**Project**” tab



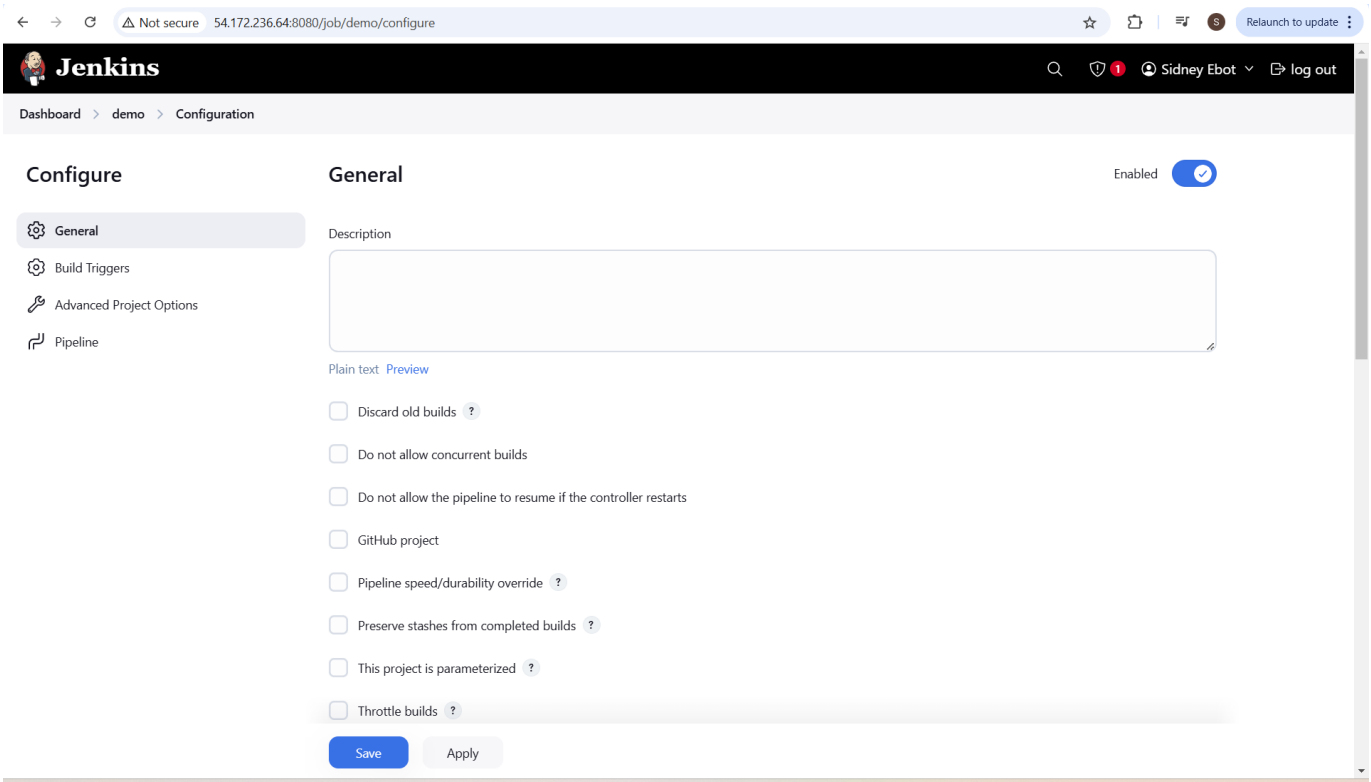
You can see that it passed and we have scanned 79 lines. Static Code Analysis is done. So, we go to the next step, that is “**Code Build**”.

STEP 12: Configure Maven for Code Build

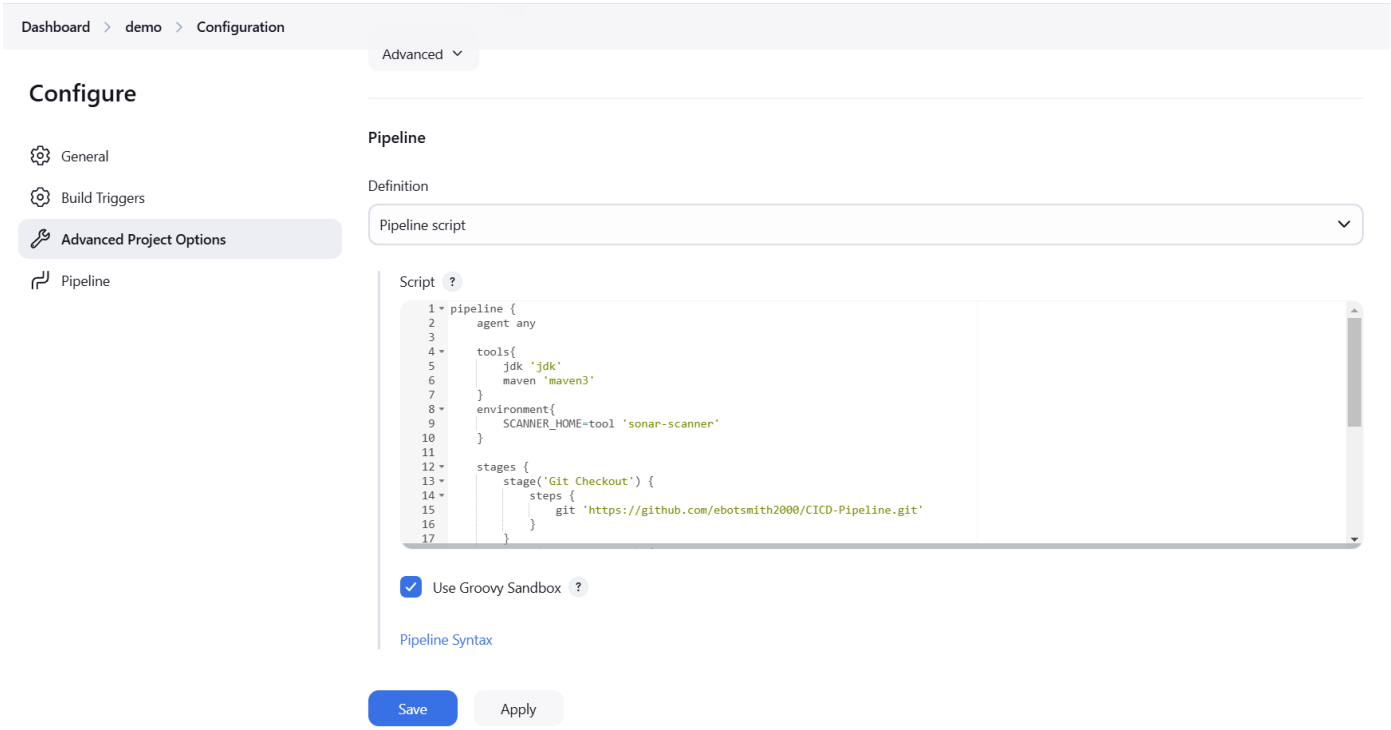
To do Code Build, we have to use "**Maven**". We have already configured Maven. Let us go to the Jenkins GUI and add a stage for "**Code Build**".



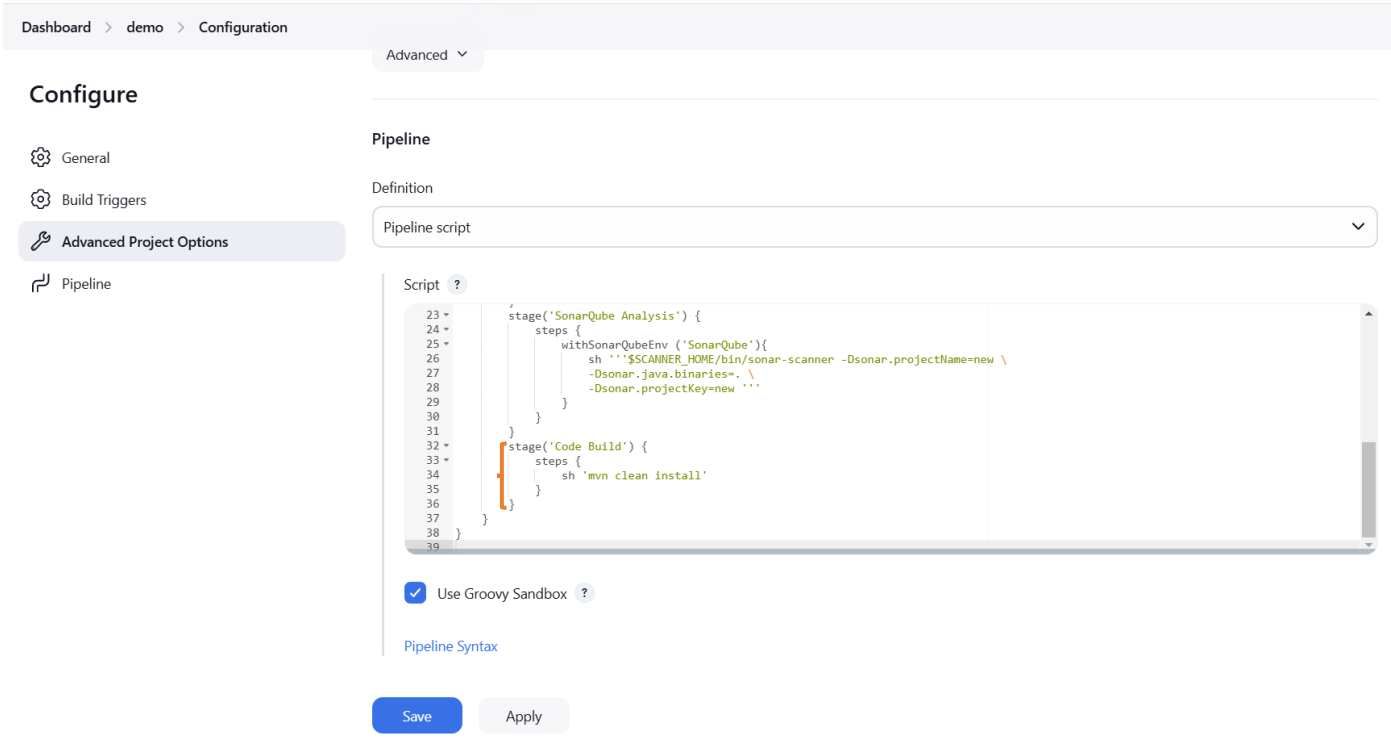
Click on "**Configure**"

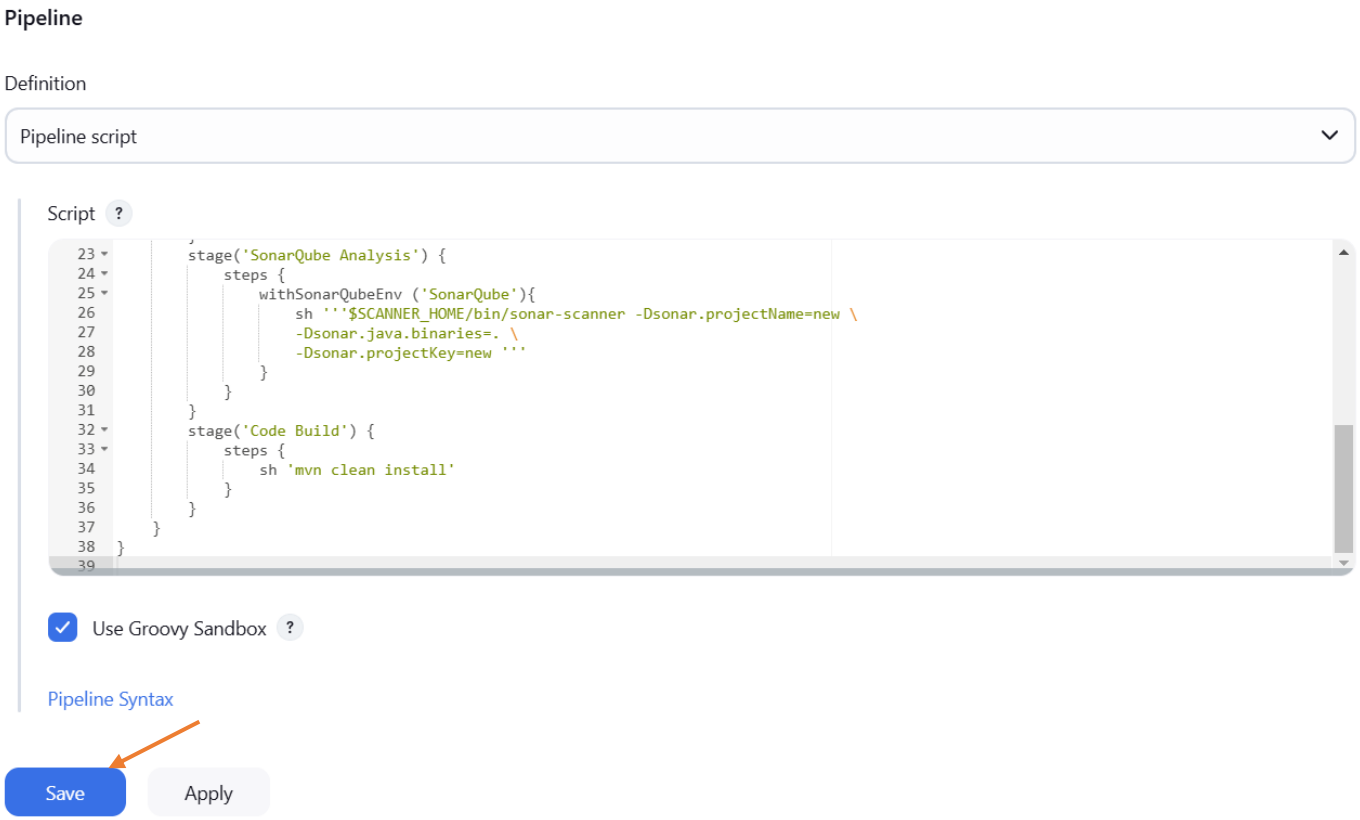


Scroll down to "**Pipeline**"



Let us add another stage for "**Code Build**"

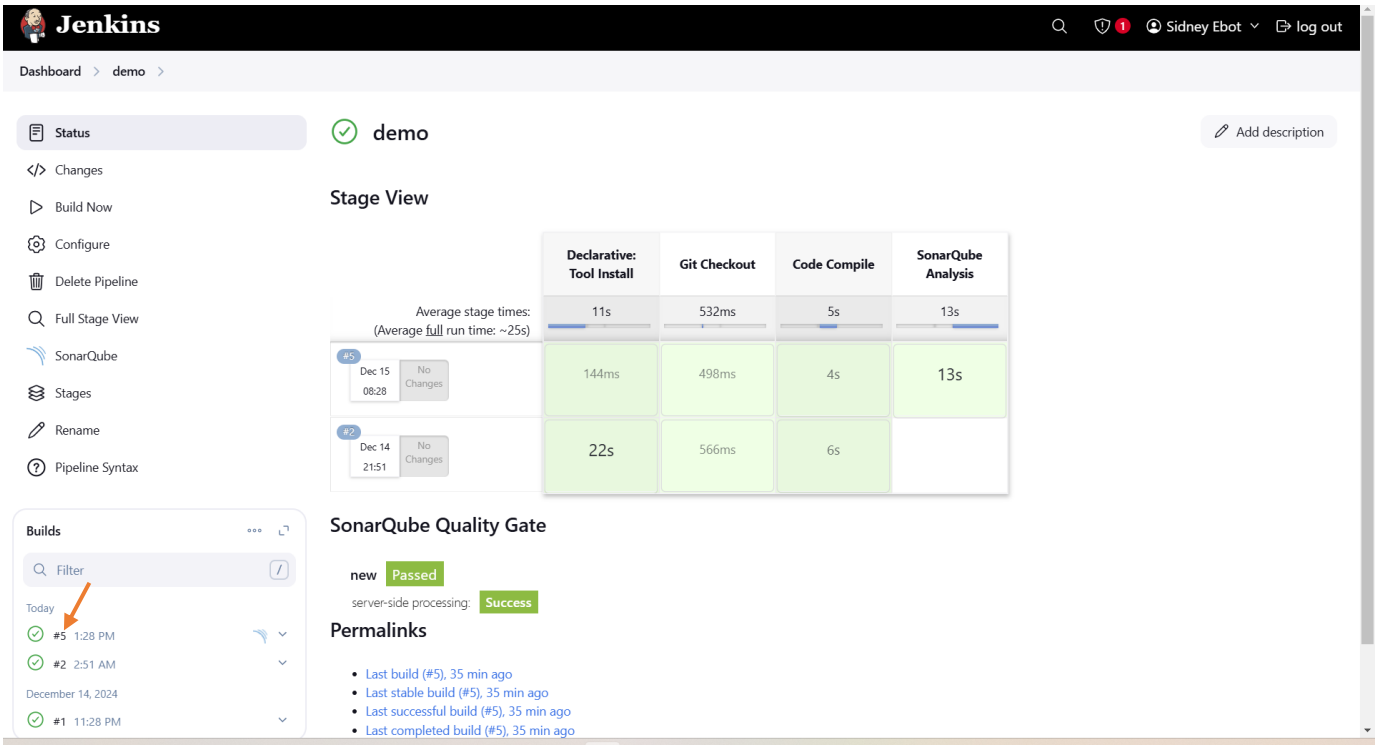




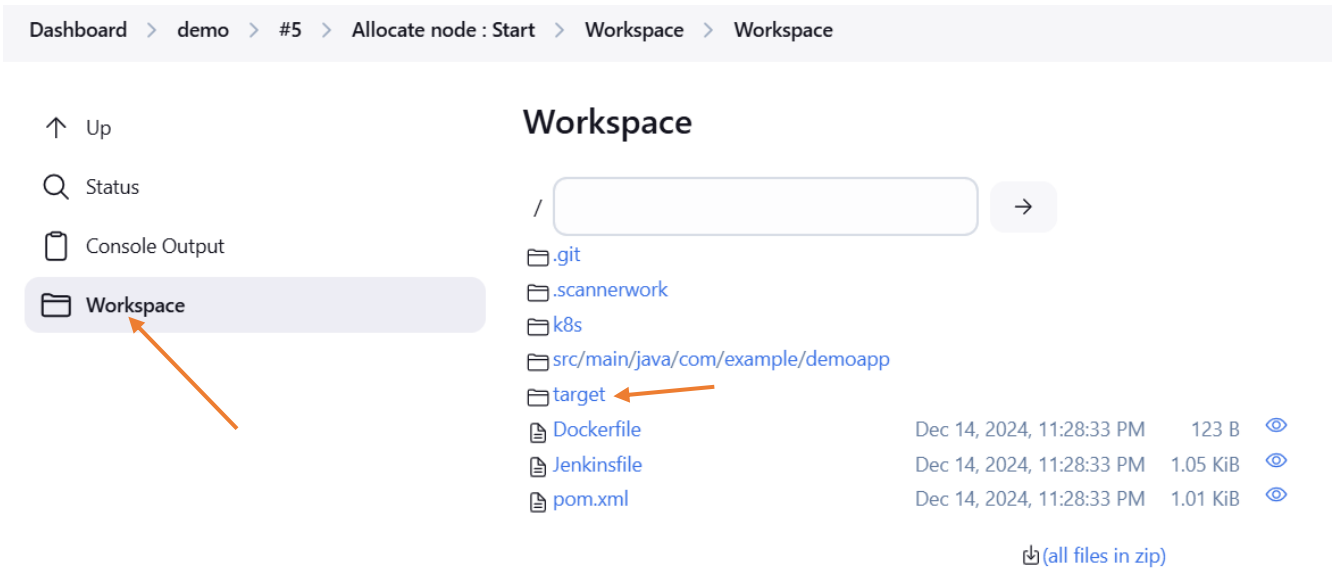

```
pipeline {
    agent any

    tools{
        jdk 'jdk'
        maven 'maven3'
    }
    environment{
        SCANNER_HOME= tool 'sonar-scanner'
    }

    stages {
        stage('Git Checkout') {
            steps {
                git 'https://github.com/ebotsmith2000/jenkins-pipeline.git'
            }
        }
        stage('Code Compile') {
            steps {
                sh 'mvn clean compile'
            }
        }
        stage('SonarQube Analysis') {
            steps {
                withSonarQubeEnv('SonarQube'){
                    sh '''$SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=new \
                    -Dsonar.java.binaries=. \
                    -Dsonar.projectKey=new '''
                }
            }
```

```
        }
        stage('Code Build') {
            steps {
                sh 'mvn clean install'
            }
        }
    }
}
```

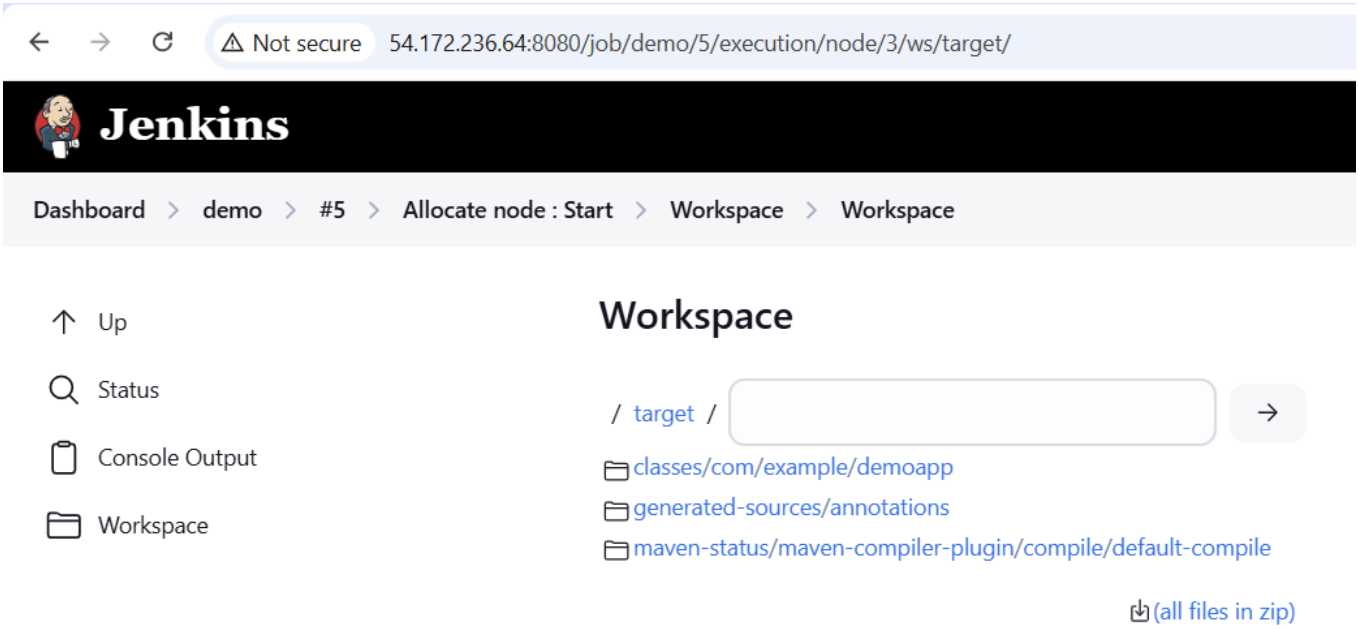
Install will build the code. Click on "**Save**"



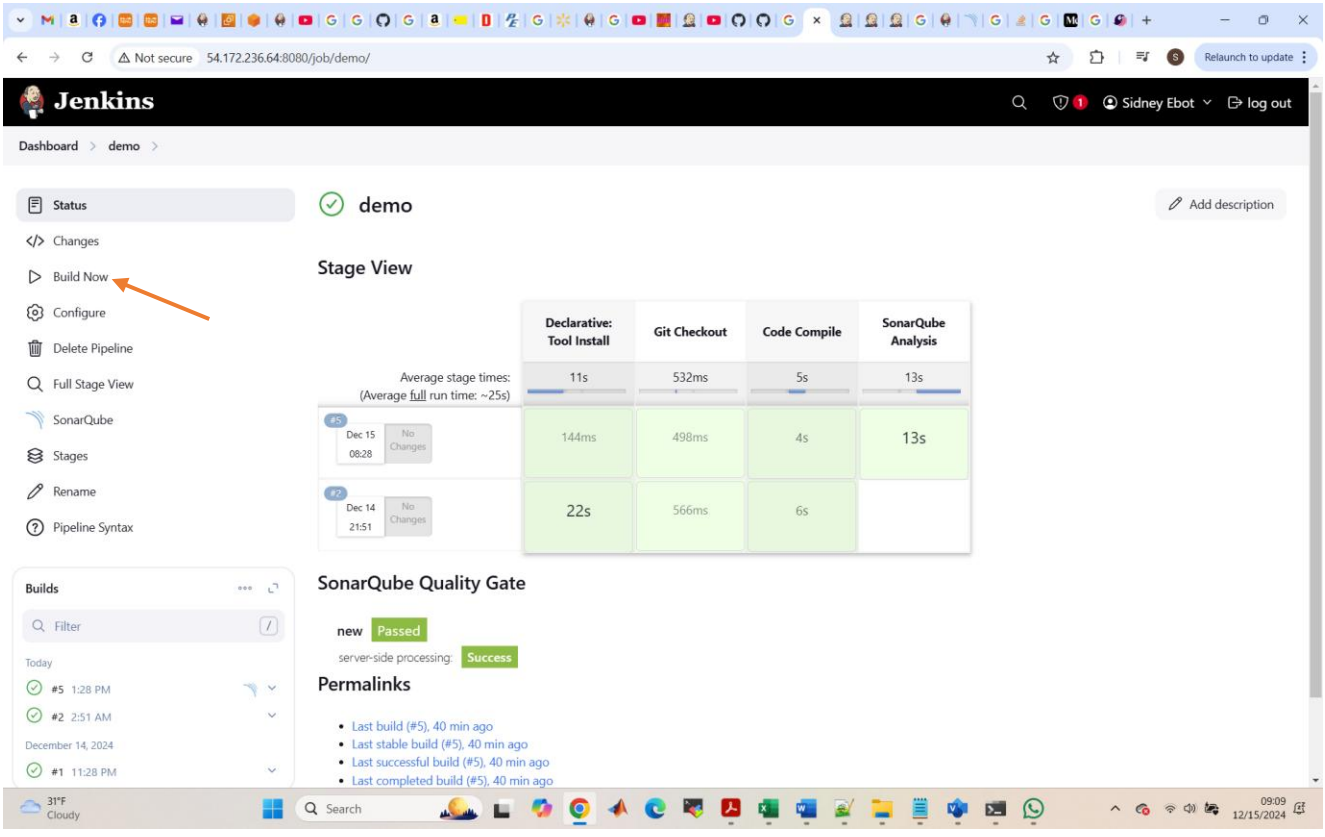
Go to "**Workspace**" by clicking on **#5** -> Then click on "**Workspace**" and click on "/var/lib/jenkins/workspace/demo on built-in"



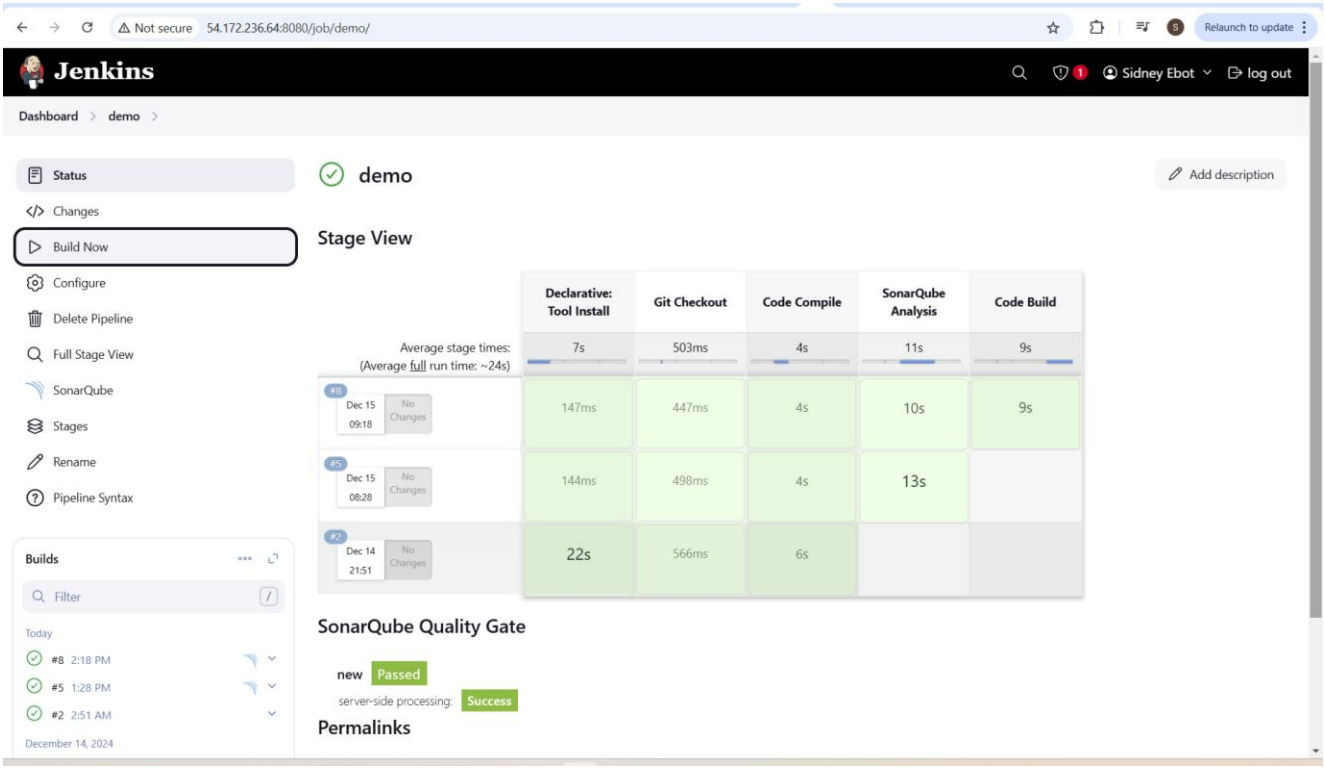
Click on “**target**”



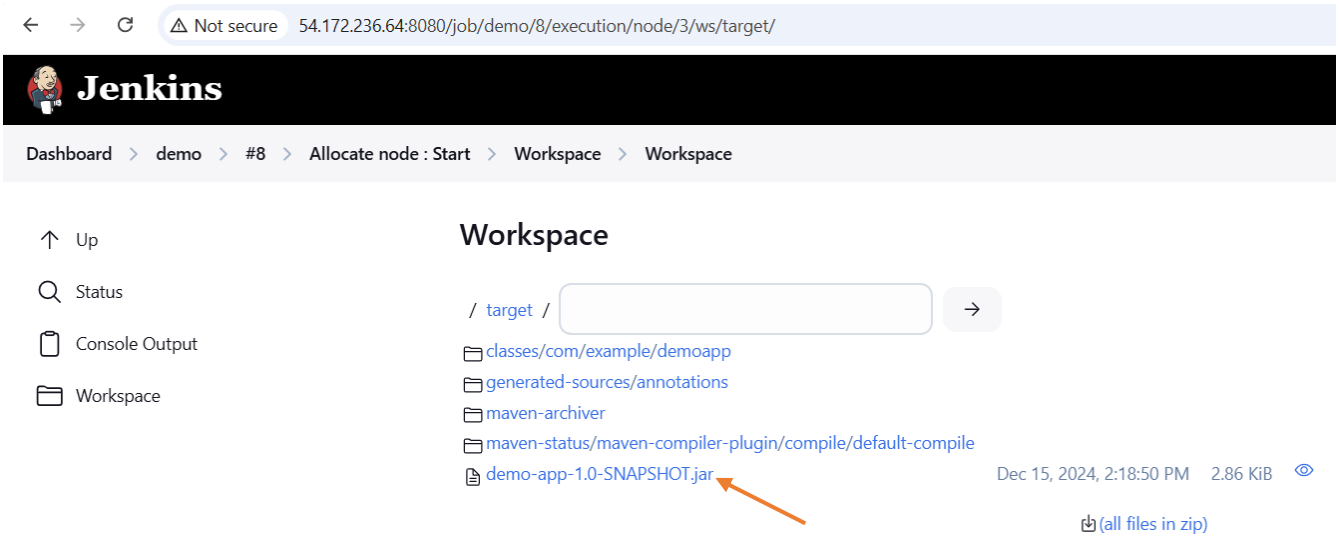
Here you can see there is **no Jar file**. We will expect a Jar file when the “**Code Build**” is successful. Now let us go back to our pipeline



Click on “**Build Now**”, we will get another stage called “Code Build”



The "**Code Build**" stage is successful. Now let us check the "**Workspace**" again



You can now see that a Jar file has been created. The "**Code Build**" is successful.

STEP 13: Install Docker and Build Docker Image

Let us go to the next step which is "**Docker Build**". If we want to execute Docker, we need to first install Docker to our EC2 instance.

Installation of Docker in Ubuntu:

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

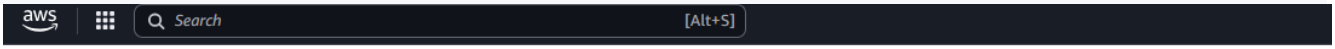
Copy the above code and paste on the terminal of the instance

```
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

```
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Ign:5 https://pkg.jenkins.io/debian binary/ InRelease
Hit:6 https://pkg.jenkins.io/debian binary/ Release
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 Components [151 kB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Components [309 kB]
Get:9 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Components [212 B]
Get:10 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Components [940 B]
Get:11 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/main amd64 Components [208 B]
Get:12 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/universe amd64 Components [11.7 kB]
Get:13 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/restricted amd64 Components [216 B]
Get:14 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/multiverse amd64 Components [212 B]
Get:16 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [7188 B]
Get:17 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Components [51.8 kB]
Get:18 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Components [212 B]
Get:19 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Components [212 B]
Fetched 912 kB in 1s (1592 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20240203).
ca-certificates set to manually installed.
curl is already the newest version (8.5.0-2ubuntu10.5).
curl set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 28 not upgraded.
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Get:4 https://download.docker.com/linux/ubuntu noble InRelease [48.8 kB]
Ign:5 https://pkg.jenkins.io/debian binary/ InRelease
Hit:6 https://pkg.jenkins.io/debian binary/ Release
Hit:7 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:8 https://download.docker.com/linux/ubuntu noble/stable amd64 Packages [16.6 kB]
Fetched 65.4 kB in 1s (119 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-31-55:~$
```

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

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




```
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Ign:5 https://pkg.jenkins.io/debian binary/ InRelease
Hit:6 https://pkg.jenkins.io/debian binary/ Release
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 Components [151 kB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Components [309 kB]
Get:9 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Components [212 B]
Get:10 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Components [940 B]
Get:11 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/main amd64 Components [208 B]
Get:12 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/universe amd64 Components [11.7 kB]
Get:13 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/restricted amd64 Components [216 B]
Get:14 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/multiverse amd64 Components [212 B]
Get:16 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [7188 B]
Get:17 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Components [51.8 kB]
Get:18 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Components [212 B]
Get:19 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Components [212 B]
Fetched 912 kB in 1s (1592 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20240203).
ca-certificates set to manually installed.
curl is already the newest version (8.5.0-2ubuntu10.5).
curl set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 28 not upgraded.
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Get:4 https://download.docker.com/linux/ubuntu noble InRelease [48.8 kB]
Ign:5 https://pkg.jenkins.io/debian binary/ InRelease
Hit:6 https://pkg.jenkins.io/debian binary/ Release
Hit:7 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:8 https://download.docker.com/linux/ubuntu noble/stable amd64 Packages [16.6 kB]
Fetched 65.4 kB in 1s (119 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-31-55:~$ echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:5 https://download.docker.com/linux/ubuntu noble InRelease
Ign:6 https://pkg.jenkins.io/debian binary/ InRelease
Hit:7 https://pkg.jenkins.io/debian binary/ Release
Reading package lists... Done
ubuntu@ip-172-31-31-55:~$
```

i-03a828fc255dc93a5 (My Server)

PublicIPs: 54.172.236.64 PrivateIPs: 172.31.31.55

Then run the command:

To Install the latest Version:

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

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

```
ubuntu@ip-172-31-31-55:~$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  docker-ce-rootless-extras libltdl7 libslirp0 pigz slirp4netns
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
The following NEW packages will be installed:
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libltdl7 libslirp0 pigz slirp4netns
0 upgraded, 10 newly installed, 0 to remove and 28 not upgraded.
Need to get 124 MB of archives.
After this operation, 445 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Type "**Y**" and press ENTER

```
Unpacking docker-compose-plugin (2.31.0-1~ubuntu.24.04~noble) ...
Selecting previously unselected package libltdl7:amd64.
Preparing to unpack .../7-libltdl7_2.4.7-7build1_amd64.deb ...
Unpacking libltdl7:amd64 (2.4.7-7build1) ...
Selecting previously unselected package libslirp0:amd64.
Preparing to unpack .../8-libslirp0_4.7.0-1ubuntu3_amd64.deb ...
Unpacking libslirp0:amd64 (4.7.0-1ubuntu3) ...
Selecting previously unselected package slirp4netns.
Preparing to unpack .../9-slirp4netns_1.2.1-1build2_amd64.deb ...
Unpacking slirp4netns (1.2.1-1build2) ...
Setting up docker-buildx-plugin (0.19.2-1~ubuntu.24.04~noble) ...
Setting up containerd.io (1.7.24-1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/containerd.service → /usr/lib/systemd/system/containerd.service.
Setting up docker-compose-plugin (2.31.0-1~ubuntu.24.04~noble) ...
Setting up libltdl7:amd64 (2.4.7-7build1) ...
Setting up docker-ce-cli (5:27.4.0-1~ubuntu.24.04~noble) ...
Setting up libslirp0:amd64 (4.7.0-1ubuntu3) ...
Setting up pigz (2.8-1) ...
Setting up docker-ce-rootless-extras (5:27.4.0-1~ubuntu.24.04~noble) ...
Setting up slirp4netns (1.2.1-1build2) ...
Setting up docker-ce (5:27.4.0-1~ubuntu.24.04~noble) ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /usr/lib/systemd/system/docker.socket.
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.3) ...
Scanning processes...
Scanning candidates...
Scanning linux images...

Pending kernel upgrade!
Running kernel version:
  6.8.0-1018-aws
Diagnostics:
  The currently running kernel version is not the expected kernel version 6.8.0-1020-aws.

Restarting the system to load the new kernel will not be handled automatically, so you should consider rebooting.

Restarting services...

Service restarts being deferred:
 /etc/needrestart/restart.d/dbus.service
 systemctl restart networkd-dispatcher.service
 systemctl restart unattended-upgrades.service

No containers need to be restarted.

User sessions running outdated binaries:
 ubuntu @ session #54: java[8655]

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-31-55:~$
```

i-03a828fc255dc93a5 (My Server)

PublicIPs: 54.172.236.64 PrivateIPs: 172.31.31.55

Docker has been installed. Let us verify the installation using this command:

sudo docker run hello-world

```
ubuntu@ip-172-31-31-55:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:5b3cc85e16e3058003c13b7821318369dad01dac3dbb877aac3c28182255c724
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

ubuntu@ip-172-31-31-55:~$
```

This confirms that Docker has been installed. But incase we have to use this Docker with Jenkins; we have to give some permissions for Docker and Jenkins.

We will do this by running the command:

sudo usermod -aG docker jenkins

```
ubuntu@ip-172-31-31-55:~$ sudo usermod -aG docker jenkins
ubuntu@ip-172-31-31-55:~$
```

We have given permission. So, if we want to use them, we have to restart Jenkins and Docker, else it won't work.

Let us first restart Jenkins using the command:

sudo systemctl restart jenkins

```
ubuntu@ip-172-31-31-55:~$ sudo systemctl restart jenkins
```

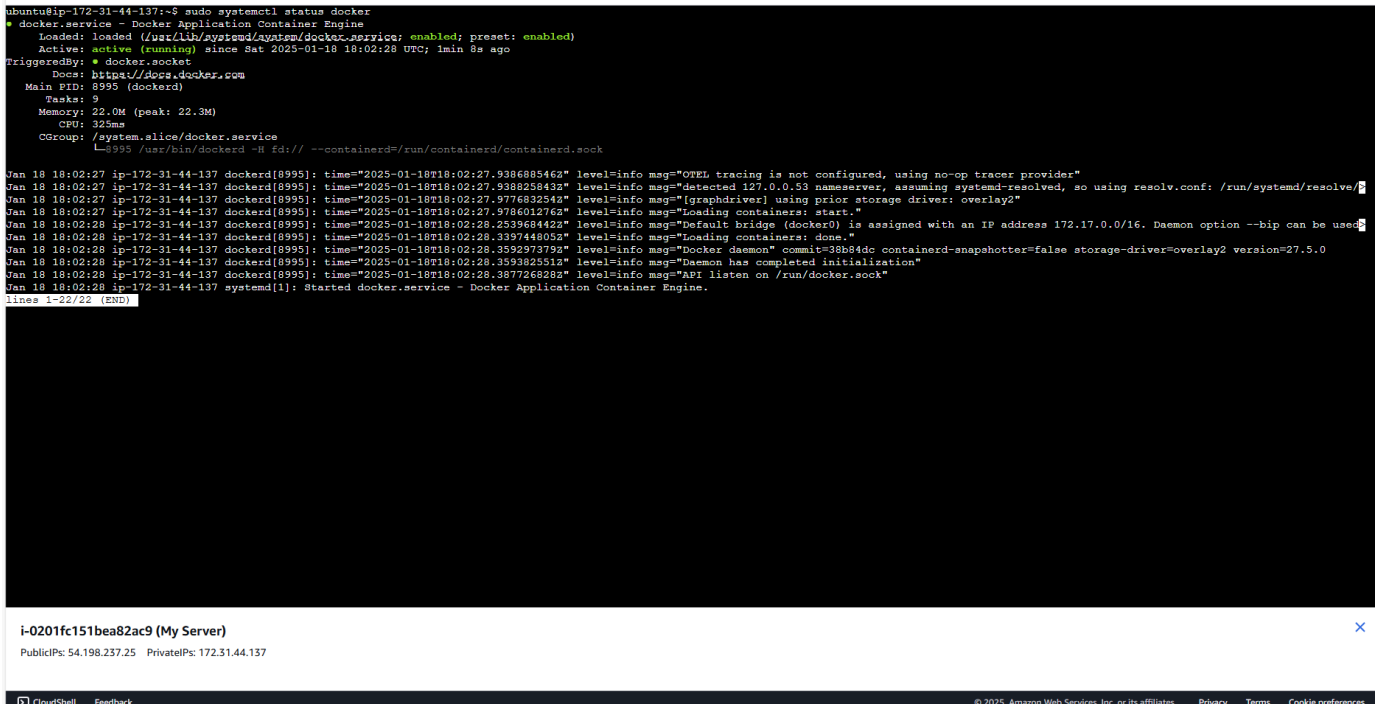
Now, let us restart Docker using the command:

sudo systemctl restart docker

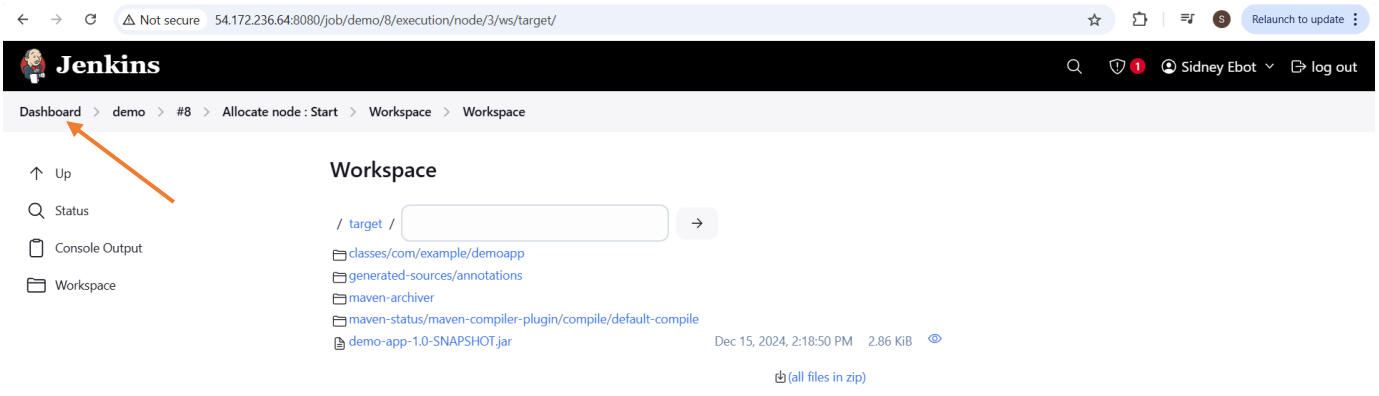
```
ubuntu@ip-172-31-31-55:~$ sudo systemctl restart jenkins
ubuntu@ip-172-31-31-55:~$ sudo systemctl restart docker
ubuntu@ip-172-31-31-55:~$
```

Verify Docker status using the command:

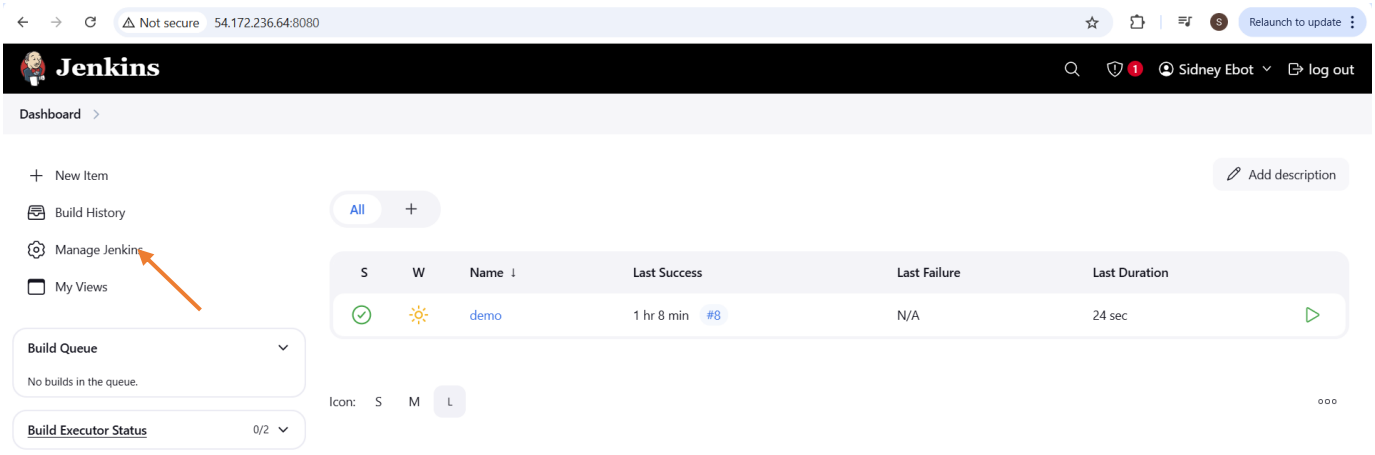
`sudo systemctl status docker`



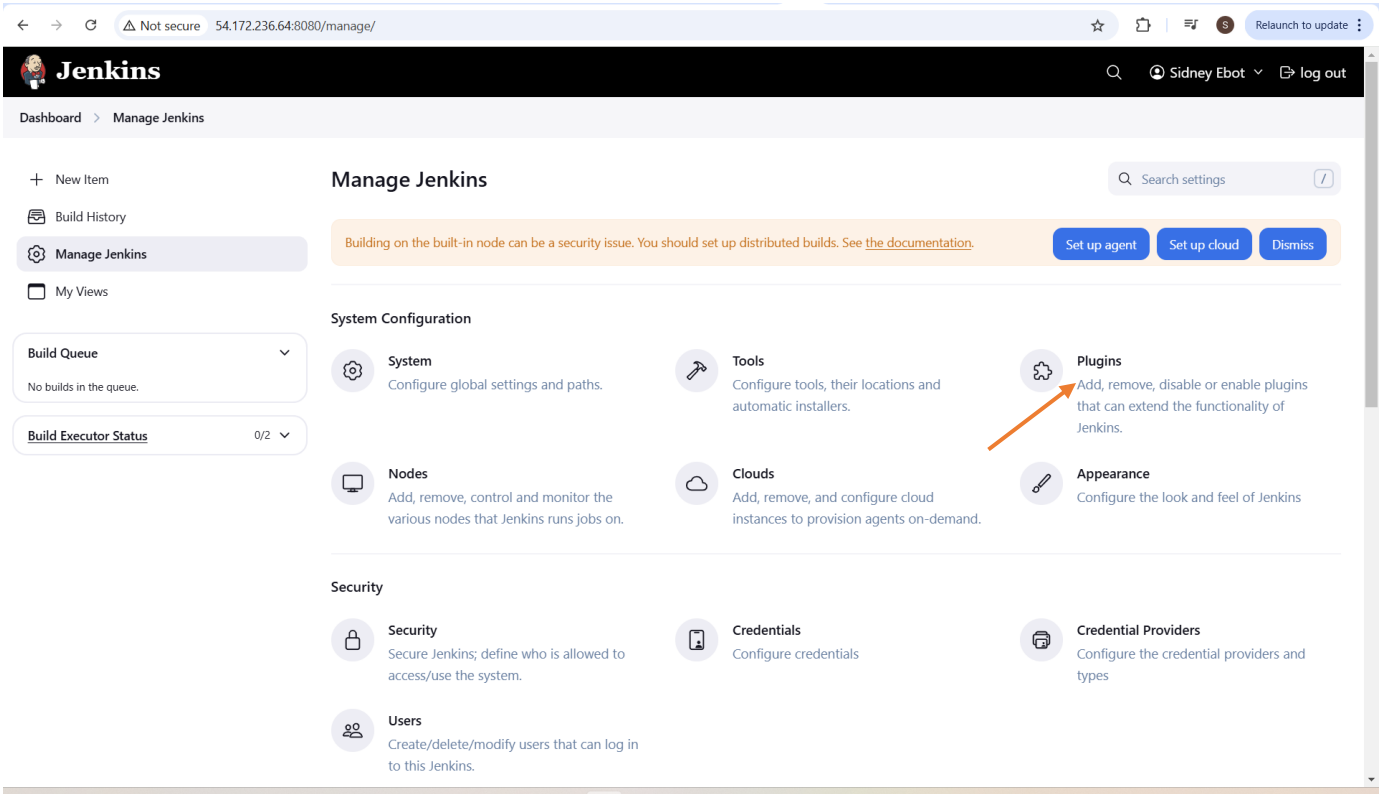
Go to the Jenkins GUI and refresh the page



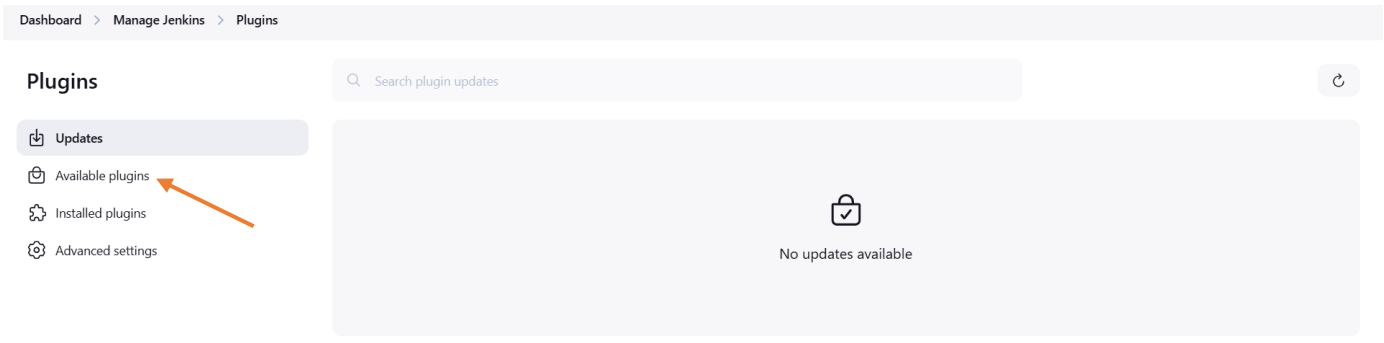
Click on "**Dashboard**"



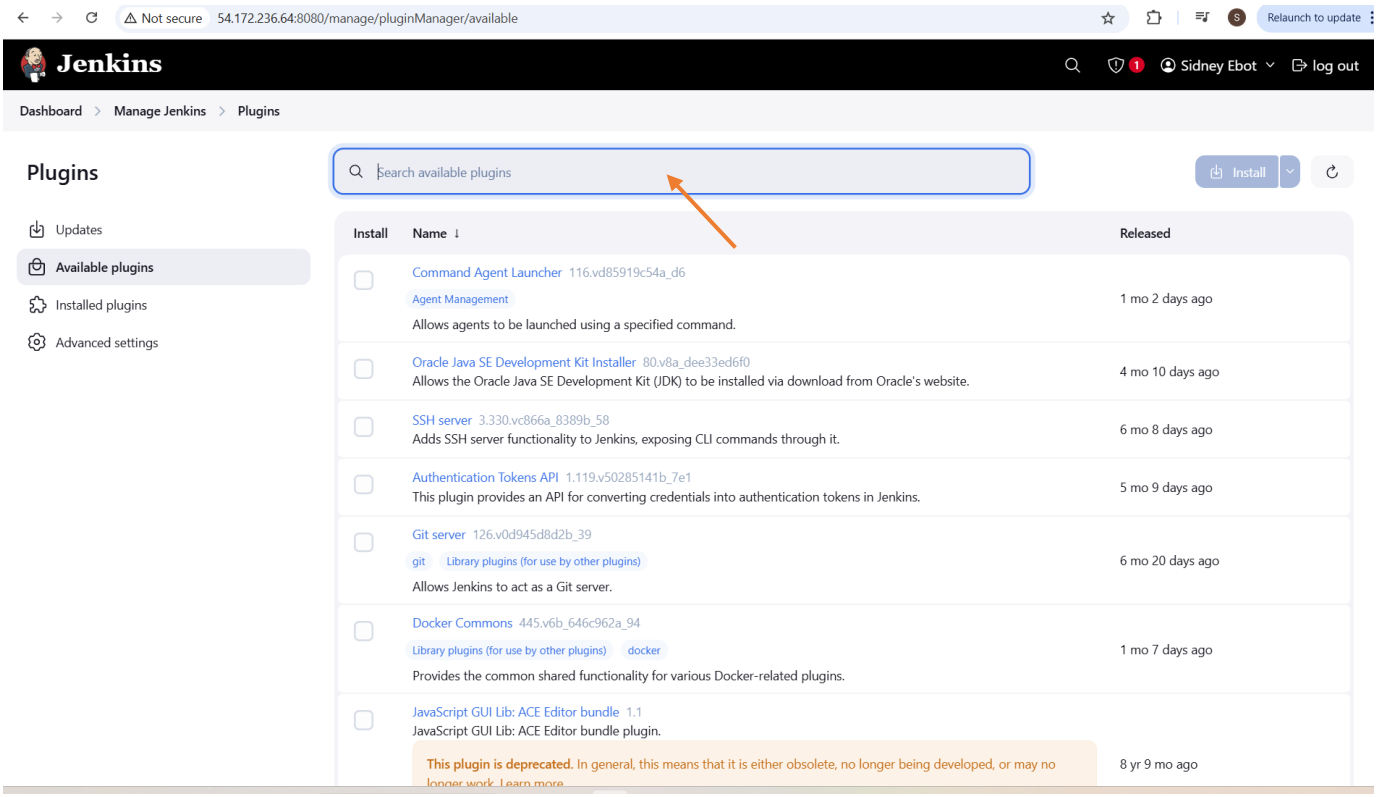
Click on “**Manage Jenkins**”



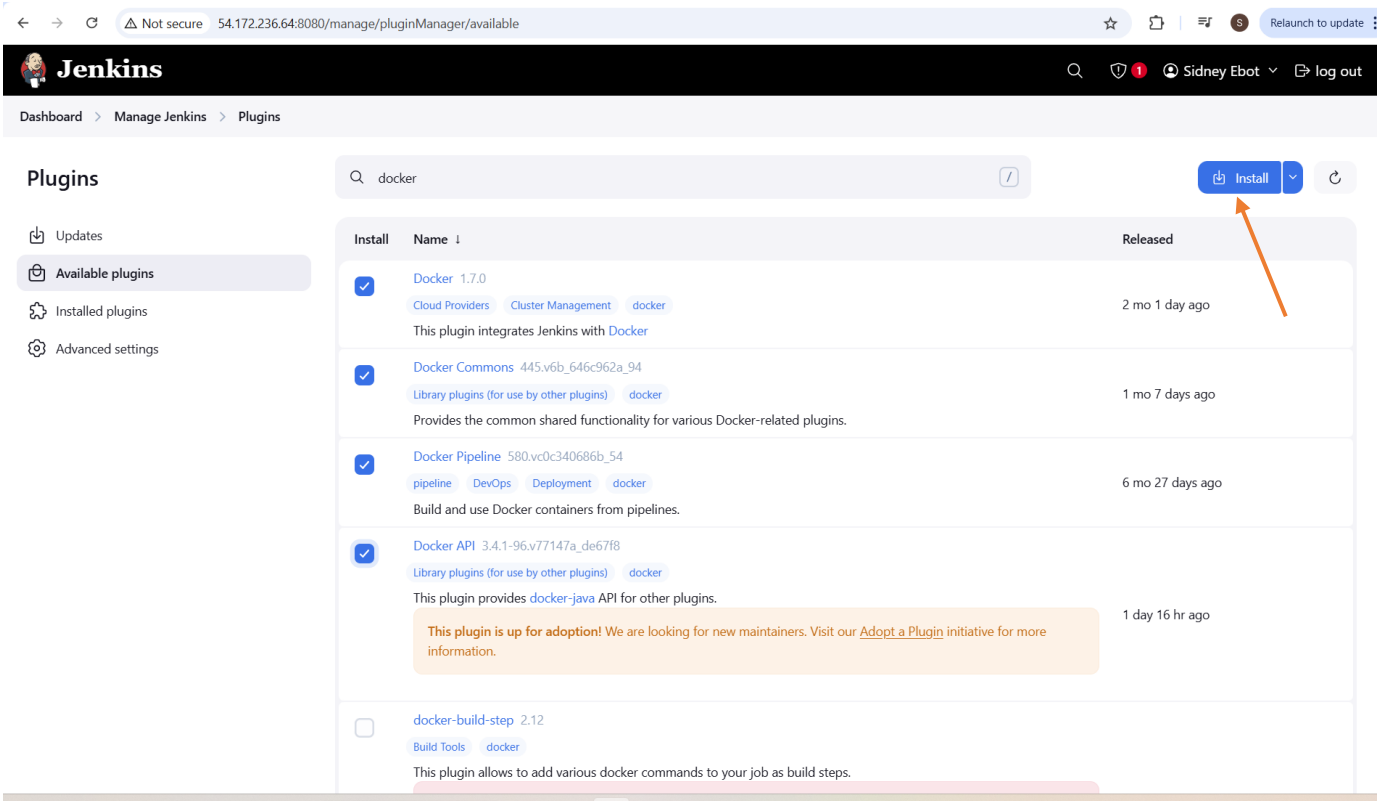
Click on “**Plugins**”



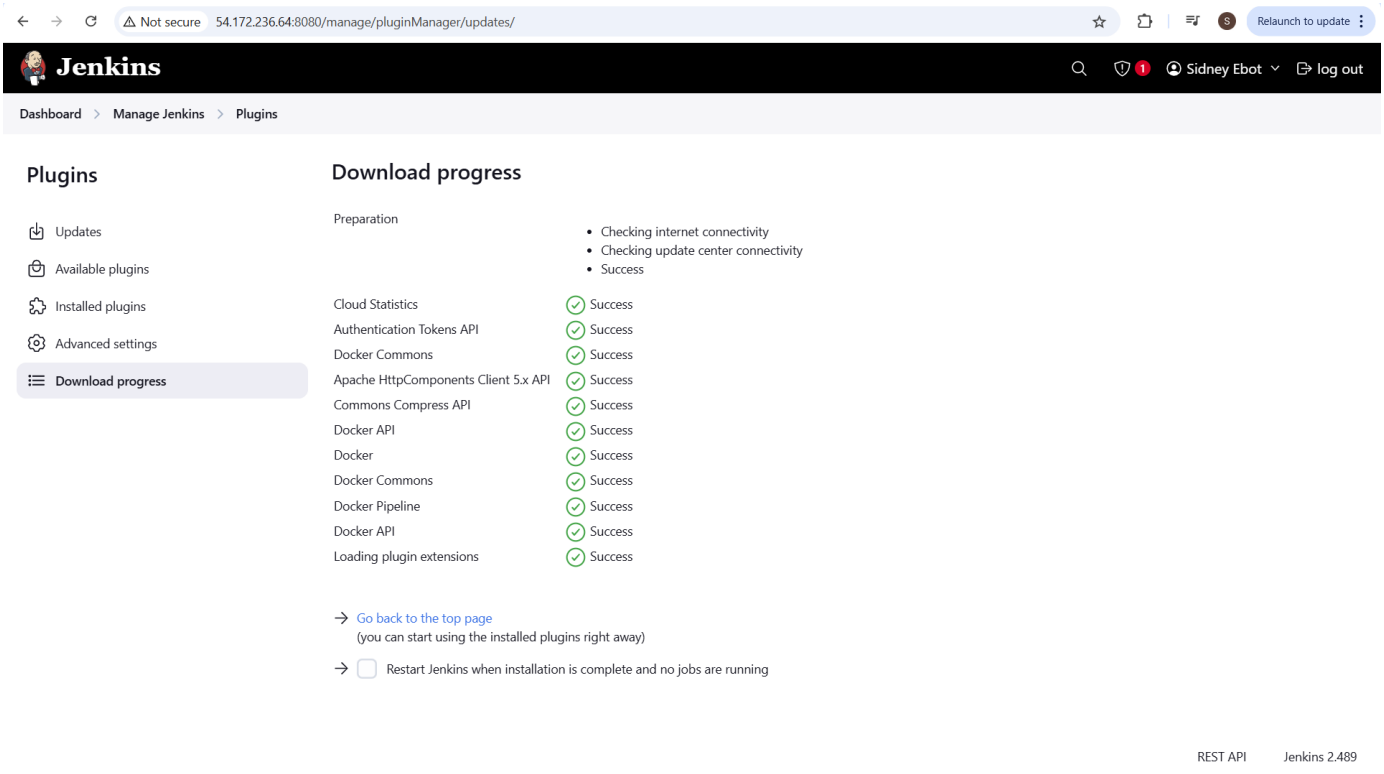
Click on “**Available Plugins**”



Search for “**Docker**”

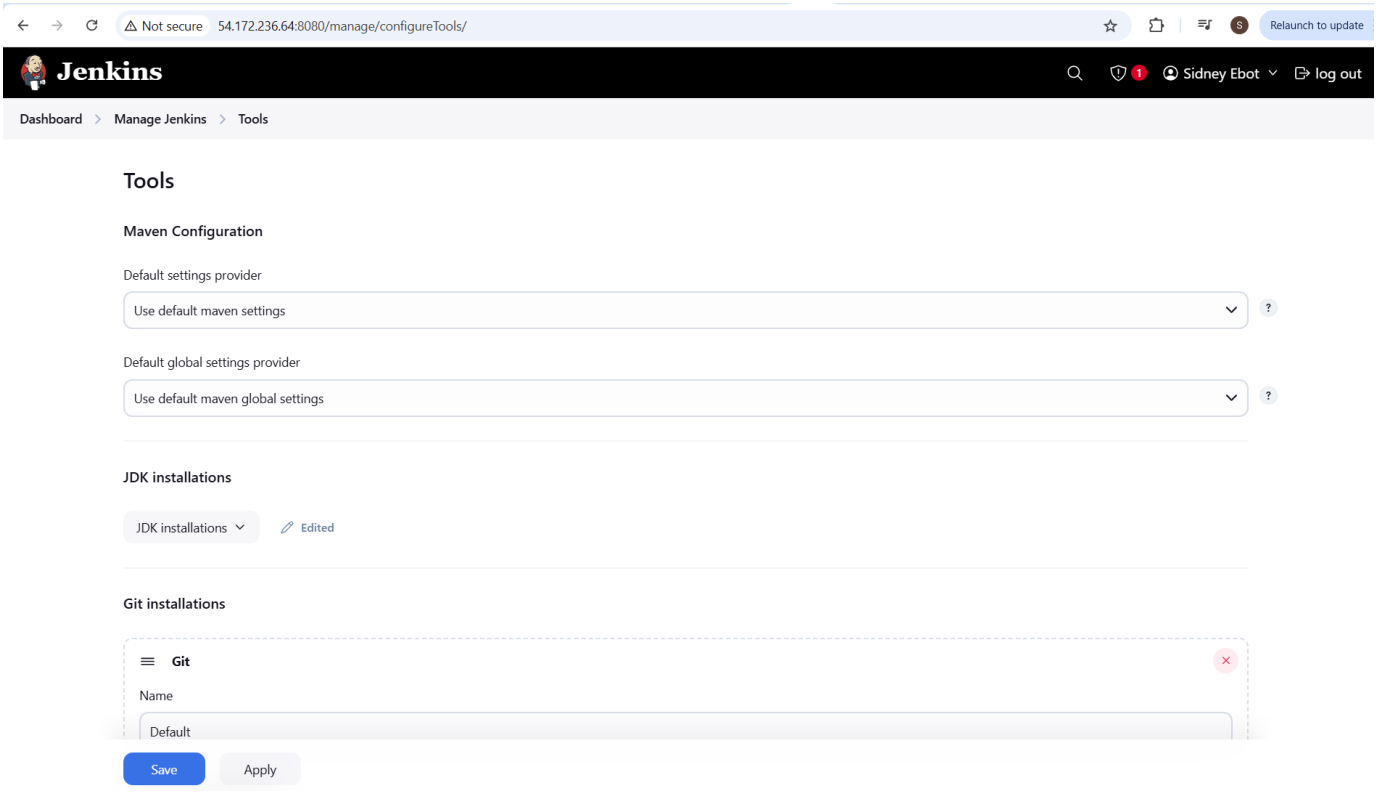


Make the selections as shown above and click on “**Install**”

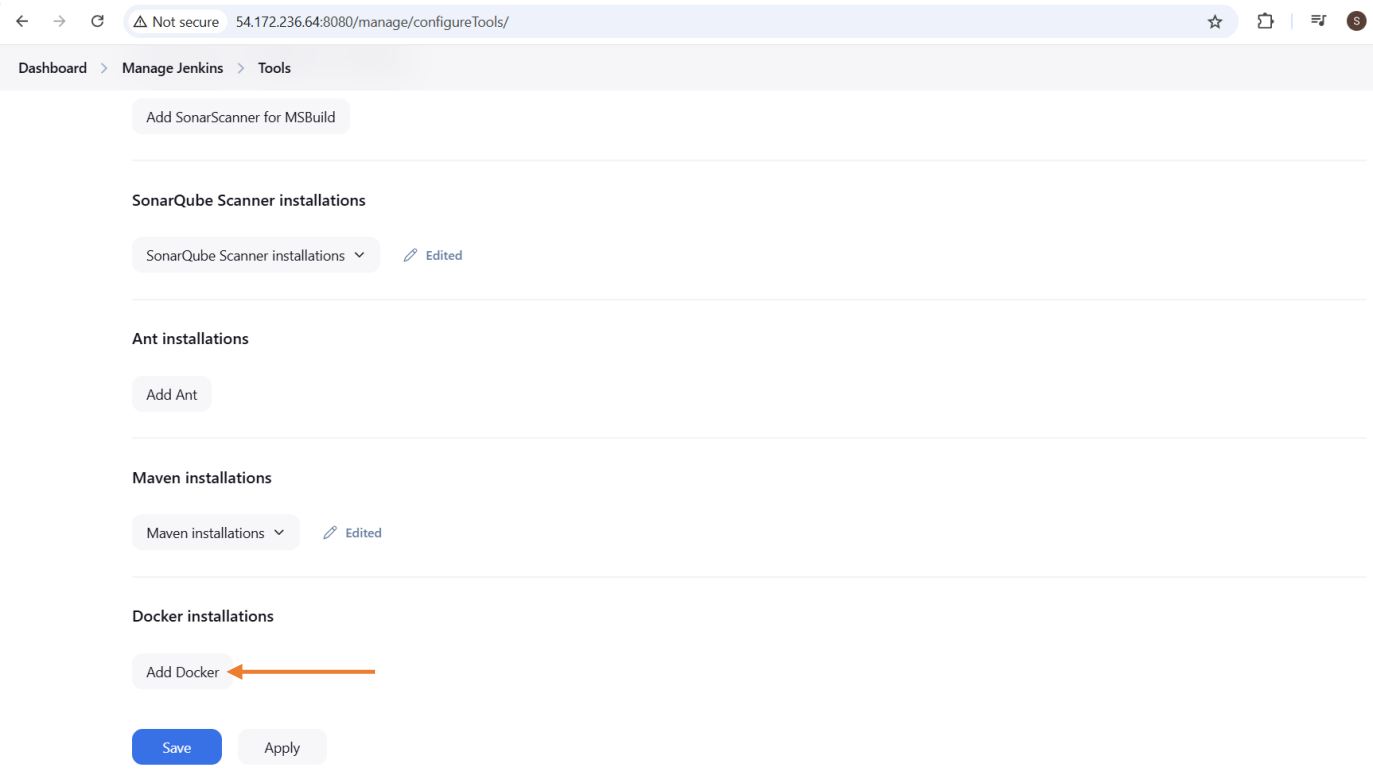


The plugins have been installed.

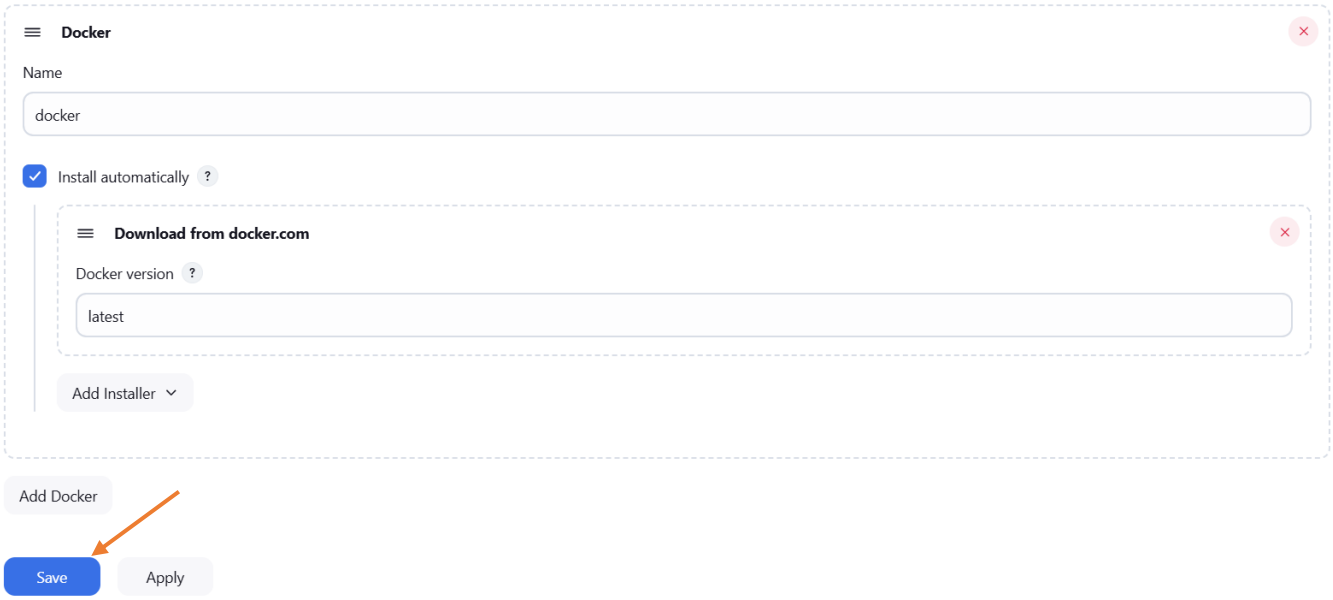
Go back to “**Manage Jenkins**”, then “**Tools**”



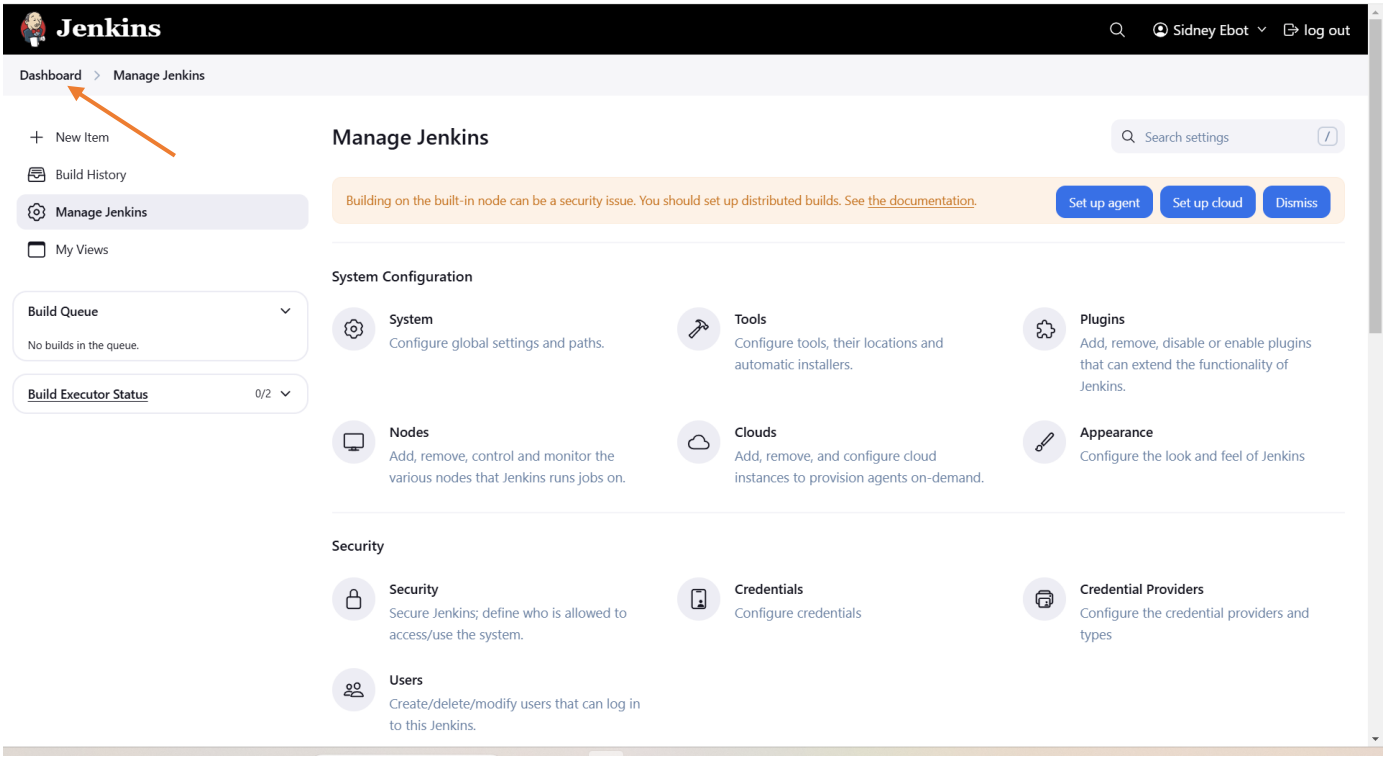
Scroll down to “**Docker Installations**”



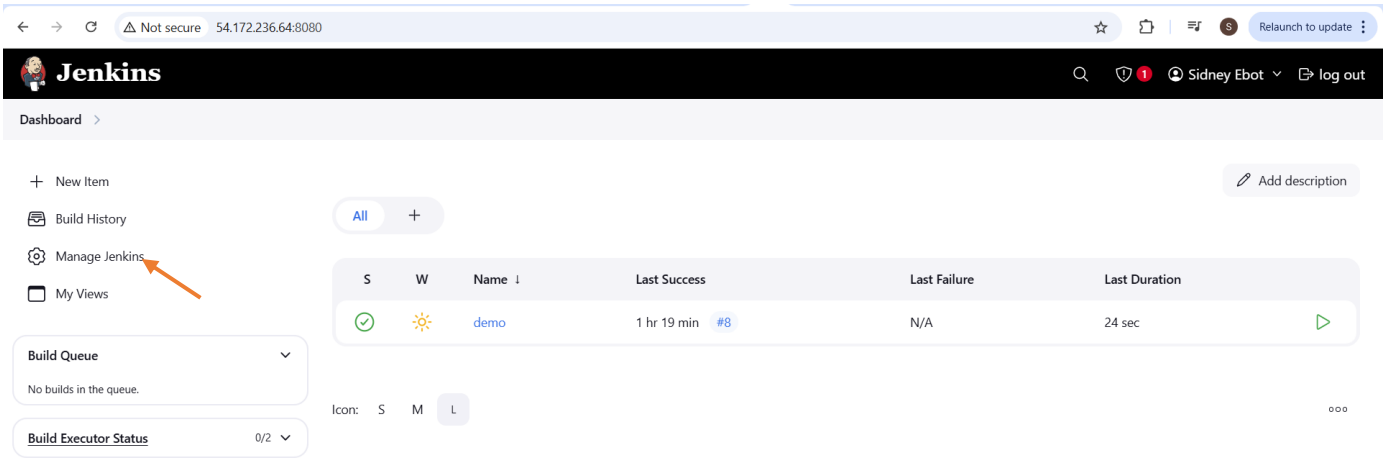
Click on "**Add Docker**". Give it the name "**docker**", select "**Install Automatically**" and click on "**Add Installer**" and choose "**Download from Docker.com**"



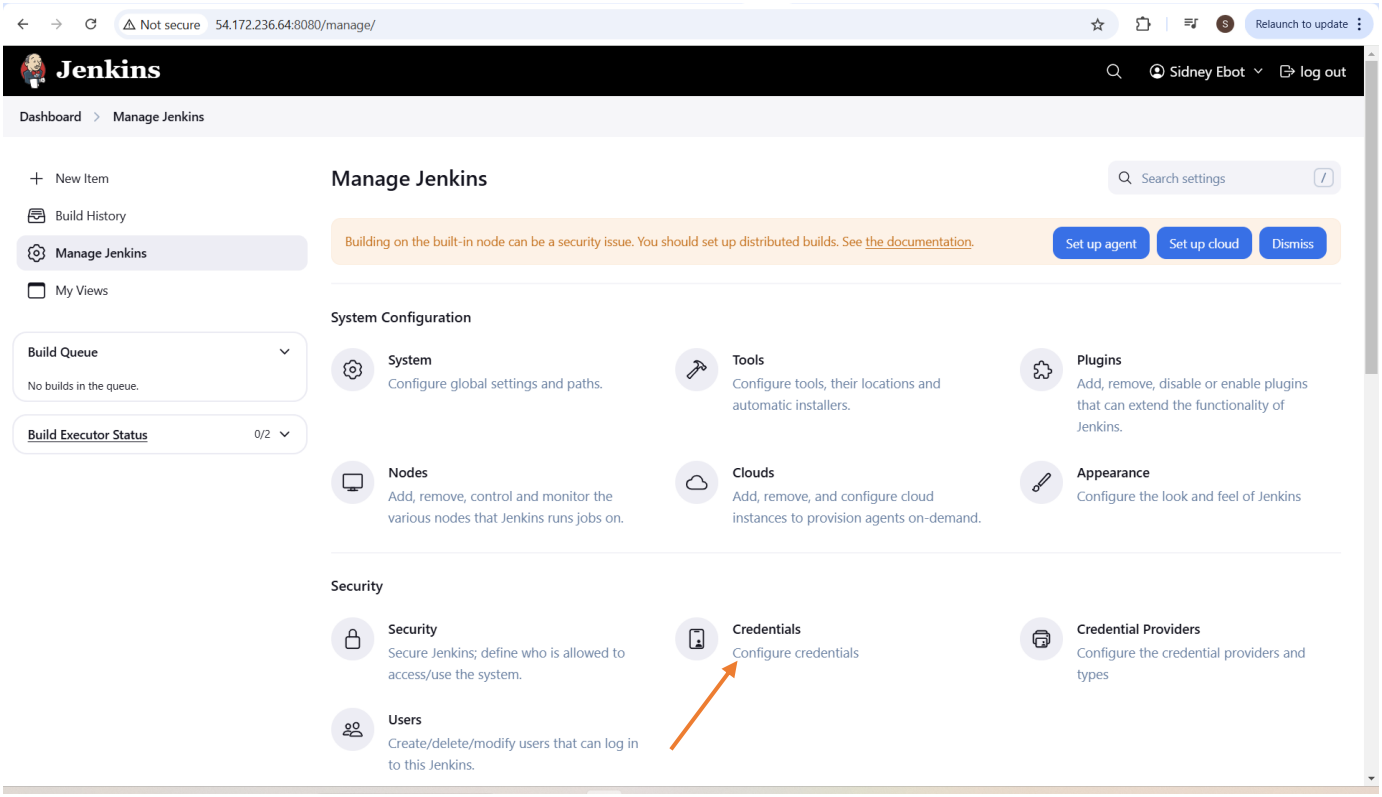
Click on "**Save**"



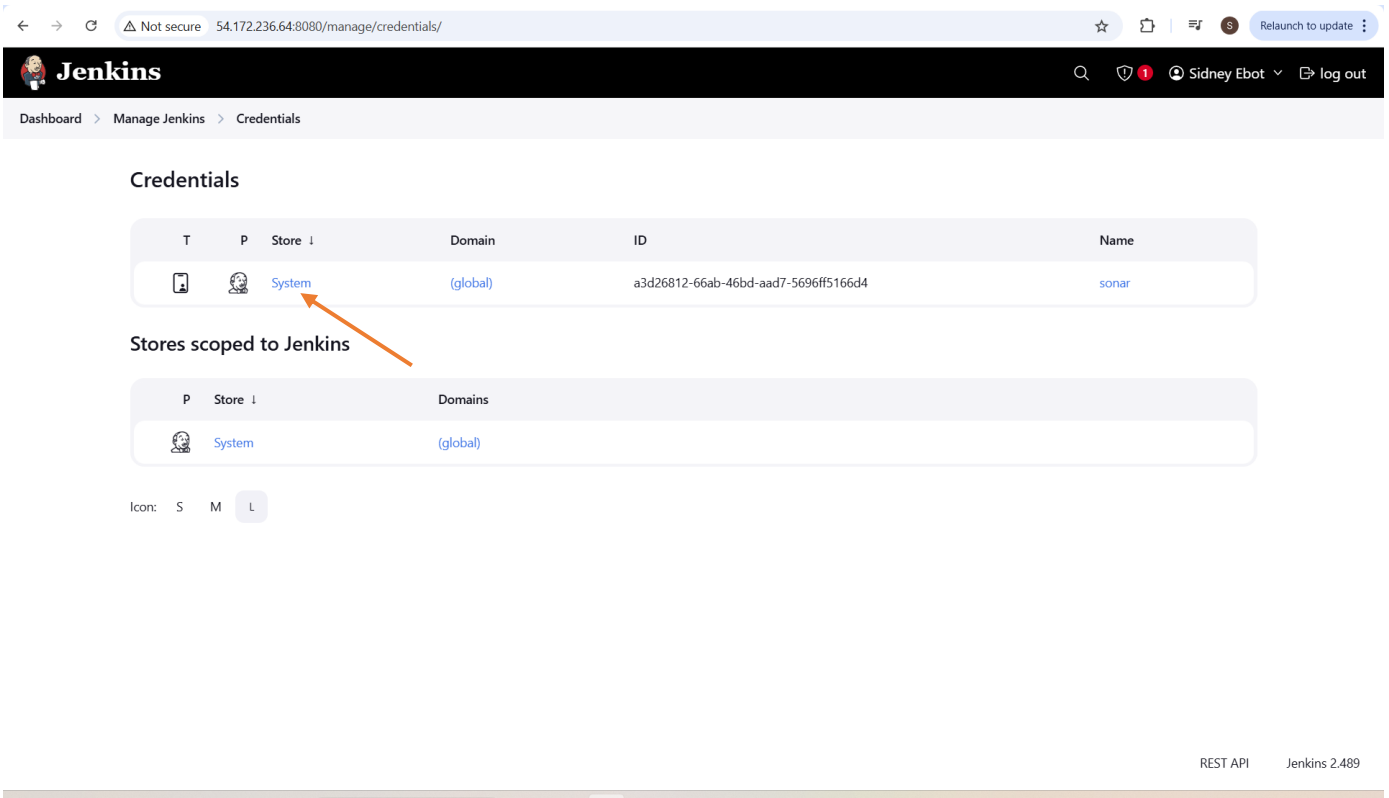
Click on “**Dashboard**”



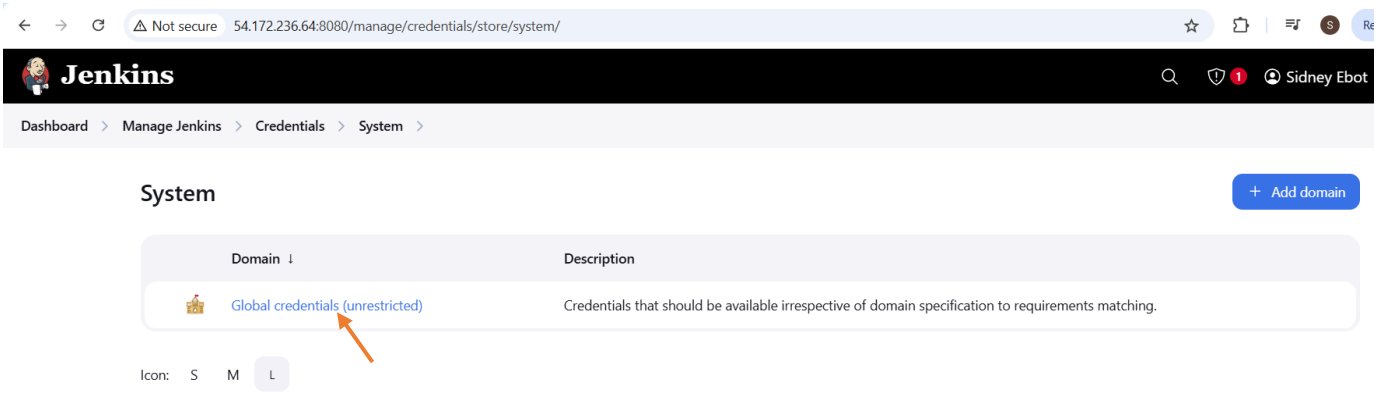
Click on “**Manage Jenkins**”



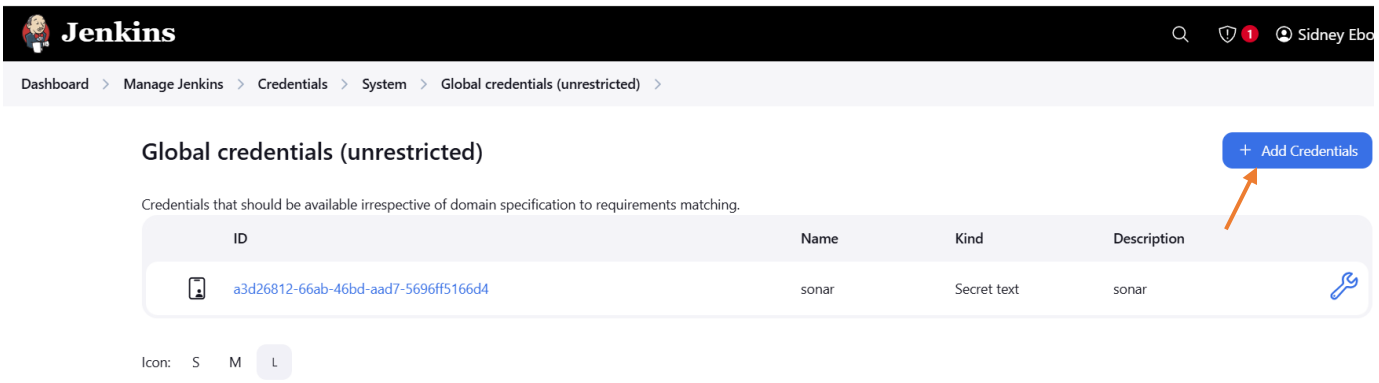
Click on "**Credentials**"



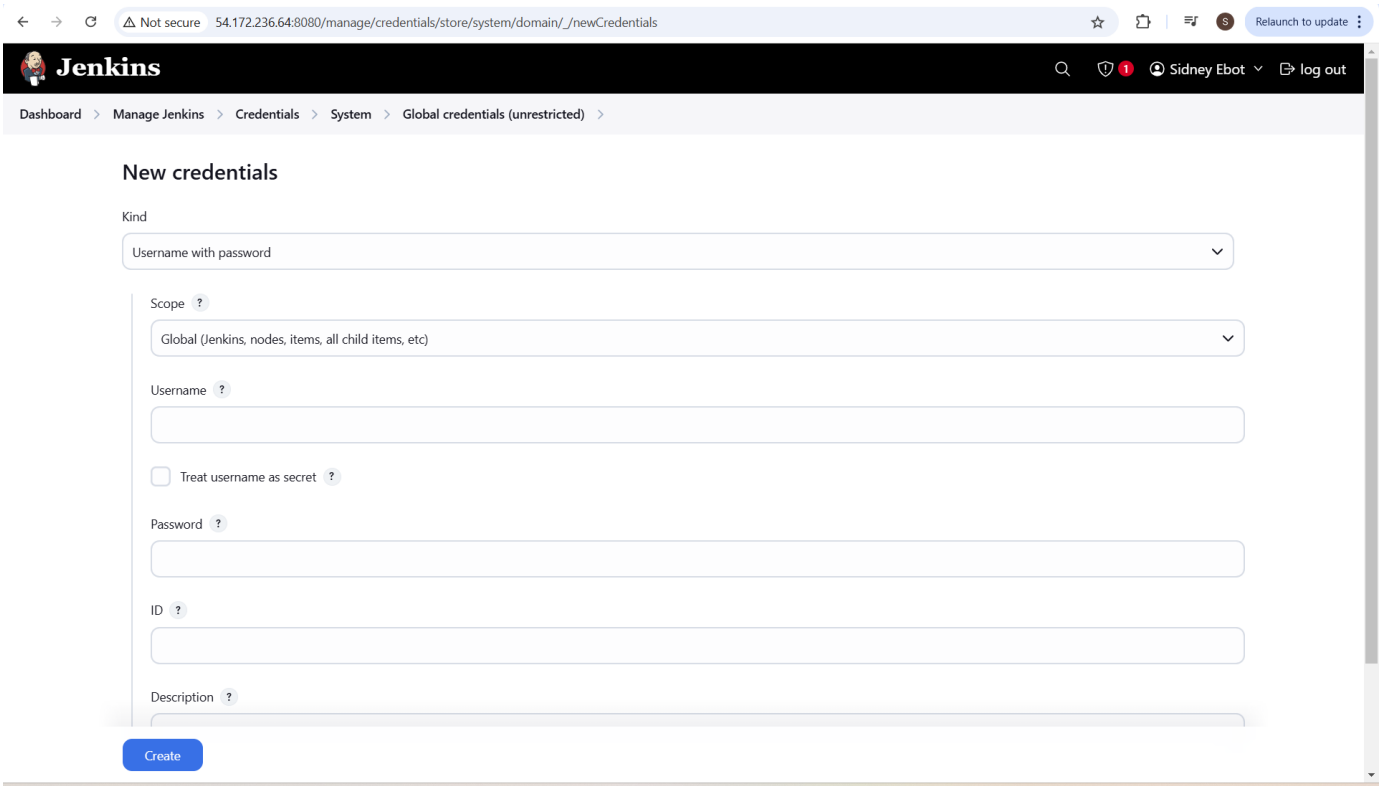
Click on "**System**"



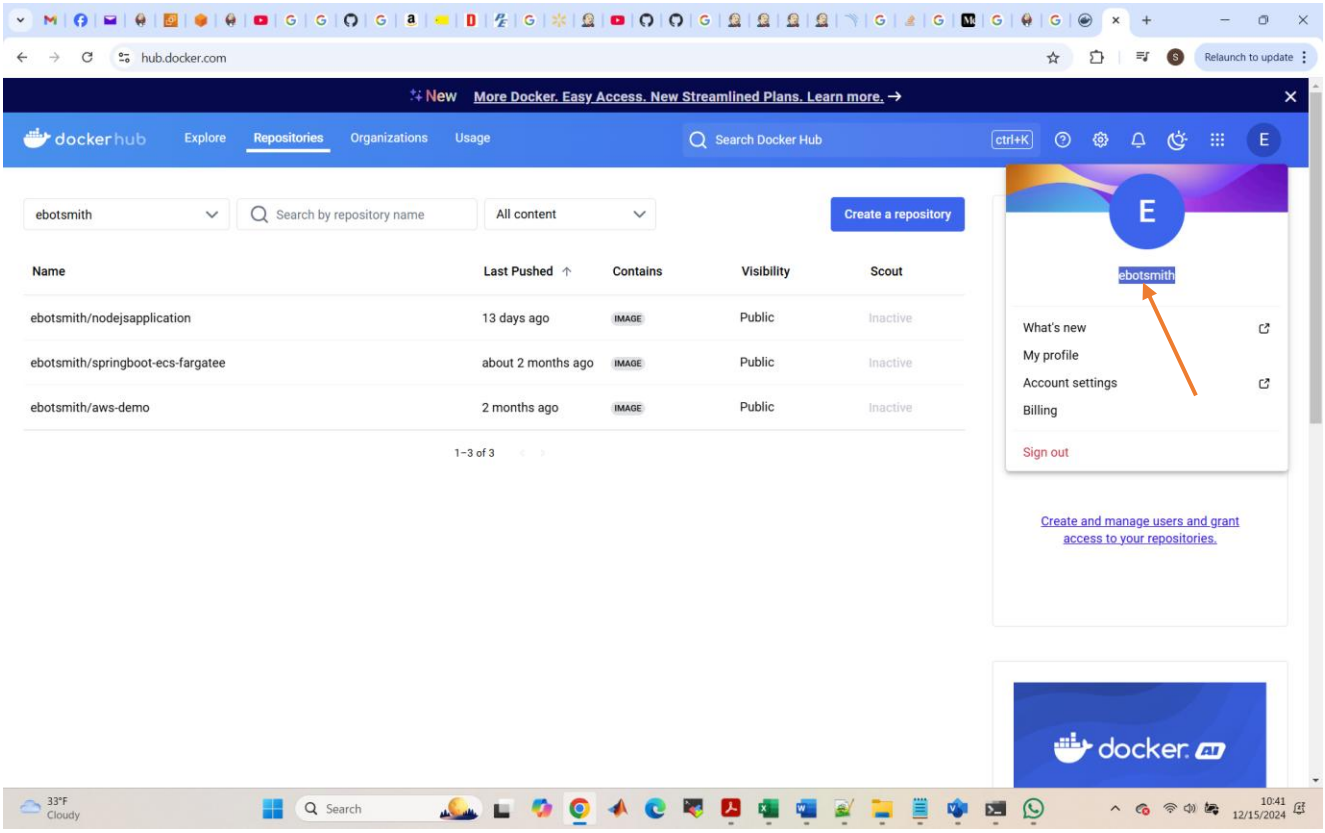
Click on "**Global Credential**"



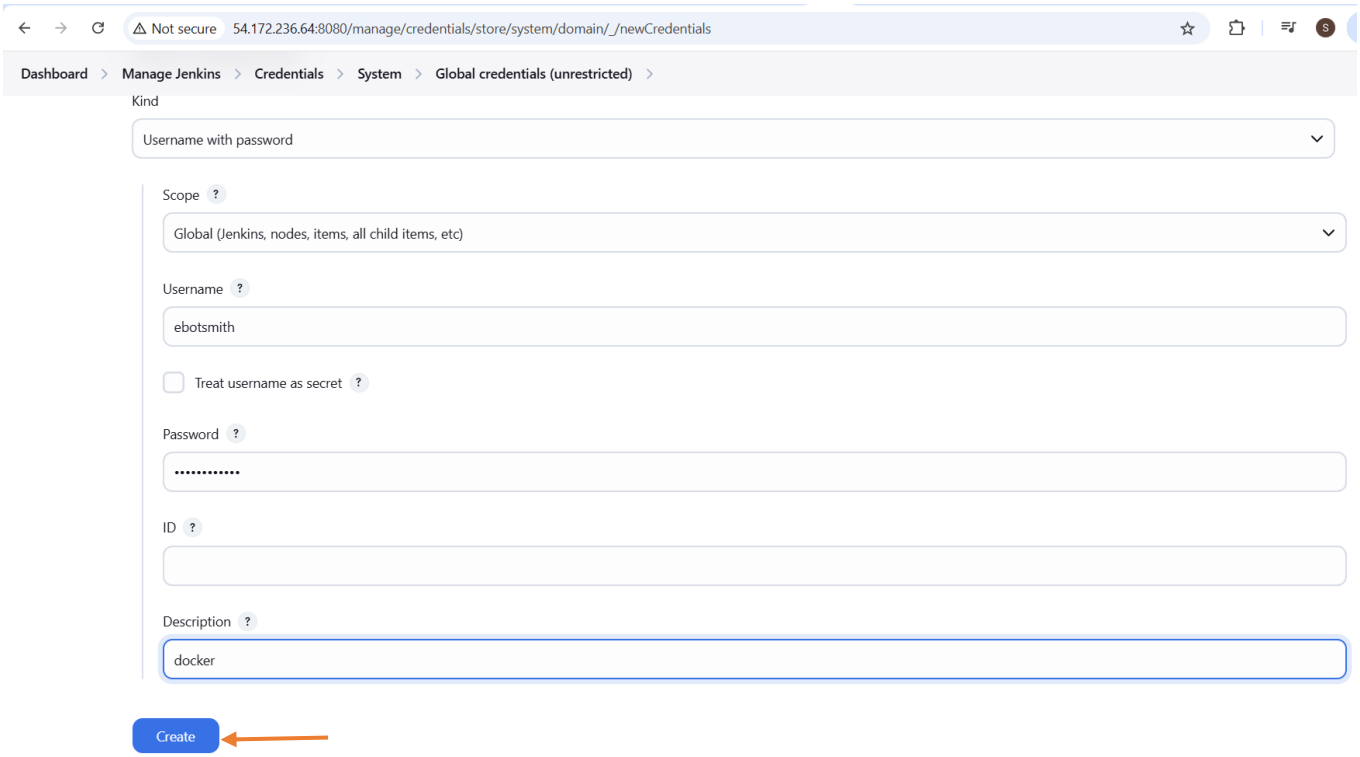
Click on "**Add Credential**"



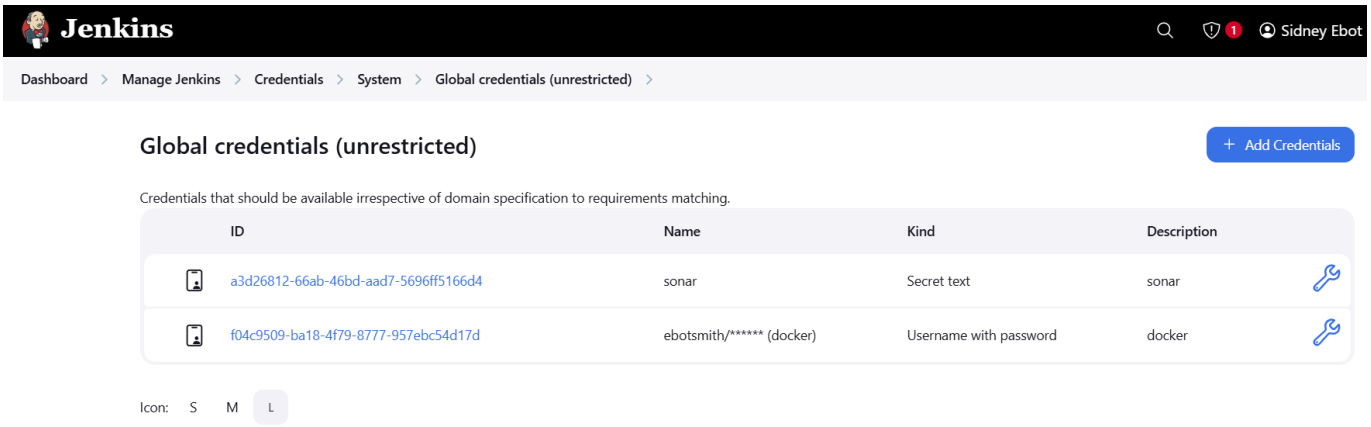
Login to the **docker hub**



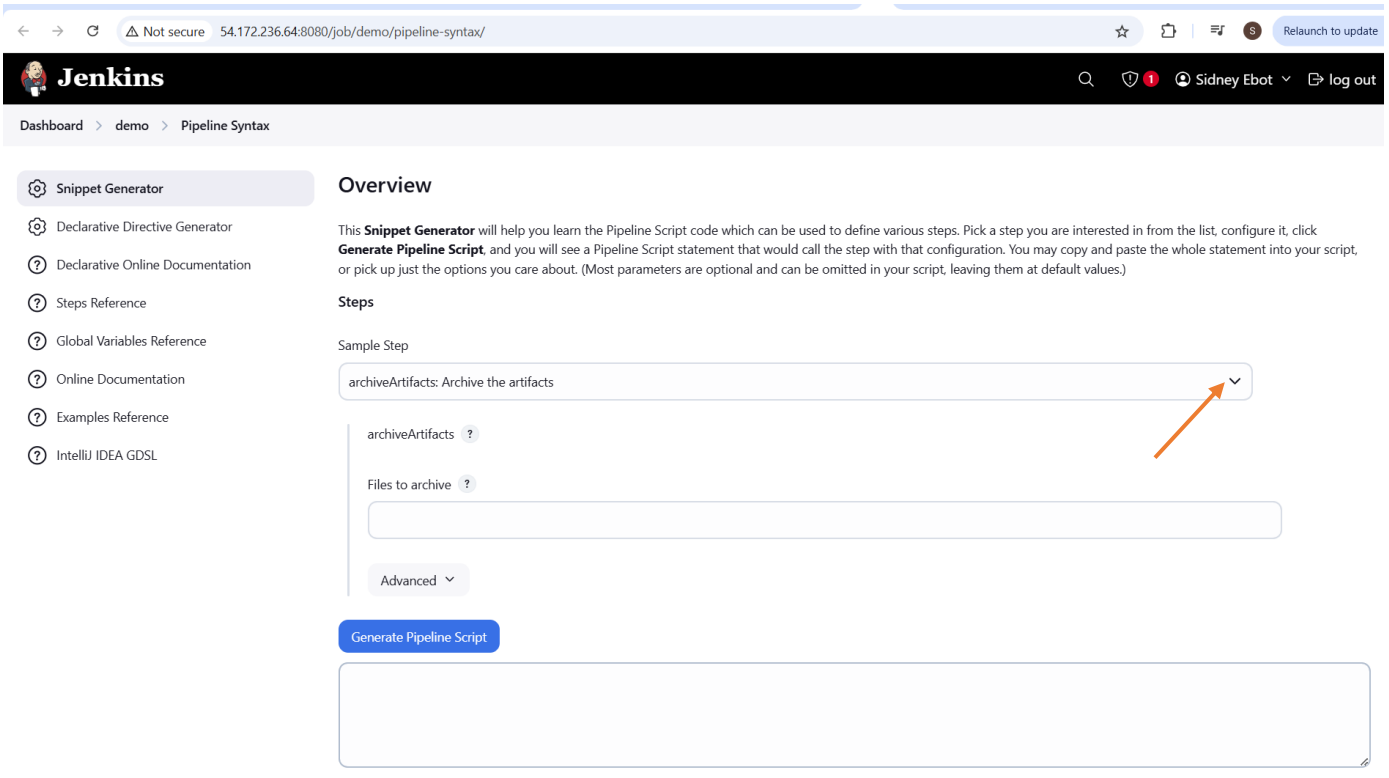
Copy your username: **ebotsmith**. Paste this user name in the Jenkins GUI and add the Docker Hub password.



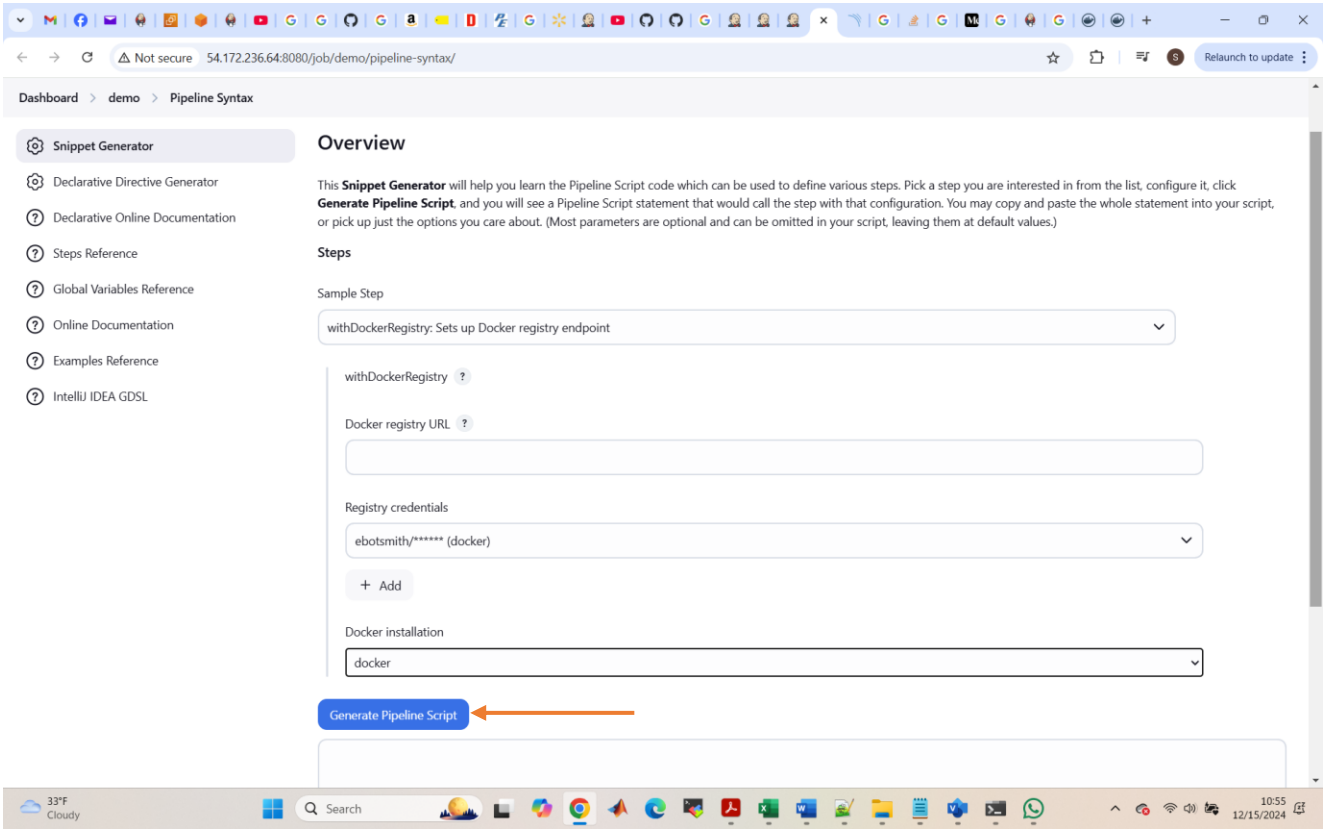
Click on "**Create**"



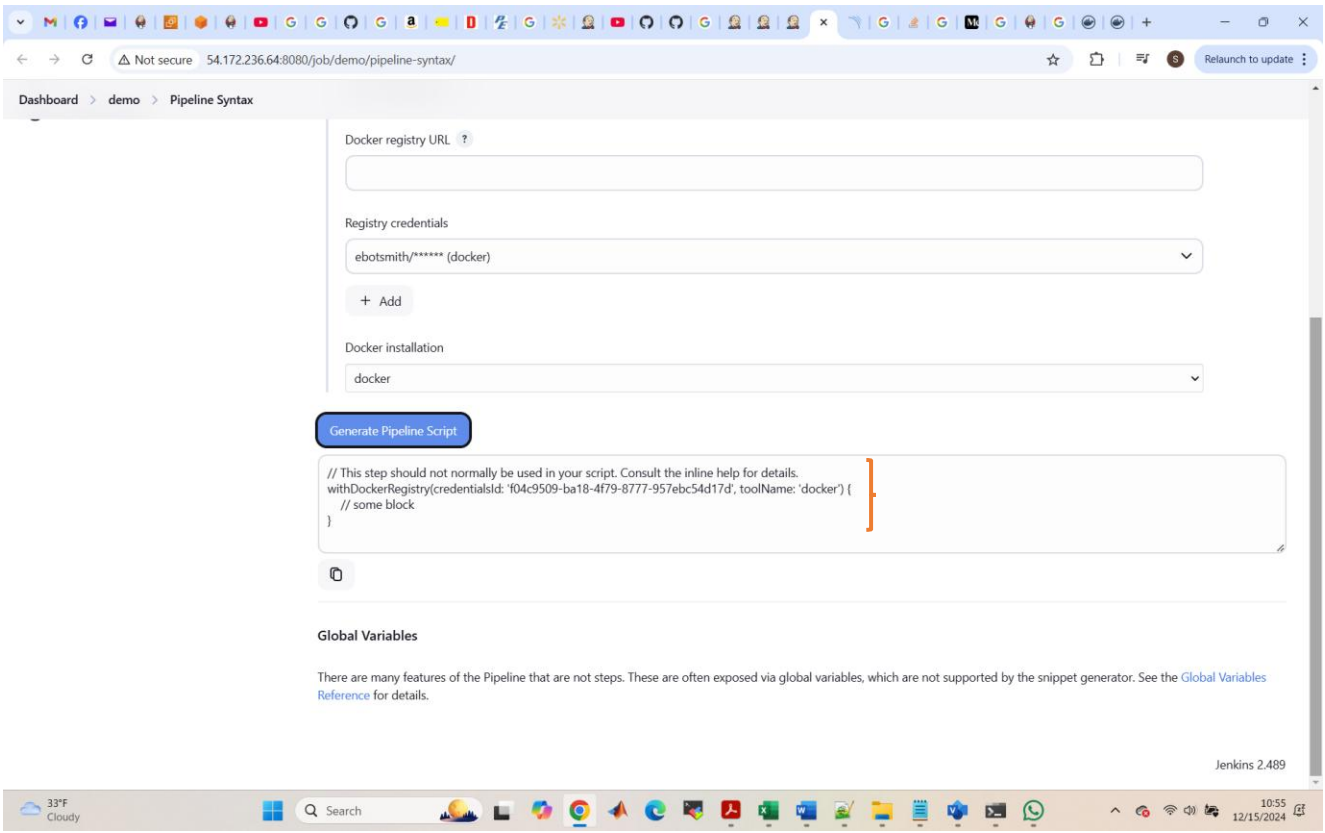
Go to “**Pipeline Syntax**” page and refresh



Click on the drop down on “**Sample Step**” and celect “**withDockerRegistry**” and under Docker installation select “**docker**”

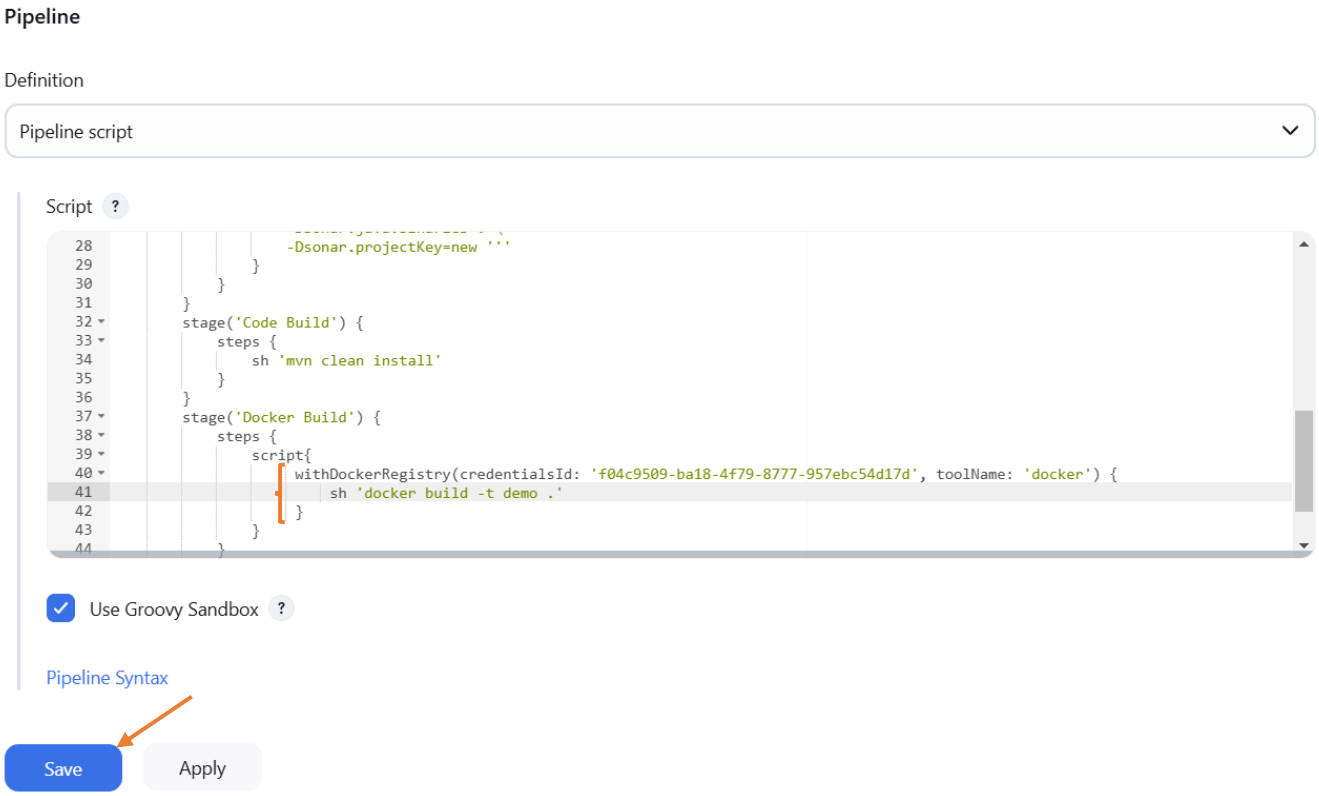


Click on "**Generate Pipeline Script**"



Copy the generated code and add a new stage with it

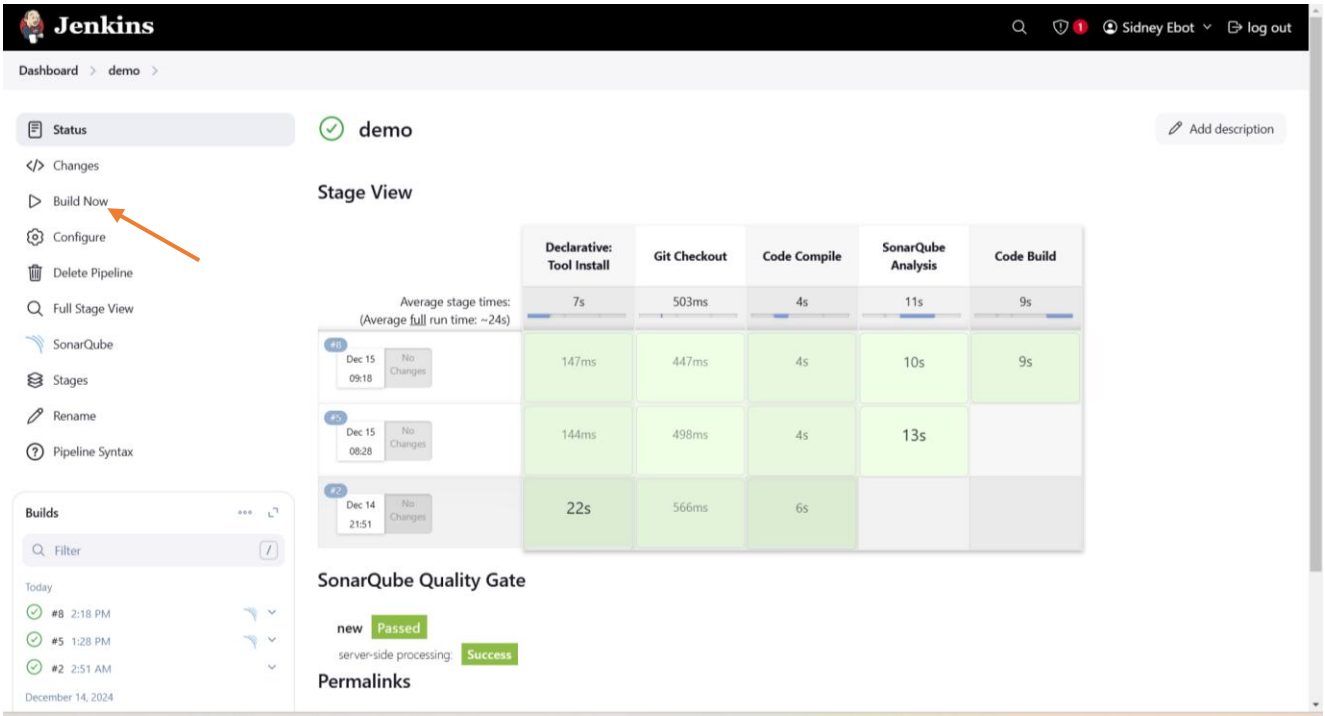
So, we go back to our Jenkins configuration page




```
pipeline {
    agent any

    tools{
        jdk 'jdk'
        maven 'maven3'
    }
    environment{
        SCANNER_HOME= tool 'sonar-scanner'
    }

    stages {
        stage('Git Checkout') {
            steps {
                git 'https://github.com/ebotsmith2000/jenkins-pipeline.git'
            }
        }
        stage('Code Compile') {
            steps {
                sh 'mvn clean compile'
            }
        }
        stage('SonarQube Analysis') {
            steps {
                withSonarQubeEnv('SonarQube'){
                    sh '''$SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=new \
                    -Dsonar.java.binaries=. \
                    -Dsonar.projectKey=new '''
                }
            }
```

```
        }
        stage('Code Build') {
            steps {
                sh 'mvn clean install'
            }
        }
        stage('Docker Build') {
            steps {
               script{
                   withDockerRegistry(credentialsId: '7e596026-5cee-4b1f-9d13-
97efcaf07959', toolName: 'docker') {
                       sh 'docker build -t demo .'
                   }

                }

            }
        }
    }
}
```

Click on "**Save**"



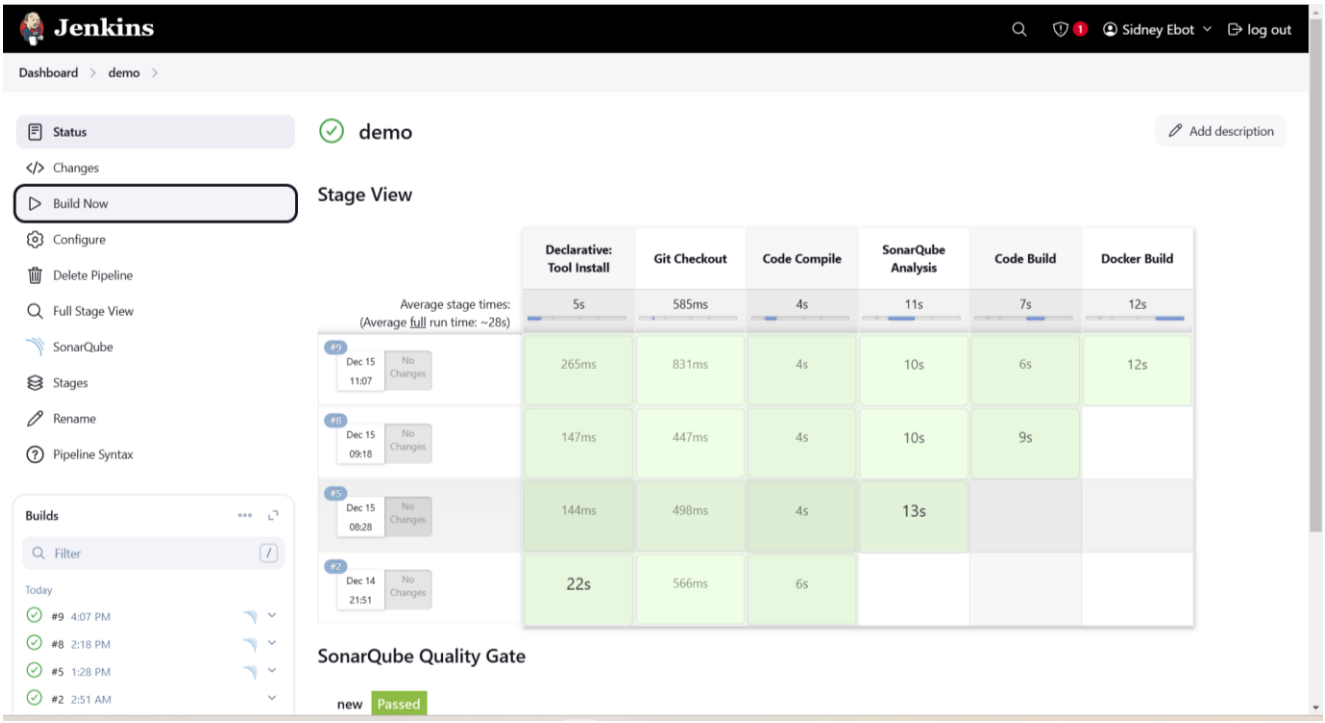
Let us check if there is any docker image by using the command:

sudo docker images

```
ubuntu@ip-172-31-31-55:~$ sudo systemctl restart jenkins
ubuntu@ip-172-31-31-55:~$ sudo systemctl restart docker
ubuntu@ip-172-31-31-55:~$ sudo docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
hello-world   latest    d2c94e258dcb   19 months ago   13.3kB
ubuntu@ip-172-31-31-55:~$
```

You can see that we have just one docker image called “**hello-world**”

Let us click on “**Build Now**” on the Jenkins GUI



Docker build is successful, let us check our docker images again by using the command:

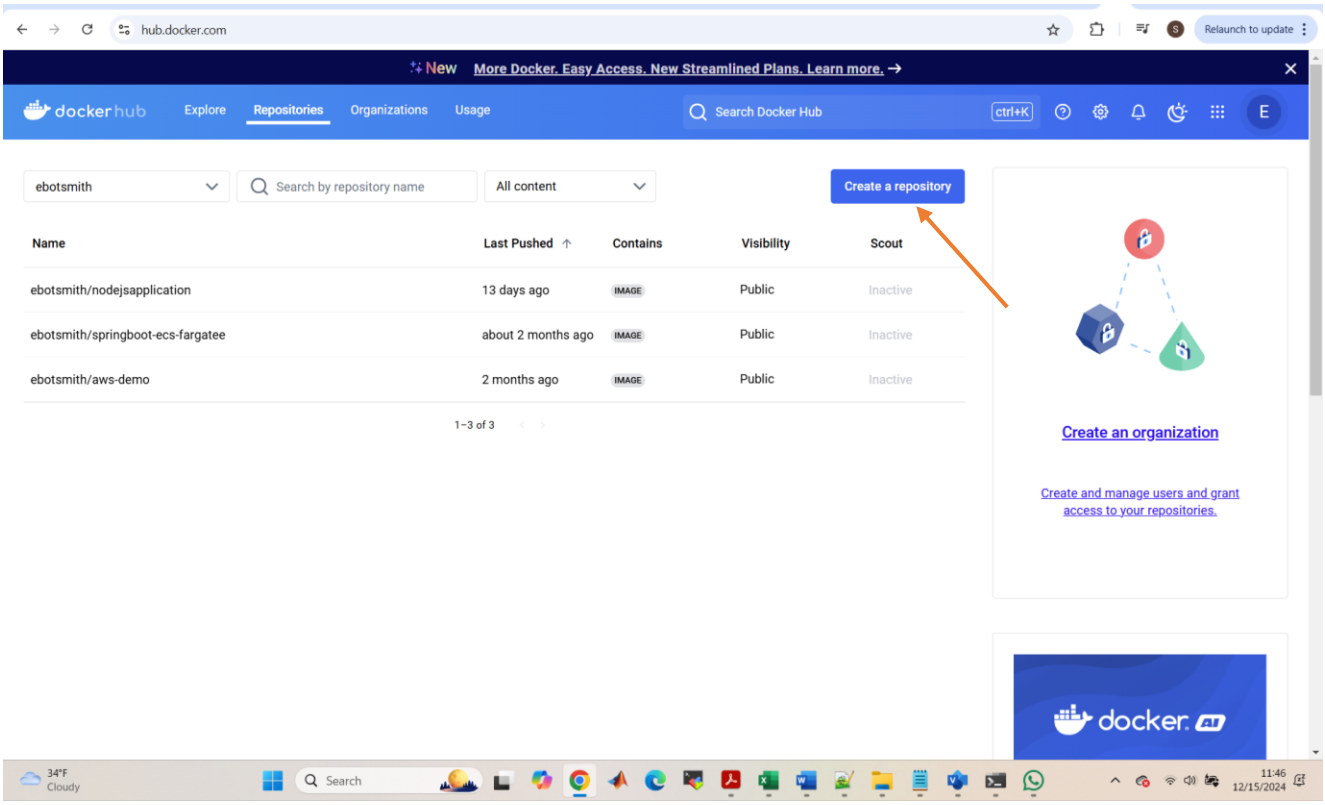
`sudo docker images`

```
ubuntu@ip-172-31-31-55:~$ sudo docker images
REPOSITORY    TAG       IMAGE ID       CREATED              SIZE
demo          latest    bdae5e8b965c   About a minute ago   424MB
hello-world   latest    d2c94e258dcb   19 months ago        13.3kB
ubuntu@ip-172-31-31-55:~$
```

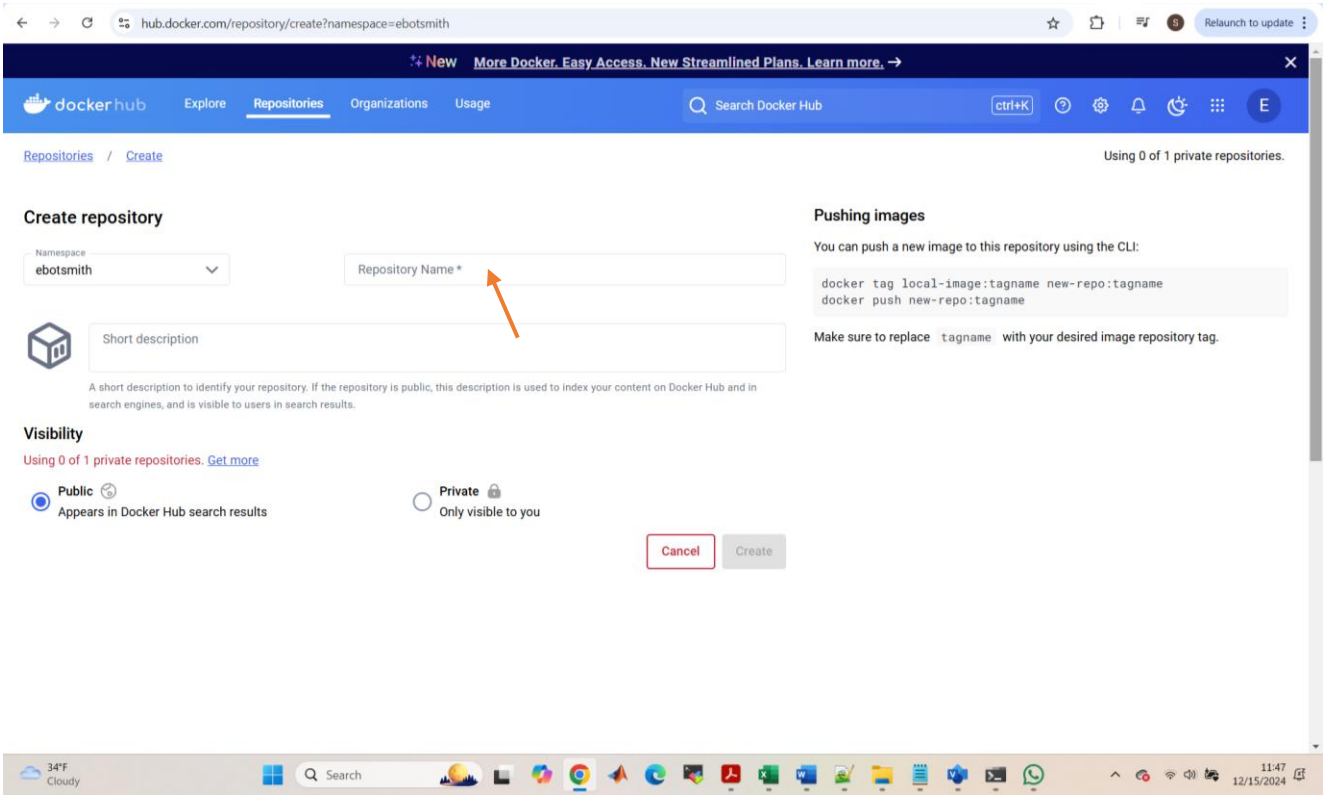
You can see that we now have two docker images “**hello-world**” and “**demo**”. So, we have successfully built the docker image of our application. So, the step “**Docker Build**” is done, we have to move to the next step that is “**Push the image to Docker Hub**”

STEP 14: Push the image to Docker Hub

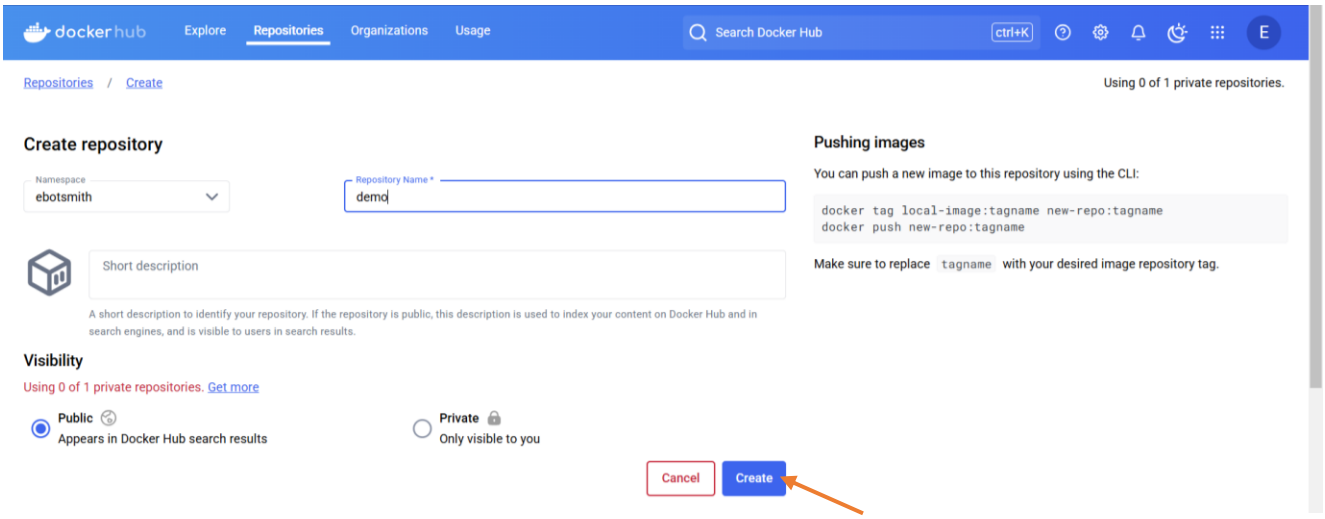
Let us go to the Docker hub and **create a repository**.



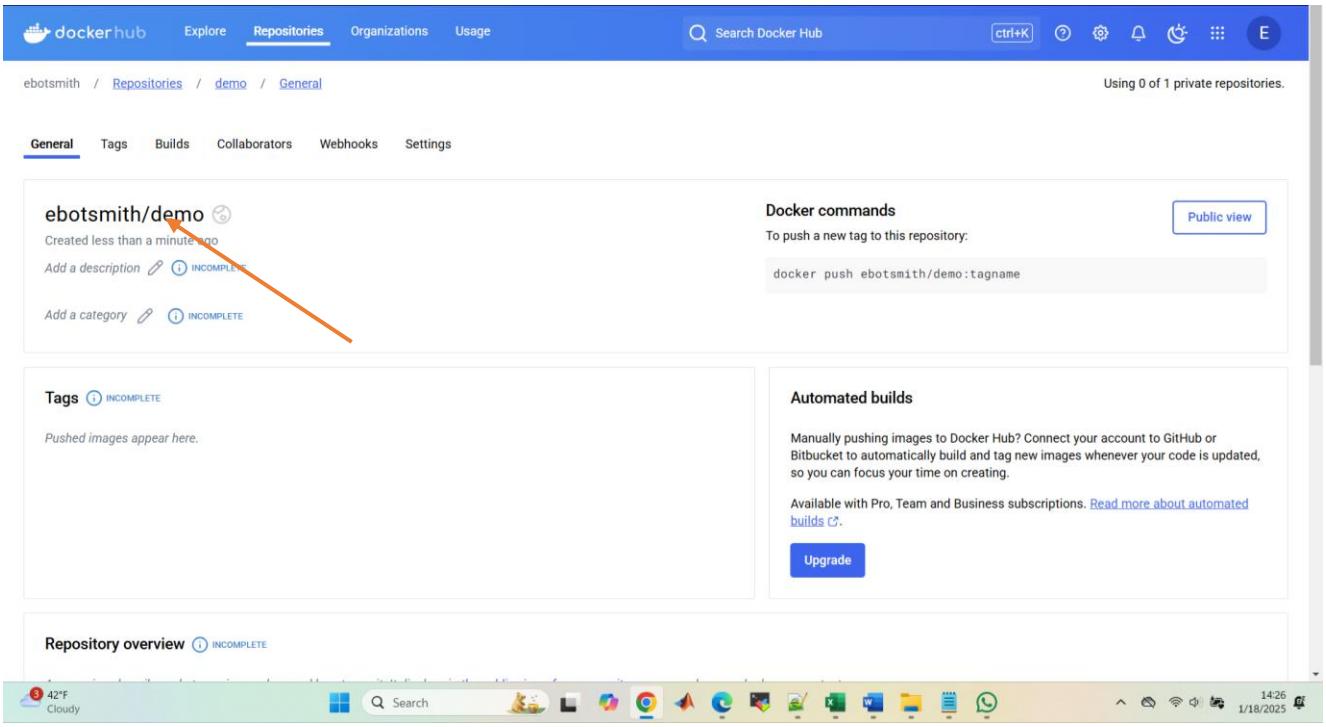
Click on "**Create a Repository**"



Let us give it the name “**demo**”

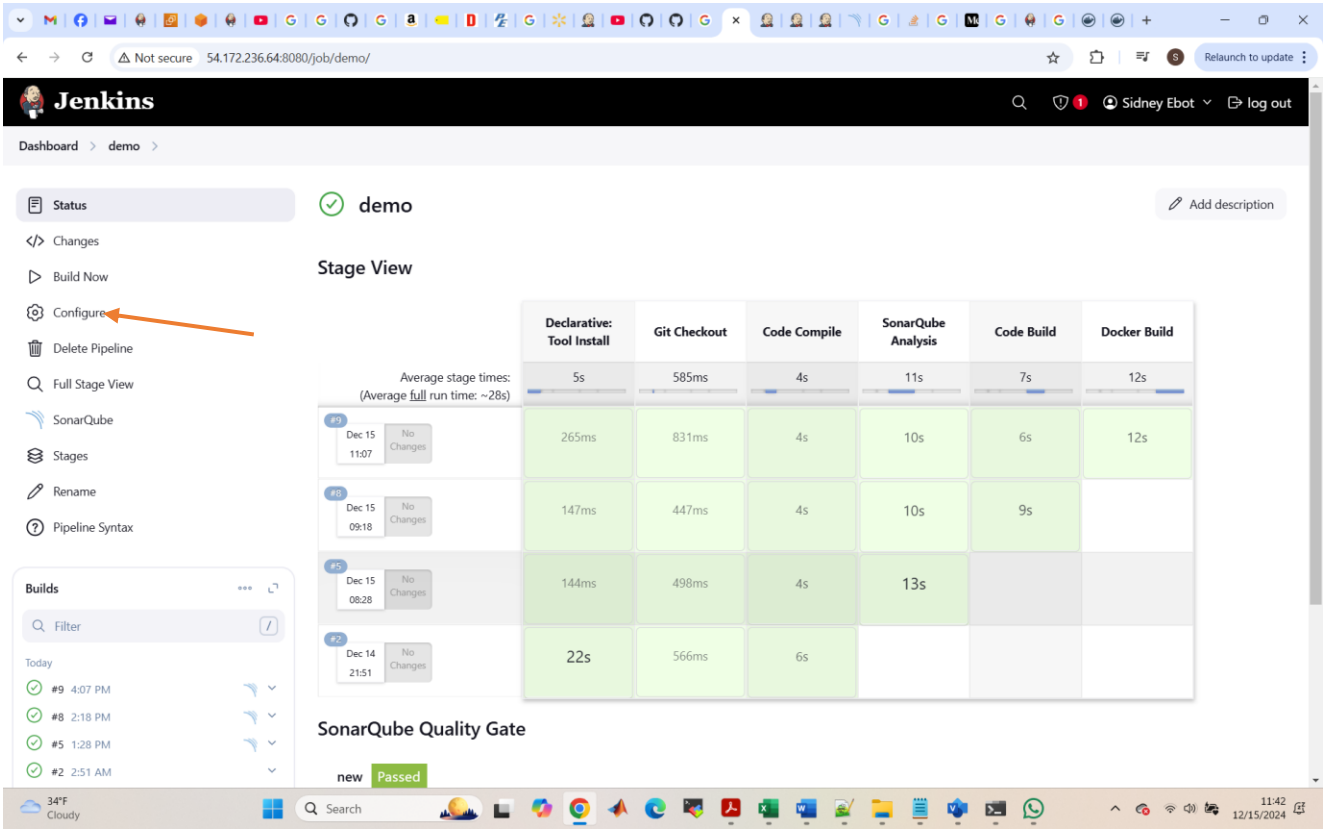


Click on “**Create**”

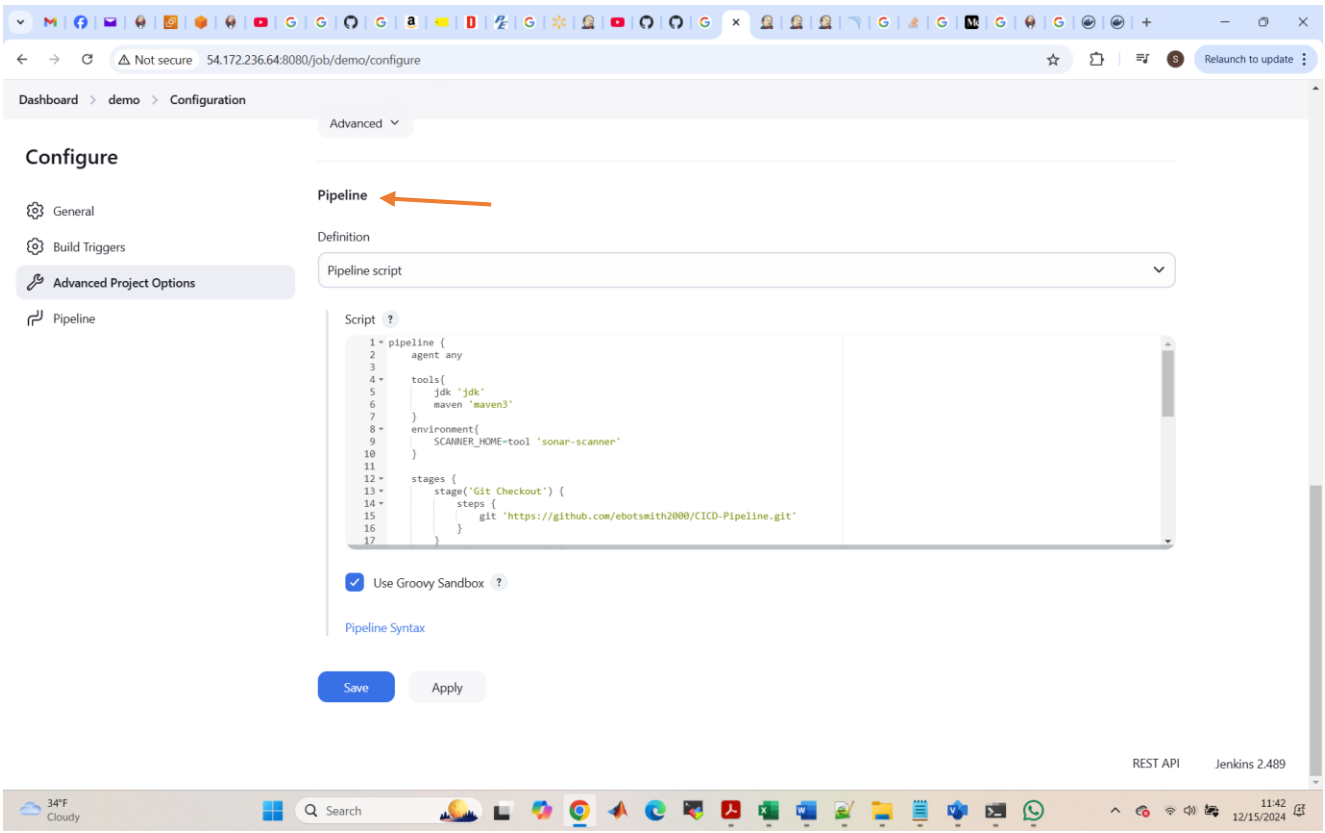


Copy this: **ebotsmith/demo**

We have to create a new stage to push the image to Docker hub. So go back to Jenkins GUI



Click on "**Configure**" and scroll down to "**Pipeline**"



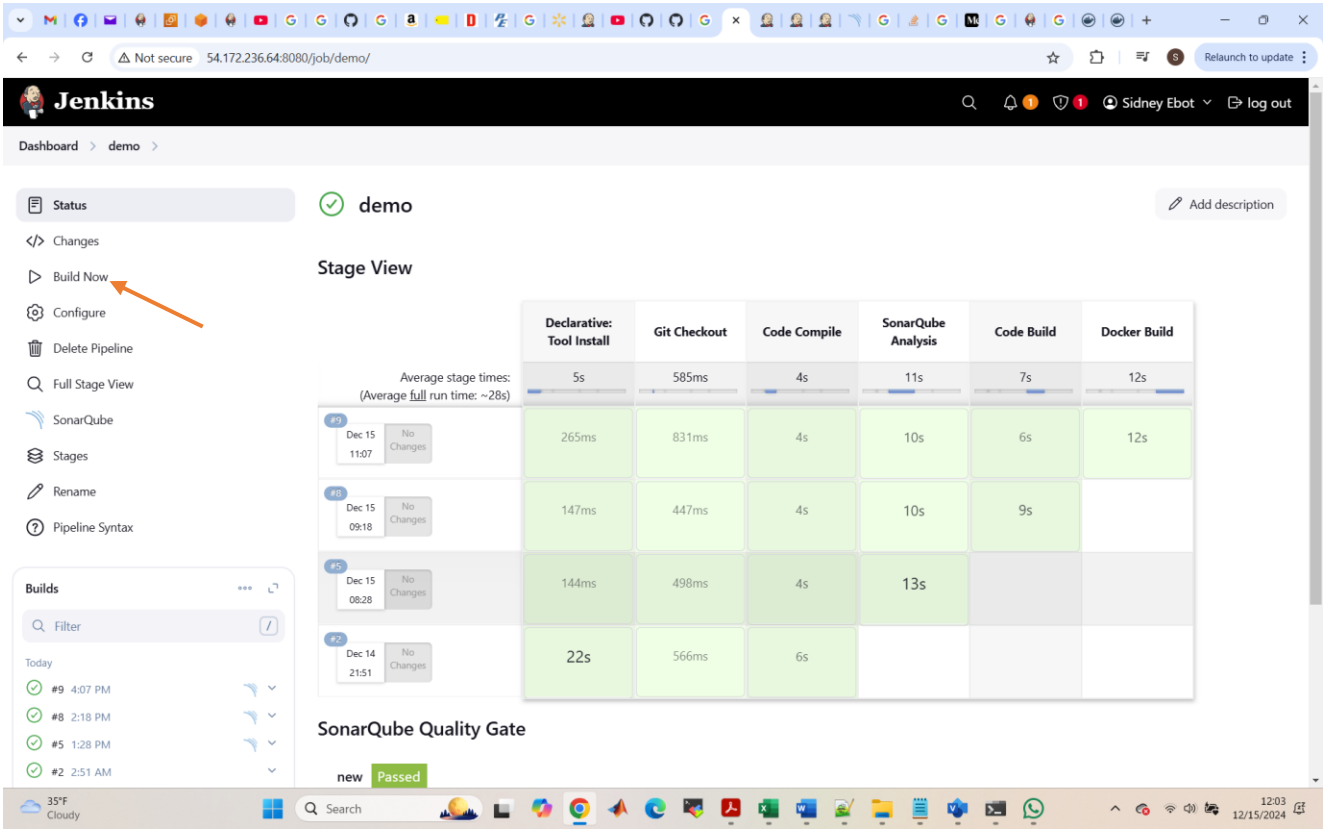
Add another stage

This '**docker tag demo ebotsmith/demo**'. Means we are tagging the image "**demo**" into our Docker hub repository.

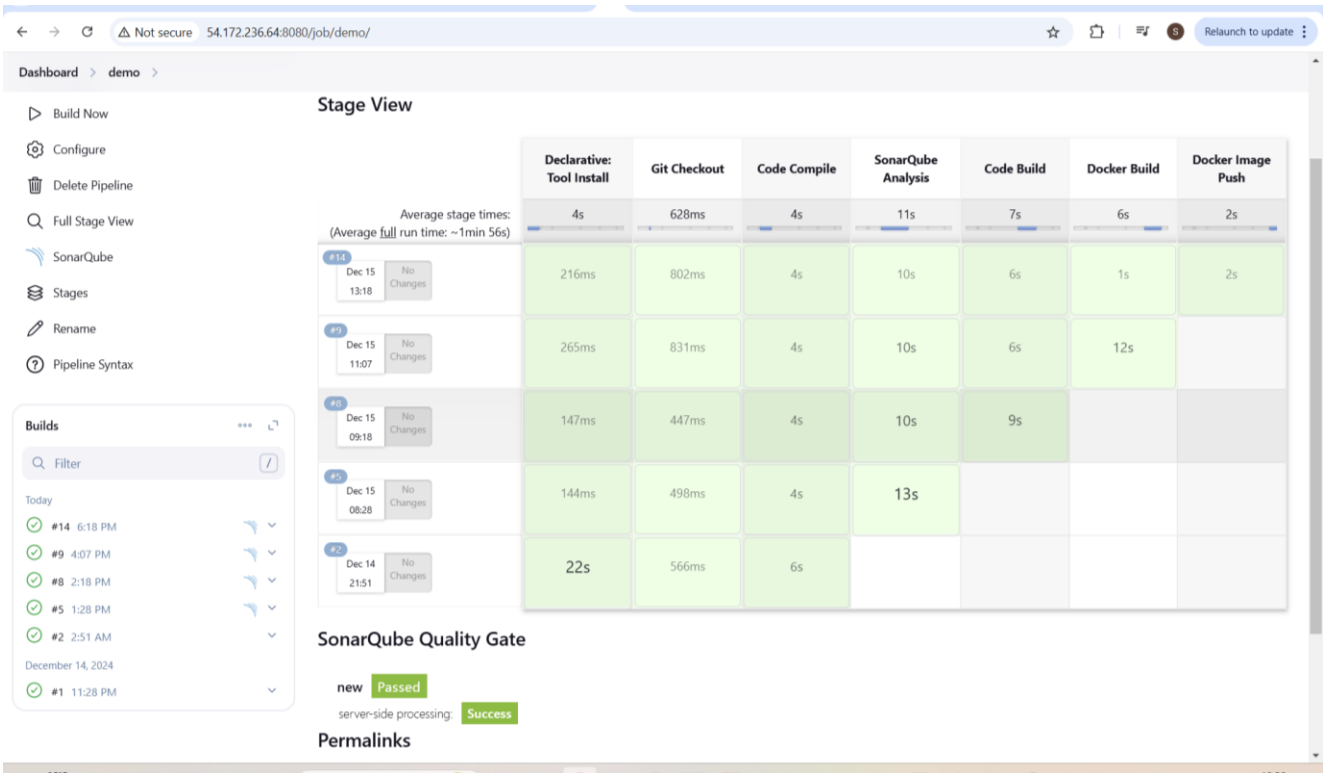

```
pipeline {
    agent any

    tools{
        jdk 'jdk'
        maven 'maven3'
    }
    environment{
        SCANNER_HOME= tool 'sonar-scanner'
    }

    stages {
        stage('Git Checkout') {
            steps {
                git 'https://github.com/ebotsmith2000/jenkins-pipeline.git'
            }
        }
        stage('Code Compile') {
            steps {
                sh 'mvn clean compile'
            }
        }
        stage('SonarQube Analysis') {
            steps {
                withSonarQubeEnv('SonarQube'){
                    sh '''$SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=new \
                    -Dsonar.java.binaries=. \
```

```
                    -Dsonar.projectKey=new '''
                }
            }
        }
        stage('Code Build') {
            steps {
                sh 'mvn clean install'
            }
        }
        stage('Docker Build') {
            steps {
               script{
                   withDockerRegistry(credentialsId: '7e596026-5cee-4b1f-9d13-
97efcaf07959', toolName: 'docker') {
                        sh 'docker build -t demo .'
                    }

                }

            }
        }
        stage('Docker Image Push') {
            steps {
               script{
                   withDockerRegistry(credentialsId: '7e596026-5cee-4b1f-9d13-
97efcaf07959', toolName: 'docker') {
                        sh 'docker tag demo ebotsmith/demo:$BUILD_ID'
                        sh 'docker push ebotsmith/demo:$BUILD_ID'
                    }

                }

            }
        }
    }
}
```

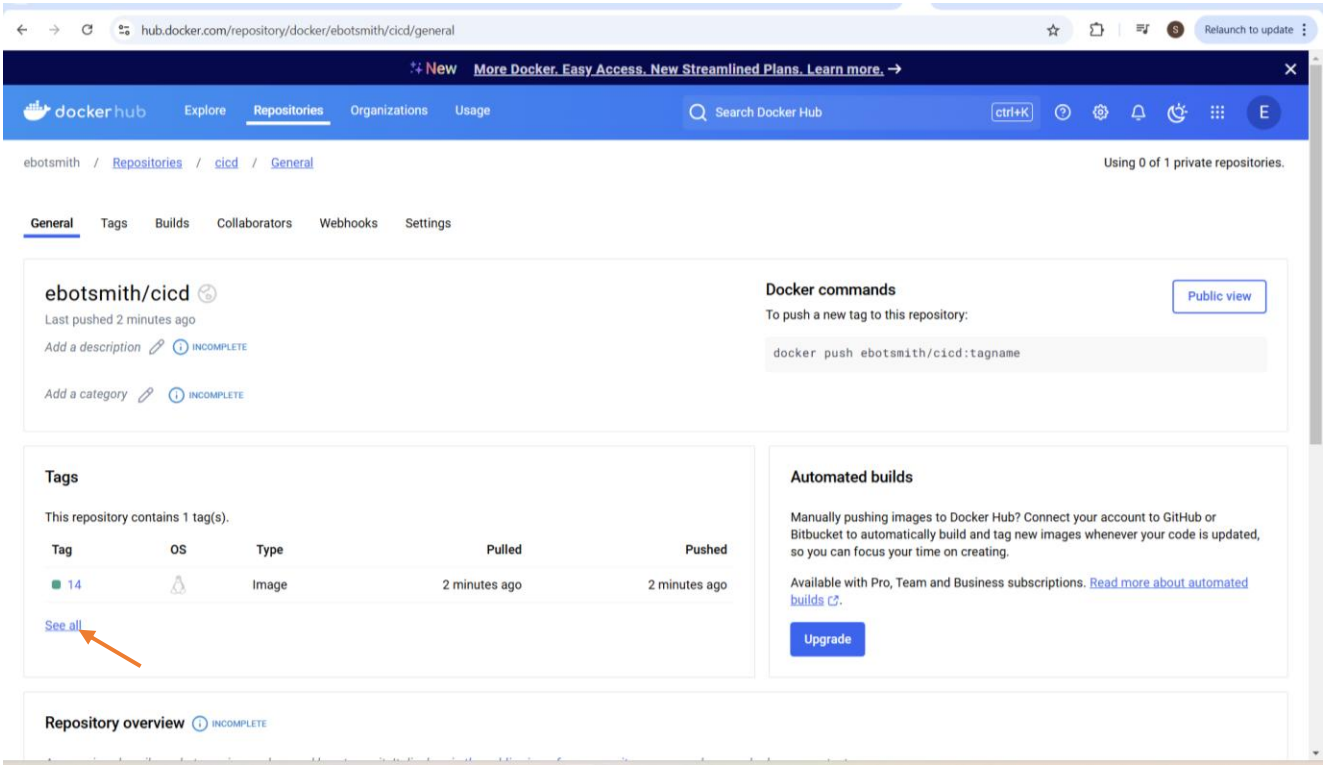
Click on “**Save**”



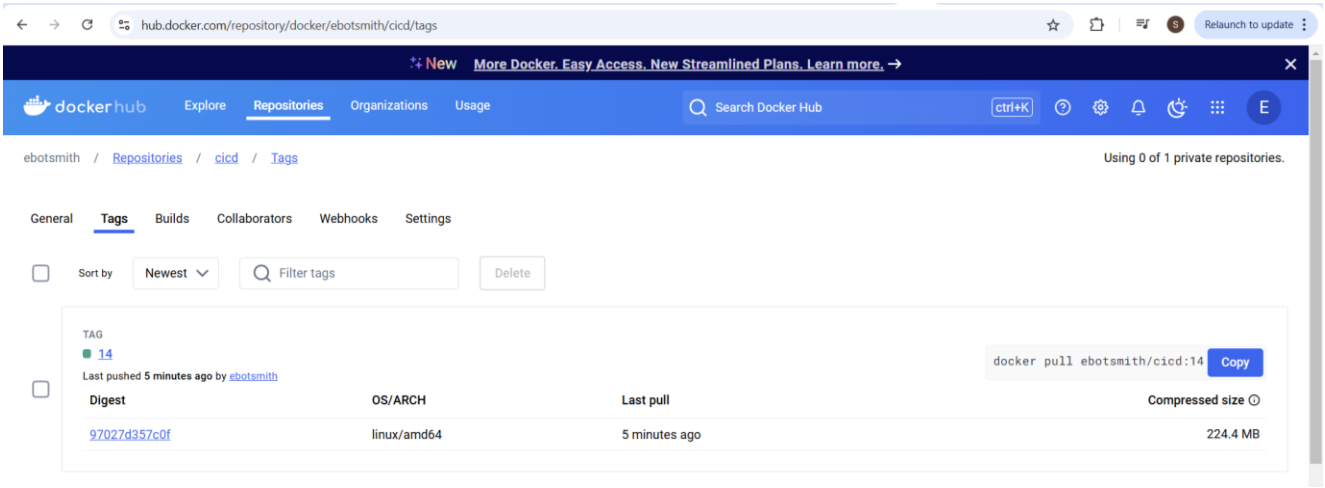
Then, click on "**Build Now**" to build



The image has been pushed. Go now to the docker hub and check the repository if the image is there.



You can now see that the image has been tagged. Click on "**See All**" to check if the image is ready



The image is ready. So, we have pushed the docker image to Docker hub. Now we have to deploy the image to Kubernetes.

STEP 15: Deploy the Image to Kubernetes.

We have to deploy the image from Docker hub to Kubernetes.

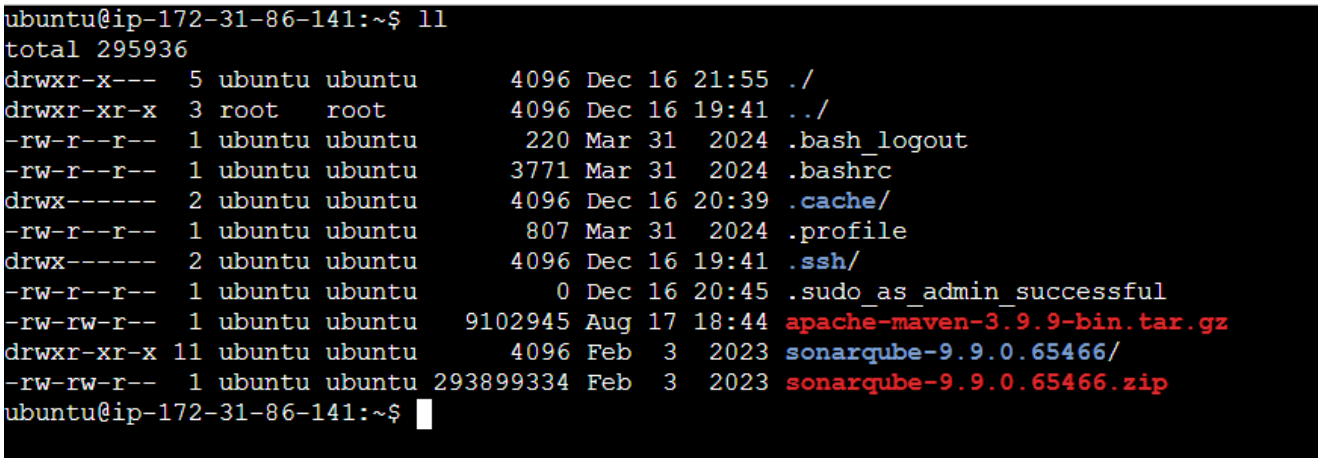
PART 1: Install Kubernetes on Server

To deploy to Kubernetes, we just need to install Kubernetes and its plugins.

First run the command: **`cd /home/ubuntu/`**

Then run the command:

`ll`



To install Kubernetes, open a new vi name Kubeadm.sh using the command:

`sudo vi kubeadm.sh`

Paste the below code on the open vi

```
#!/bin/bash

echo ".........----------------#################._.-.-INSTALLING KUBERNETES-.-
._.#################----------------........."

# Update and install dependencies
apt-get update
apt-get install -y curl apt-transport-https ca-certificates gnupg lsb-release

# Configure Kubernetes package repository
KUBE_LATEST=$(curl -L -s https://dl.k8s.io/release/stable.txt | awk 'BEGIN { FS="." } {
printf "%s.%s", $1, $2 }')
mkdir -p /etc/apt/keyrings
curl -fsSL https://pkgs.k8s.io/core:/stable:/${KUBE_LATEST}/deb/Release.key | gpg --dearmor
-o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/${KUBE_LATEST}/deb/ /" >
/etc/apt/sources.list.d/kubernetes.list

# Update package list and install Kubernetes components
apt-get update
apt-get install -y kubelet kubectl kubeadm kubernetes-cni containerd

# Configure containerd
mkdir -p /etc/containerd
containerd config default | sed 's/SystemdCgroup = false/SystemdCgroup = true/' >
/etc/containerd/config.toml
systemctl restart containerd
systemctl enable kubelet

# Initialize Kubernetes cluster
kubeadm reset -f
kubeadm init --pod-network-cidr='10.244.0.0/16' --service-cidr='10.96.0.0/16' --skip-token-
print

# Configure kubectl for the root user
mkdir -p ~/.kube
cp -i /etc/kubernetes/admin.conf ~/.kube/config

# Install a pod network (WeaveNet)
kubectl apply -f "https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-
daemonset-k8s-1.11.yaml"
kubectl rollout status daemonset weave-net -n kube-system --timeout=90s

# Remove taints from the control-plane node to schedule workloads
node=$(kubectl get nodes -o=jsonpath='{.items[0].metadata.name}')
for taint in $(kubectl get node $node -o=jsonpath='{range
.spec.taints[*]}{.key}{":"}{.effect}{"-"}{end}')
do
    kubectl taint node $node $taint
done

kubectl get nodes -o wide

echo ".........----------------#################._.-.-KUBERNETES INSTALLATION COMPLETED-.-
._.#################----------------........."
```

```
#!/bin/bash

echo ".....----------################.-.-INSTALLING KUBERNETES-.-._.################----------....."

# Update and install dependencies
apt-get update
apt-get install -y curl apt-transport-https ca-certificates gnupg lsb-release

# Configure Kubernetes package repository
KUBE_LATEST=$(curl -L -s https://dl.k8s.io/release/stable.txt | awk 'BEGIN { FS="." } { printf "%s.%s", $1, $2 }')
mkdir -p /etc/apt/keyrings
curl -fsSL https://pkgs.k8s.io/core:/stable:/${KUBE_LATEST}/deb/Release.key | gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/${KUBE_LATEST}/deb/ /" > /etc/apt/sources.list.d/kubernetes.list

# Update package list and install Kubernetes components
apt-get update
apt-get install -y kubelet kubectl kubeadm kubernetes-cni containerd

# Configure containerd
mkdir -p /etc/containerd
containerd config default | sed 's/SystemdCgroup = false/SystemdCgroup = true/' > /etc/containerd/config.toml
systemctl restart containerd
systemctl enable kubelet

# Initialize Kubernetes cluster
kubeadm reset -f
kubeadm init --pod-network-cidr='10.244.0.0/16' --service-cidr='10.96.0.0/16' --skip-token-print

# Configure kubectl for the root user
mkdir -p ~/.kube
cp -i /etc/kubernetes/admin.conf ~/.kube/config

# Install a pod network (WeaveNet)
kubectl apply -f "https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s-1.11.yaml"
kubectl rollout status daemonset weave-net -n kube-system --timeout=90s

# Remove taints from the control-plane node to schedule workloads
node=$(kubectl get nodes -o=jsonpath='{.items[0].metadata.name}')
for taint in $(kubectl get node $node -o=jsonpath='{range .spec.taints[*]}{.key}{":"}{.effect}{"~"}{end}')
do
    kubectl taint node $node $taint
done

kubectl get nodes -o wide

echo ".....----------################.-.-KUBERNETES INSTALLATION COMPLETED-.-._.################----------.....
~
~
~
~
~                                                                                                    46,136        All
```

Then to save and quit the vi, type (**:wq**) and press **ENTER**

```
ubuntu@ip-172-31-86-141:~$ ll
total 295936
drwxr-x---  5 ubuntu ubuntu      4096 Dec 16 21:55 ./
drwxr-xr-x  3 root   root        4096 Dec 16 19:41 ../
-rw-r--r--  1 ubuntu ubuntu       220 Mar 31  2024 .bash_logout
-rw-r--r--  1 ubuntu ubuntu      3771 Mar 31  2024 .bashrc
drwx------  2 ubuntu ubuntu      4096 Dec 16 20:39 .cache/
-rw-r--r--  1 ubuntu ubuntu       807 Mar 31  2024 .profile
drwx------  2 ubuntu ubuntu      4096 Dec 16 19:41 .ssh/
-rw-r--r--  1 ubuntu ubuntu         0 Dec 16 20:45 .sudo_as_admin_successful
-rw-rw-r--  1 ubuntu ubuntu   9102945 Aug 17 18:44 apache-maven-3.9.9-bin.tar.gz
drwxr-xr-x 11 ubuntu ubuntu      4096 Feb  3  2023 sonarqube-9.9.0.65466/
-rw-rw-r--  1 ubuntu ubuntu 293899334 Feb  3  2023 sonarqube-9.9.0.65466.zip
ubuntu@ip-172-31-86-141:~$ sudo vi kubeadm.sh
ubuntu@ip-172-31-86-141:~$
```

To install the script, we first execute permission with the command:

sudo chmod +x kubeadm.sh

```
ubuntu@ip-172-31-86-141:~$ ll
total 295936
drwxr-x---  5 ubuntu ubuntu      4096 Dec 16 21:55 ./
drwxr-xr-x  3 root   root        4096 Dec 16 19:41 ../
-rw-r--r--  1 ubuntu ubuntu       220 Mar 31  2024 .bash_logout
-rw-r--r--  1 ubuntu ubuntu      3771 Mar 31  2024 .bashrc
drwx------  2 ubuntu ubuntu      4096 Dec 16 20:39 .cache/
-rw-r--r--  1 ubuntu ubuntu       807 Mar 31  2024 .profile
drwx------  2 ubuntu ubuntu      4096 Dec 16 19:41 .ssh/
-rw-r--r--  1 ubuntu ubuntu         0 Dec 16 20:45 .sudo_as_admin_successful
-rw-rw-r--  1 ubuntu ubuntu   9102945 Aug 17 18:44 apache-maven-3.9.9-bin.tar.gz
drwxr-xr-x 11 ubuntu ubuntu      4096 Feb  3  2023 sonarqube-9.9.0.65466/
-rw-rw-r--  1 ubuntu ubuntu 293899334 Feb  3  2023 sonarqube-9.9.0.65466.zip
ubuntu@ip-172-31-86-141:~$ sudo vi kubeadm.sh
ubuntu@ip-172-31-86-141:~$ sudo chmod +x kubeadm.sh
ubuntu@ip-172-31-86-141:~$
```

Run the command

ll

```
ubuntu@ip-172-31-86-141:~$ sudo chmod +x kubeadm.sh
ubuntu@ip-172-31-86-141:~$ ll
total 295940
drwxr-x---  5 ubuntu ubuntu      4096 Dec 16 23:08 ./
drwxr-xr-x  3 root   root        4096 Dec 16 19:41 ../
-rw-r--r--  1 ubuntu ubuntu       220 Mar 31  2024 .bash_logout
-rw-r--r--  1 ubuntu ubuntu      3771 Mar 31  2024 .bashrc
drwx------  2 ubuntu ubuntu      4096 Dec 16 20:39 .cache/
-rw-r--r--  1 ubuntu ubuntu       807 Mar 31  2024 .profile
drwx------  2 ubuntu ubuntu      4096 Dec 16 19:41 .ssh/
-rw-r--r--  1 ubuntu ubuntu         0 Dec 16 20:45 .sudo_as_admin_successful
-rw-rw-r--  1 ubuntu ubuntu   9102945 Aug 17 18:44 apache-maven-3.9.9-bin.tar.gz
-rwxr-xr-x  1 root   root        2029 Dec 16 23:08 kubeadm.sh*
drwxr-xr-x 11 ubuntu ubuntu      4096 Feb  3  2023 sonarqube-9.9.0.65466/
-rw-rw-r--  1 ubuntu ubuntu 293899334 Feb  3  2023 sonarqube-9.9.0.65466.zip
ubuntu@ip-172-31-86-141:~$
```

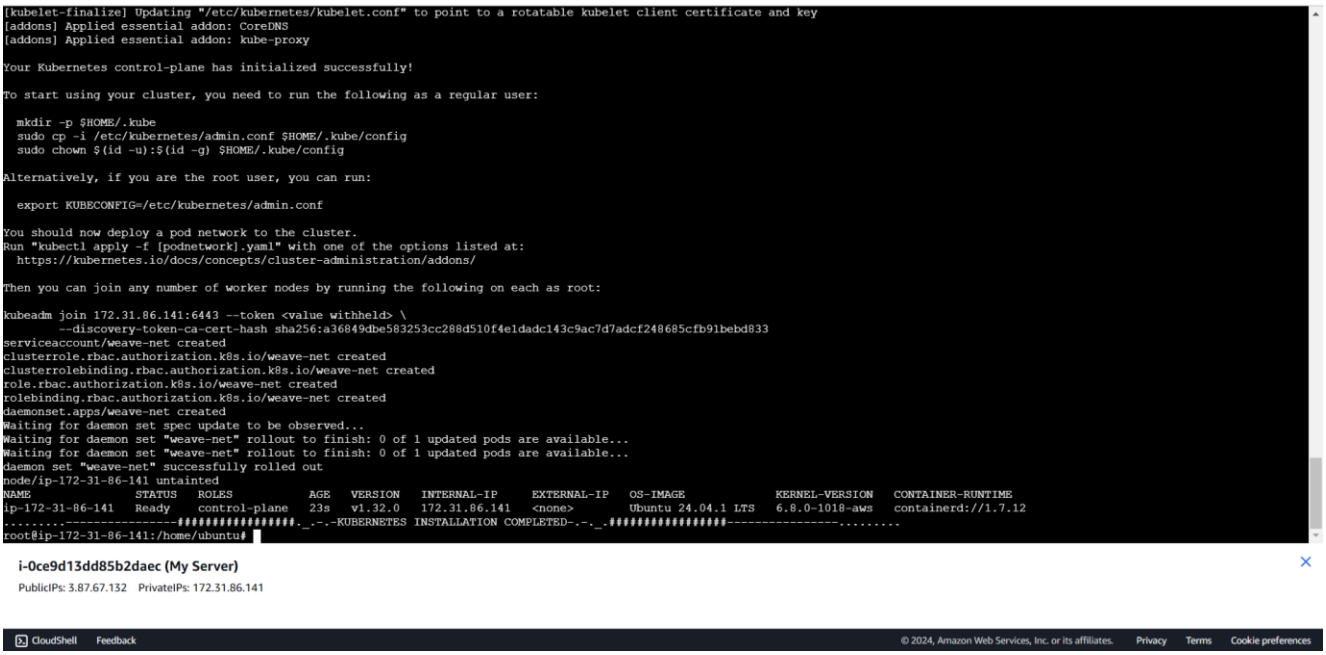
Run the command to switch to root user:

sudo su

```
ubuntu@ip-172-31-86-141:~$ sudo chmod +x kubeadm.sh
ubuntu@ip-172-31-86-141:~$ ll
total 295940
drwxr-x---  5 ubuntu ubuntu      4096 Dec 16 23:08 ./
drwxr-xr-x  3 root   root        4096 Dec 16 19:41 ../
-rw-r--r--  1 ubuntu ubuntu       220 Mar 31  2024 .bash_logout
-rw-r--r--  1 ubuntu ubuntu      3771 Mar 31  2024 .bashrc
drwx------  2 ubuntu ubuntu      4096 Dec 16 20:39 .cache/
-rw-r--r--  1 ubuntu ubuntu       807 Mar 31  2024 .profile
drwx------  2 ubuntu ubuntu      4096 Dec 16 19:41 .ssh/
-rw-r--r--  1 ubuntu ubuntu         0 Dec 16 20:45 .sudo_as_admin_successful
-rw-rw-r--  1 ubuntu ubuntu   9102945 Aug 17 18:44 apache-maven-3.9.9-bin.tar.gz
-rwxr-xr-x  1 root   root        2029 Dec 16 23:08 kubeadm.sh*
drwxr-xr-x 11 ubuntu ubuntu      4096 Feb  3  2023 sonarqube-9.9.0.65466/
-rw-rw-r--  1 ubuntu ubuntu 293899334 Feb  3  2023 sonarqube-9.9.0.65466.zip
ubuntu@ip-172-31-86-141:~$ sudo su
root@ip-172-31-86-141:/home/ubuntu#
```

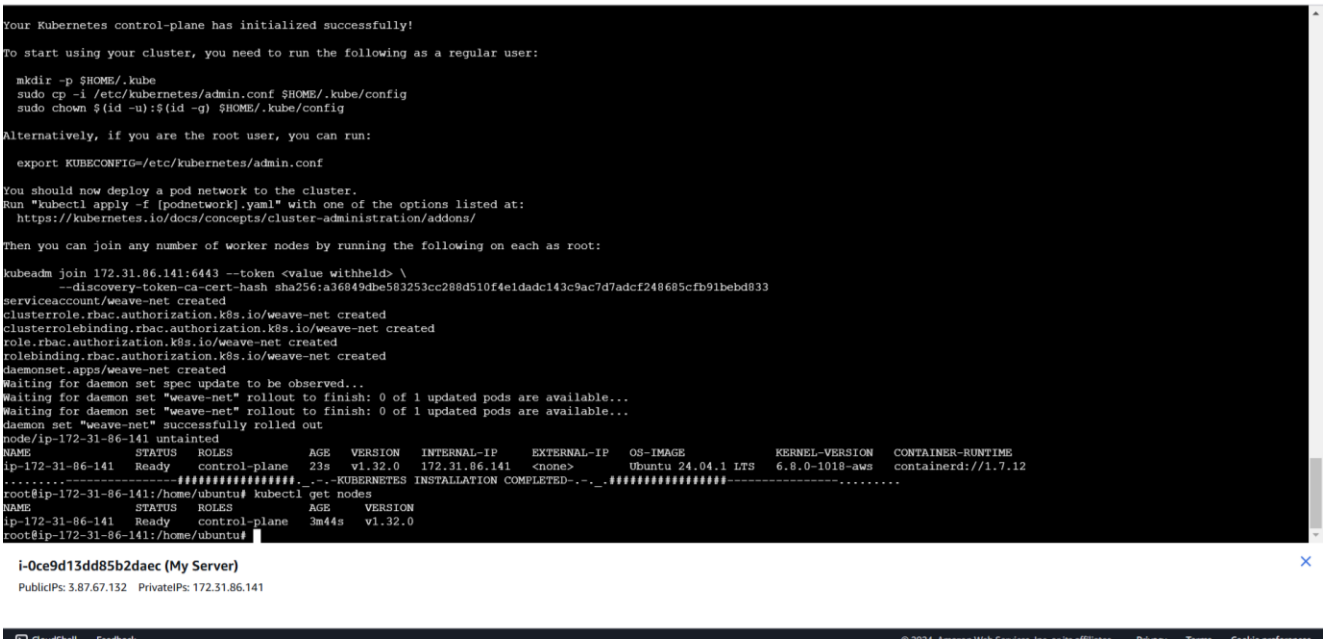
Now, let us install kubeadm.sh using the command:

./kubeadm.sh



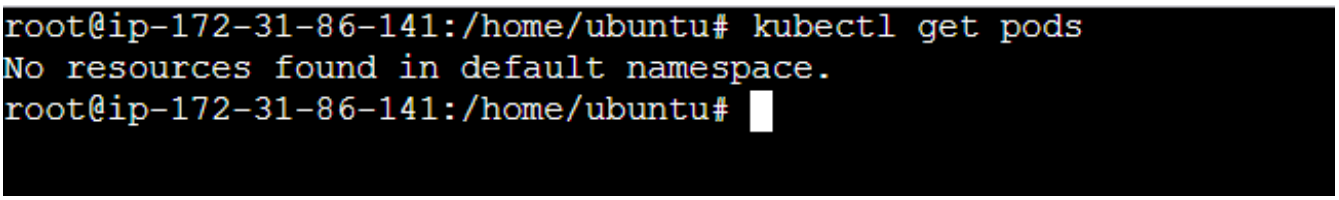
It is done. Now, let us try the command:

kubectl get nodes



So, we have one node and it is ready. Let us now run the command:

kubectl get pods



No resources found in this default namespace.

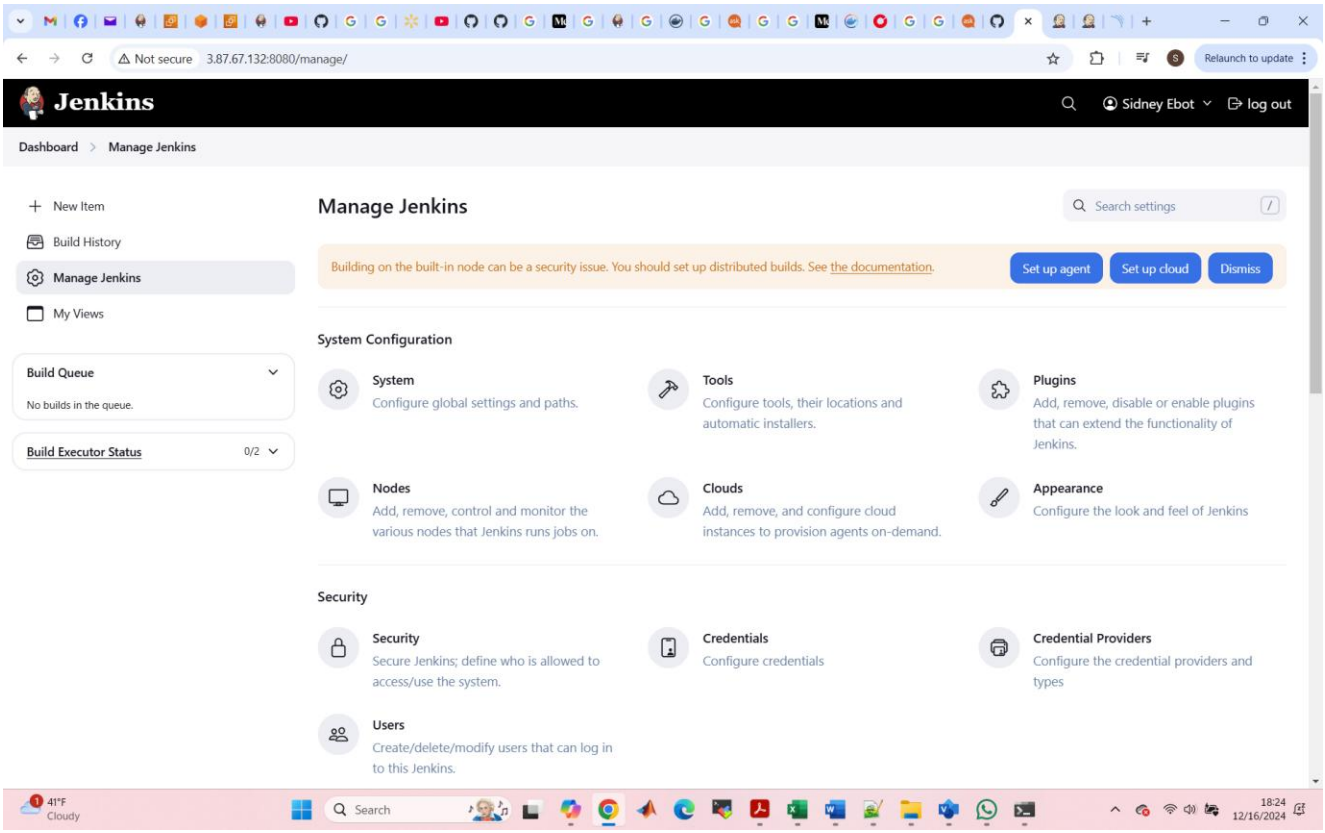
Run the command:

cd ~

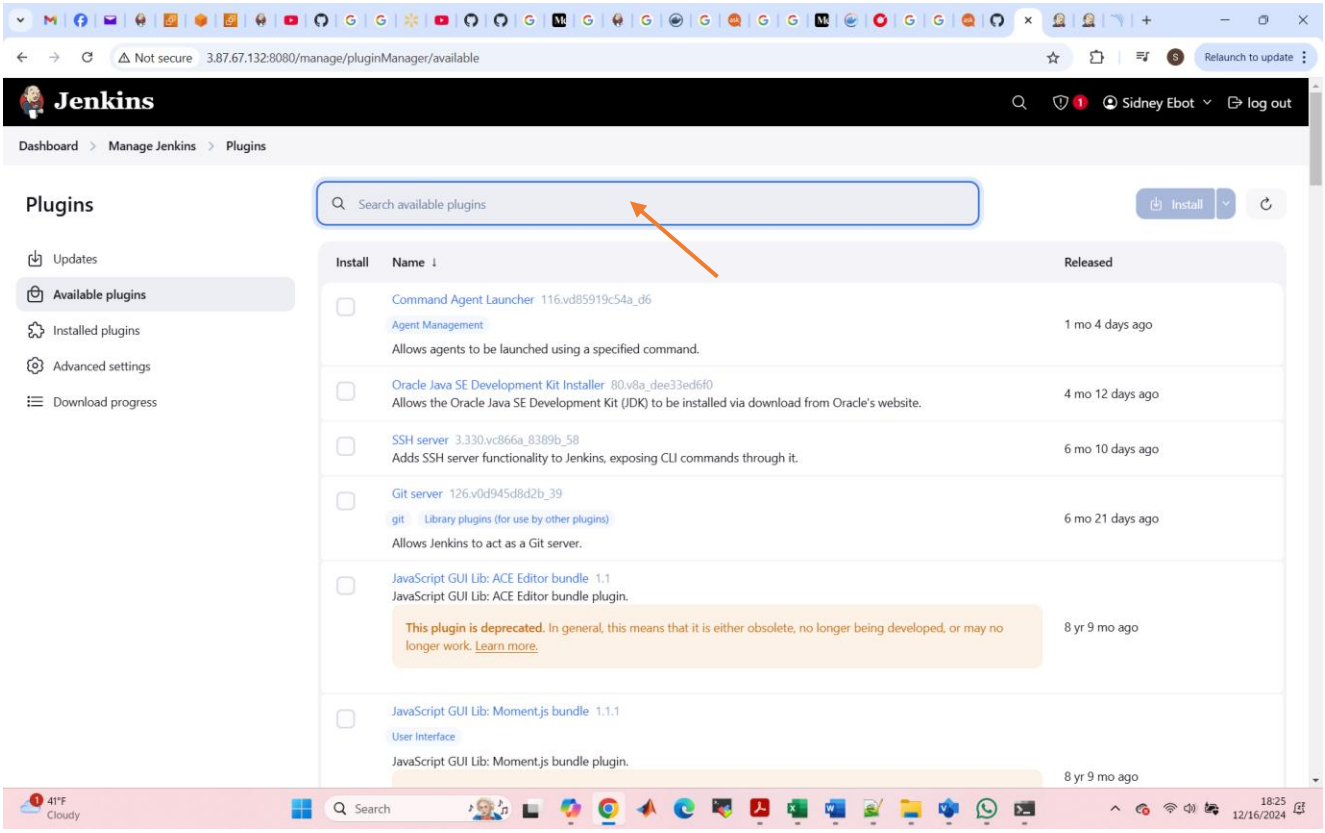
```
root@ip-172-31-86-141:/home/ubuntu# cd ~
root@ip-172-31-86-141:~#
```

PART 2: Install Kubernetes Plugins on Jenkins

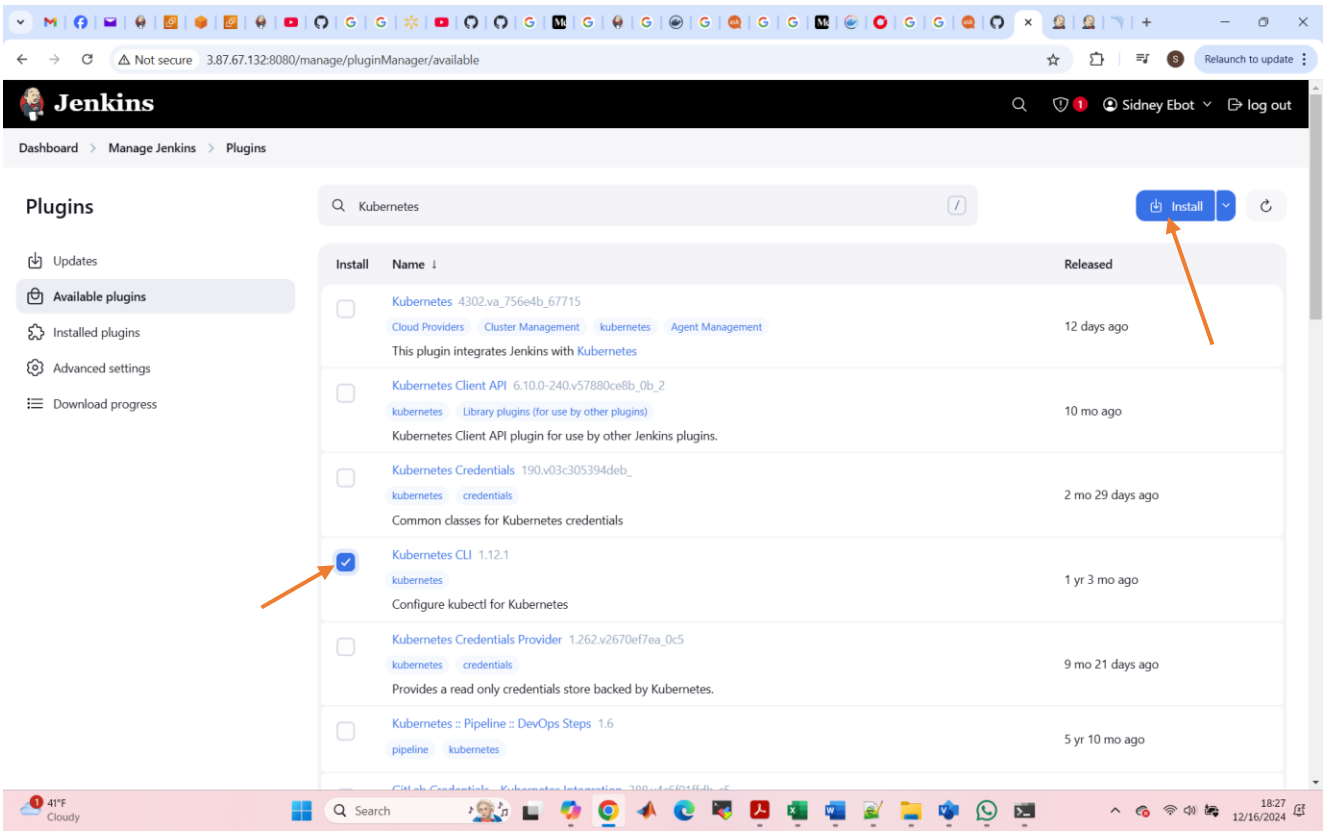
Now, let us configure the CI/CD pipeline. Go to "**Manage Jenkins**"



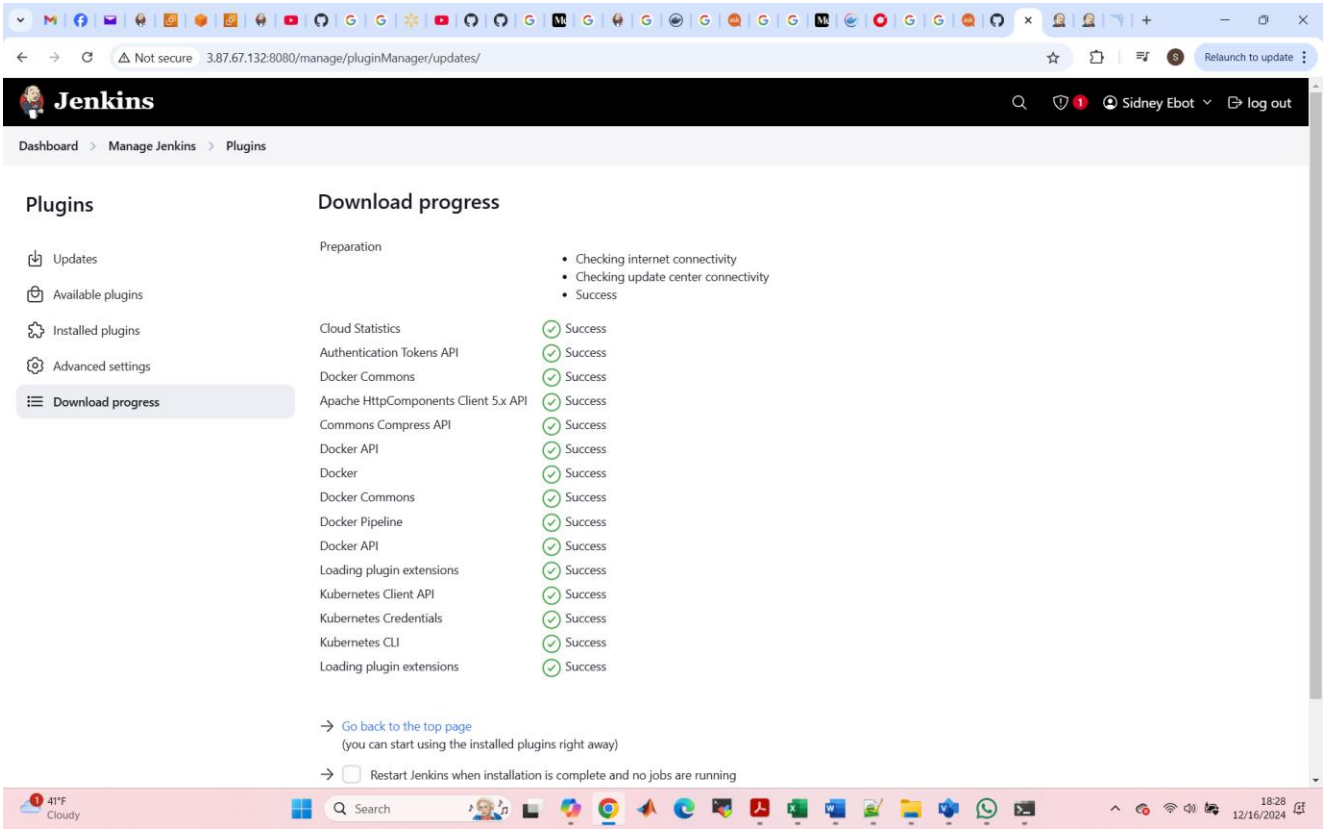
Go to "**Available Plugins**"



Search for "**Kubernetes**" and select "**Kubernetes CLI**"



Click on "**Install**"



On the Instance terminal, run the command:

ll

```
root@ip-172-31-86-141:/home/ubuntu# cd ~
root@ip-172-31-86-141:~# ll
total 40
drwx------  5 root root 4096 Dec 16 23:41 ./
drwxr-xr-x 22 root root 4096 Dec 16 19:41 ../
-rw-------  1 root root   54 Dec 16 23:41 .bash_history
-rw-r--r--  1 root root 3106 Apr 22  2024 .bashrc
drwxr-xr-x  3 root root 4096 Dec 16 23:13 .kube/
-rw-------  1 root root   20 Dec 16 21:27 .lesshst
-rw-r--r--  1 root root  161 Apr 22  2024 .profile
drwx------  2 root root 4096 Dec 16 19:41 .ssh/
-rw-------  1 root root 1831 Dec 16 23:08 .viminfo
drwx------  3 root root 4096 Dec 16 19:41 snap/
root@ip-172-31-86-141:~#
```

Then run the command:

cd .kube/

```
root@ip-172-31-86-141:/home/ubuntu# cd ~
root@ip-172-31-86-141:~# ll
total 40
drwx------  5 root root 4096 Dec 16 23:41 ./
drwxr-xr-x 22 root root 4096 Dec 16 19:41 ../
-rw-------  1 root root   54 Dec 16 23:41 .bash_history
-rw-r--r--  1 root root 3106 Apr 22  2024 .bashrc
drwxr-xr-x  3 root root 4096 Dec 16 23:13 .kube/
-rw-------  1 root root   20 Dec 16 21:27 .lesshst
-rw-r--r--  1 root root  161 Apr 22  2024 .profile
drwx------  2 root root 4096 Dec 16 19:41 .ssh/
-rw-------  1 root root 1831 Dec 16 23:08 .viminfo
drwx------  3 root root 4096 Dec 16 19:41 snap/
root@ip-172-31-86-141:~# cd .kube/
root@ip-172-31-86-141:~/.kube#
```

Run the command”

11

```
root@ip-172-31-86-141:/home/ubuntu# cd ~
root@ip-172-31-86-141:~# ll
total 40
drwx------  5 root root 4096 Dec 16 23:41 ./
drwxr-xr-x 22 root root 4096 Dec 16 19:41 ../
-rw-------  1 root root   54 Dec 16 23:41 .bash_history
-rw-r--r--  1 root root 3106 Apr 22  2024 .bashrc
drwxr-xr-x  3 root root 4096 Dec 16 23:13 .kube/
-rw-------  1 root root   20 Dec 16 21:27 .lesshst
-rw-r--r--  1 root root  161 Apr 22  2024 .profile
drwx------  2 root root 4096 Dec 16 19:41 .ssh/
-rw-------  1 root root 1831 Dec 16 23:08 .viminfo
drwx------  3 root root 4096 Dec 16 19:41 snap/
root@ip-172-31-86-141:~# cd .kube/
root@ip-172-31-86-141:~/.kube# ll
total 20
drwxr-xr-x 3 root root 4096 Dec 16 23:13 ./
drwx------ 5 root root 4096 Dec 16 23:41 ../
drwxr-x--- 4 root root 4096 Dec 16 23:13 cache/
-rw------- 1 root root 5657 Dec 16 23:13 config
root@ip-172-31-86-141:~/.kube#
```

Here, we have the “config” file. This is the file we will use to login to the cluster.

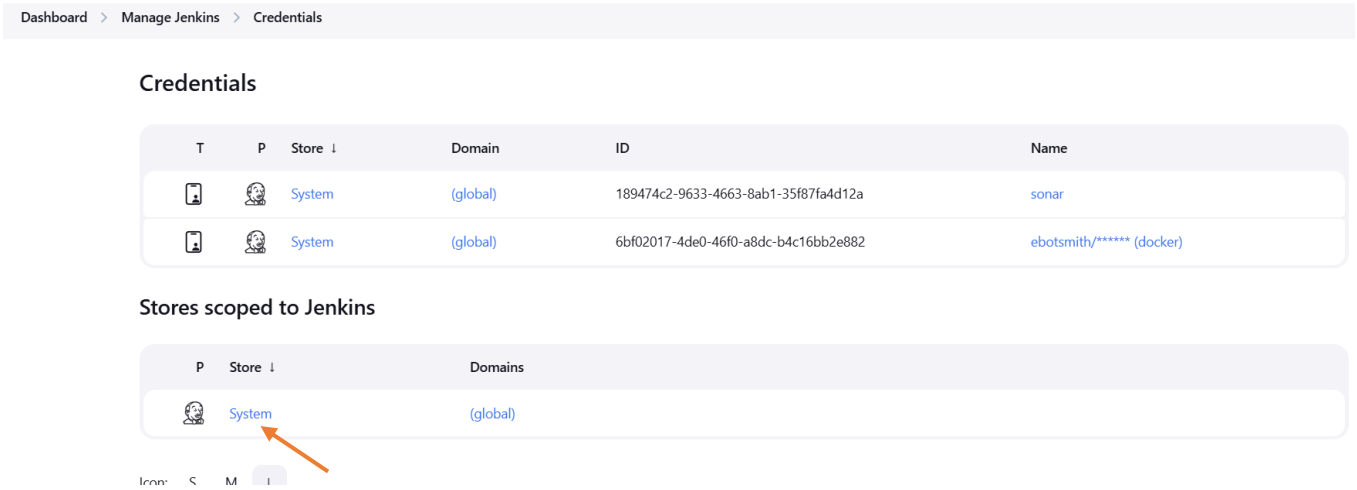
Run the command:

`cat config`

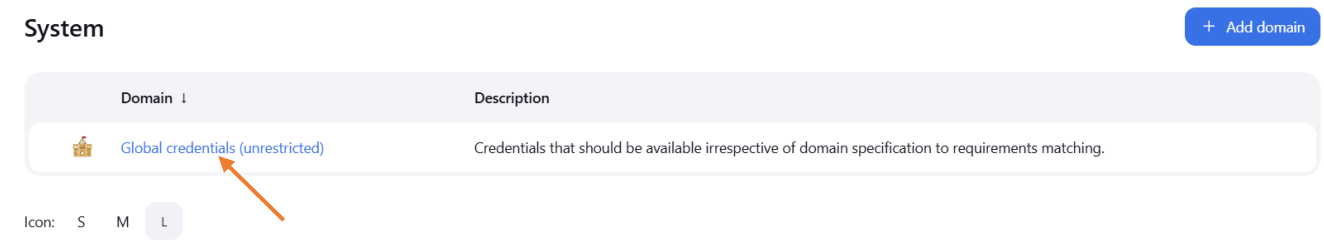
Copy this and paste on a notepad and save as “**kubeconfig**”

PART 3: Add credentials for Kubernetes

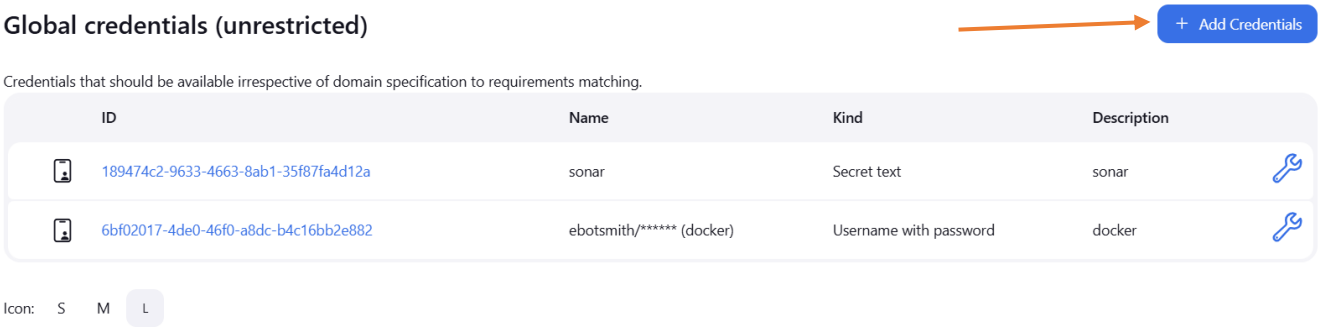
Now go to Jenkins GUI and navigate to “Manage Jenkins”, then to “credentials” to add a credential



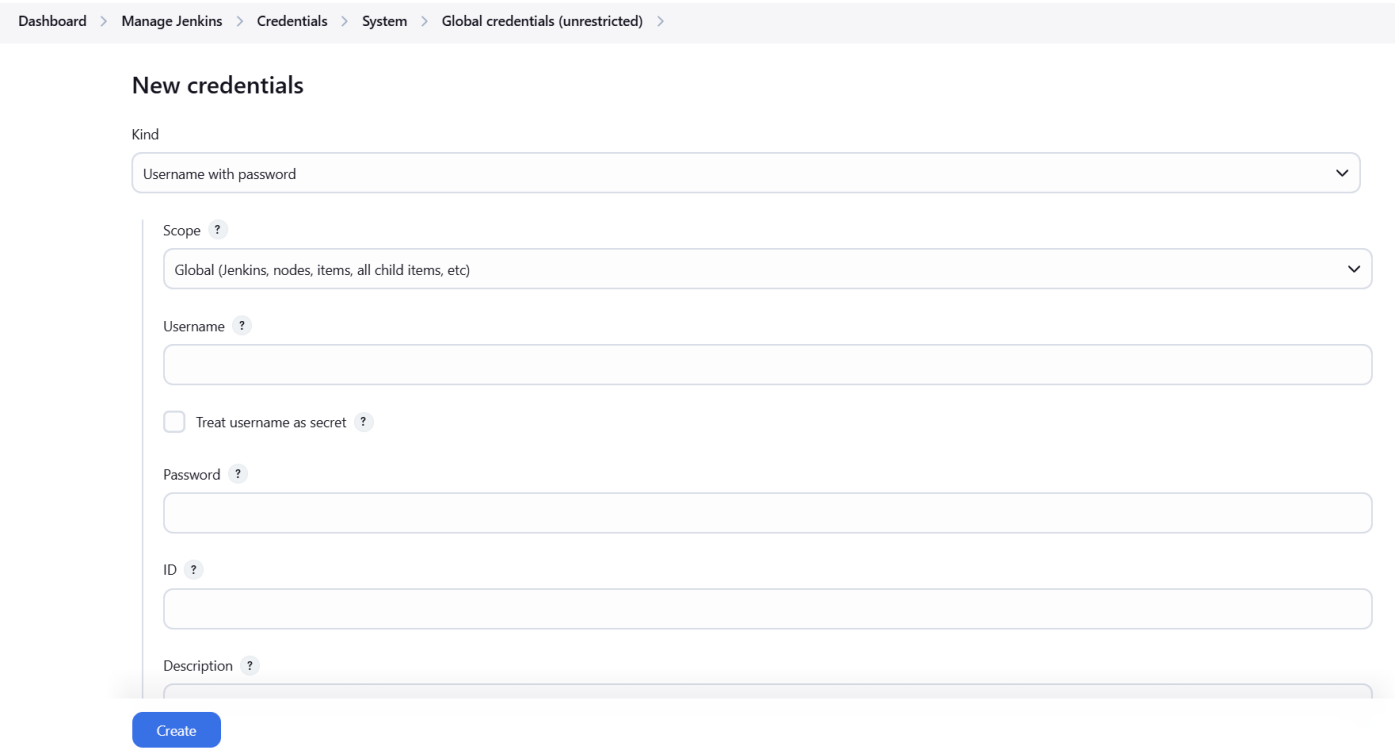
Click on “**System**”



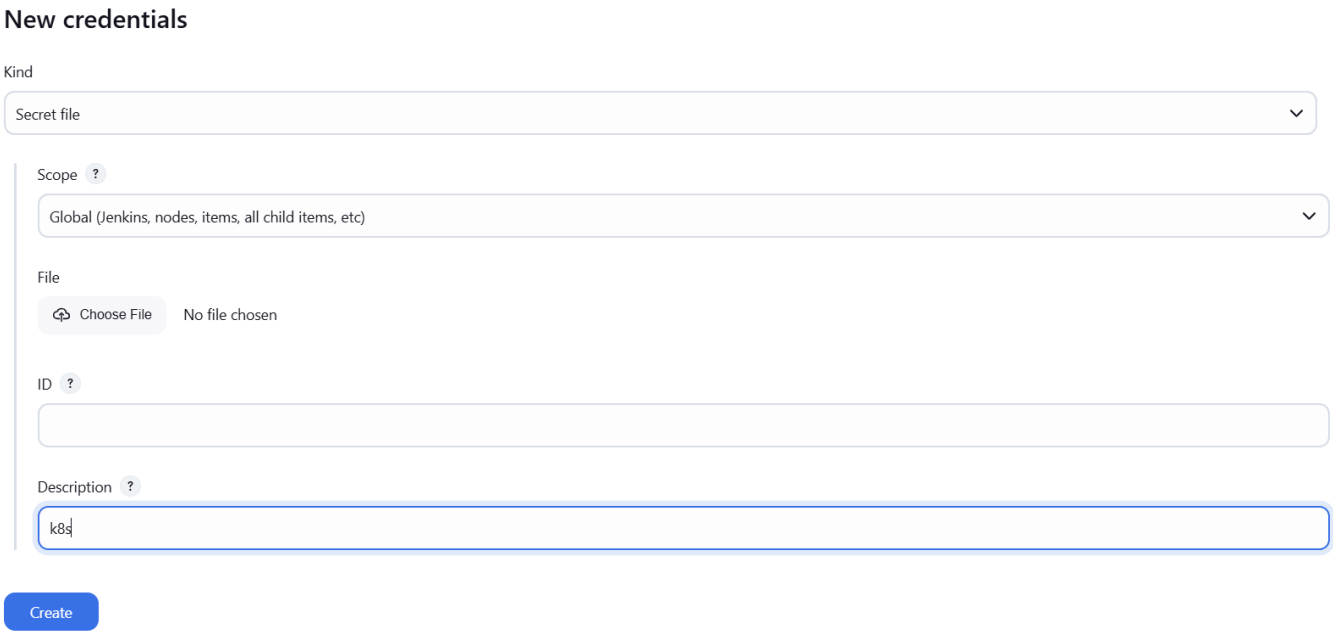
Click on “**Global Credentials**”



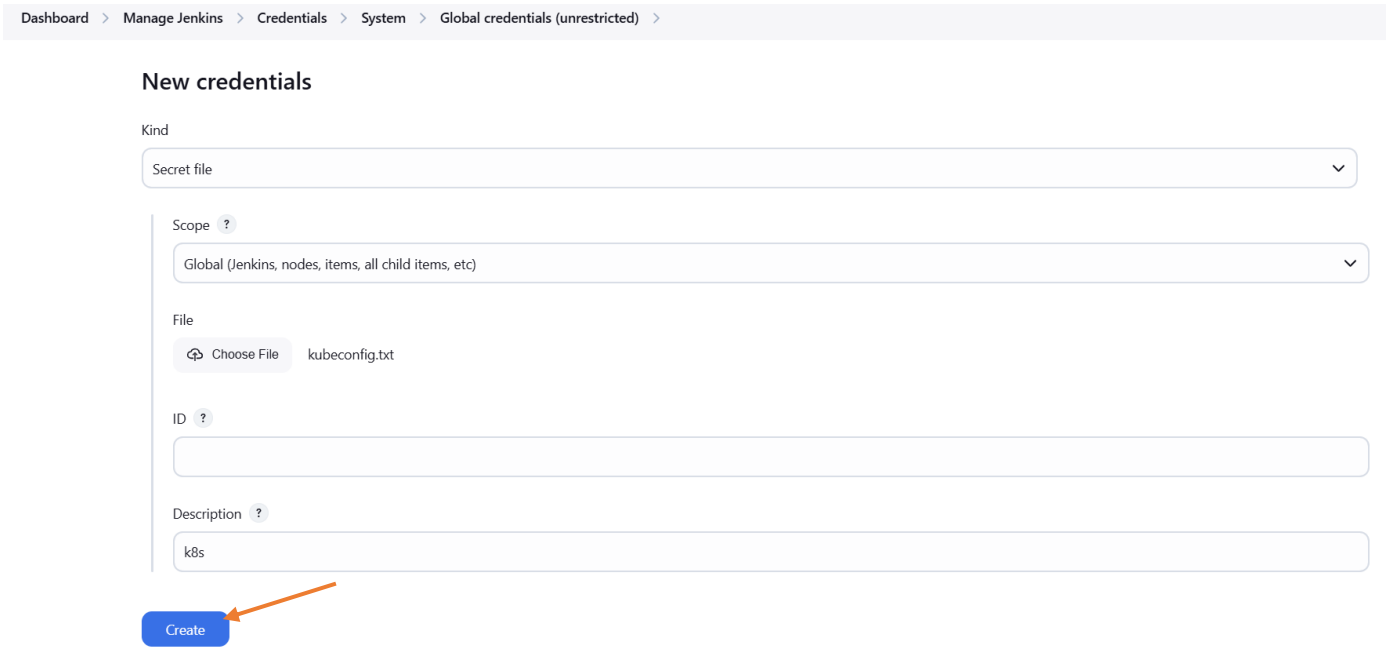
Click on “**add Credentials**”



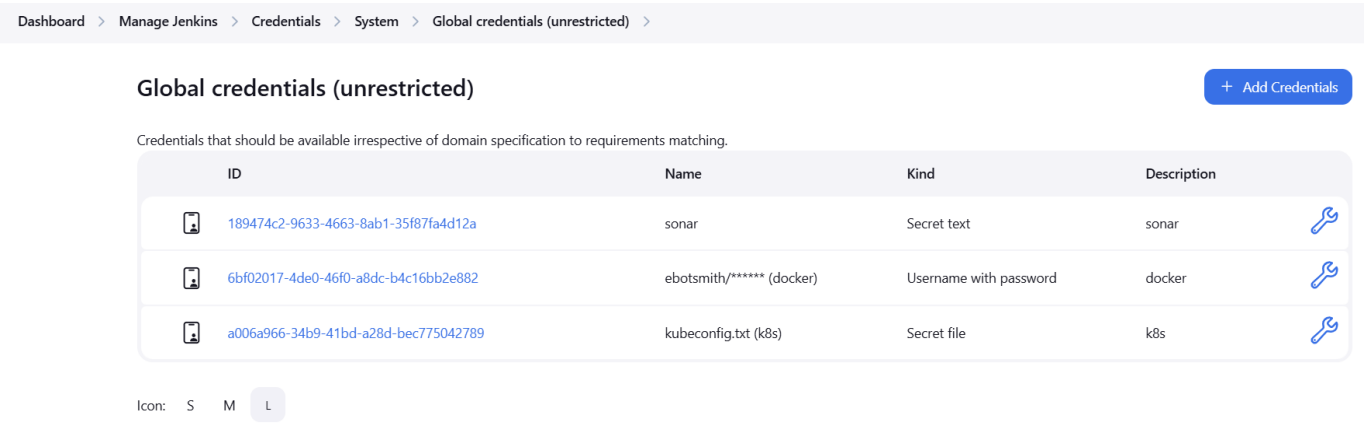
Click on the drop down on “**kind**”, select “**Secret File**”



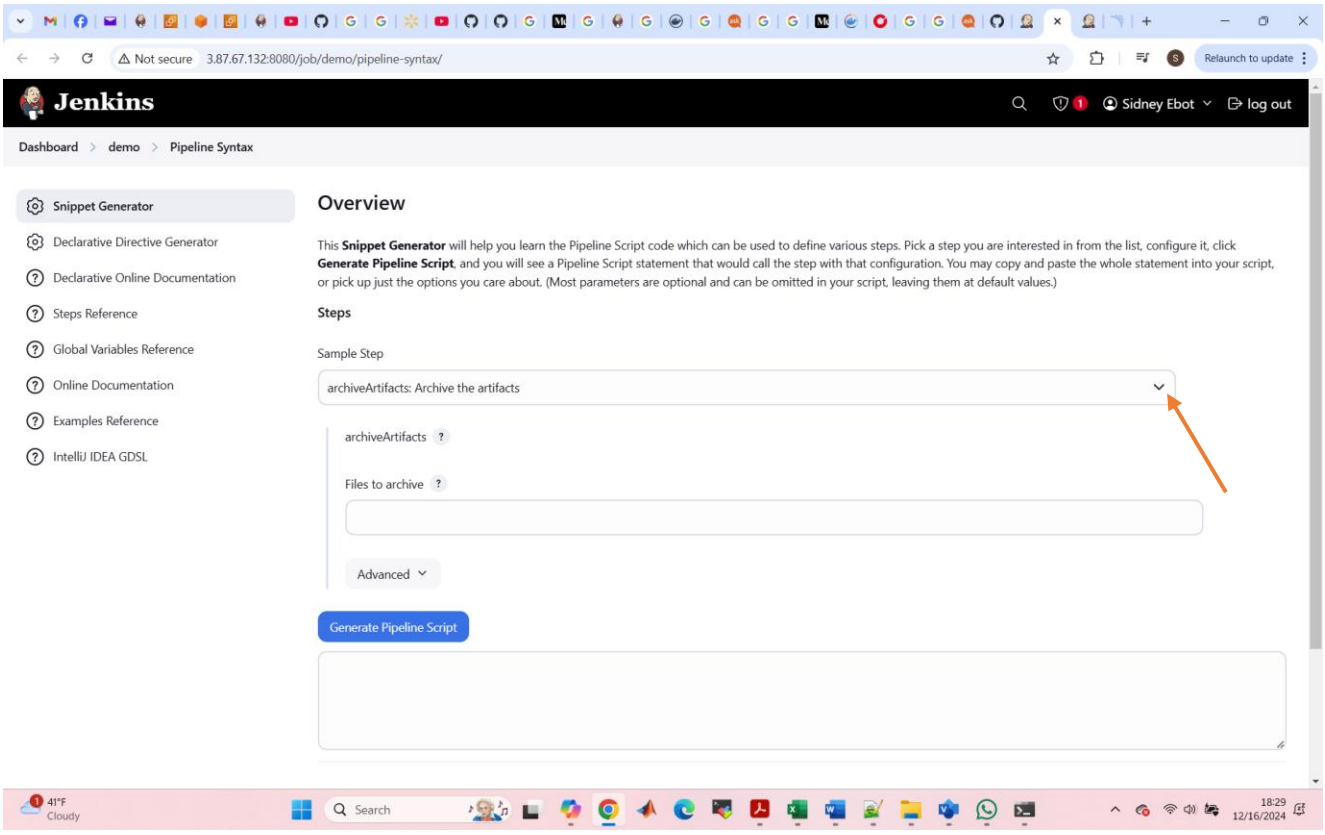
Then click on "**Choose file**" and select the file we just saved, that is "**kubeconfig**" and in description put "**k8s**"



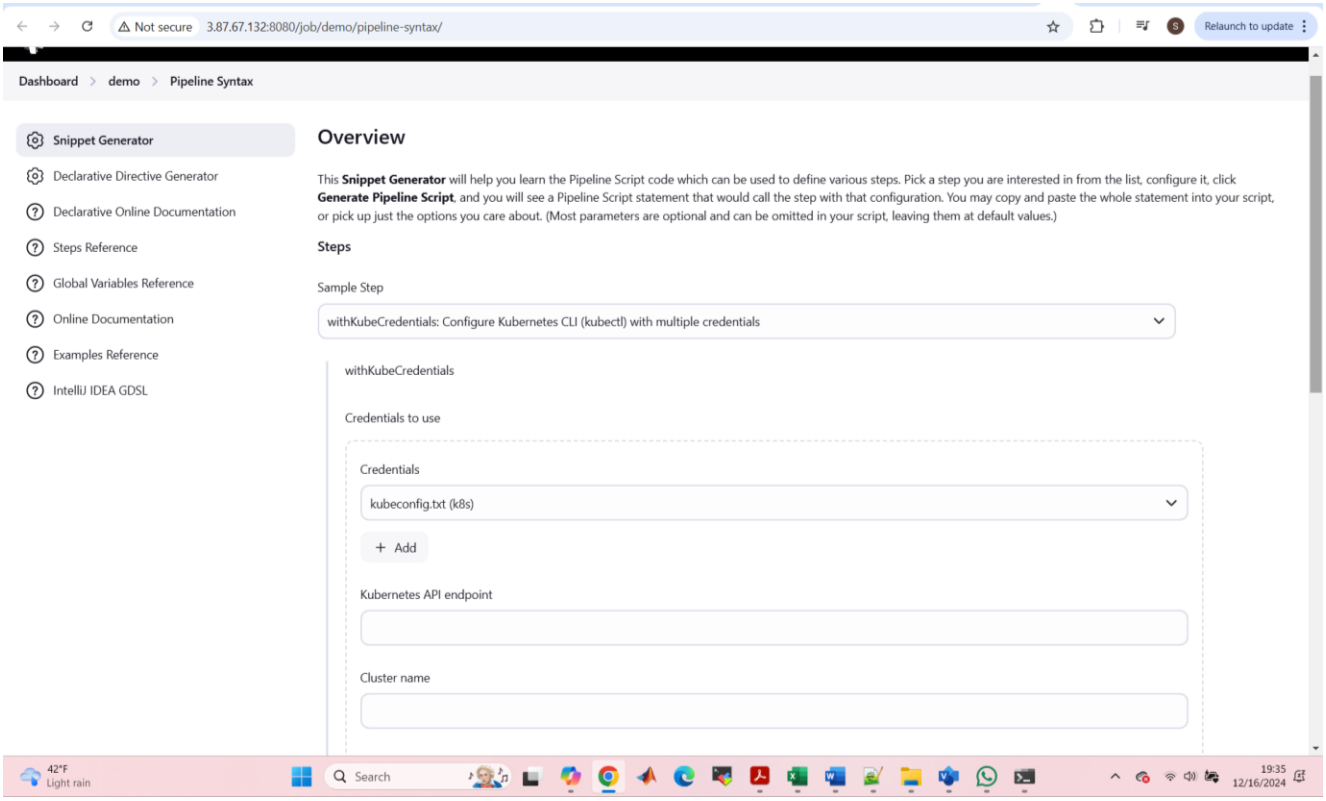
Click on "**Create**"



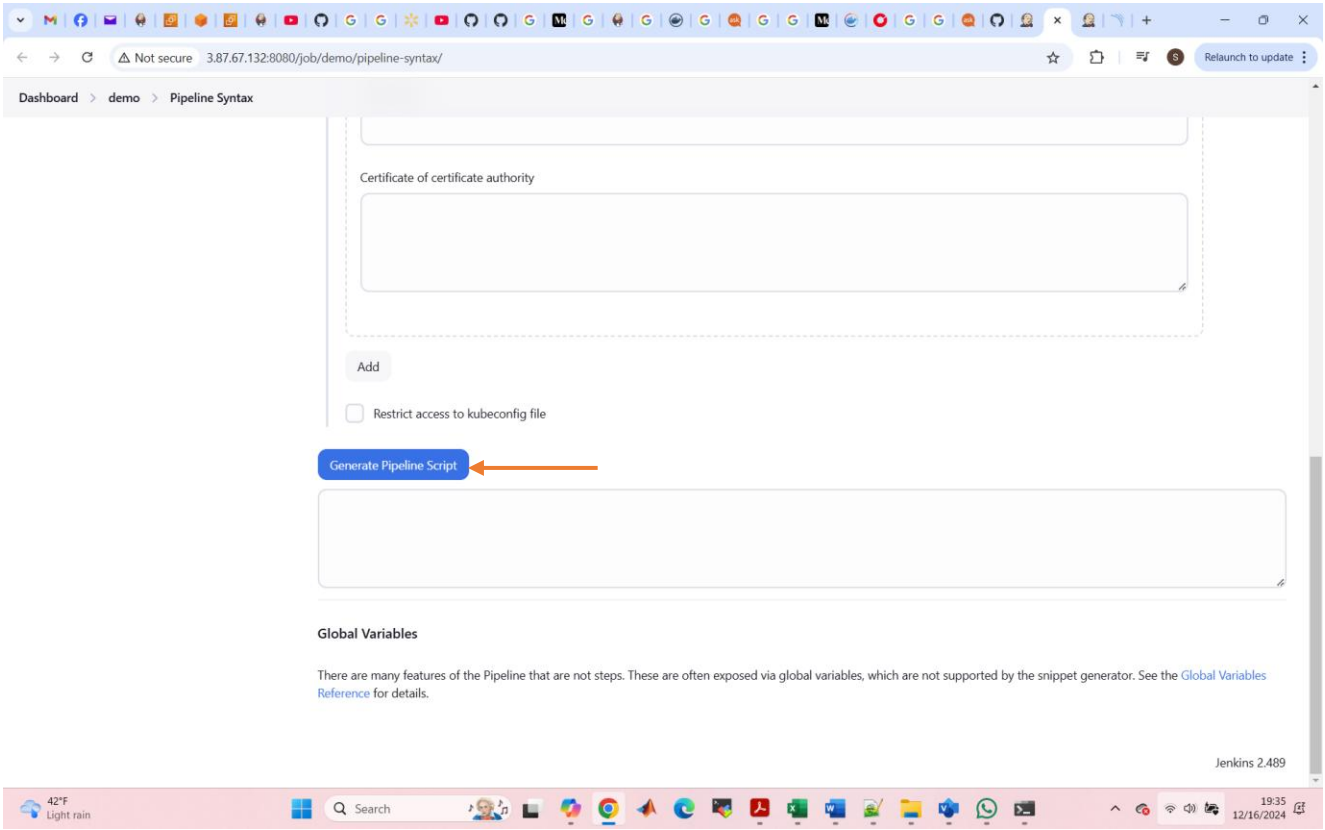
Go back to the "**Pipeline Syntax**" and refresh the page



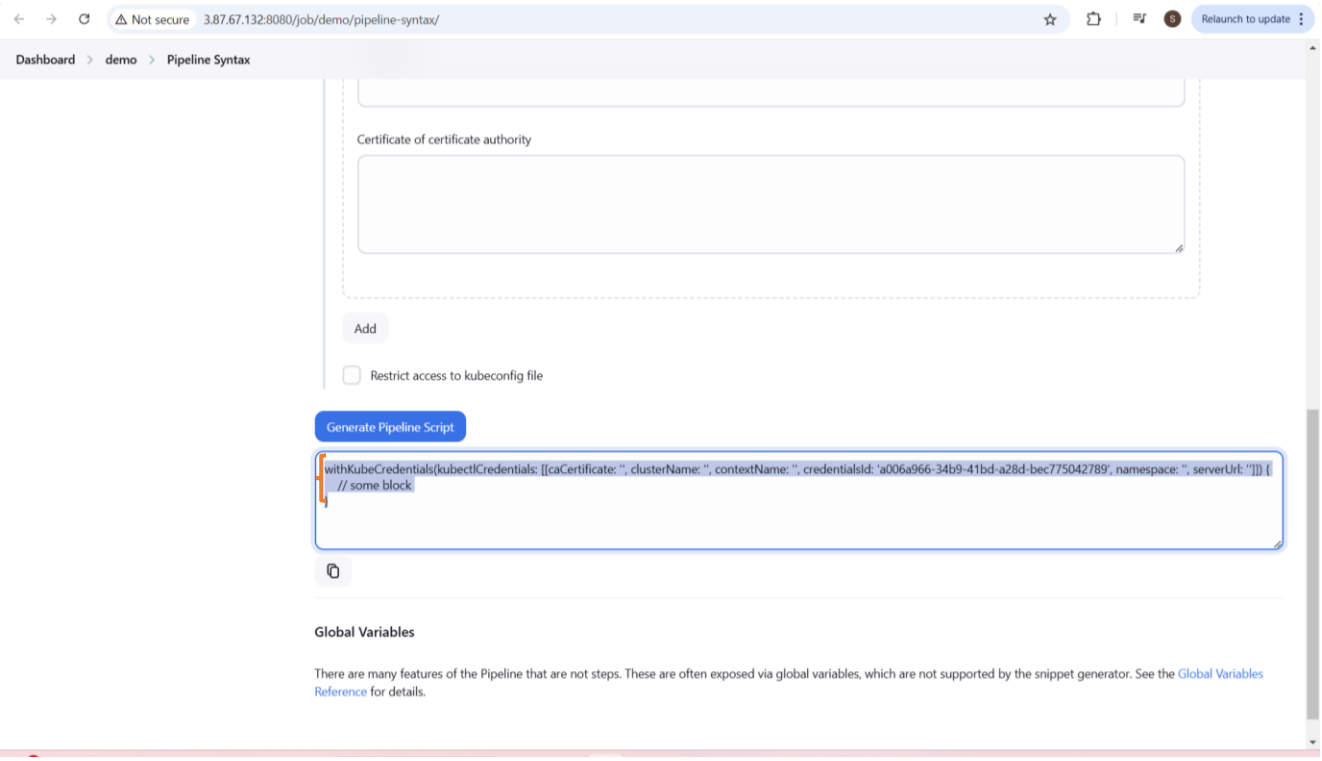
Click on the drop down on "**Sample Step**", look for "**withKubeCredentials…….**", Under "**Credentials**", select the "**kubeconfig**" file we just saved.



Scroll down

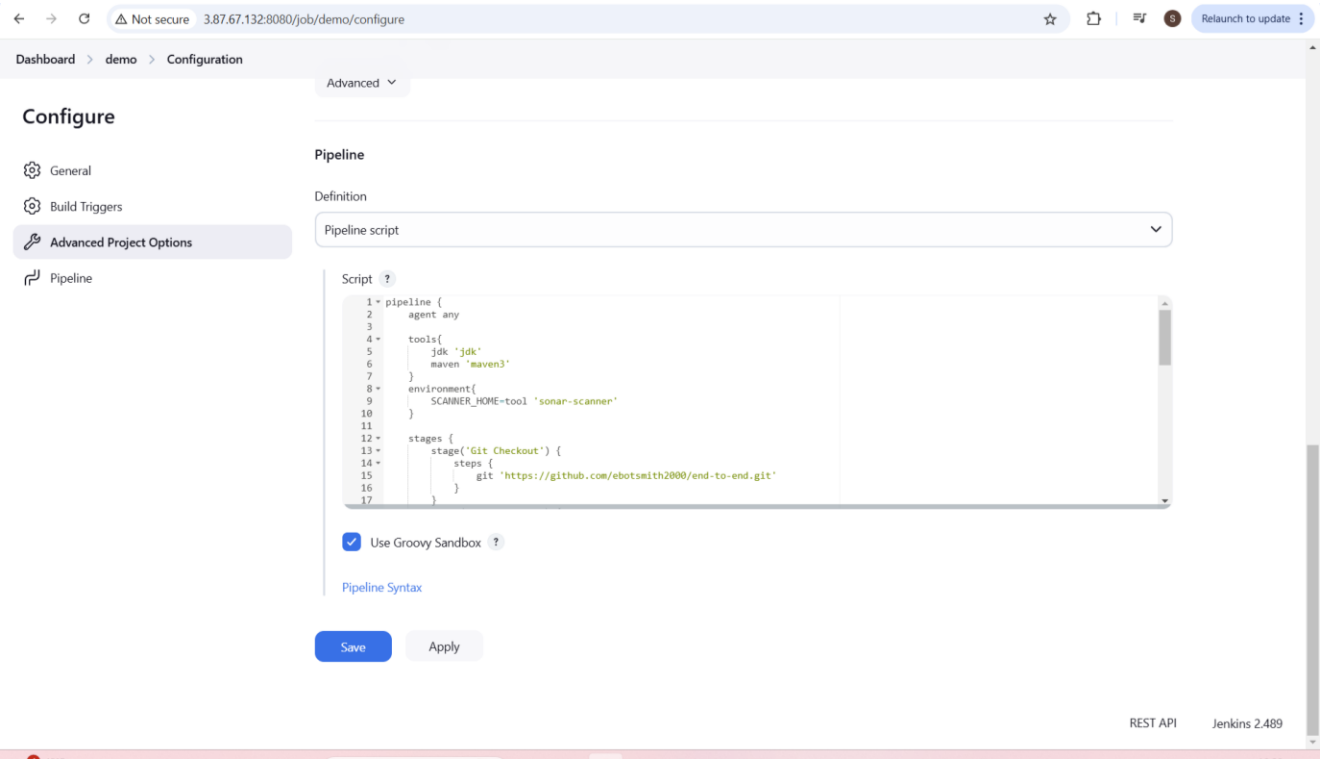


Click on “**Generate Pipeline Script**”



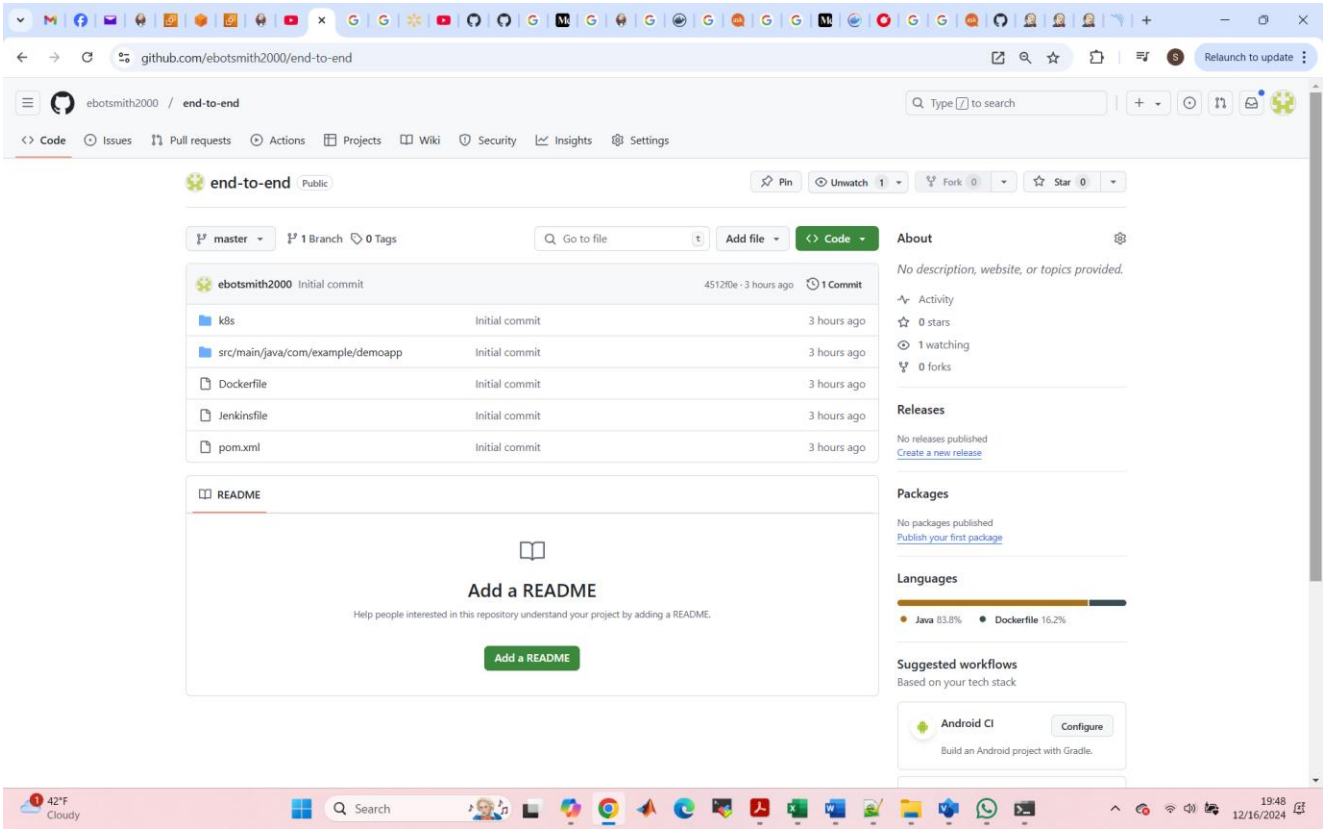
Copy the generated script and use in our next Pipeline stage

Go to **Dashboard** - > "**demo**" -> **Configure**

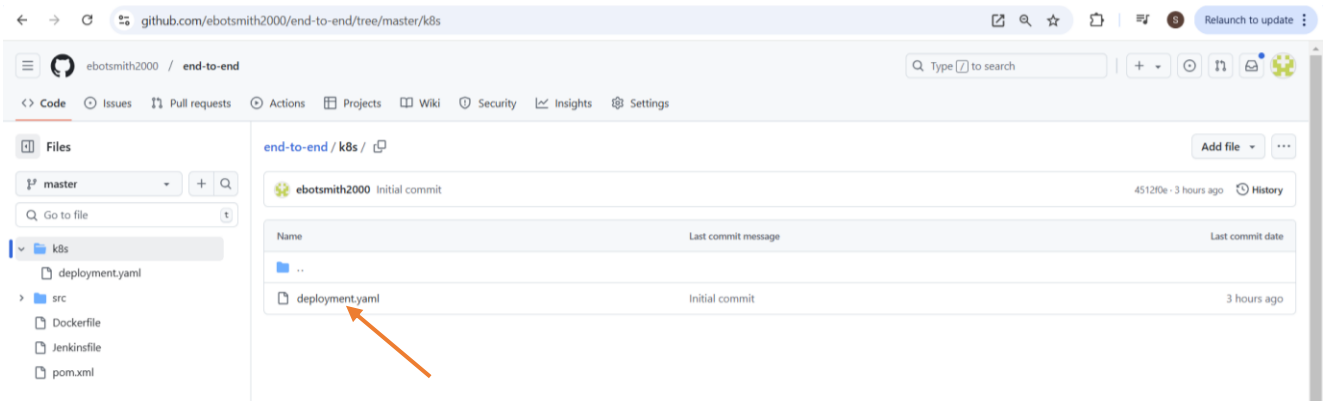


Add a new stage to the pipeline script and we will use the copied generated script here. Then modify the script.

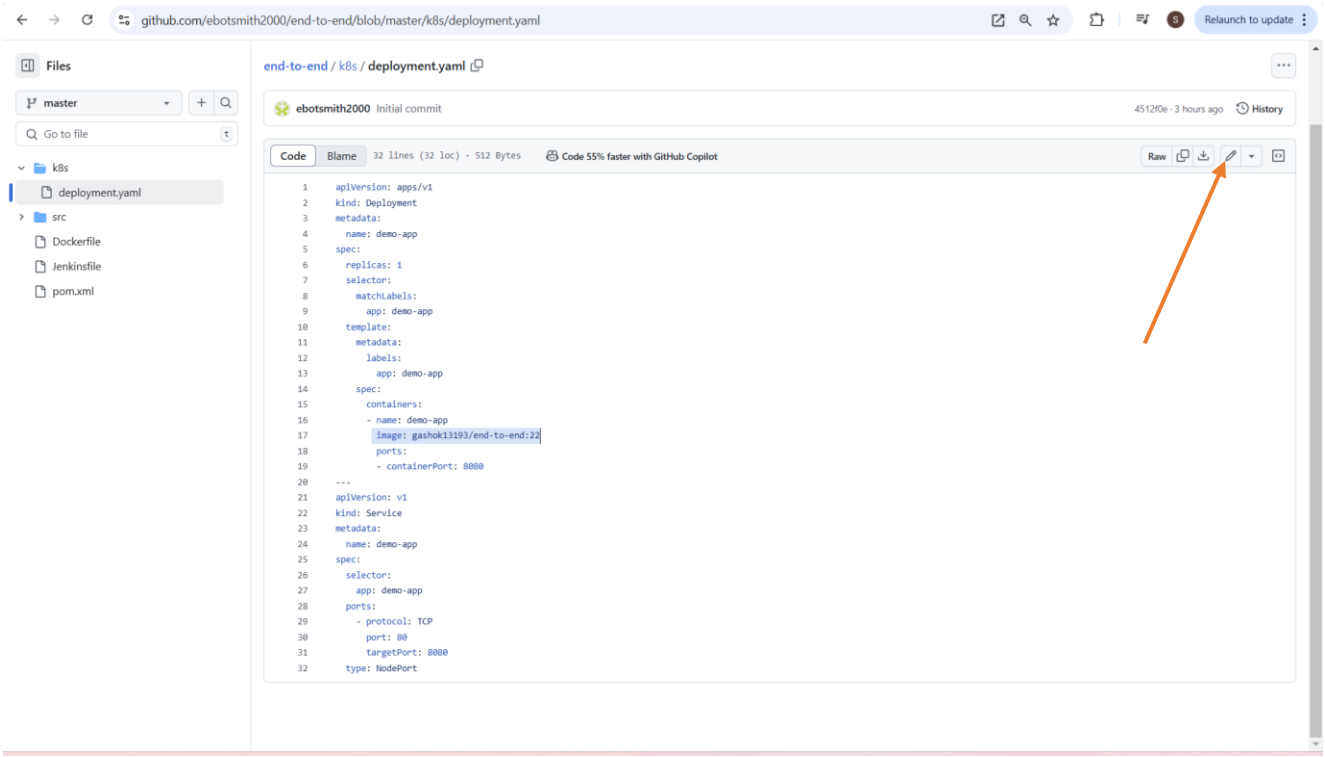
But let us first go to our GitHub repository



Click on "**k8s**" folder

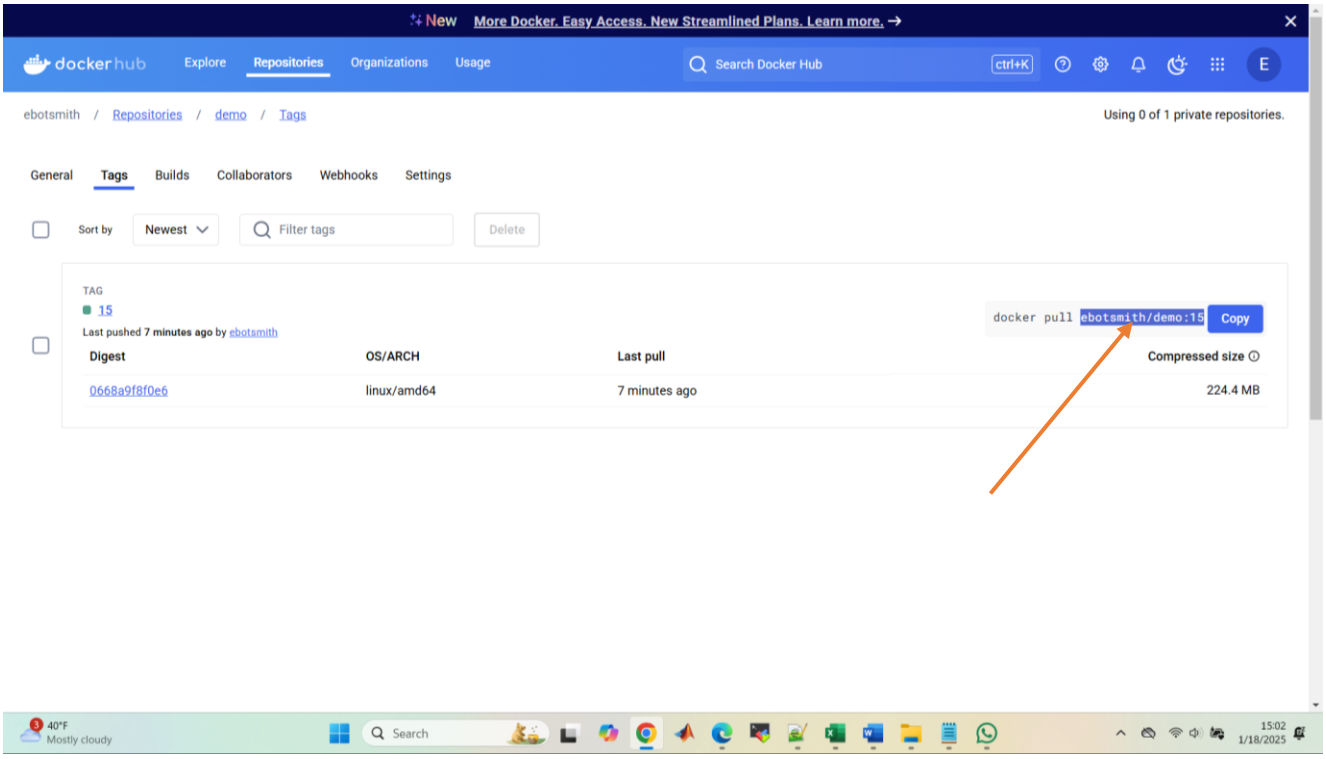


Click on "**deployment.yaml**" file



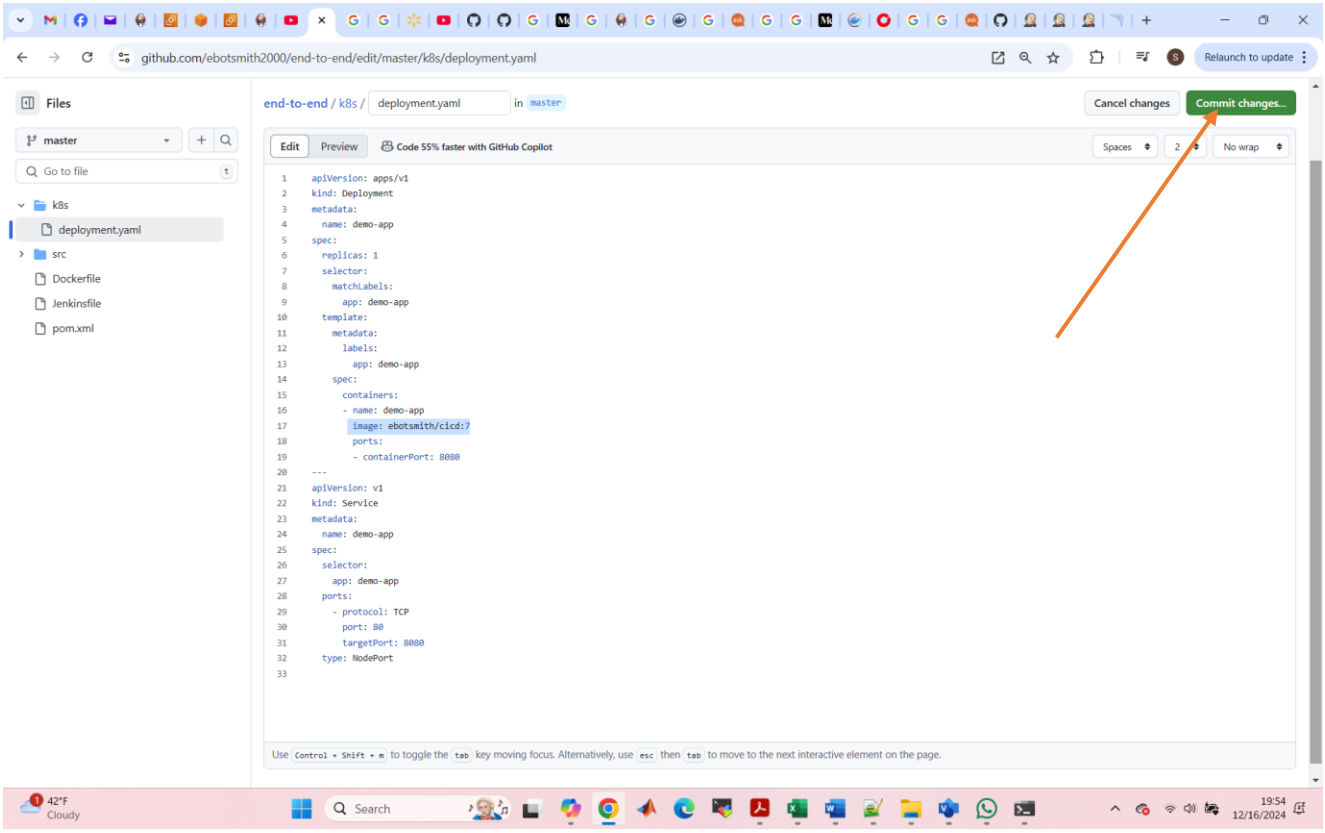
Edit the highlighted line in this script to match with your "**docker Image**". You will click on "**Edit**"

Open your Docker hub

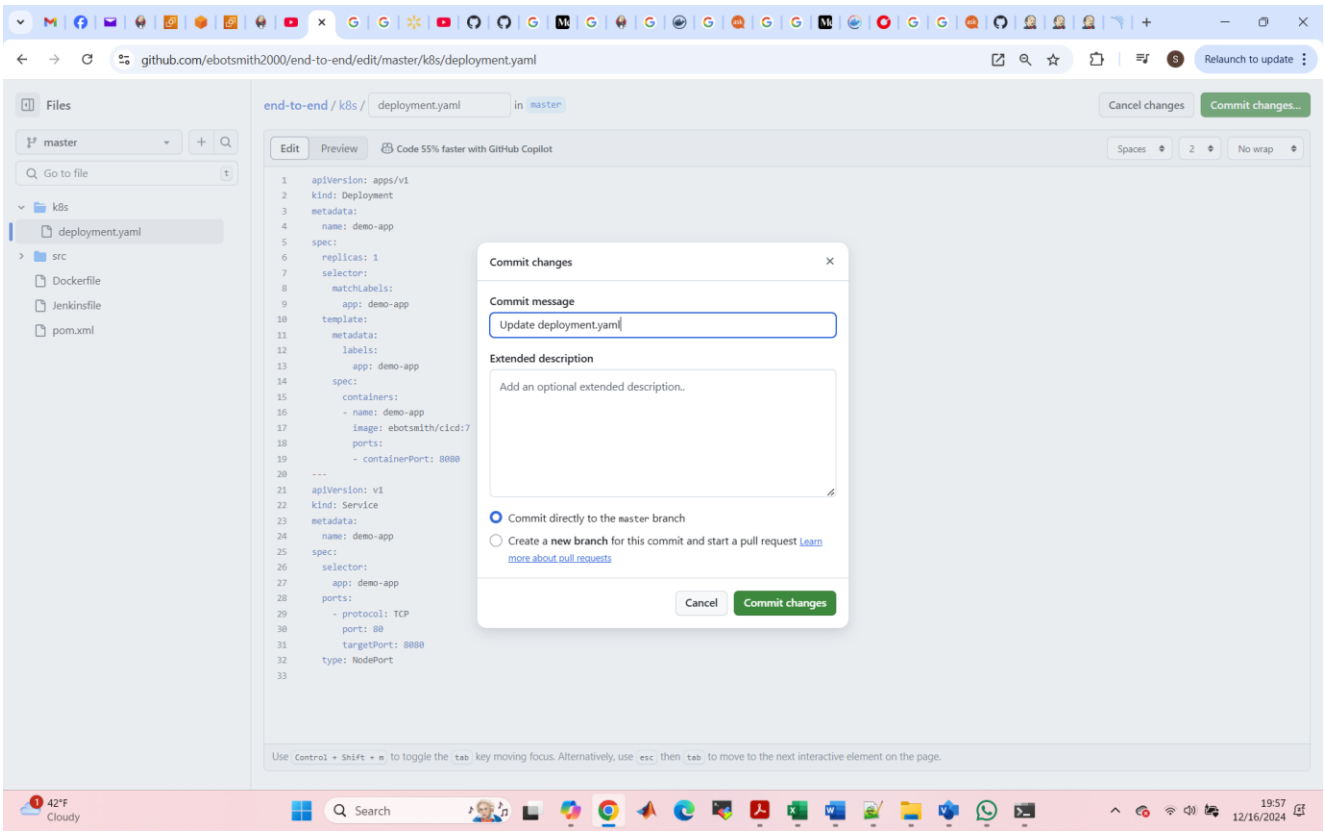


Copy this: **ebotsmith/demo:15**

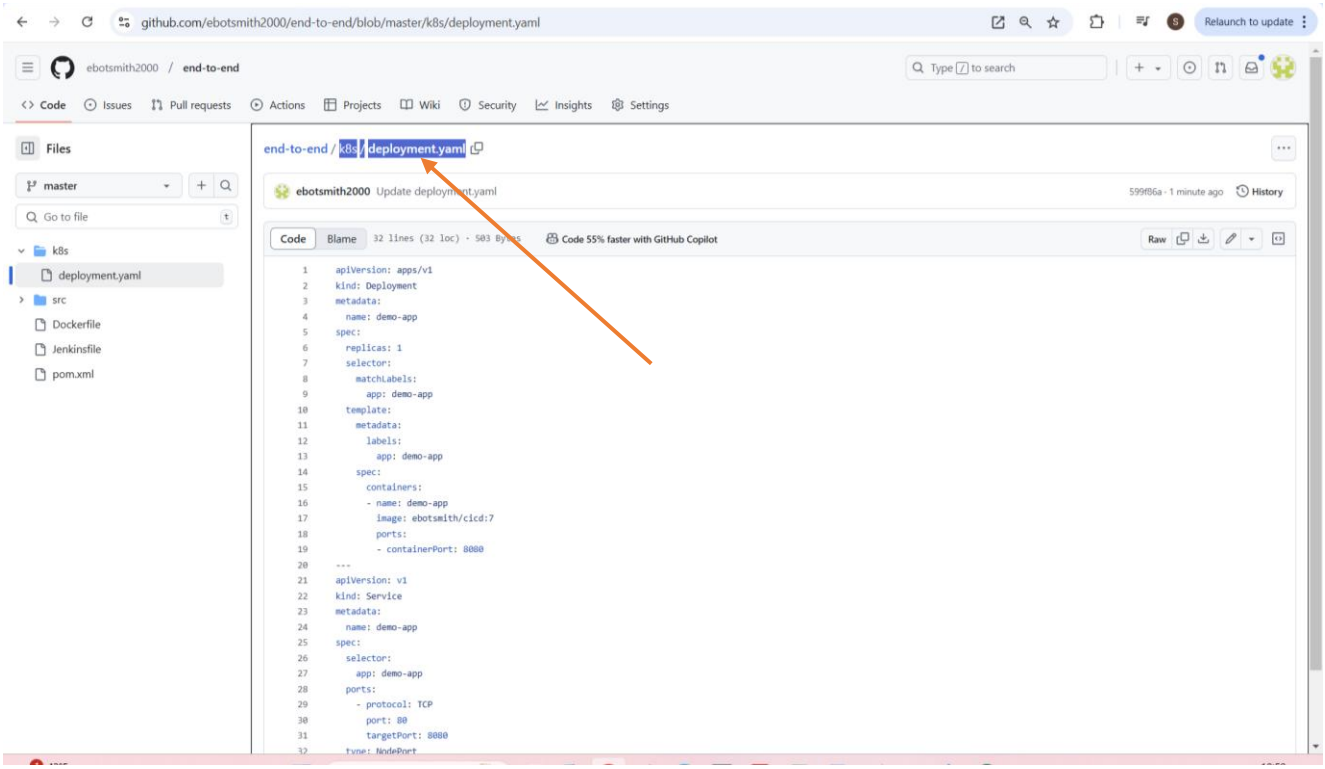
Paste it in the highlighted part of the "**deployment.yaml**" file above



Then click on "**Commit Changes**"



Click on "**Commit Changes**" again



Copy the highlighted part above: **k8s/deployment.yaml**

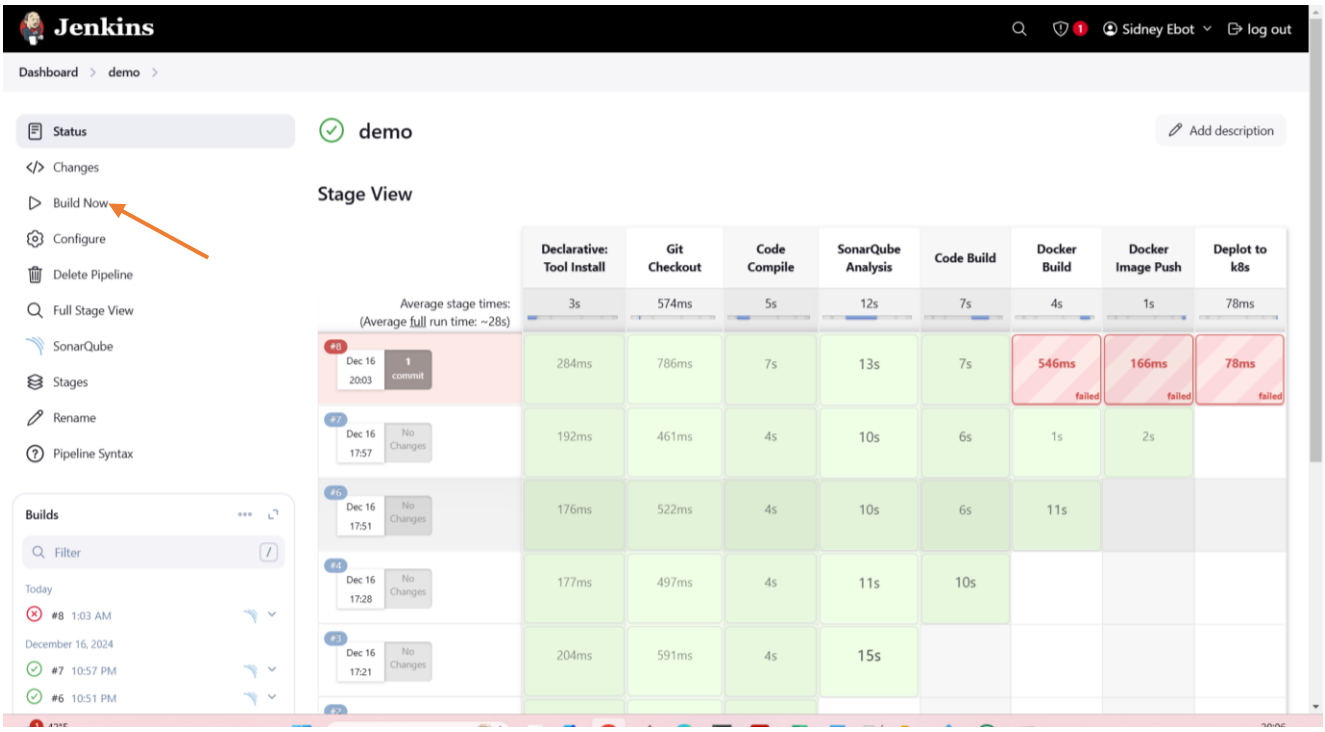
Then go back to Jenkins GUI to complete the modification of the Pipeline Script


```
pipeline {
    agent any
```

```
    tools{
        jdk 'jdk'
        maven 'maven3'
    }
    environment{
        SCANNER_HOME= tool 'sonar-scanner'
    }

    stages {
        stage('Git Checkout') {
            steps {
                git 'https://github.com/ebotsmith2000/jenkins-pipeline.git'
            }
        }
        stage('Code Compile') {
            steps {
                sh 'mvn clean compile'
            }
        }
        stage('SonarQube Analysis') {
            steps {
                withSonarQubeEnv('SonarQube'){
                    sh '''$SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=new \
                    -Dsonar.java.binaries=. \
                    -Dsonar.projectKey=new '''
                }
            }
        }
        stage('Code Build') {
            steps {
                sh 'mvn clean install'
            }
        }
        stage('Docker Build') {
            steps {
               script{
                   withDockerRegistry(credentialsId: '7e596026-5cee-4b1f-9d13-97efcaf07959', toolName: 'docker')
{
                       sh 'docker build -t demo .'
                   }

                }

            }
        }
        stage('Docker Image Push') {
            steps {
               script{
                   withDockerRegistry(credentialsId: '7e596026-5cee-4b1f-9d13-97efcaf07959', toolName: 'docker')
{
                       sh 'docker tag demo ebotsmith/demo:$BUILD_ID'
                       sh 'docker push ebotsmith/demo:$BUILD_ID'
                   }
                }

            }
        }
        stage('Deploy to k8s') {
            steps {
               script{
                   withKubeCredentials(kubectlCredentials: [[caCertificate: '', clusterName: '', contextName:
'', credentialsId: '10e17ebc-03e7-441a-b922-32530142f75f', namespace: '', serverUrl: '']]) {
                    sh 'kubectl apply -f k8s/deployment.yaml'
                   }
                }

            }
        }
    }
}
```

Then click on "**Save**"

Let us build the pipeline again by clicking on “**Build Now**”



The build has failed. Let us go to the “Console Output”. Below is the warning message:

```
Warning: failed to get default registry endpoint from daemon (Cannot connect to
the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?).
Using system default: https://index.docker.io/v1/

Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker
daemon running?
```

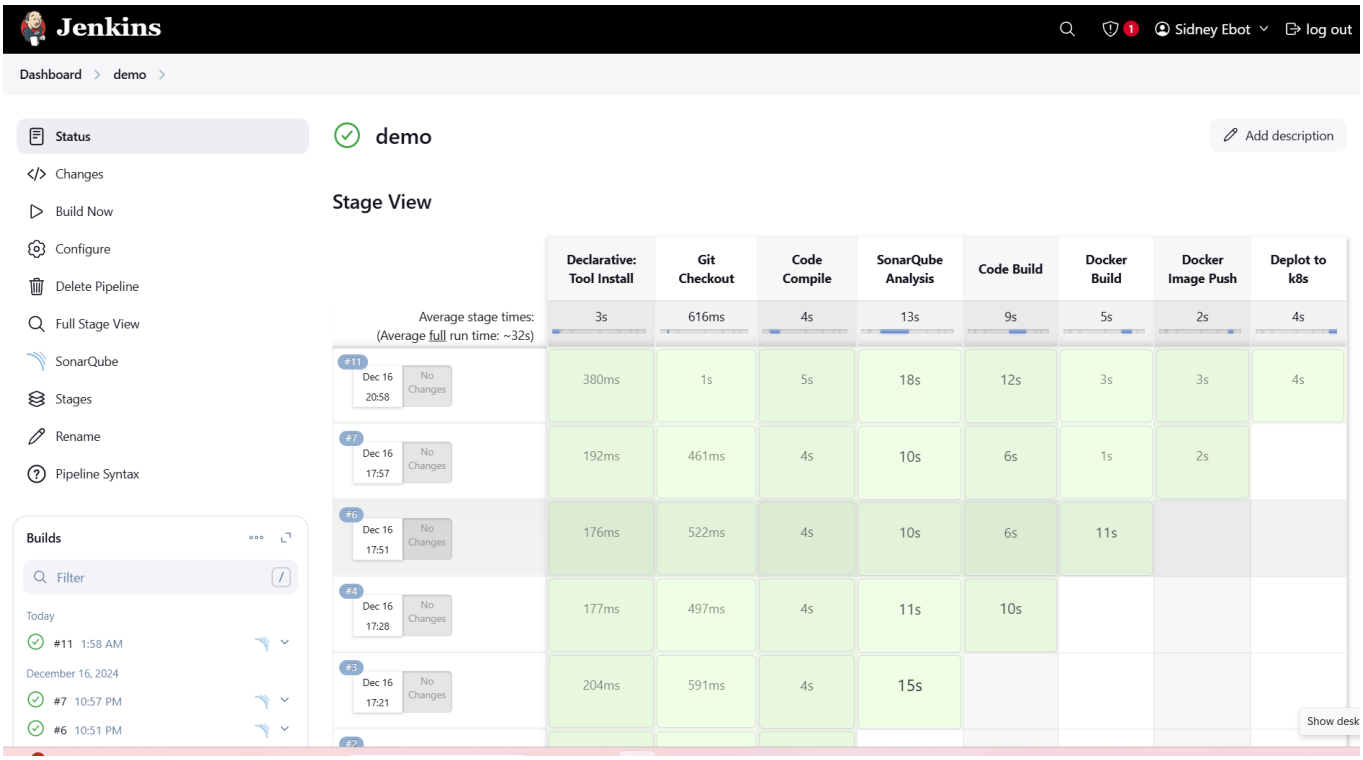
It looks like docker is having issues.

SOLUTION

To resolve this

- reboot the EC2 instance.
- Reinstall SonarQube
- Reinstall Docker

The build again



And the build is successful

Then, navigate to the kube folder using the below commands and check of the kube pod is available.

Run the command: **sudo su**

Followed by the command: **cd ~**

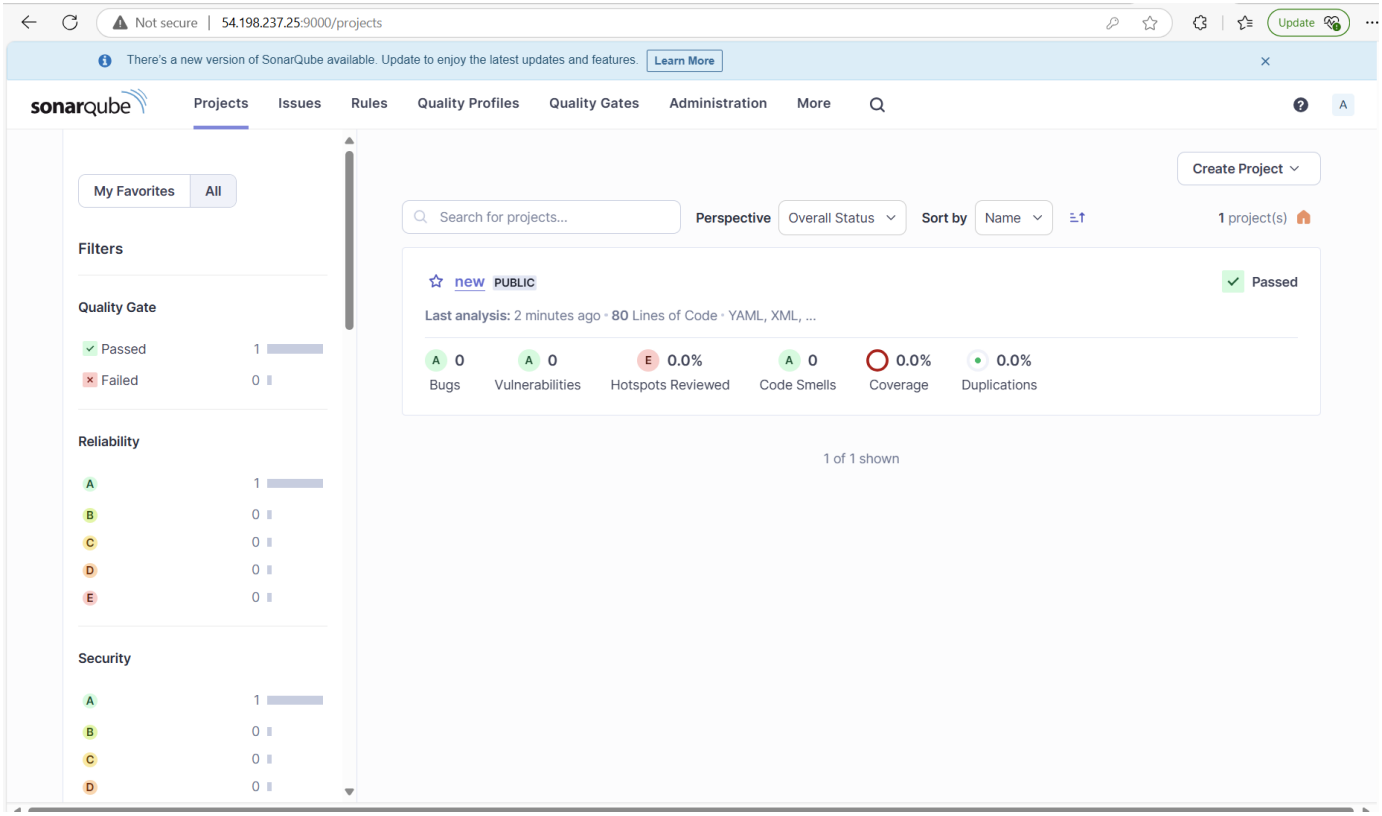
Then the command: **cd .kube/**

Now, run the command to get the pods:

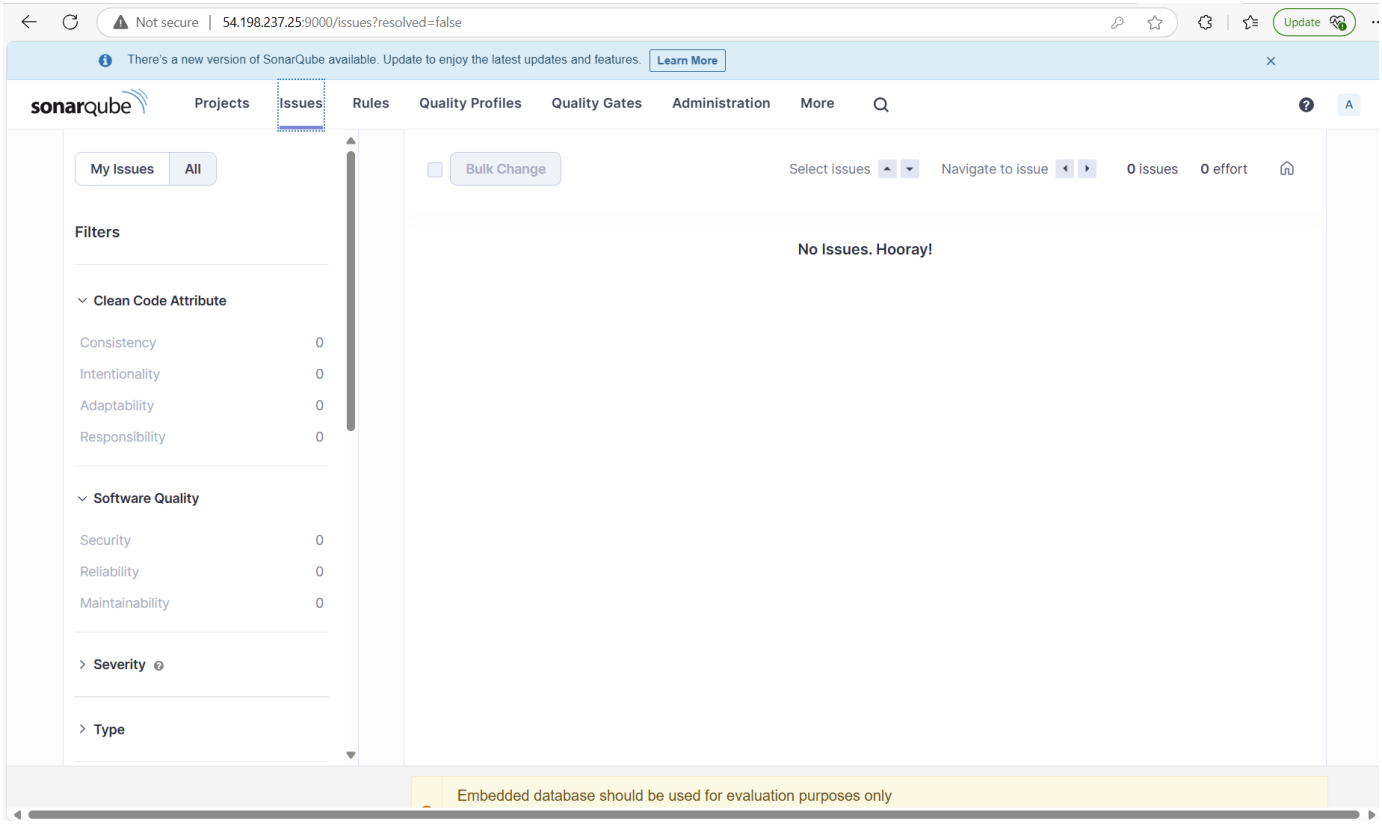
kubectl get pods

```
ubuntu@ip-172-31-44-137:~$ sudo su
root@ip-172-31-44-137:/home/ubuntu# cd ~
root@ip-172-31-44-137:~# cd .kube/
root@ip-172-31-44-137:~/.kube# kubectl get pods
NAME                          READY   STATUS             RESTARTS      AGE
demo-app-6c857b5989-m7nrc     0/1     CrashLoopBackOff   7 (88s ago)   12m
root@ip-172-31-44-137:~/.kube#
```

And check the code analysis on SonarQube.



Everything passed



You can see there is no issue

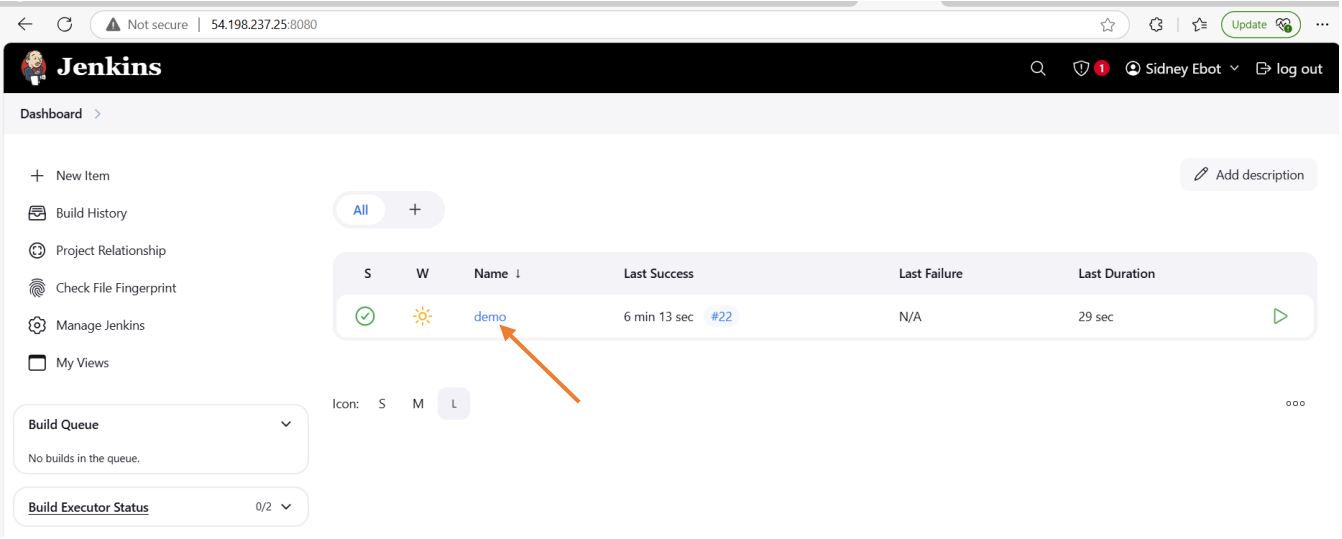
STEP 16: Automating the process

The final step is to automate the pipeline so that whenever any change is made in the GitHub repository and the changes are committed. The Jenkins pipeline is triggers to run automatically.

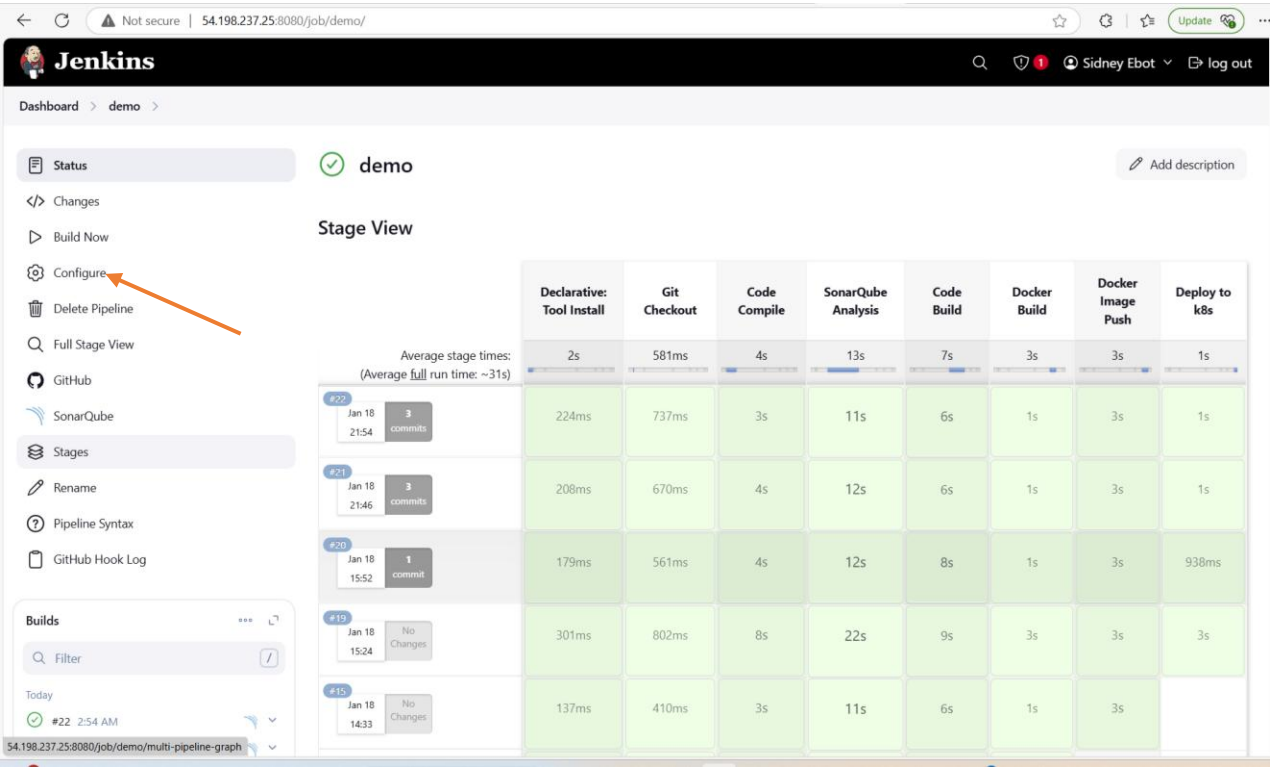
To achieve this, we have to make to changes. One in Jenkins and the other in GitHub.

PART 1: Modification on Jenkins

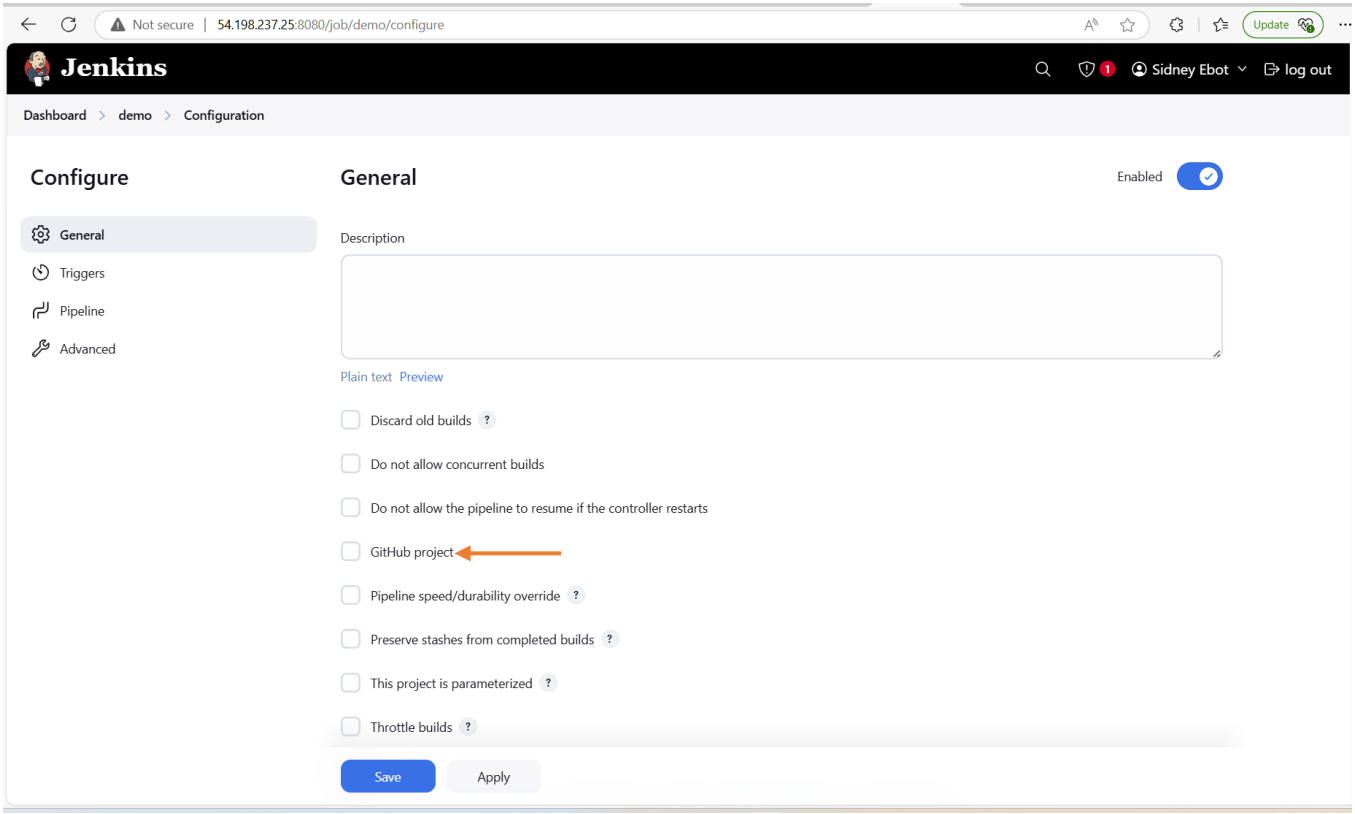
Go to Jenkins Dashboard



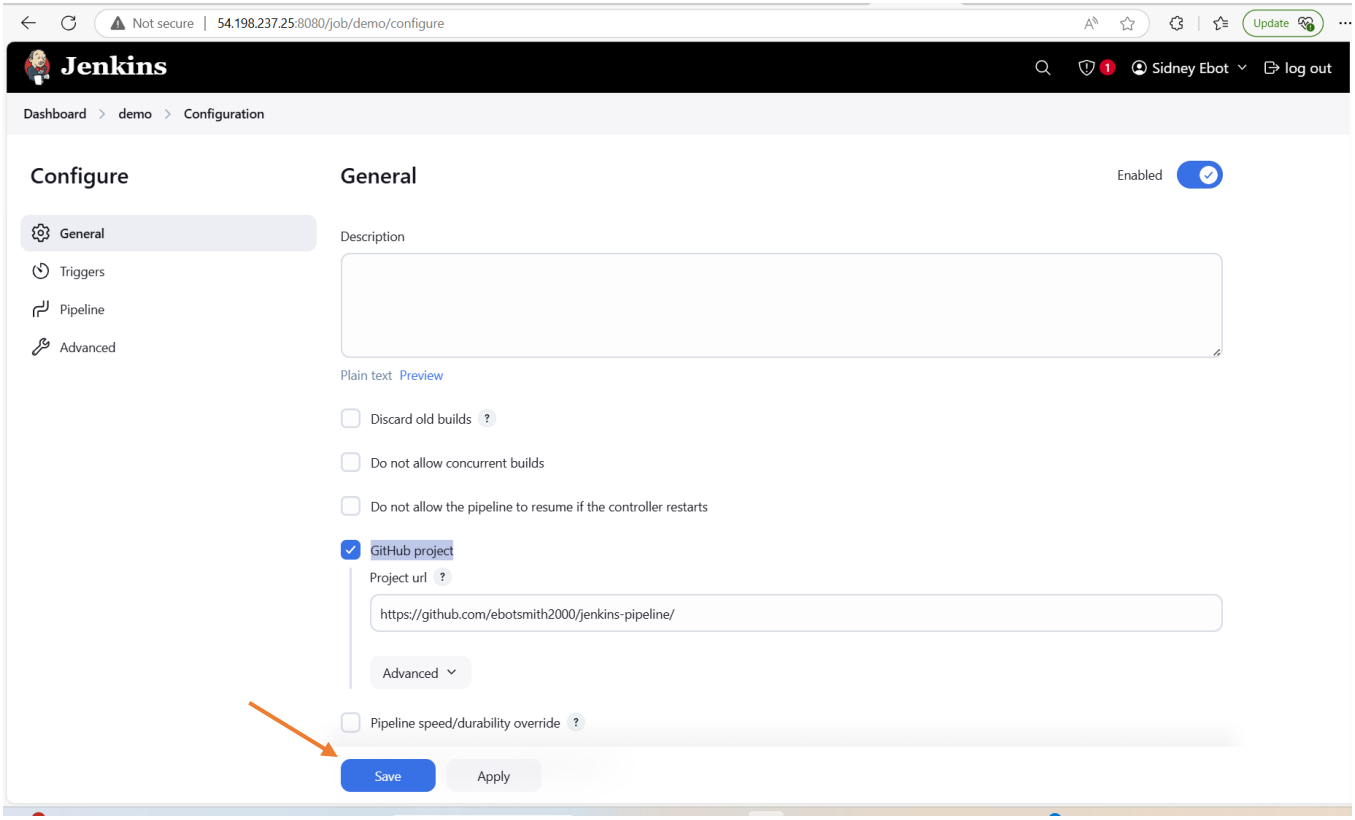
Click on the Pipeline "**demo**"



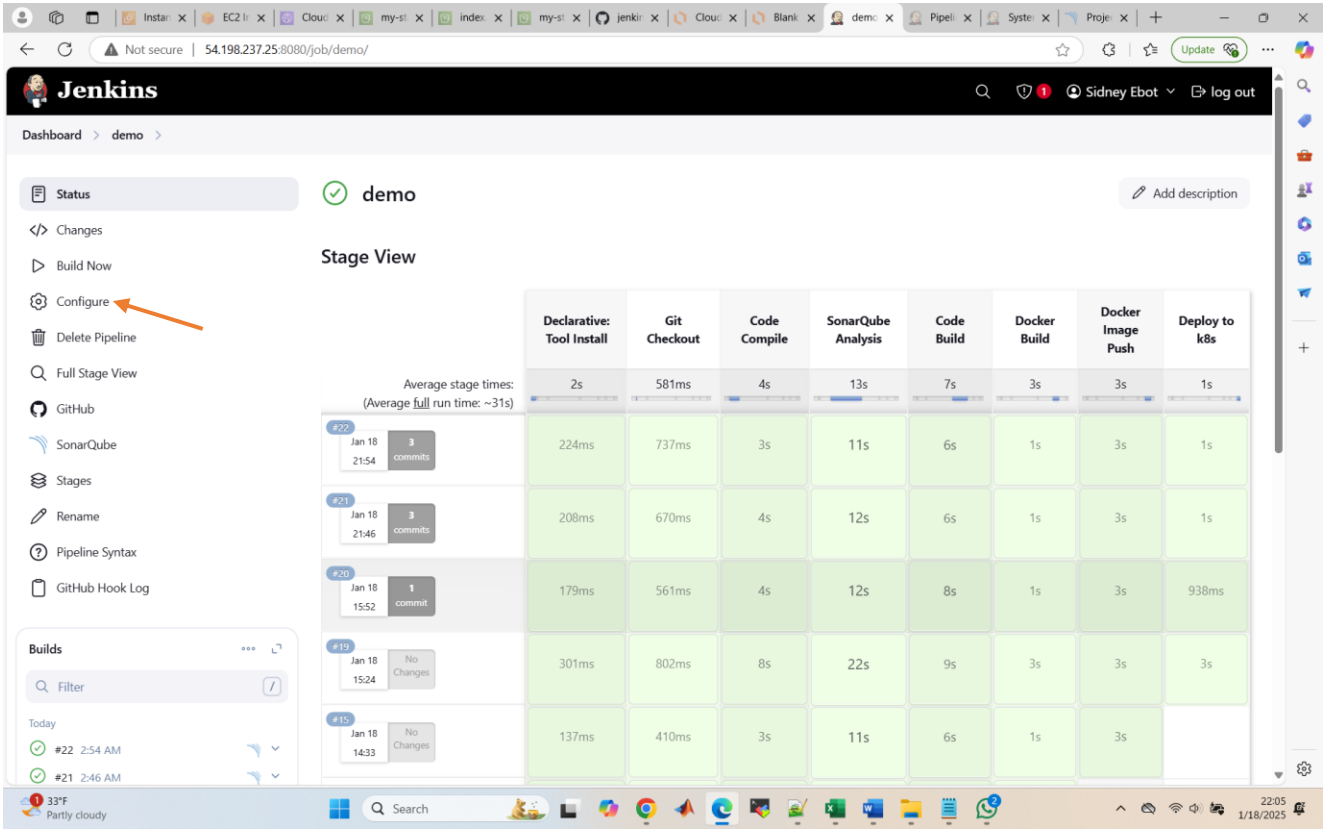
Click on “Configure”



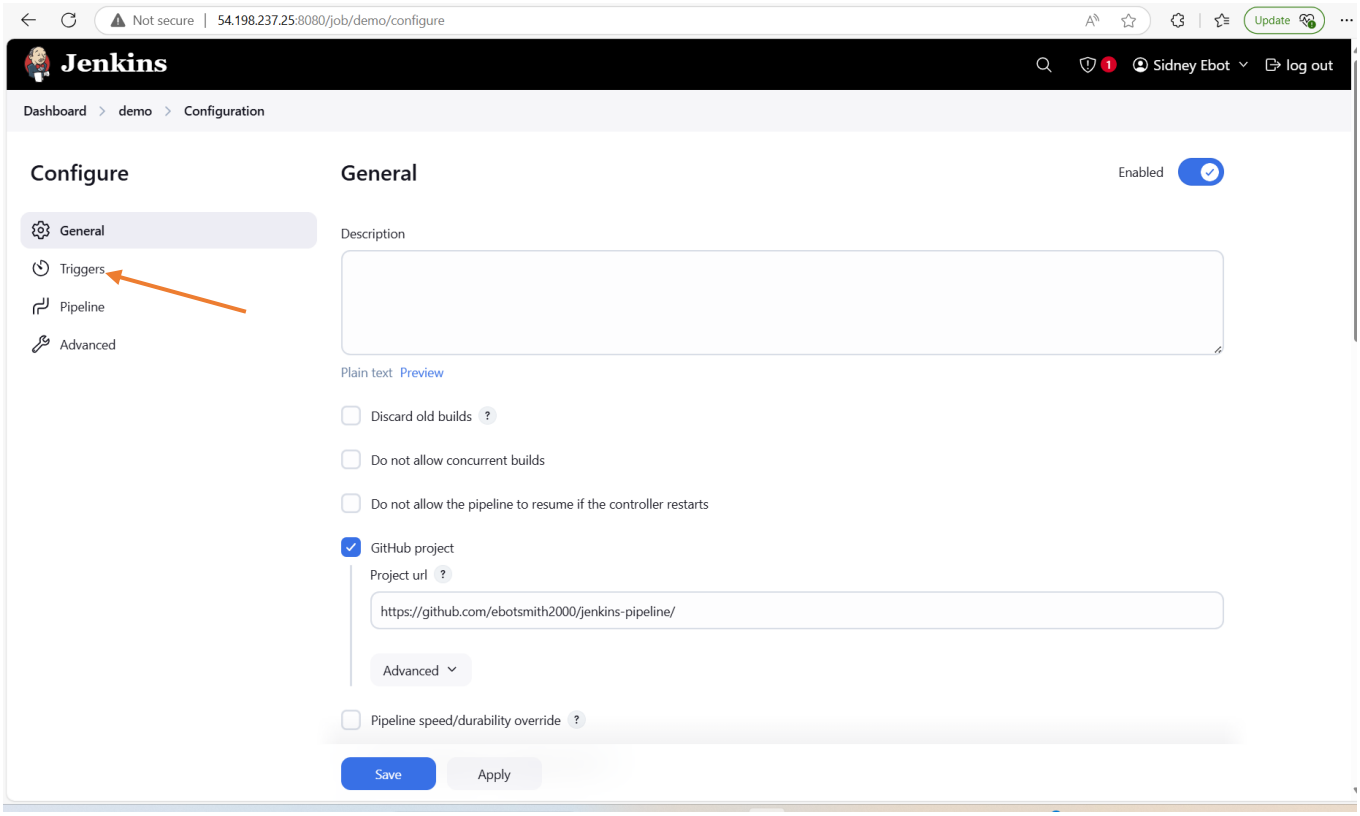
Under **“General”**, select **“GitHub project”** and enter the URL of the GitHub repository



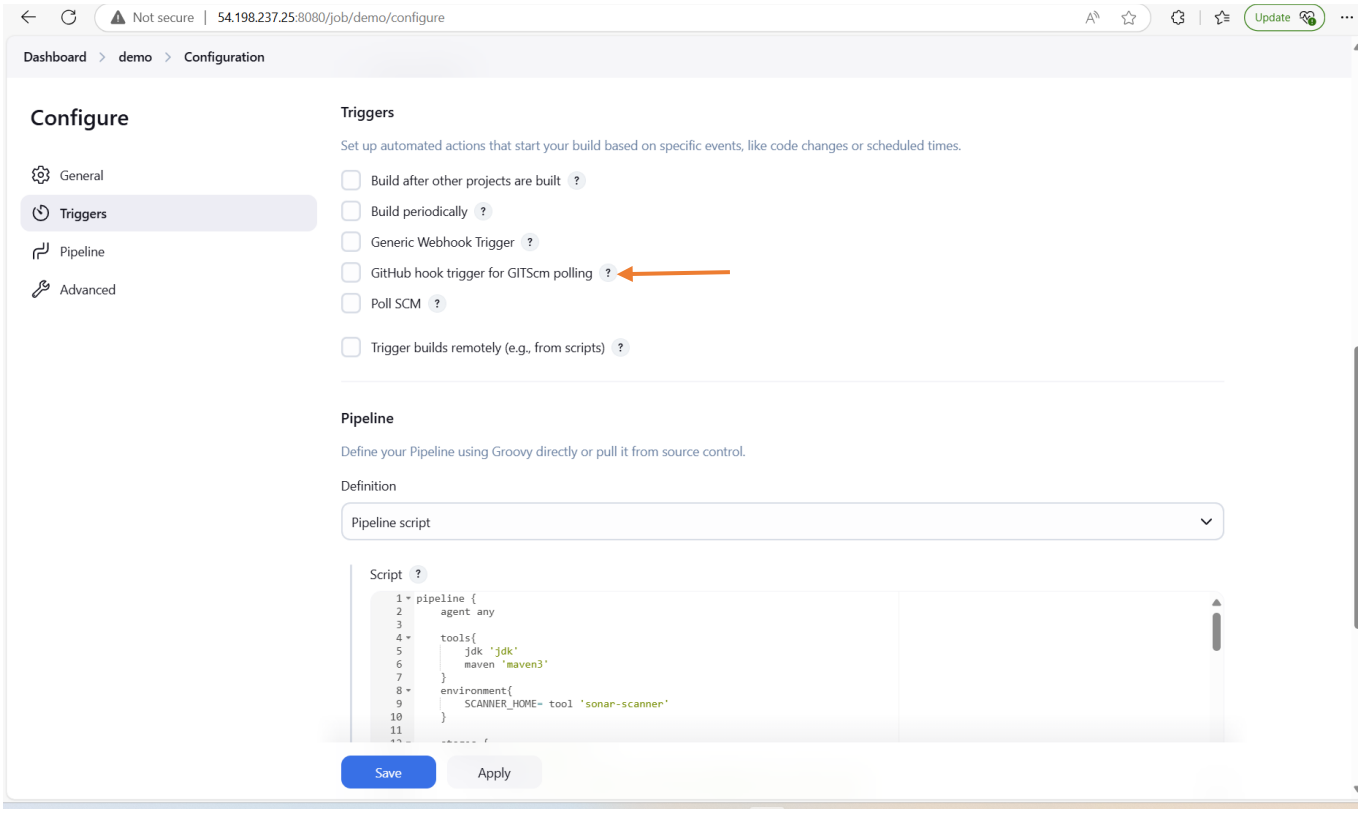
Click on **“Save”**



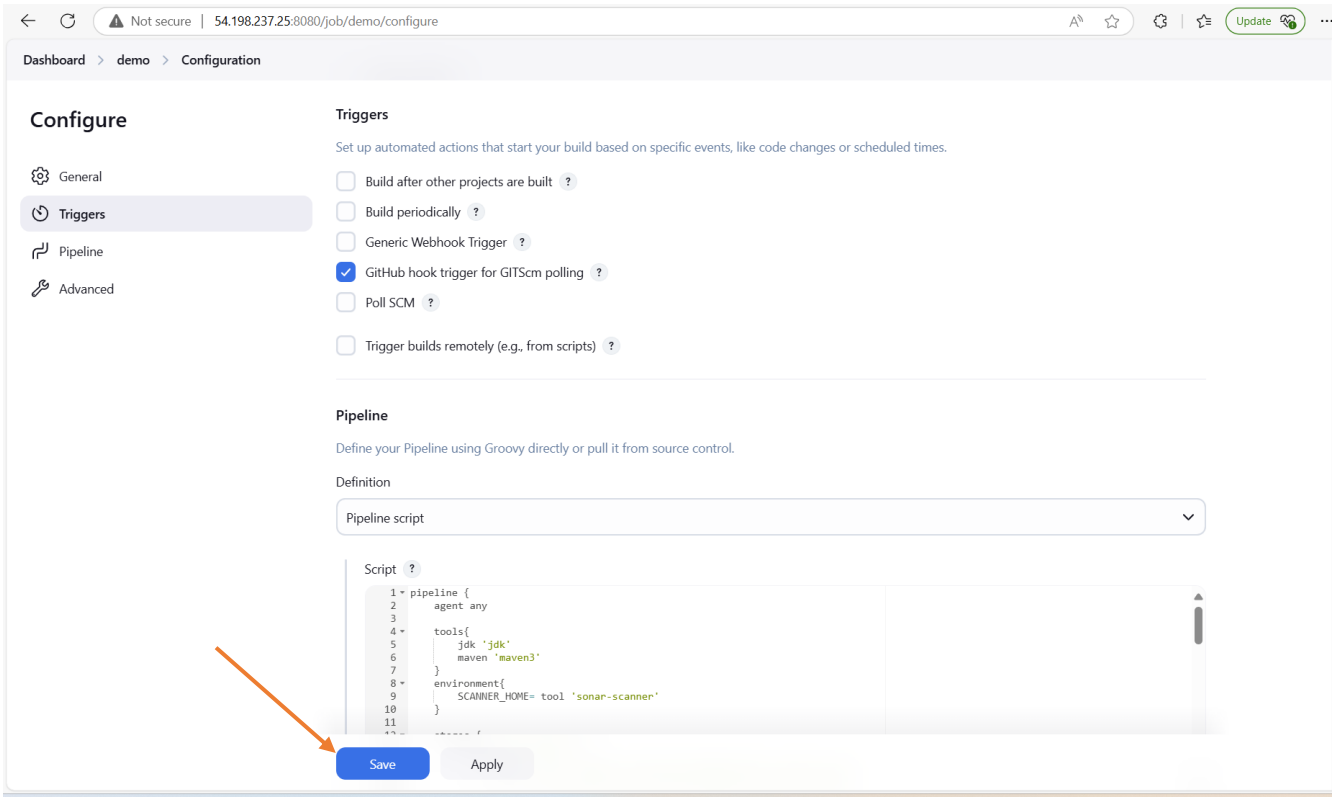
Now, we are going to modify the trigger. Click on "**Configure**" again



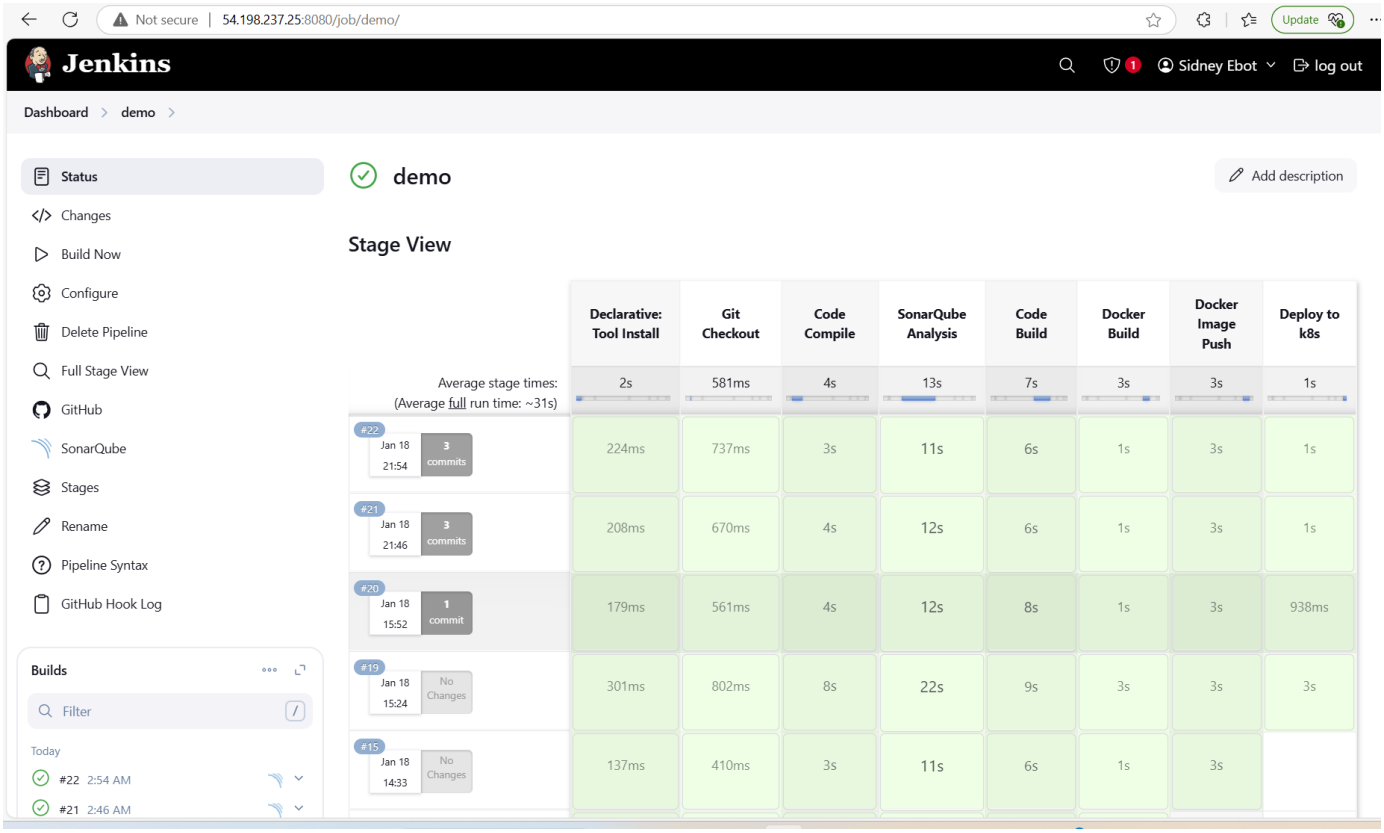
Click on "**Triggers**" on the left-hand side



Under “**Triggers**”, select “**GitHub hook trigger for GITScm polling**”

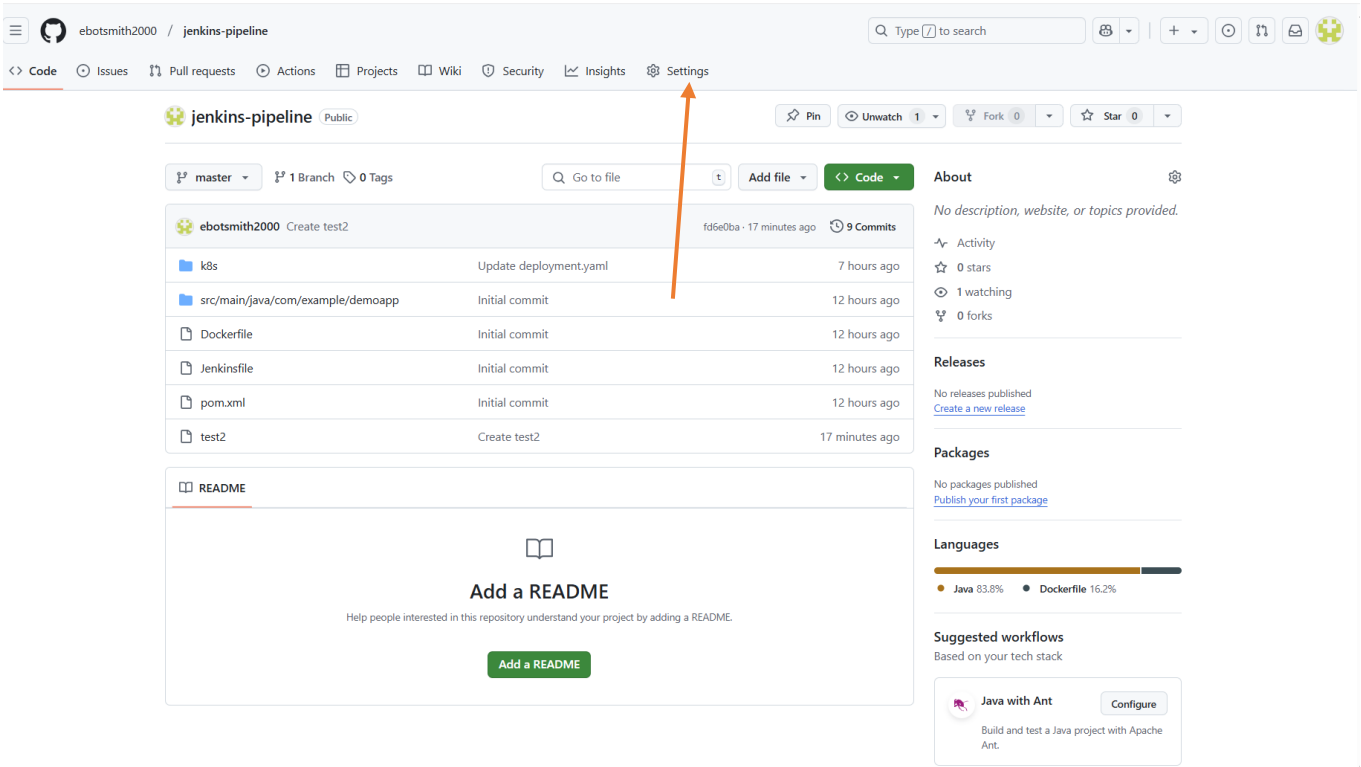


Click on “**Save**”

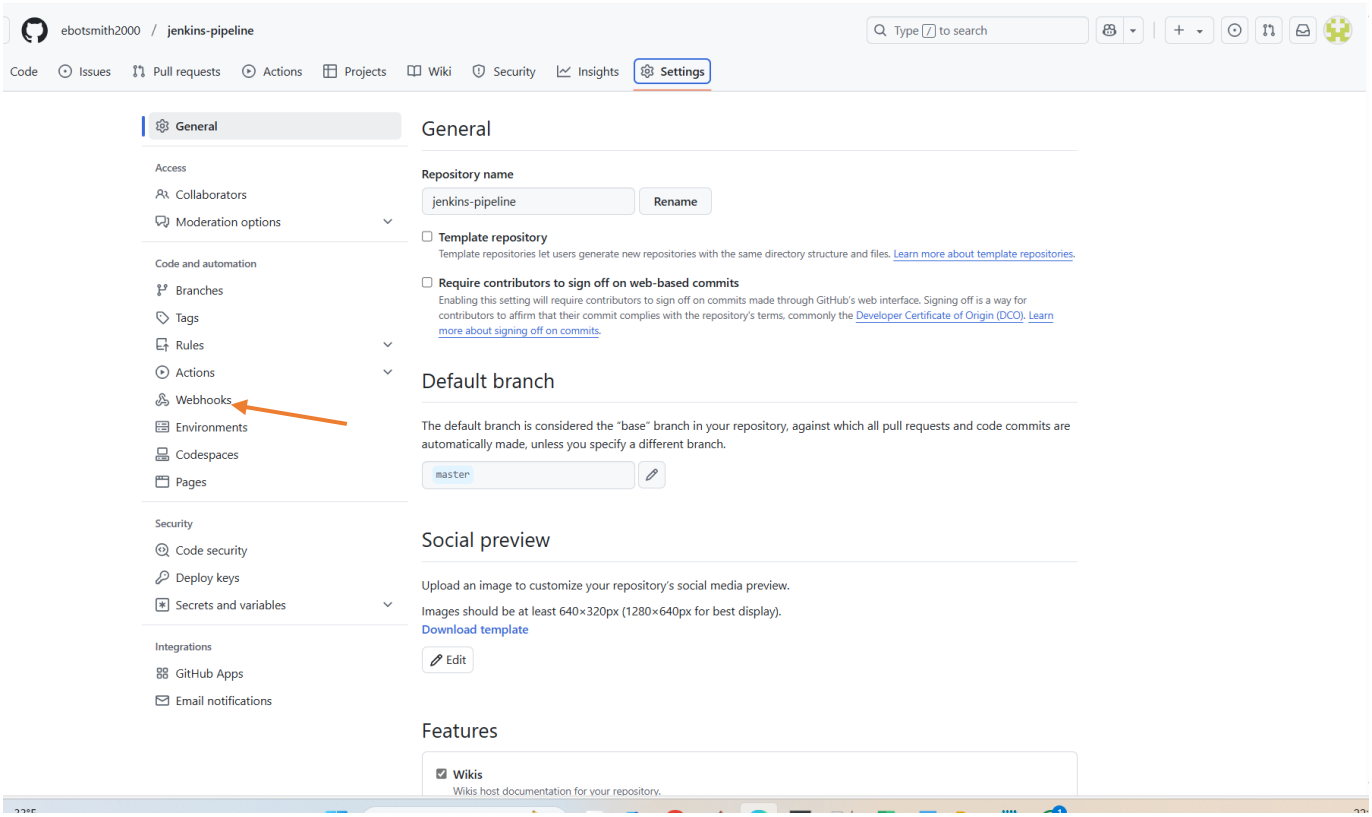


PART 2: Modification on GitHub Repository

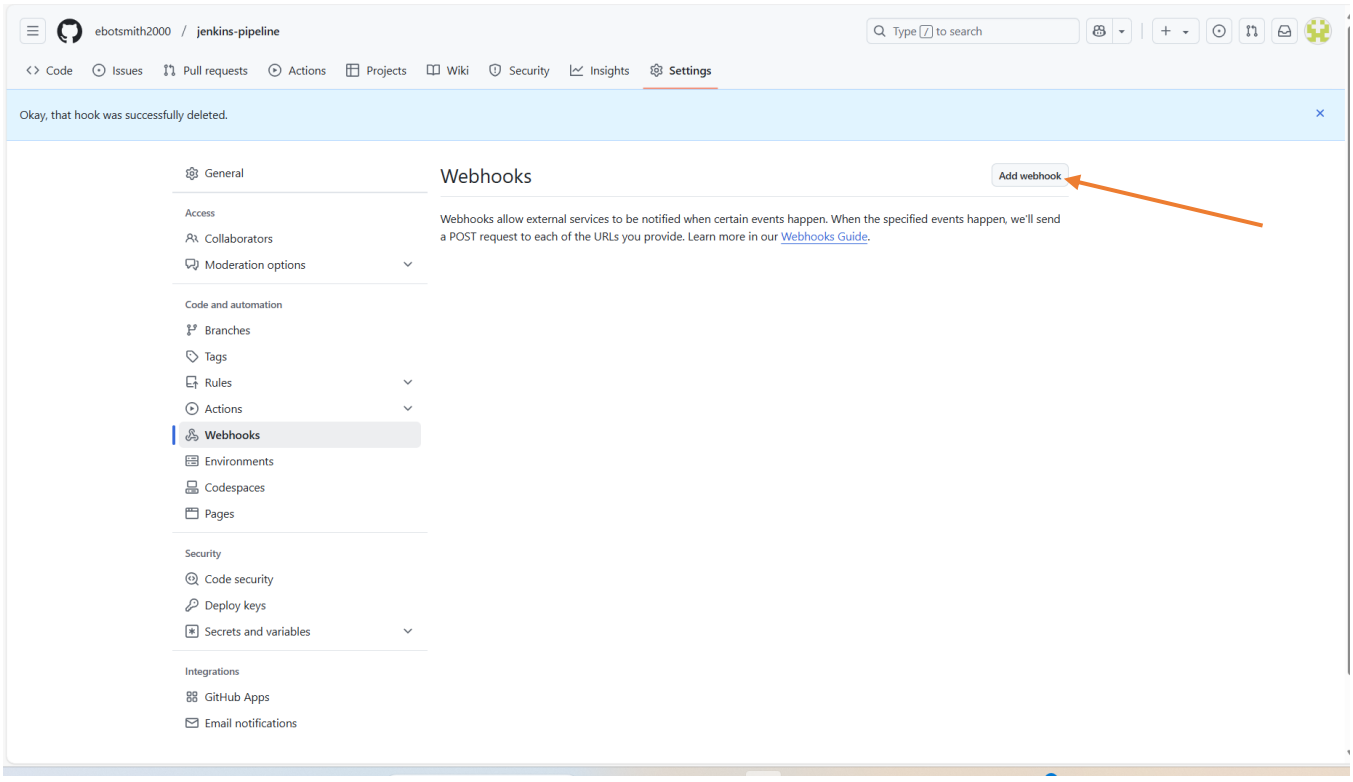
Now, we have to modify the GitHub repository. Go to the GitHub repository



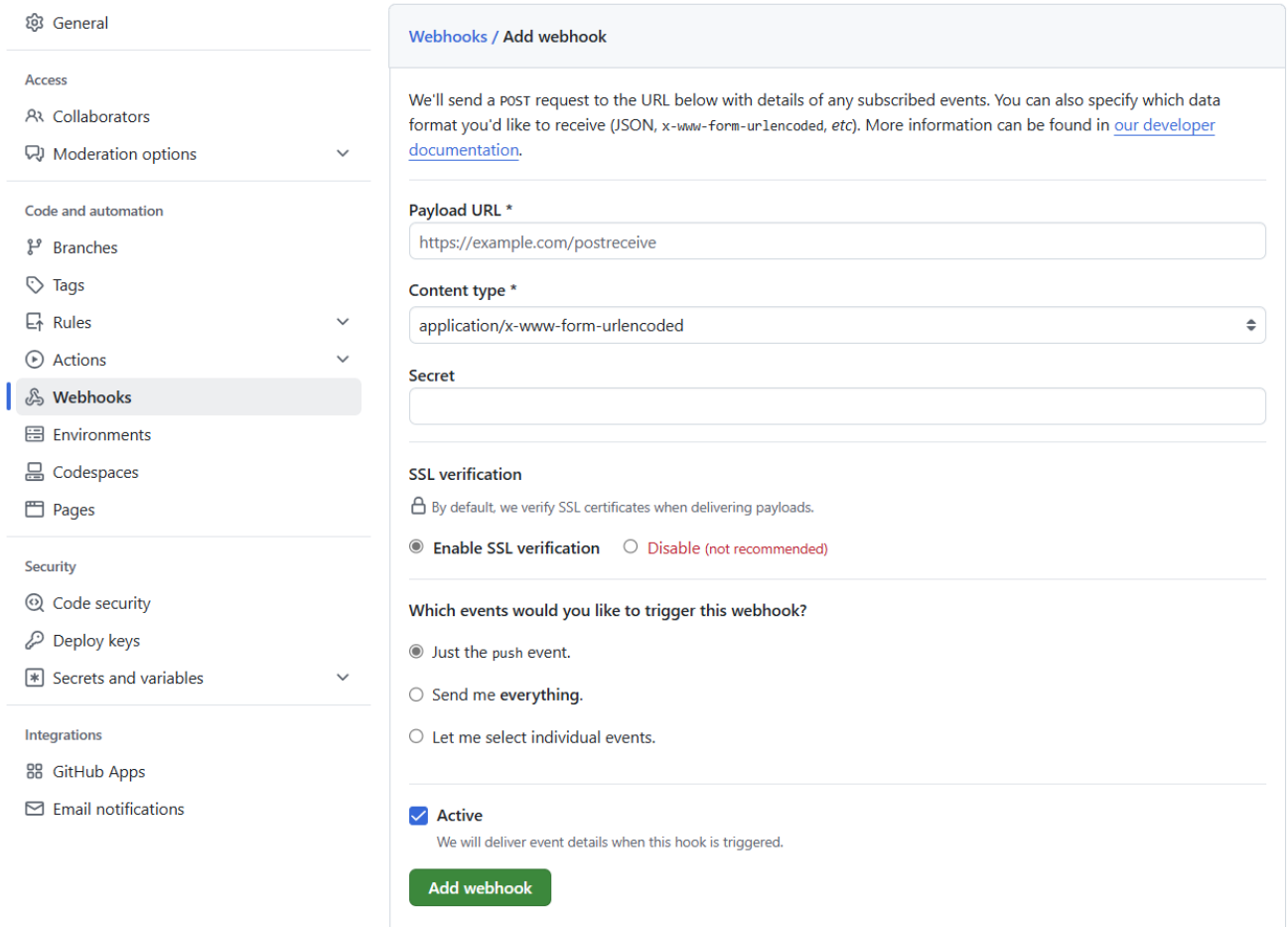
Click on "**Settings**"



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



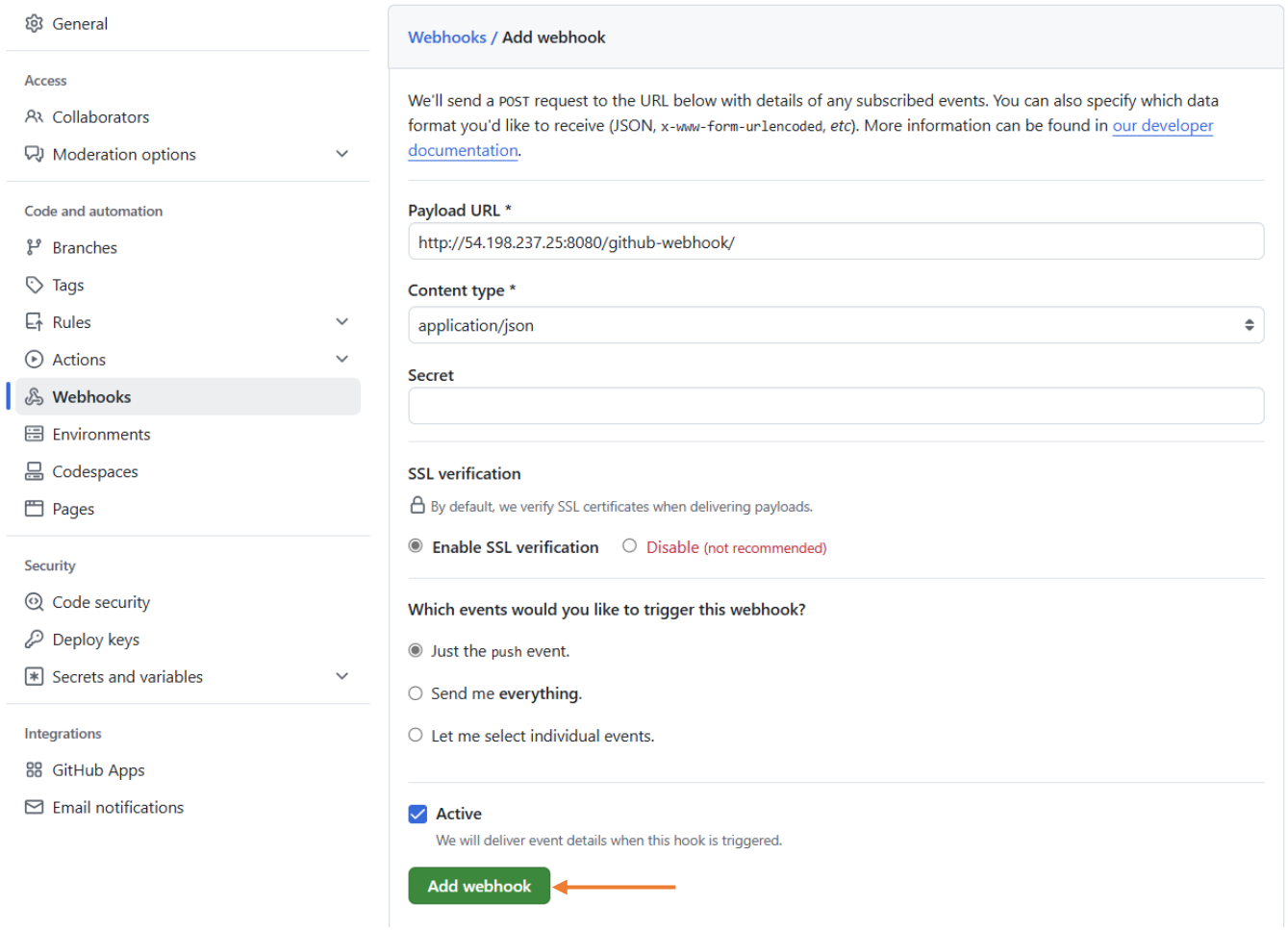
Click on "**Add Webhook**"



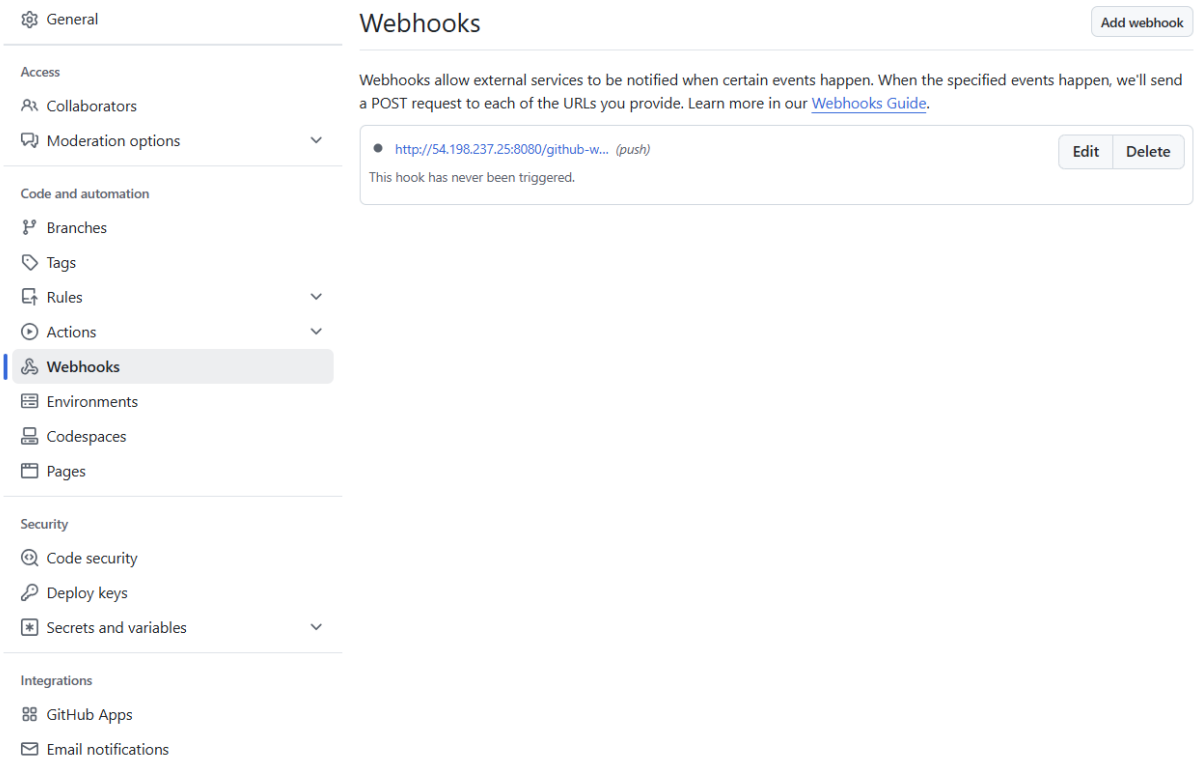
On **Payload URL**: Enter the Jenkins URL/github-webhook/

That is http://54.198.237.25:8080/github-webhook/

On **Content Type**: Select **application/json**



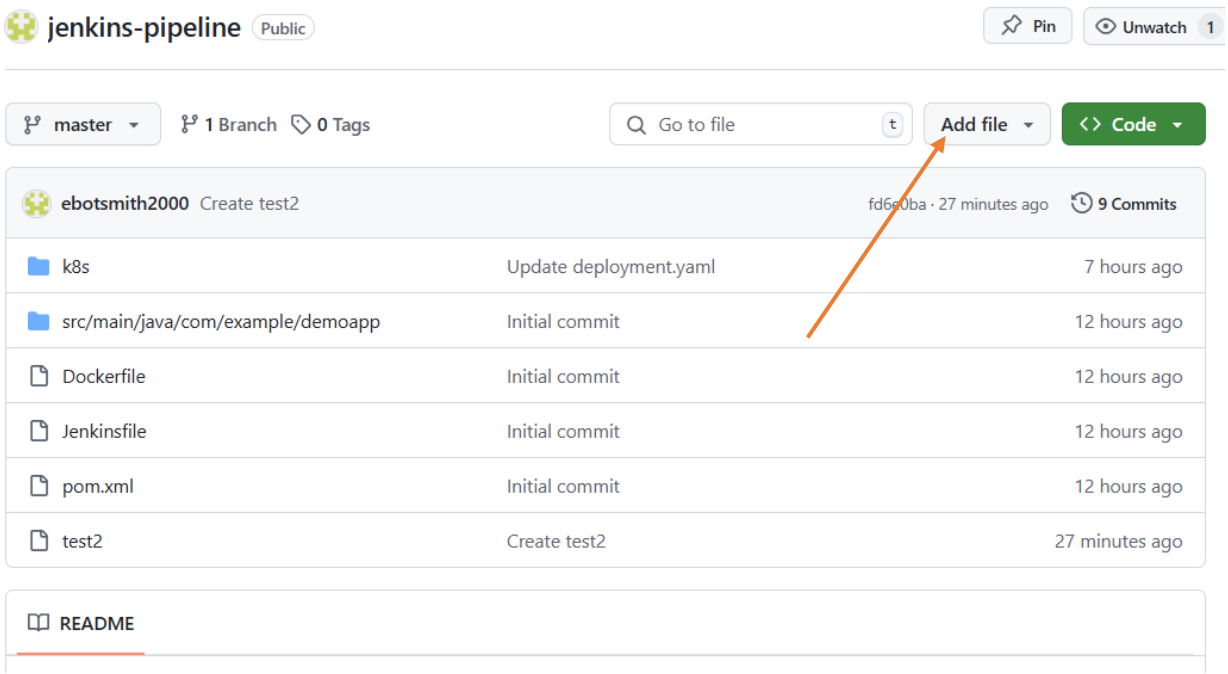
Click on "**Add webhook**"



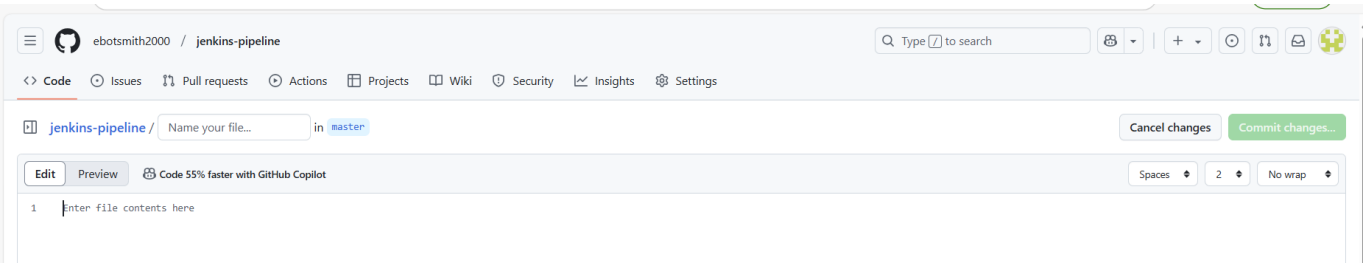
Now, let us test if the pipeline has been automated.

TESTING

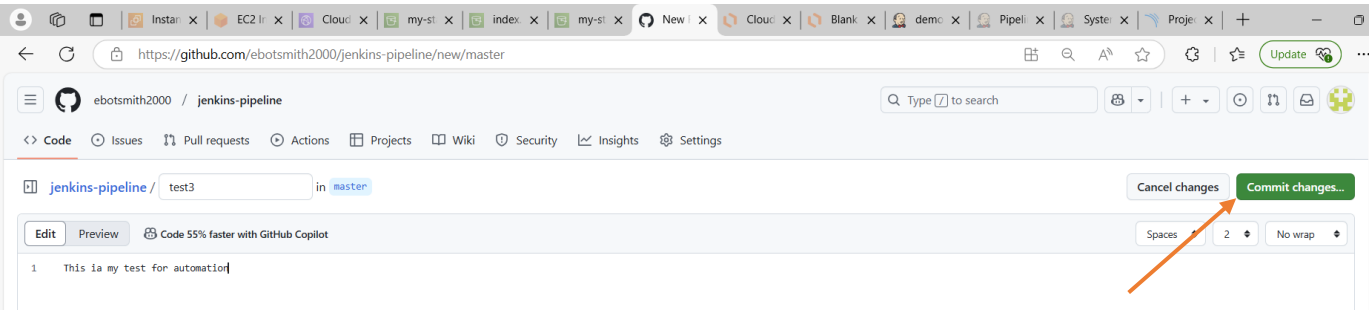
Let me add a new text file called "**test3.txt**" to our GitHub repository. Then commit to see of the pipeline will be triggered automatically.



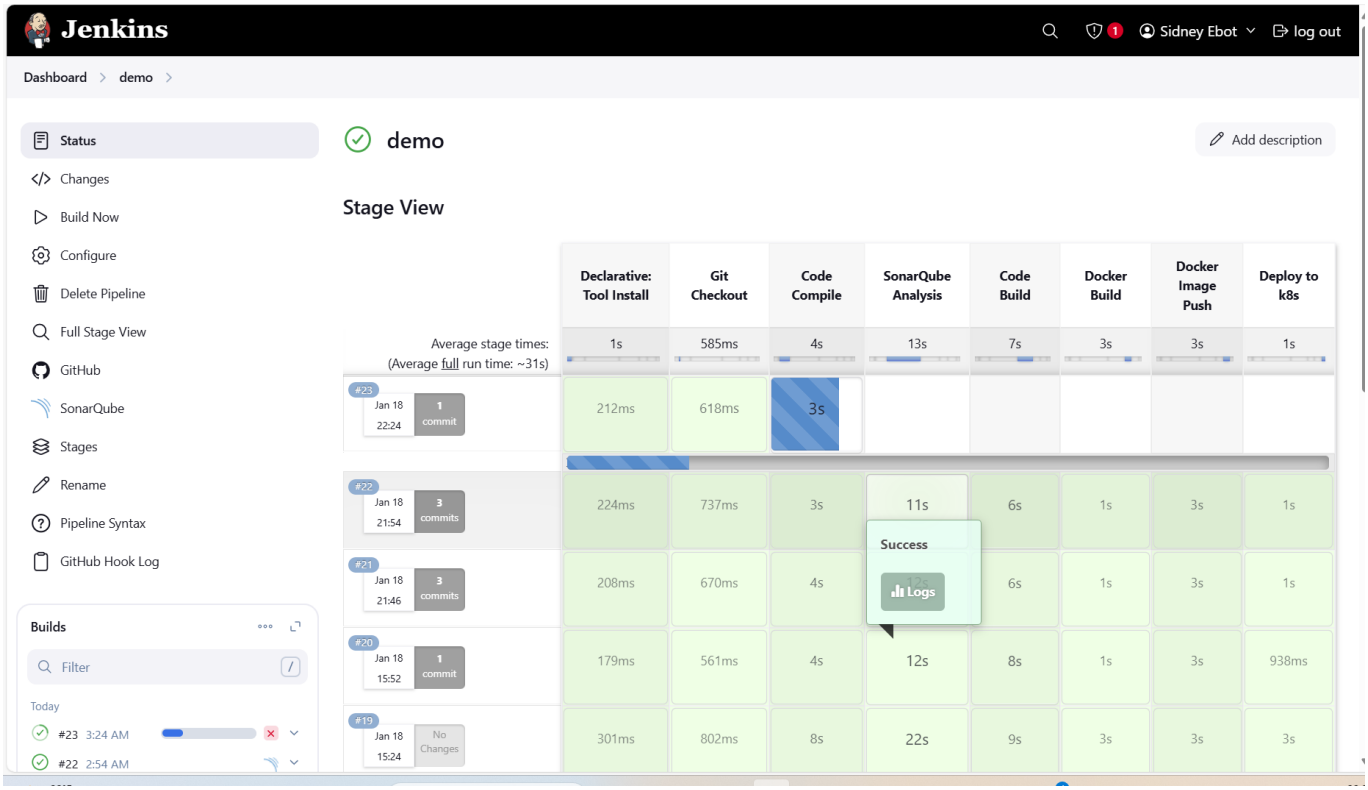
Click on "**Add file**" and select "**create a new file**"



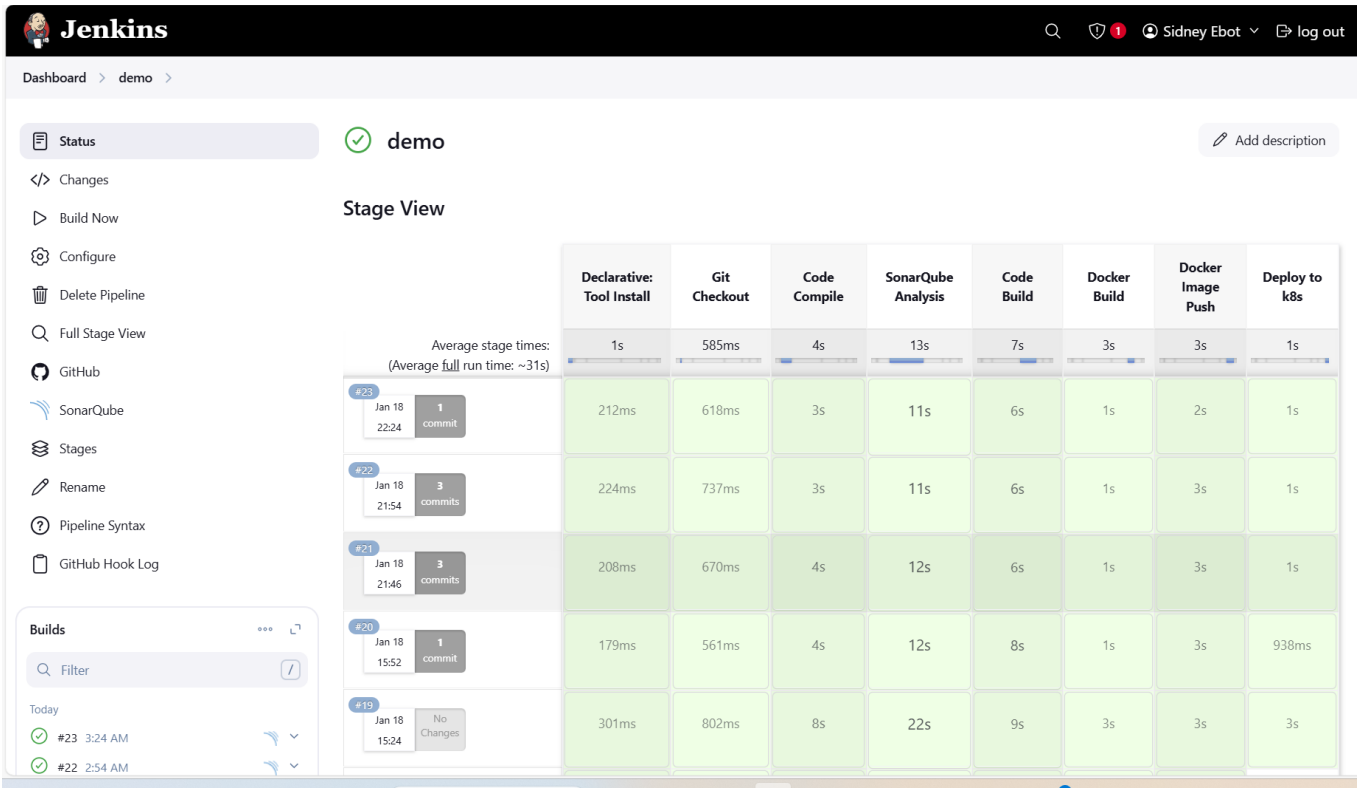
Let us add the file called "**test3**"



Click on "**Commit Changes**" and we head back to our Jenkins GUI



You can see that the Jenkins pipeline has started running automatically



You can see that the pipeline is successful. So, we have made an automated CI/CD pipeline on Jenkins