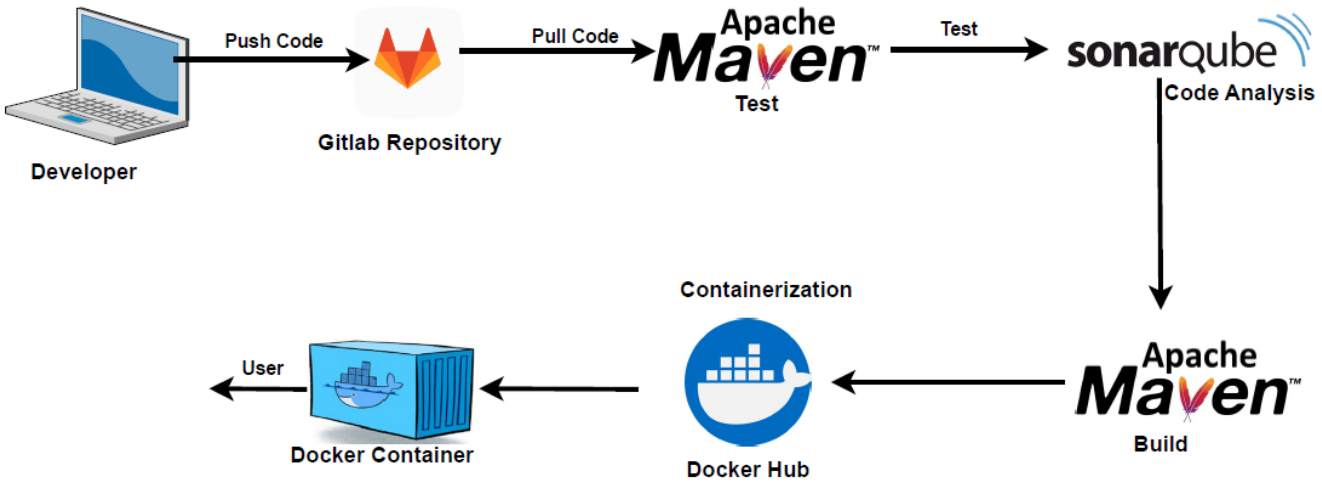
Prepared by Sidney Smith

Deploying Java Application using GitLab Pipeline Project with Maven, SonarQube, Docker

In this document, I will demonstrate how to deploy a Java-based application using Gitlab Pipeline. I will use GitLab Runner (Project Runner) on AWS with Docker Executor, SonarQube server on AWS and Docker.

At the end, you will learn how to build a full GitLab-CI pipeline for Java-based applications, with a focus on DevSecOps practices.

Architecture



The developer writes a Java-based code and pushes it to the Gitlab repository that will trigger the Pipeline. The code will be tested with Maven. Next the code analysis is done with SonarQube. Then the analyzed code is build using Maven. Finally, the image is built and containerized using Docker.

IMPLEMENTATION

The pipeline includes four key jobs:

1. Test Job for the Java application using Maven
2. Code Quality Check Job using SonarQube
3. Build and Package the Java Application Job using Maven
4. Containerize the Application and Push Image to Docker Hub Job using Docker

We will also walk through setting up the infrastructure, including:

- **GitLab Runner (Project Runner) on AWS** with Docker Executor
- **SonarQube server on AWS** using Docker for Java code quality analysis

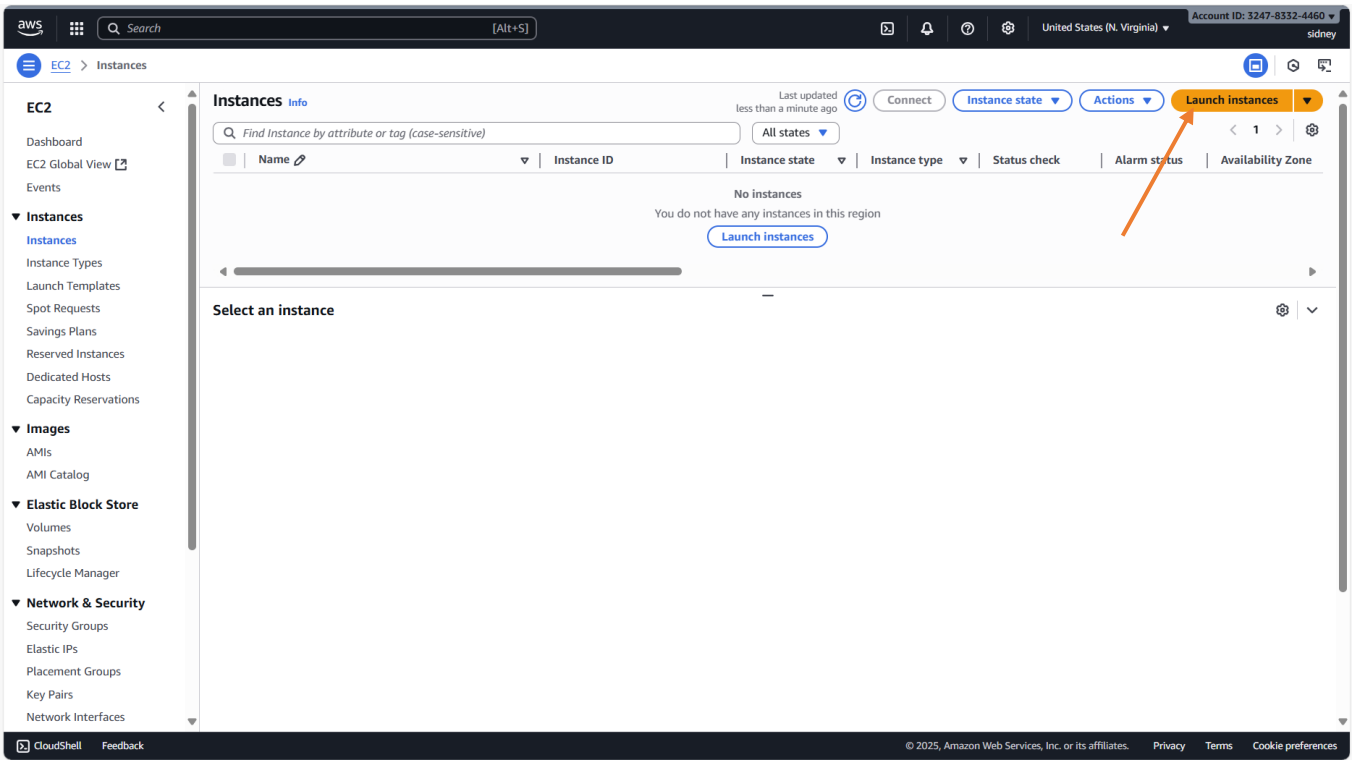
You will learn how to write pipeline scripts for each of these jobs. The Test and Build jobs for the Java app will run on the Project Runner, while the SonarQube code quality check and Containerize job will run on GitLab's Shared Runner.

STEP 1: Create an Ubuntu EC2 instance and Connect to the Instance

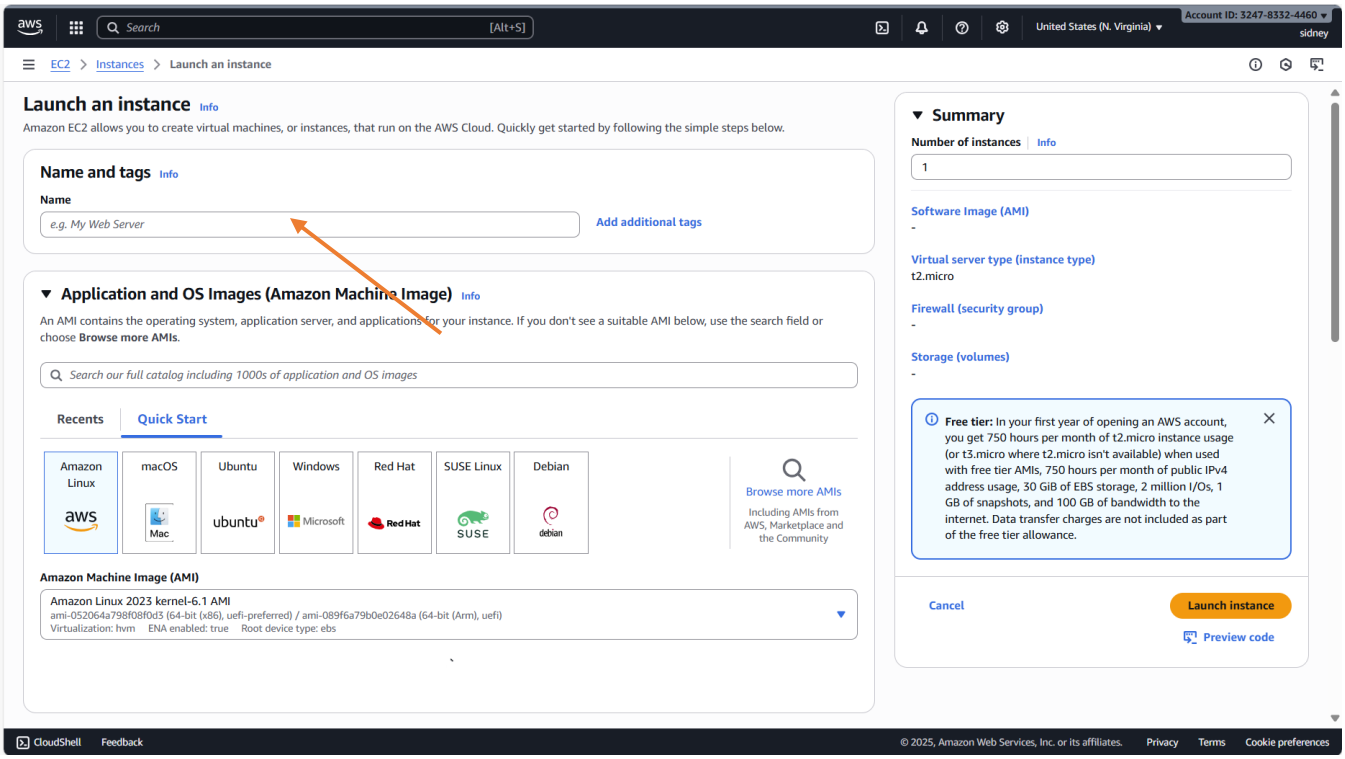
We have to create an Ubuntu EC2 instance, namely Gitlab-Server, add ports 22, 80, 43, and 9000 for SSH, HTTP, HTTPS and SonarQube respectively to the instances. Then SSH connect to the Ubuntu EC2 instance.

Part 1: Create an ubuntu EC2 instance

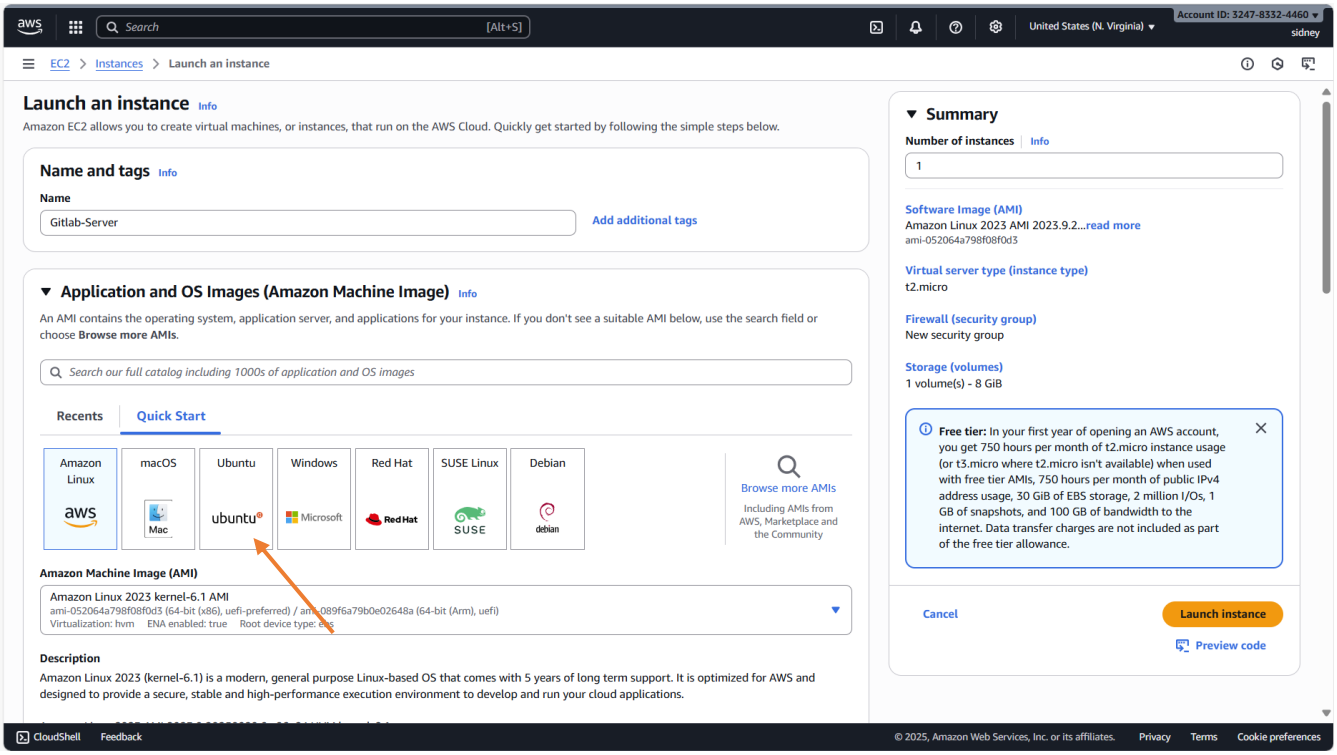
Go to EC2 dashboard



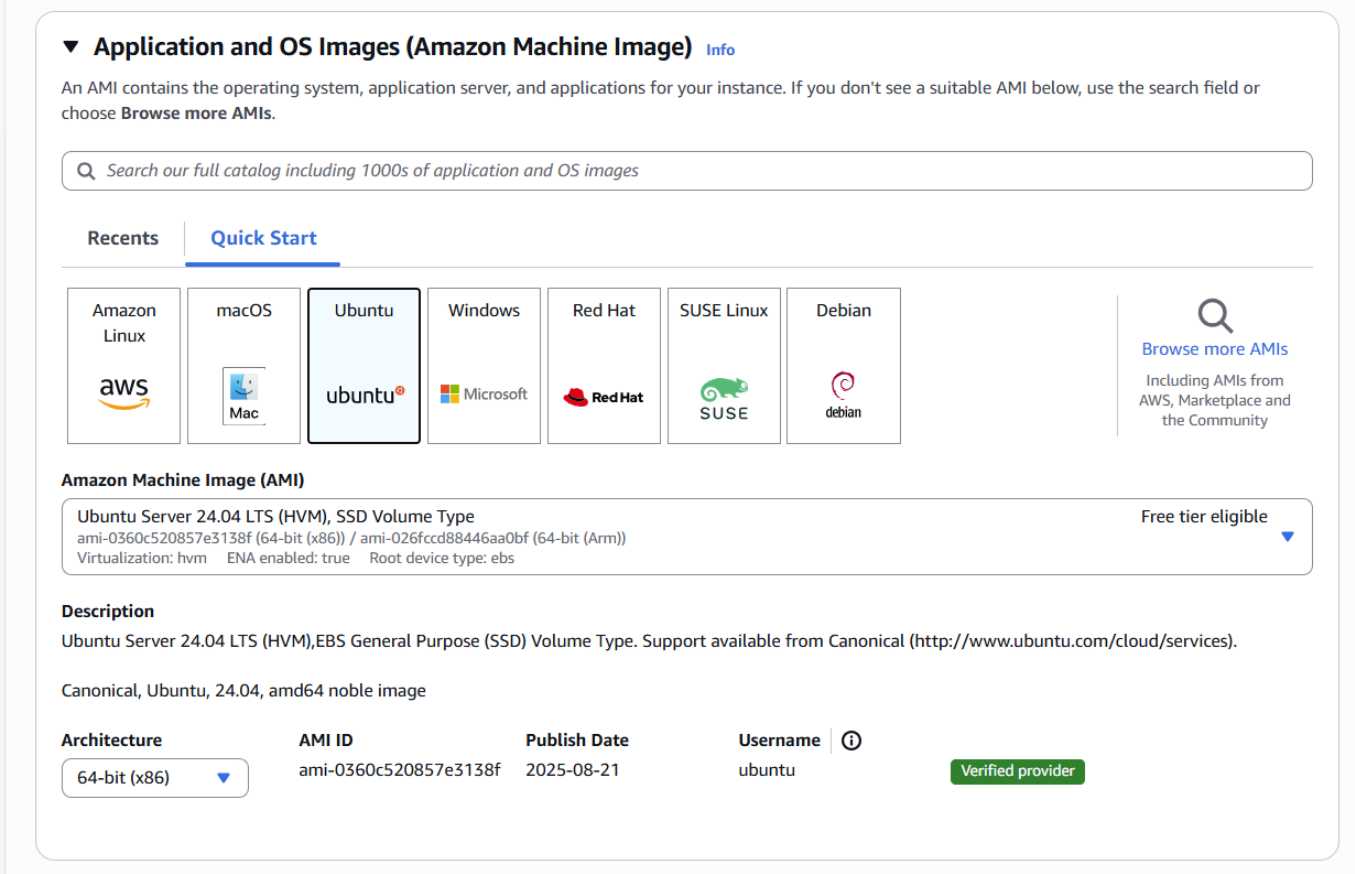
Click on "**Launch Instance**"



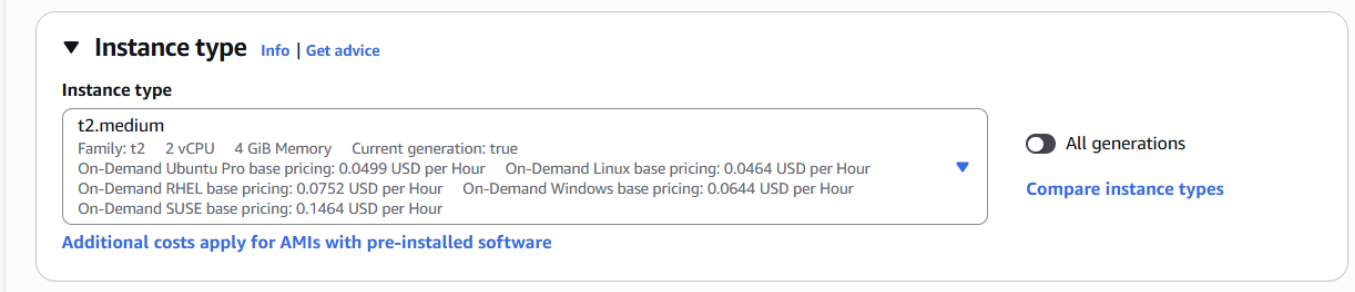
Give the instance the name "**Gitlab-Server**"



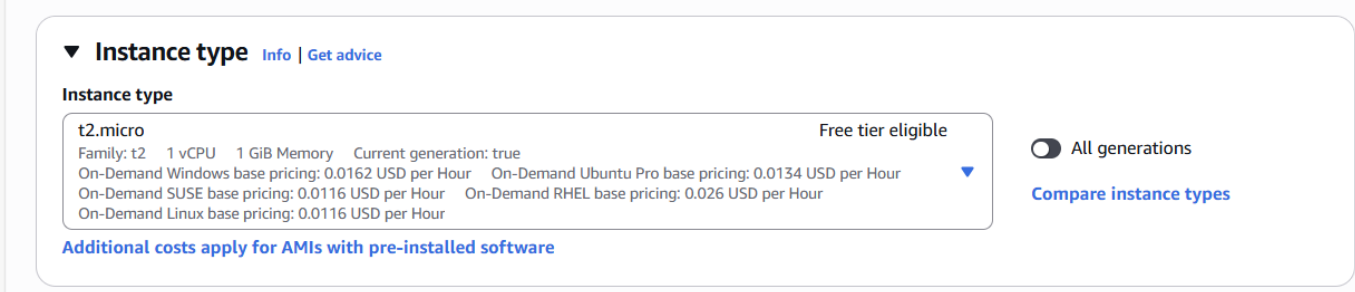
On "**Application and OS Images (Amazon Machine Image)**", select "**Ubuntu**"



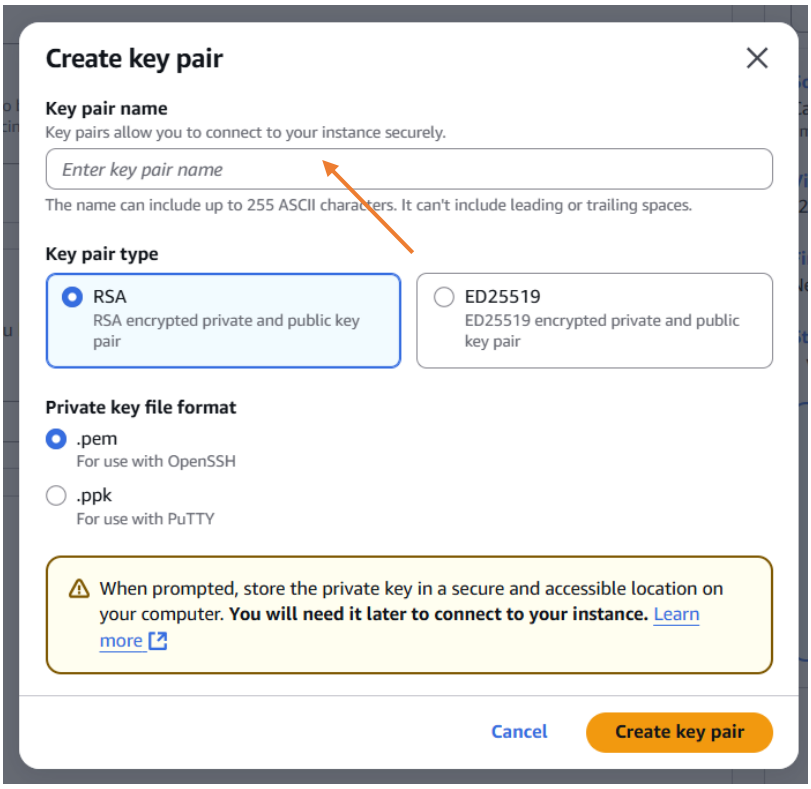
Scroll down to “**Instance Type**”, select “**t2.medium**”



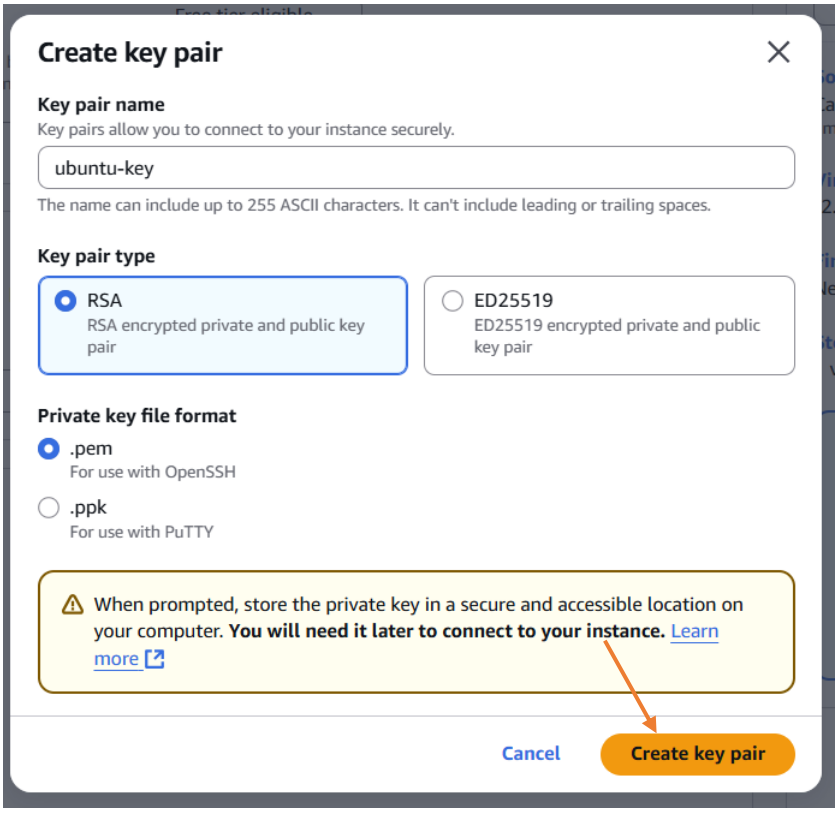
Then scroll down to “**Key Pair**”



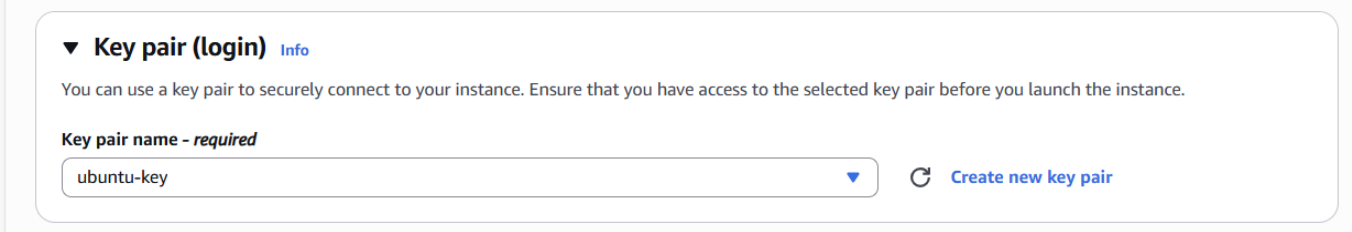
Click on “**create new key pair**”



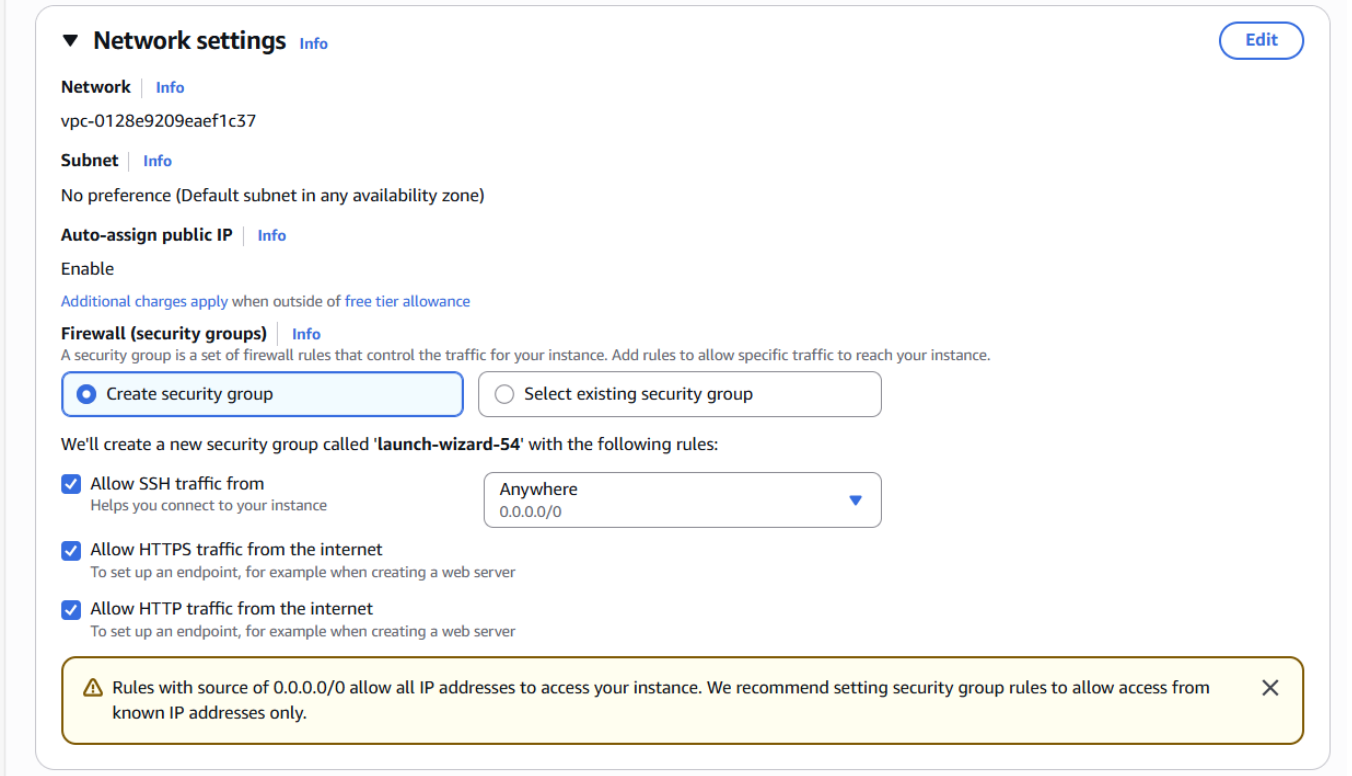
Give the key pair a name, I will call it "**ubuntu-key**"



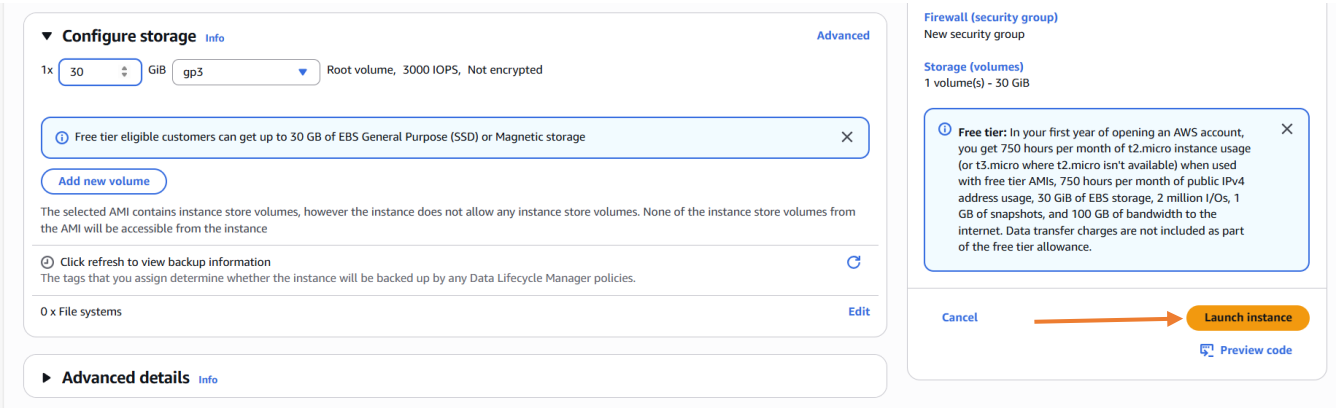
Click on "**Create key pair**"



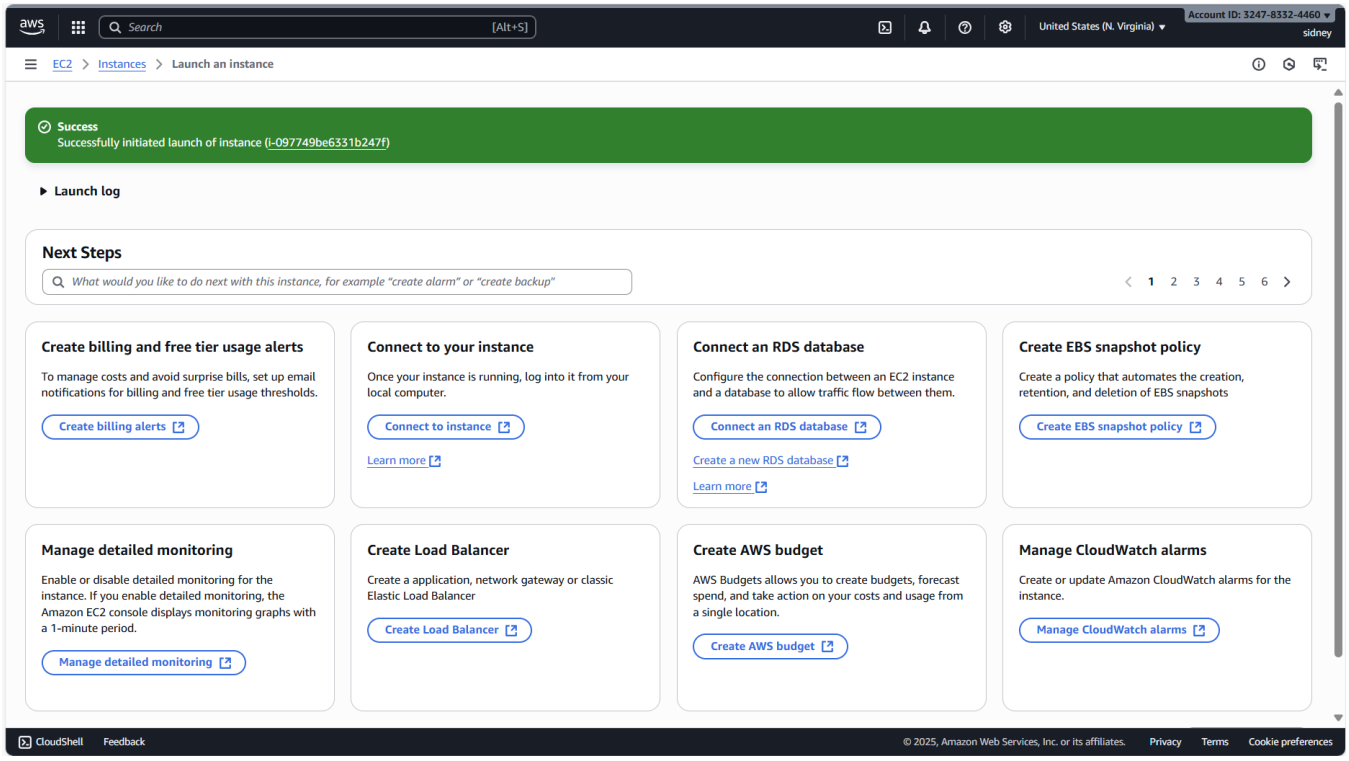
Scroll down to "**Network Settings**"



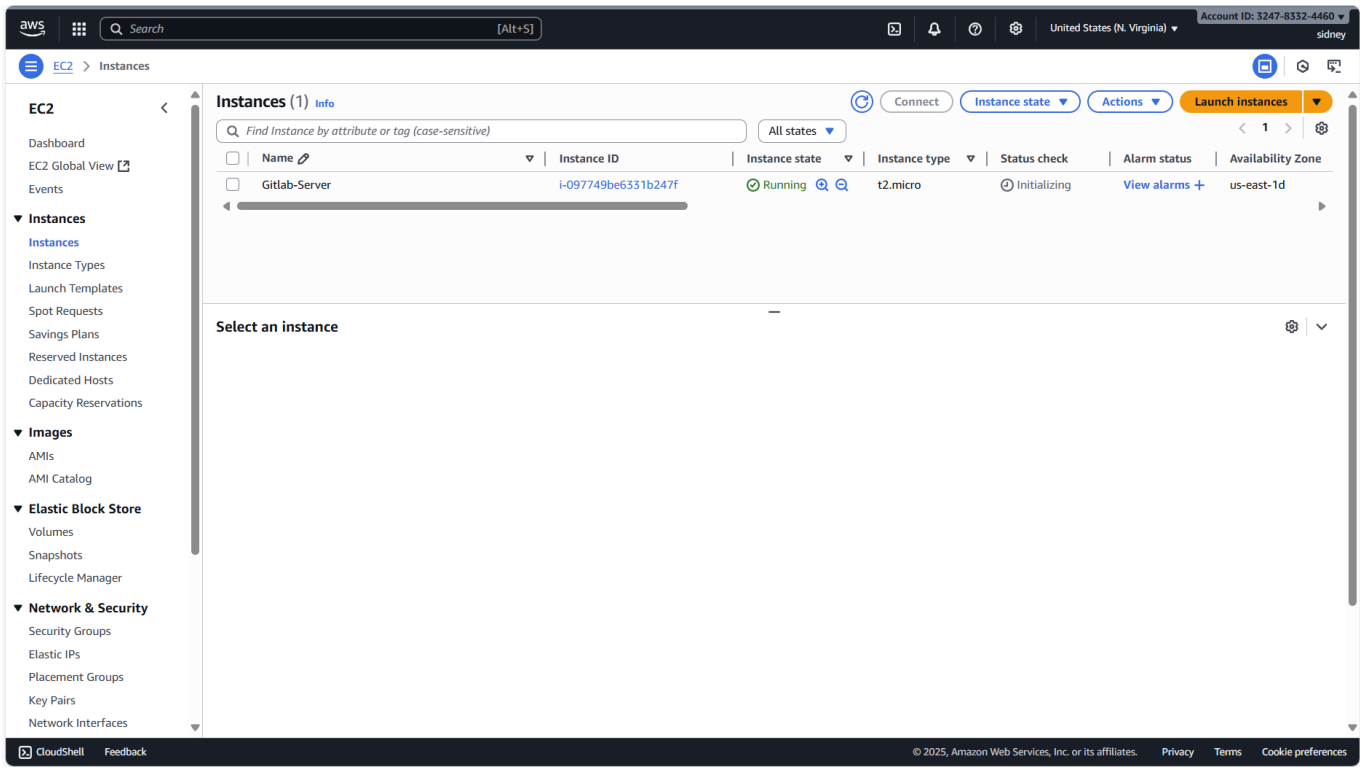
Scroll down to "**Configuration Storage**" and make the value "**30GiB**"



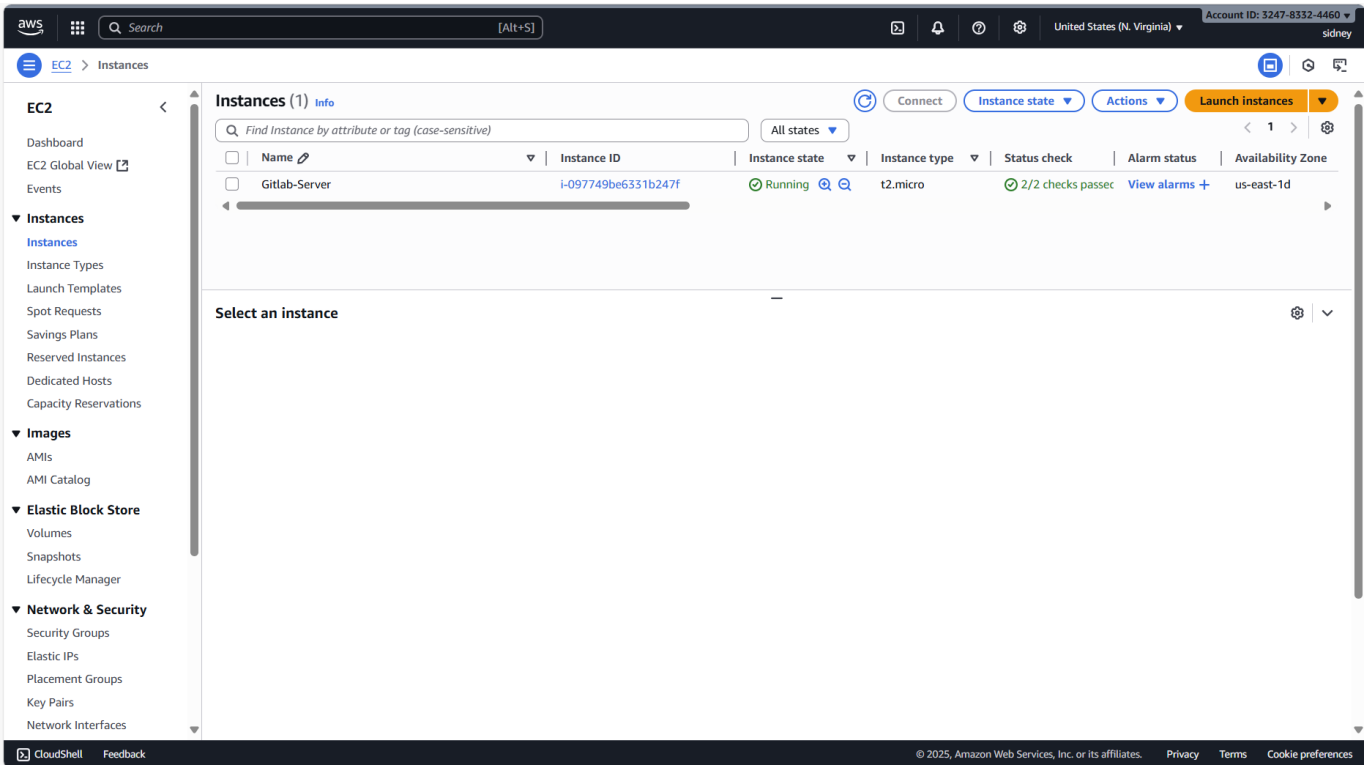
Click on "**Launch Instance**"



Click on "**Instance**"



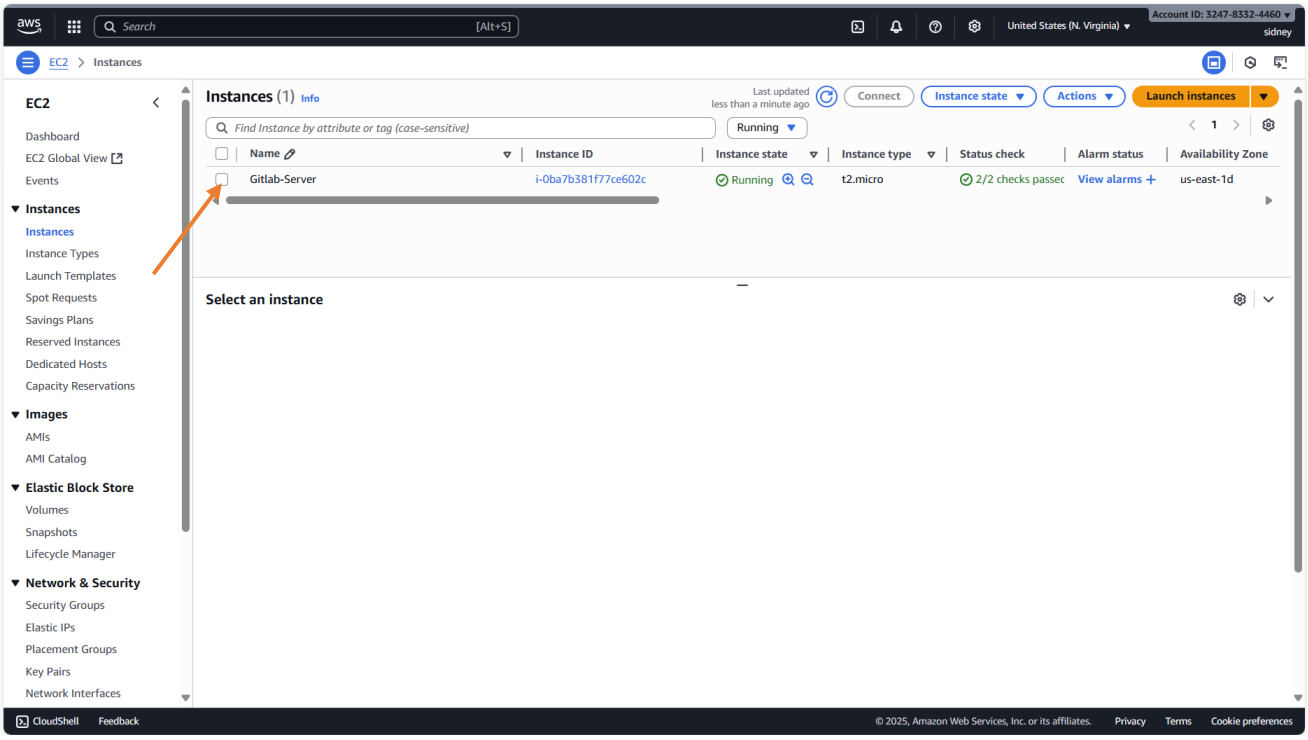
The instance is initializing, wait for it to pass the "**2/2 check**"



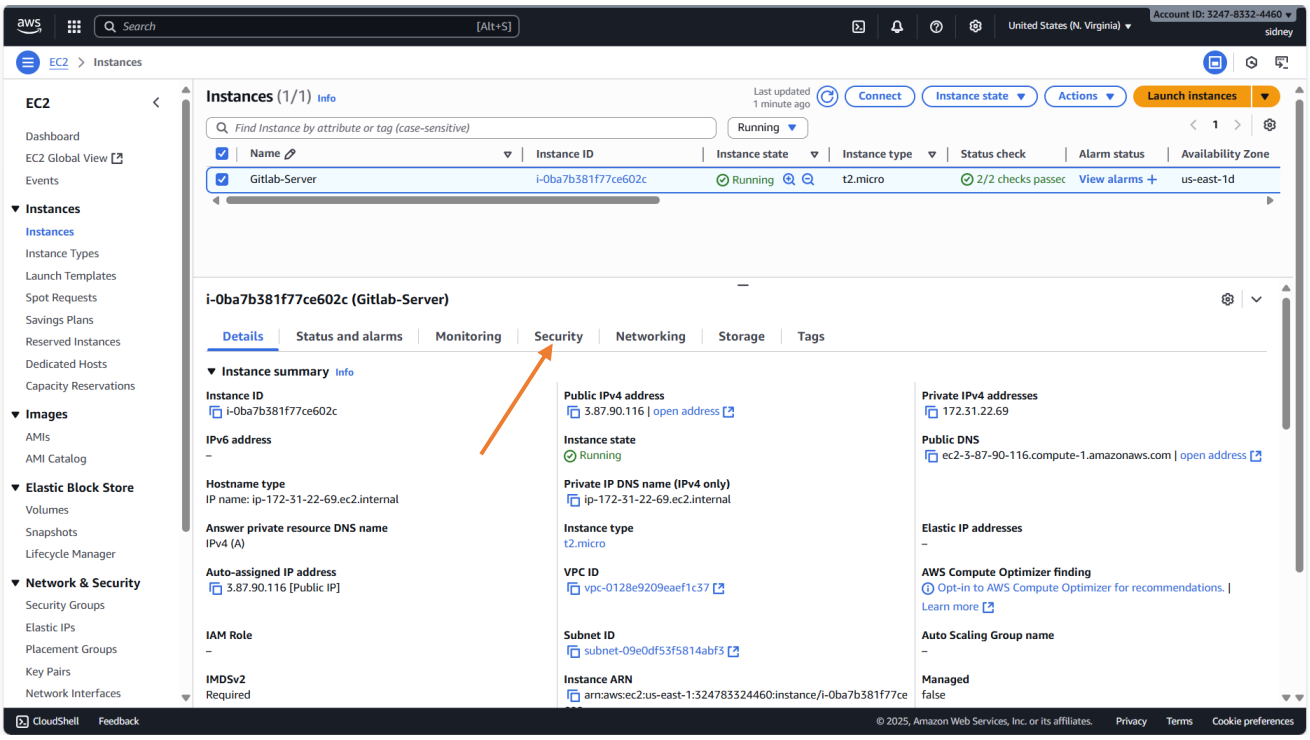
The EC2 instance have passed the "**2/2 check**"

Part 2: Add Port 9000 for SonarQube in the Inbound rule

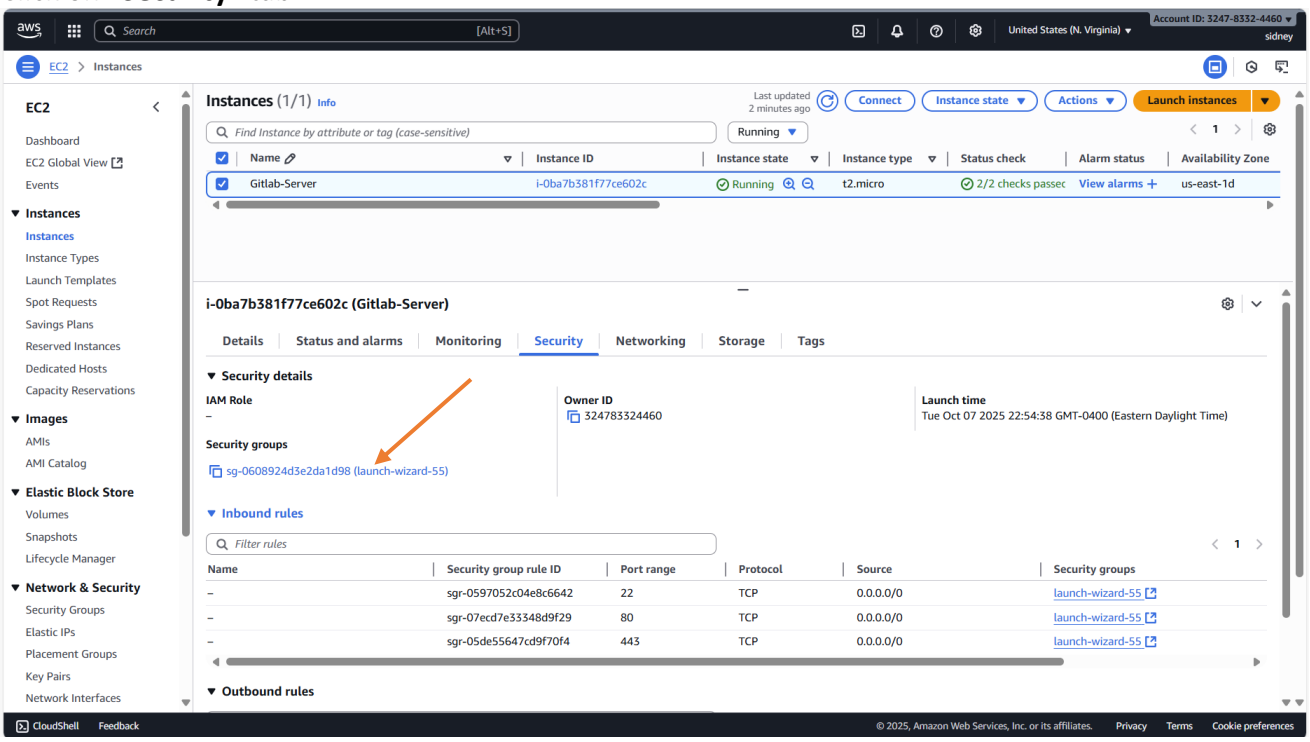
We have to add Port 9000 to be used to access SonarQube on the browser.



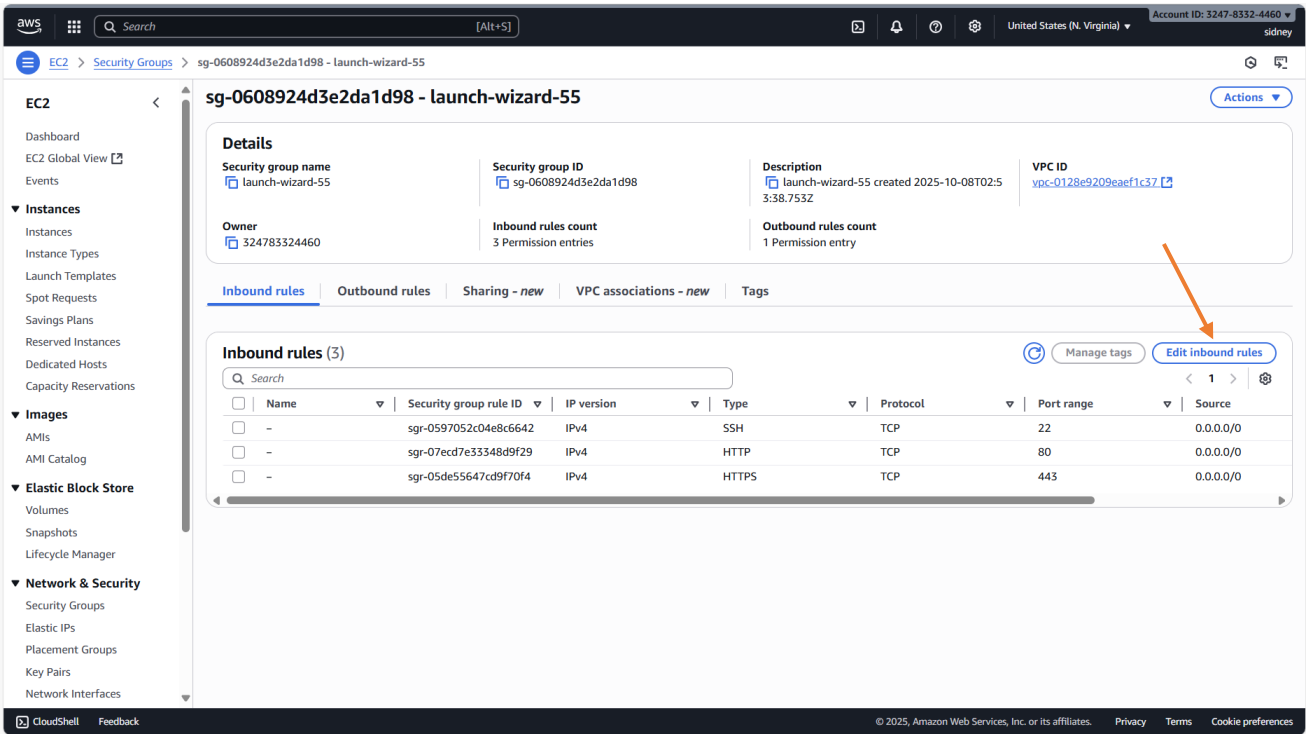
Select the EC2 instance



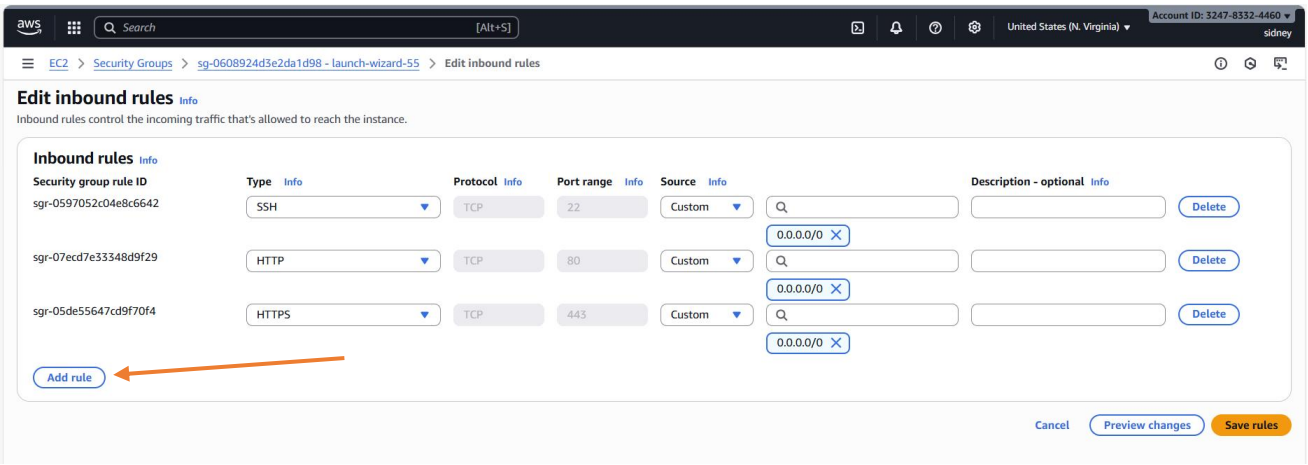
Click on “**Security**” tab



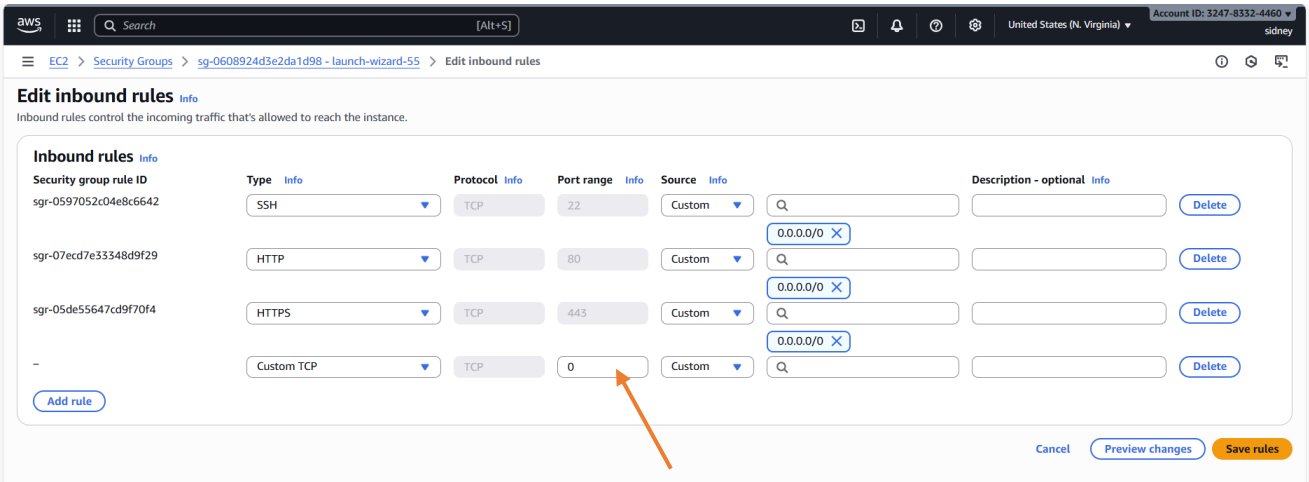
Click on “**Security Group**”



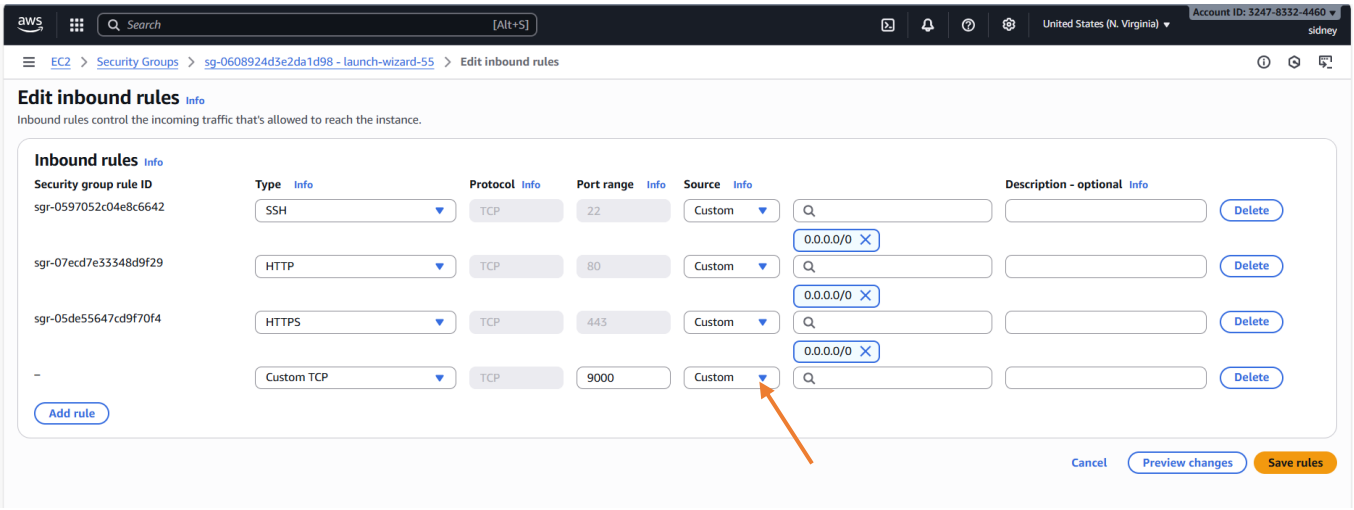
Click on "**Edit Inbound Rules**"



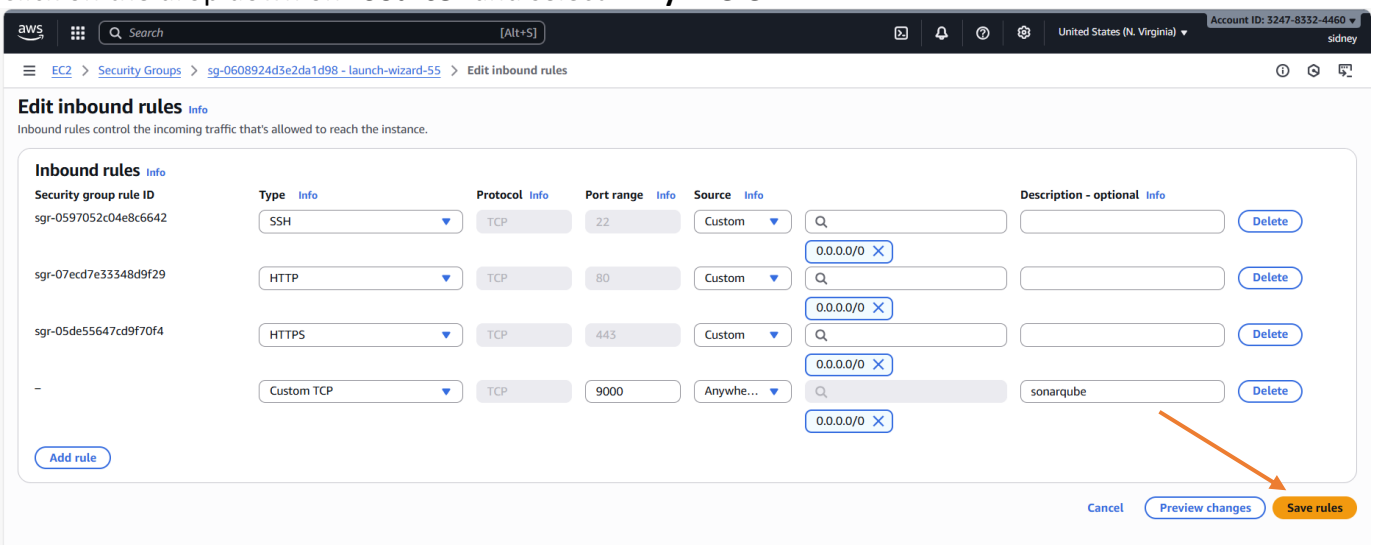
Click on "**Add Rule**"



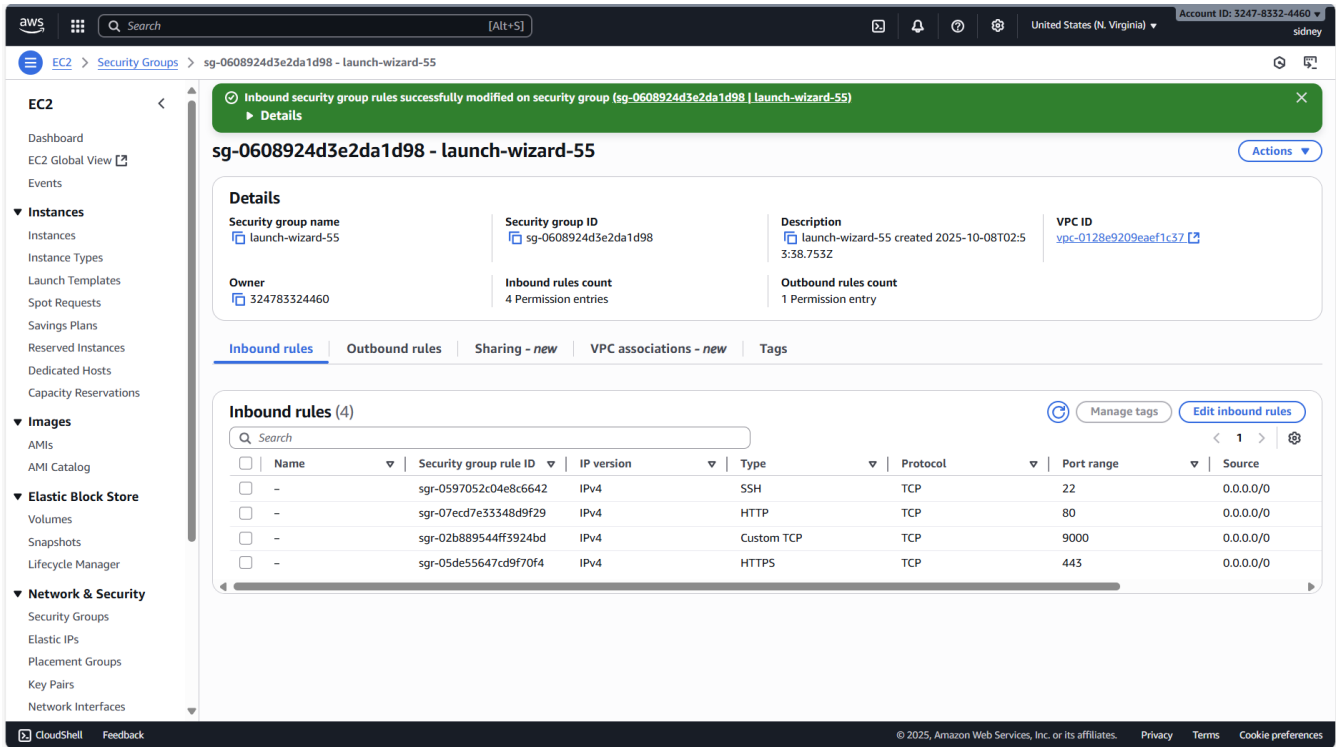
Add Port 9000



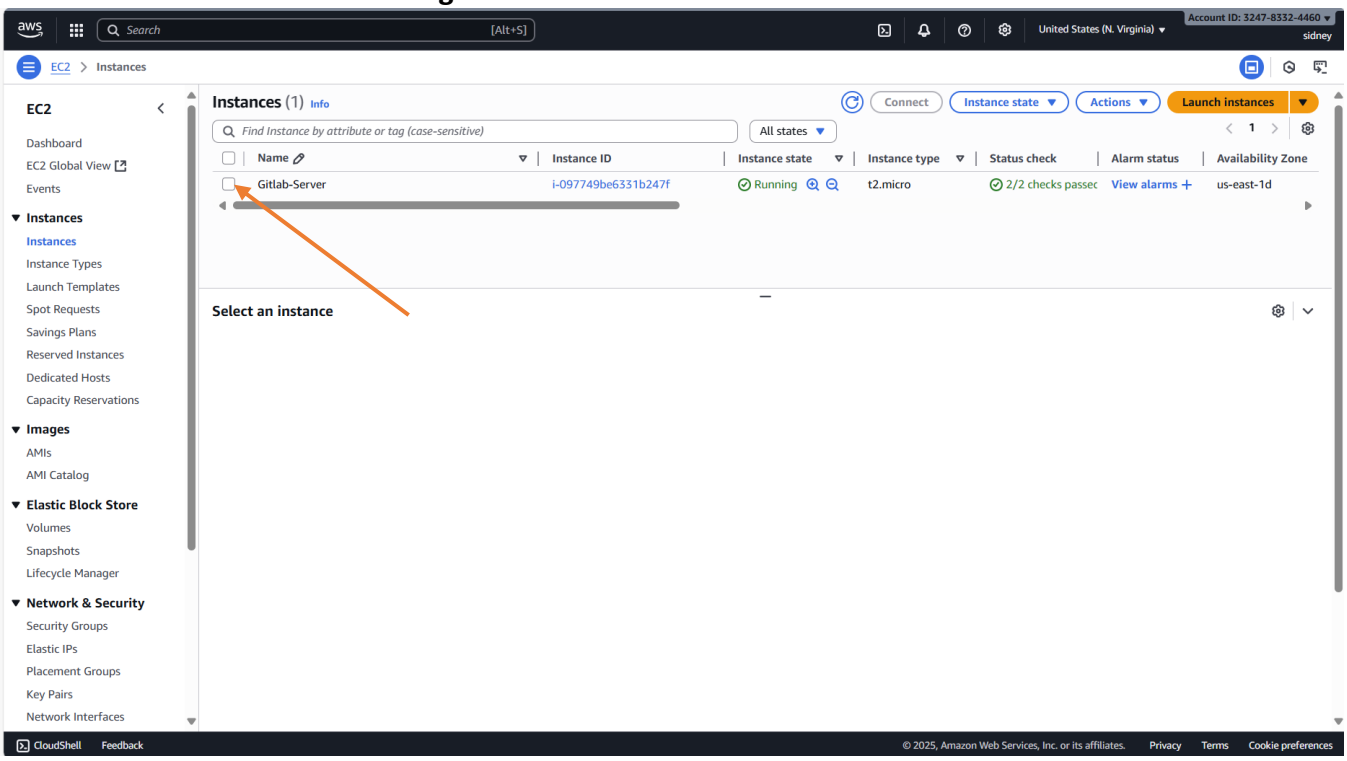
Click on the drop down on “**Source**” and select “**Anywhere-IPv4**”



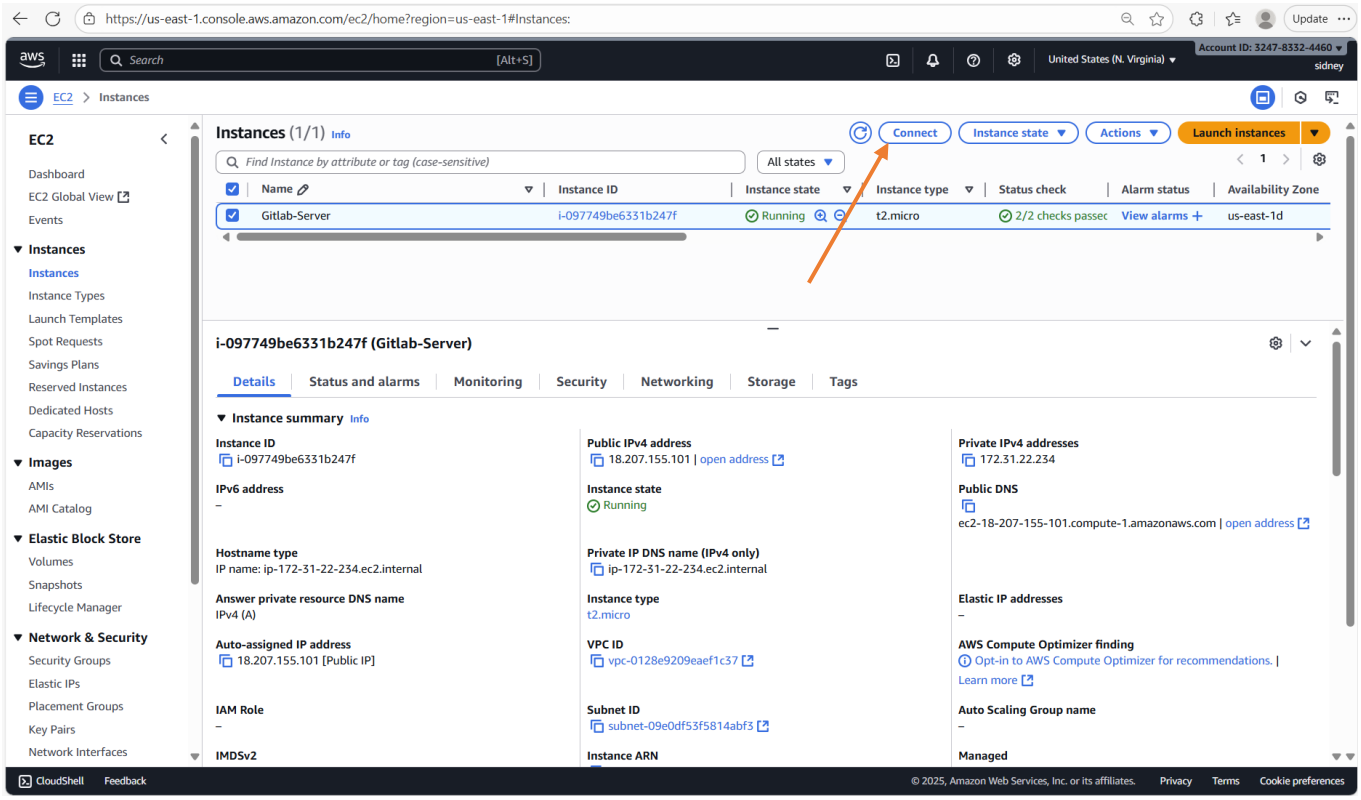
Click on “**Save Rules**”



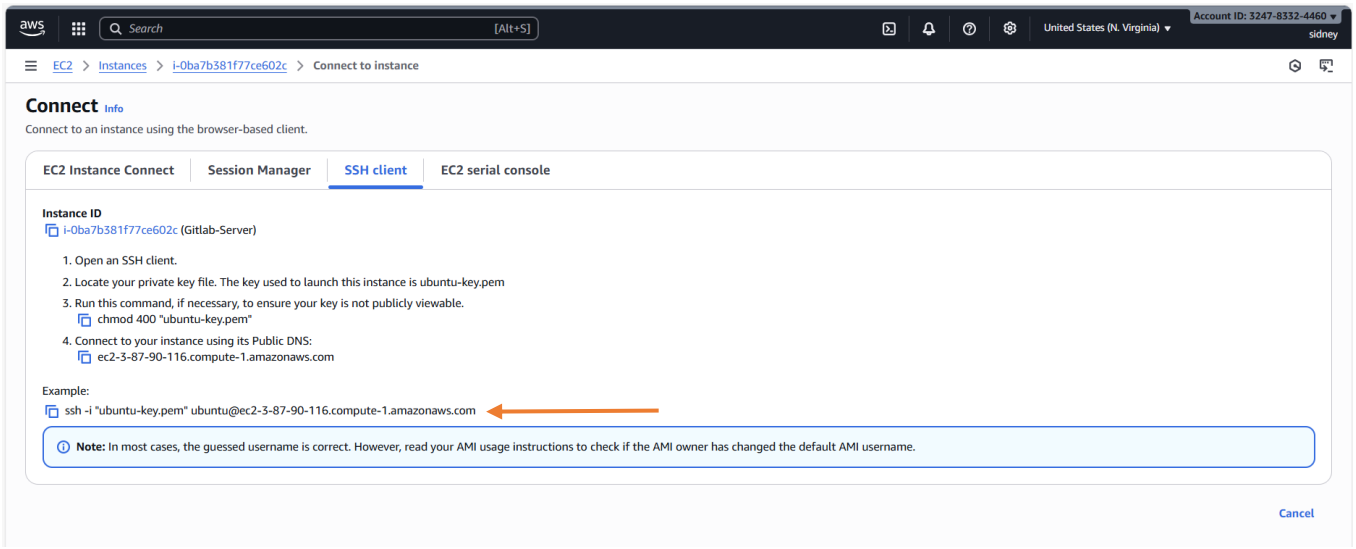
Part 3: Connect to instance through SSH



Select the EC2 instance



Click on “**Connect**”



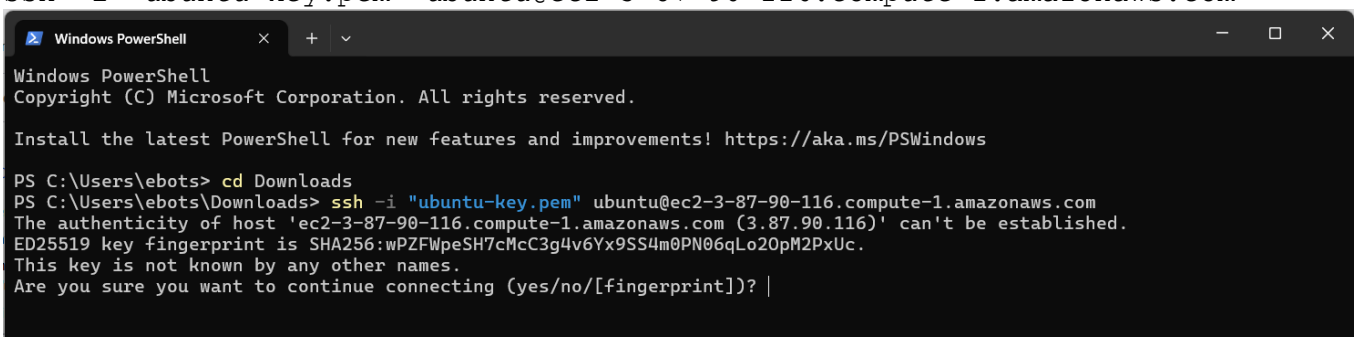
Copy the command:

```
ssh -i "ubuntu-key.pem" ubuntu@ec2-3-87-90-116.compute-1.amazonaws.com
```

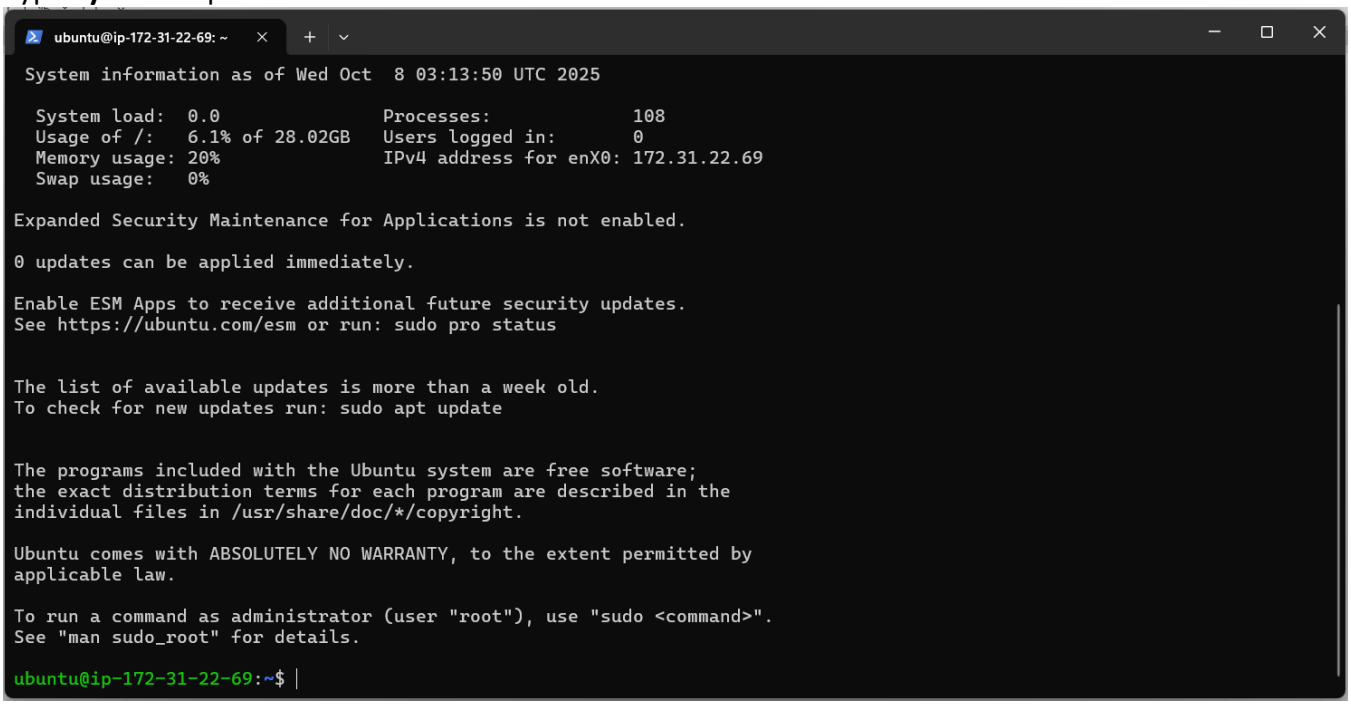
Open PowerShell and navigate to the folder where the “**key pair**” .pem file is saved. It is saved in my “**Downloads**” folder

Then run the command:

```
ssh -i "ubuntu-key.pem" ubuntu@ec2-3-87-90-116.compute-1.amazonaws.com
```

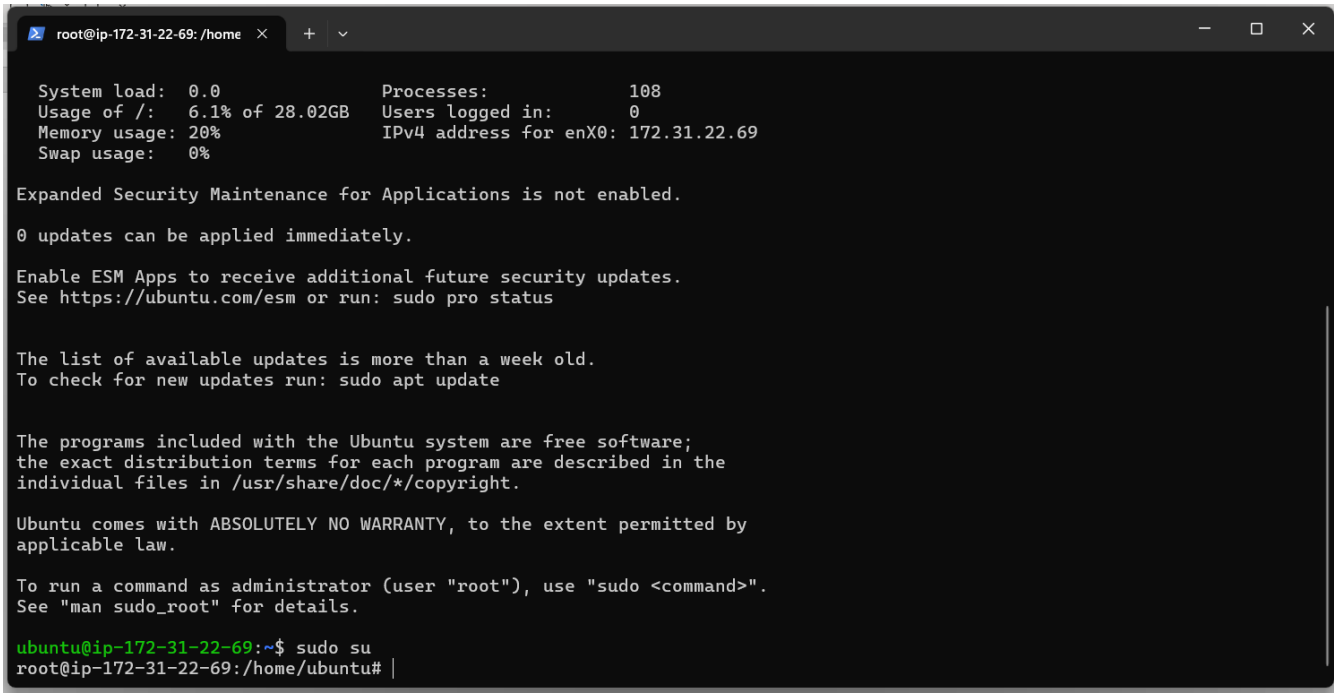


Type "**yes**" and press ENTER



Grant root user privilege using the command:

```
sudo su
```



Clear the terminal using the command:

```
clear
```

STEP 2: Install Java JDK 17 on EC2 server

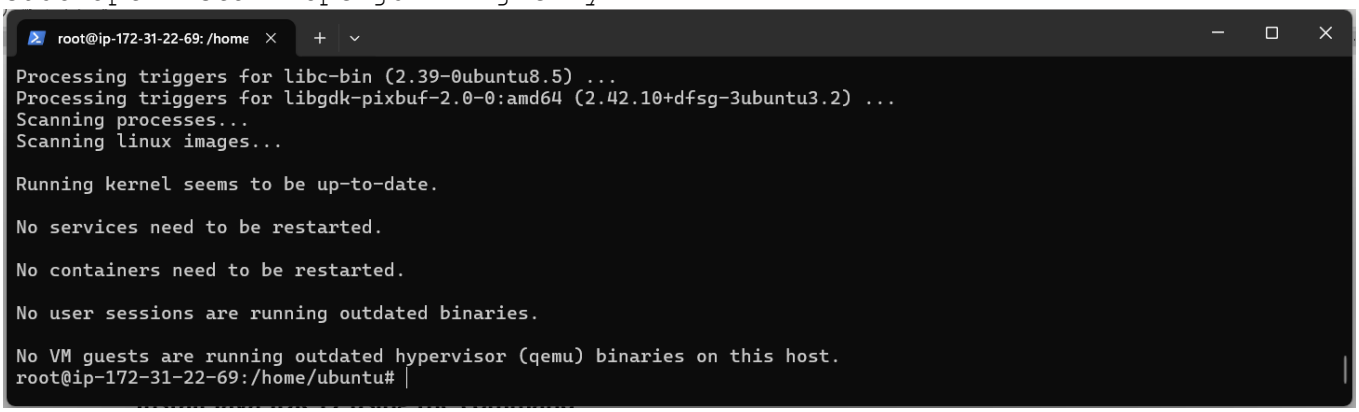
We will now install Java JDK 17 on our server.

First, we update the packages using the command:

```
sudo apt update
```

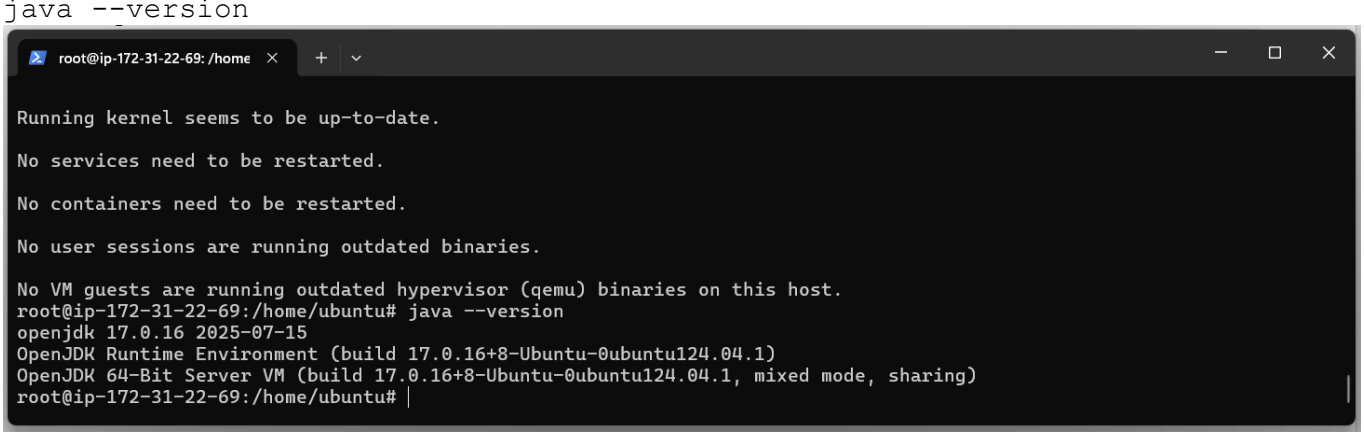
Install Java JDK 17 using the command:

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



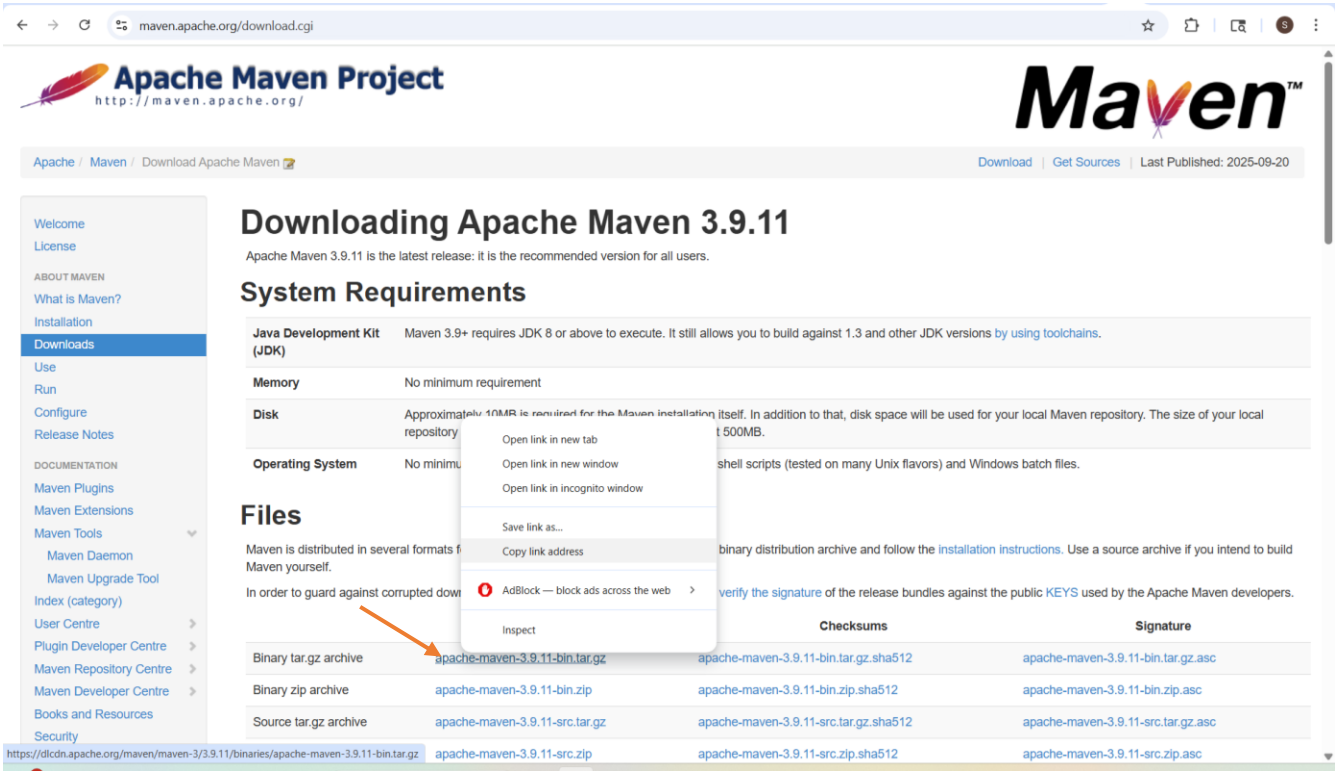
Verify Java is Installed by running the command:

```
java --version
```



STEP 3: 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:

Go to the URL: https://maven.apache.org/download.cgi the link for the “Binary tar.gz archive” file.

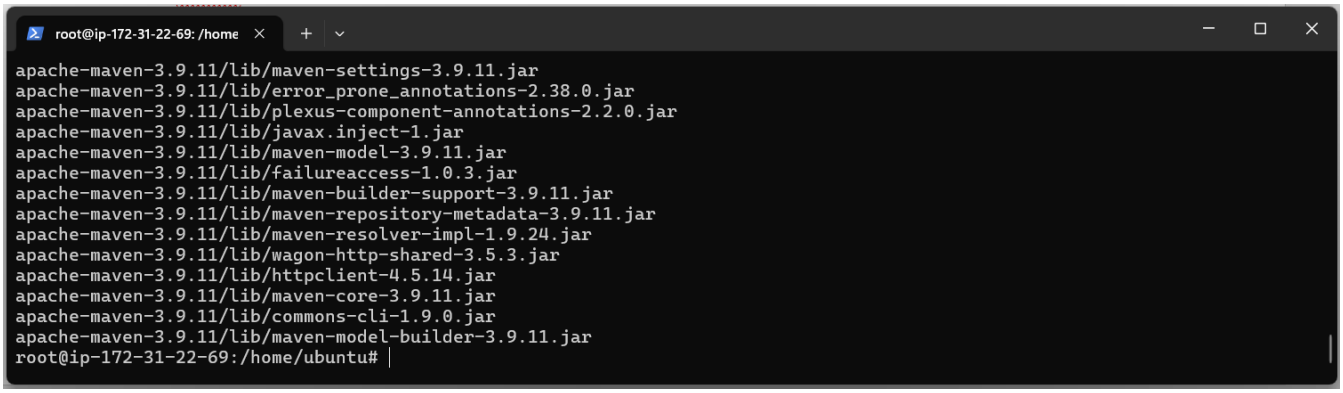


Select “**Copy Link address**”
`https://dlcdn.apache.org/maven/maven-3/3.9.11/binaries/apache-maven-3.9.11-bin.tar.gz`

Part 1: Download the Maven Binaries
Then run the following commands to download Maven
`wget [Link]`

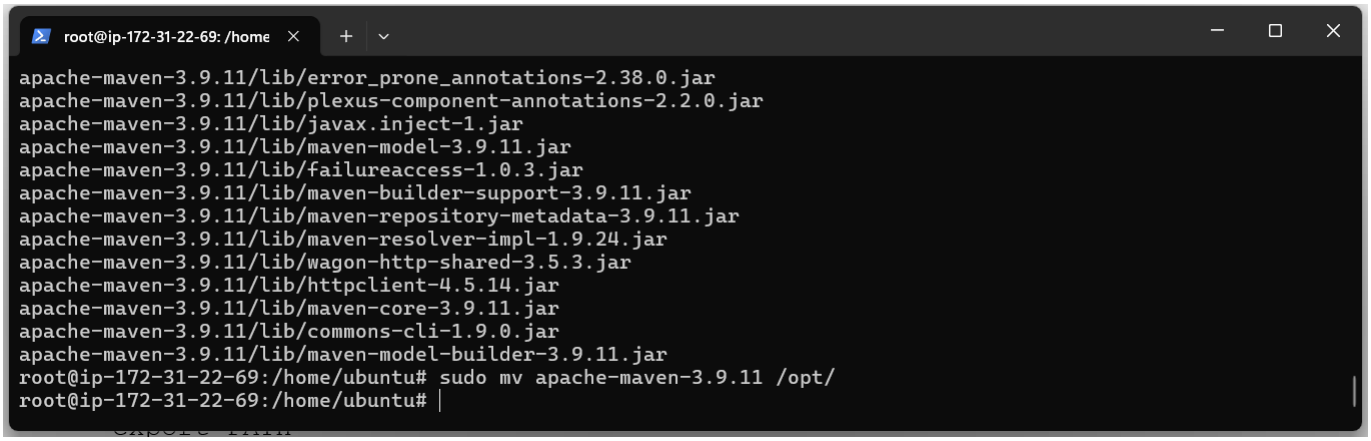
`wget https://dlcdn.apache.org/maven/maven-3/3.9.11/binaries/apache-maven-3.9.11-bin.tar.gz`

Part 2: Untar the Downloaded file
`tar -xvf apache-maven-3.9.11-bin.tar.gz`



Part 3: Untar the mvn package to the /opt folder

```
sudo mv apache-maven-3.9.11 /opt/
```



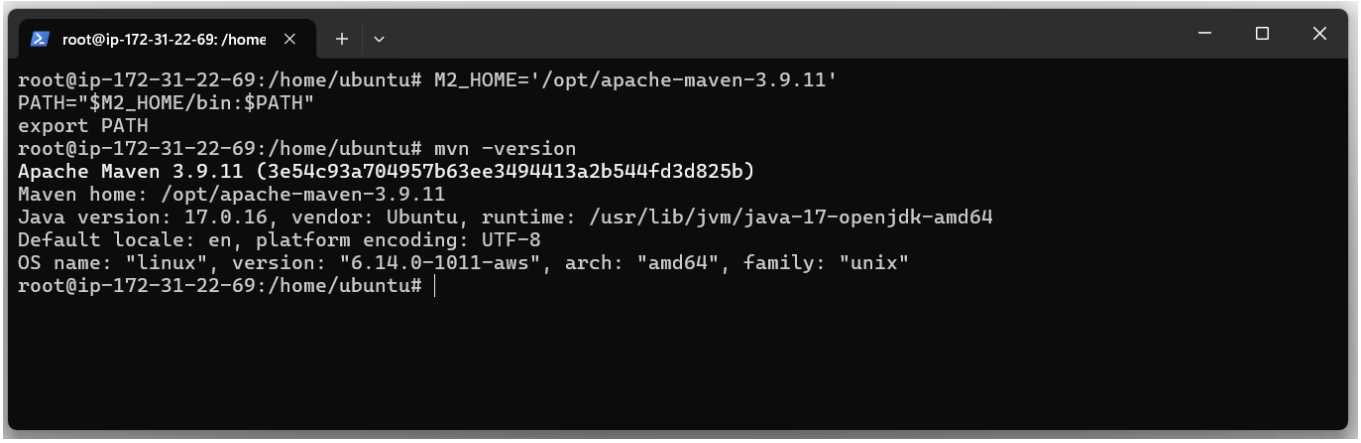
Part 4: Setting M2_HOME and Path Variables

Add the following lines to the user profile file (.profile).

```
M2_HOME='/opt/apache-maven-3.9.11'
PATH="$M2_HOME/bin:$PATH"
export PATH
```

Part 5: Verify the Maven installation by using the command:

```
mvn -version
```



Maven version 3.9.11 has been installed.

STEP 4: Install Docker

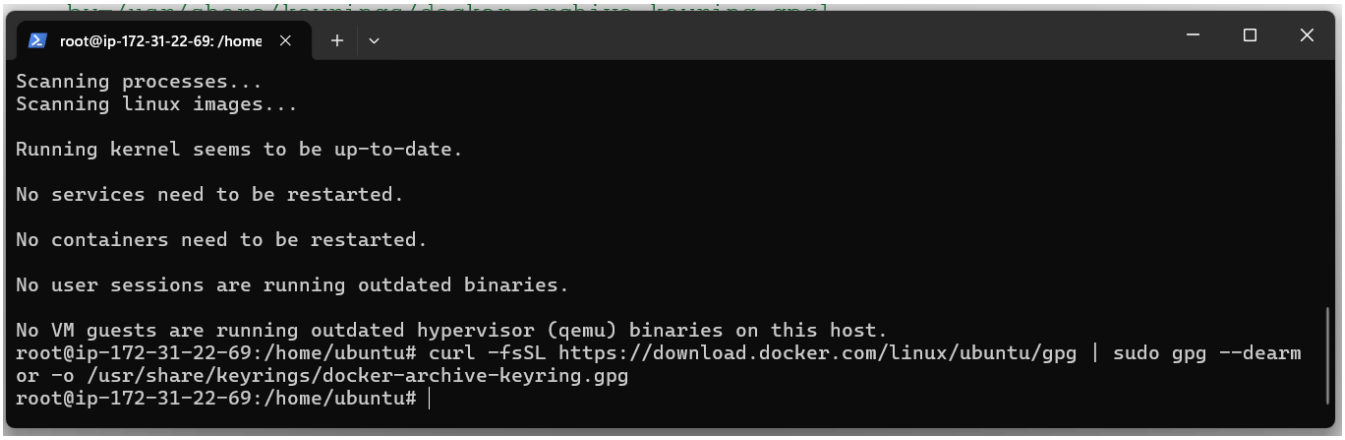
Installing Docker on Ubuntu can be done by setting up the official Docker repository and then installing the necessary packages. This method ensures you receive the latest stable versions and updates.

Part 1: Update Package Index and Install Prerequisites

```
sudo apt update
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common
```

Part 2: Add Docker's GPG Key

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg
```



Part 3: Add the Docker Repository

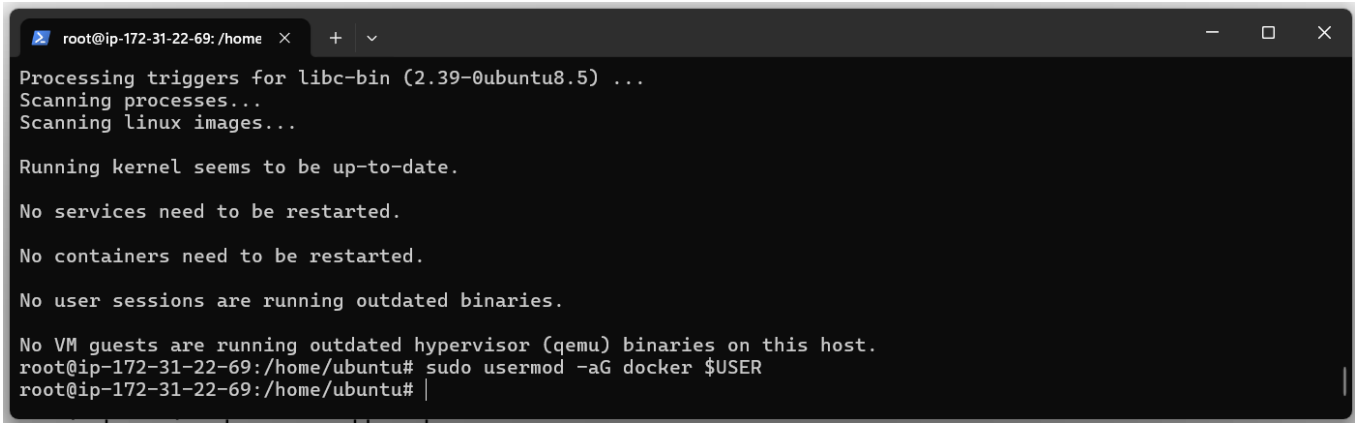
```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-
archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Part 4: Install Docker Engine

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

Part 5: Add Your User to the Docker Group (to run Docker commands without sudo):

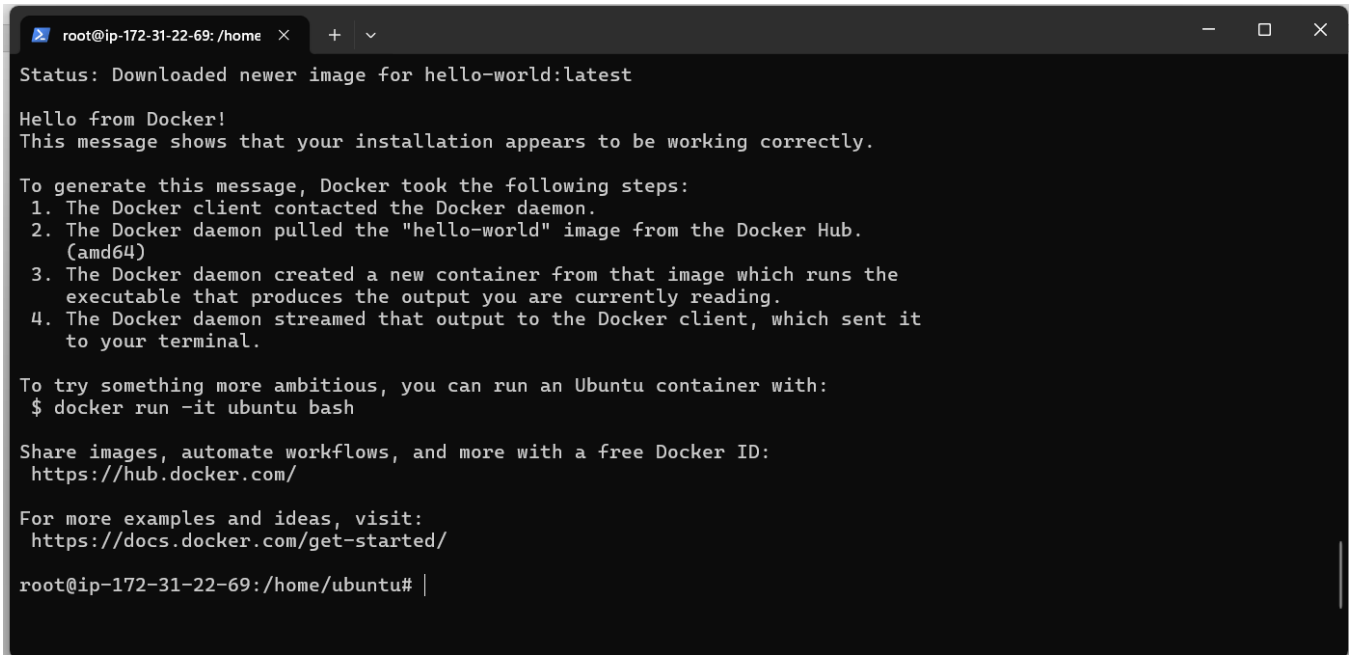
```
sudo usermod -aG docker $USER
```



You will need to log out and log back in (or restart your system) for this change to take effect.

Part 6: Verify the Installation

`docker run hello-world`



STEP 5: Install SonarQube on EC2 Server

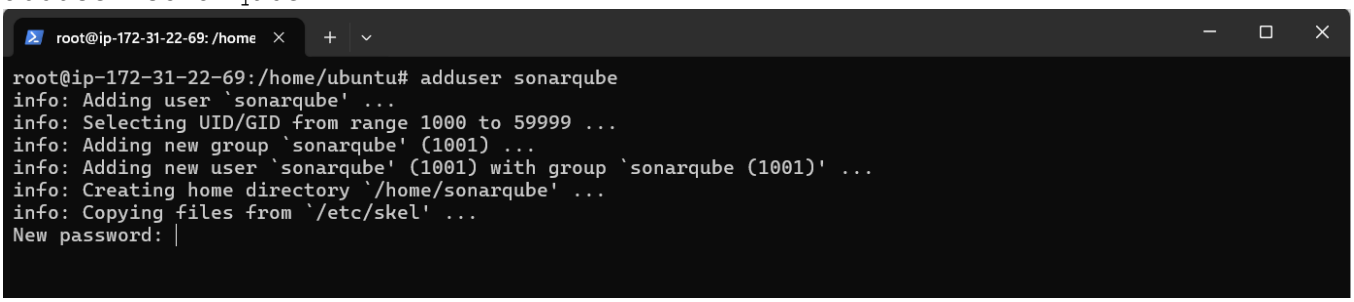
Now, we have to do "**Static Code Analysis**" using SonarQube. Installing SonarQube on an Ubuntu EC2 instance involves several key steps.

we are setting up our sonar on the same server as Jenkins and this requires opening an additional port for SonarQube to run. SonarQube on default runs on port 9000

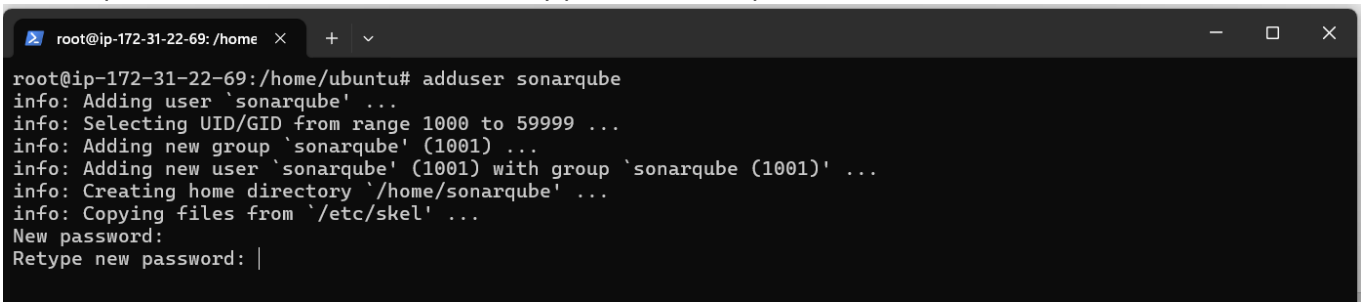
Part 1: Create a SonarQube user

First, we have to create a SonarQube user using the command:

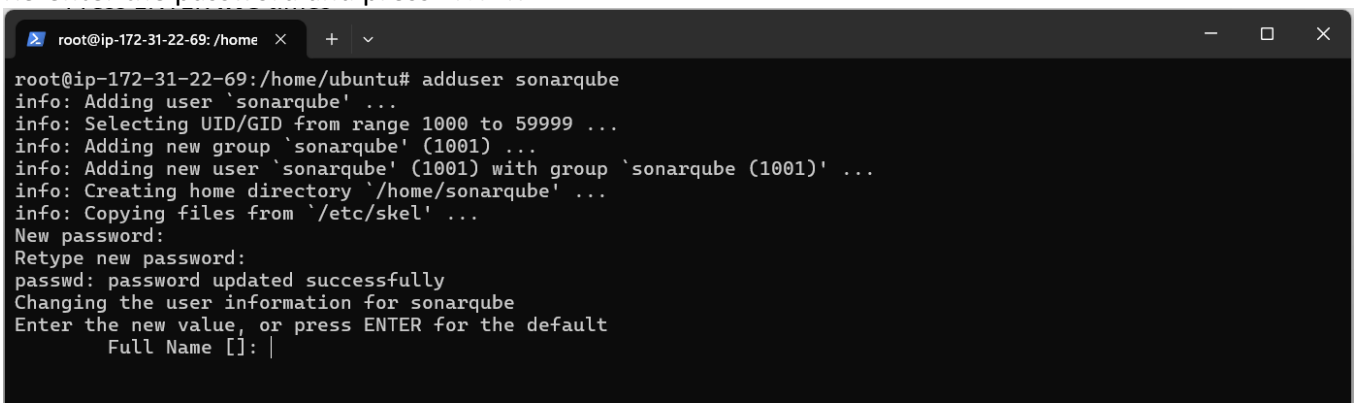
```
adduser sonarqube
```



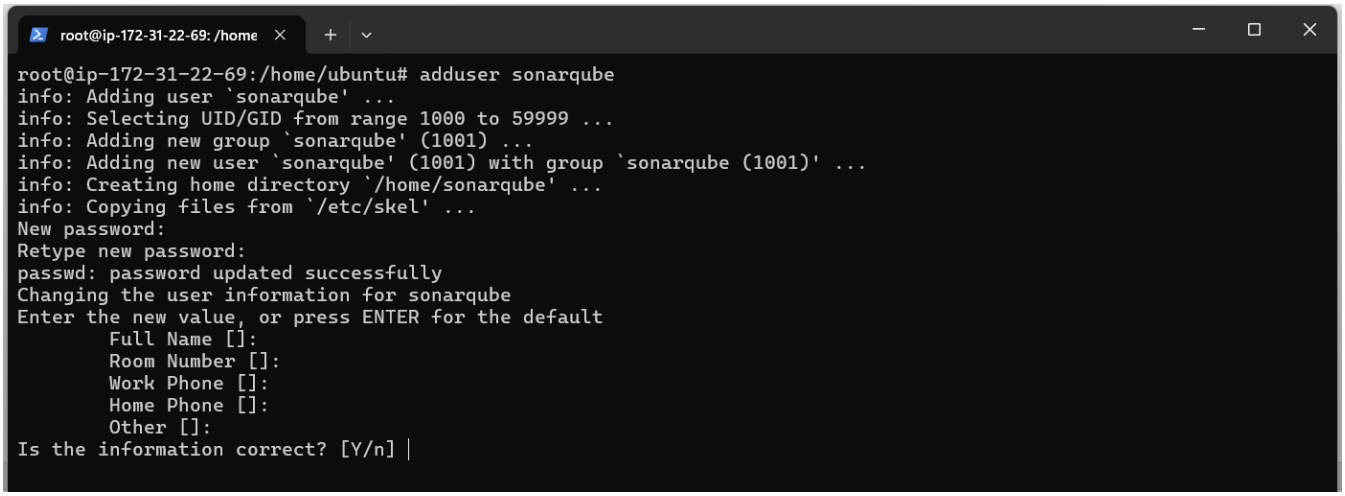
Enter a password. I will use "**admin**" as my password and press Enter



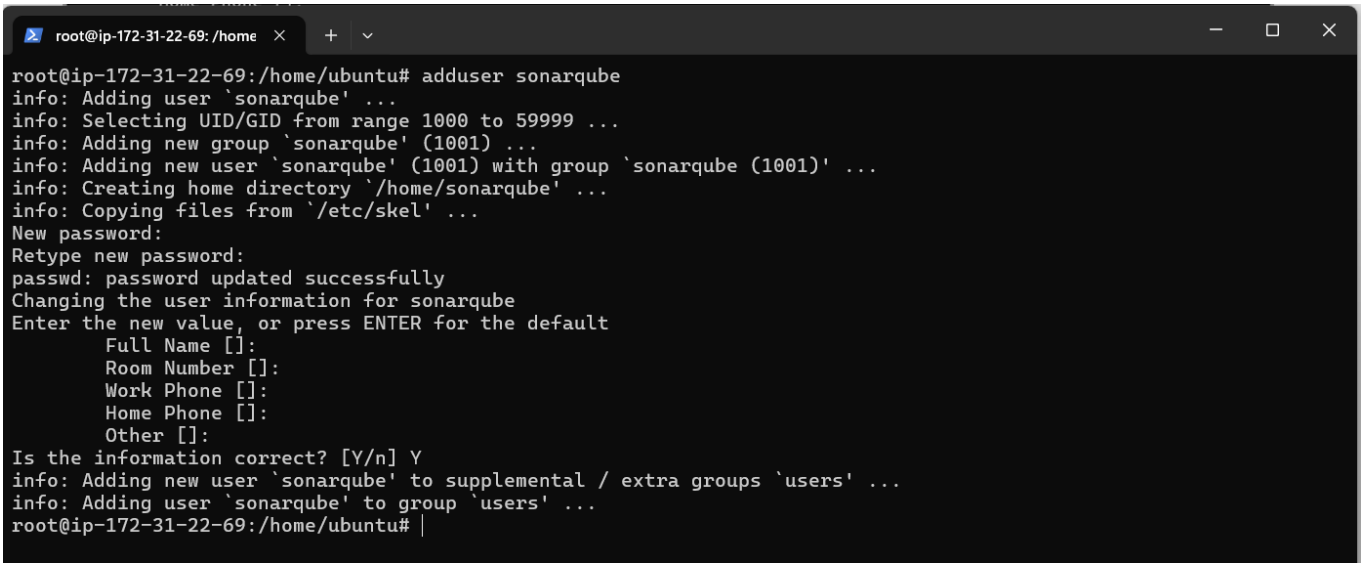
Re-enter the password and press ENTER



Press ENTER **five** times



Type "**Y**" and press ENTER

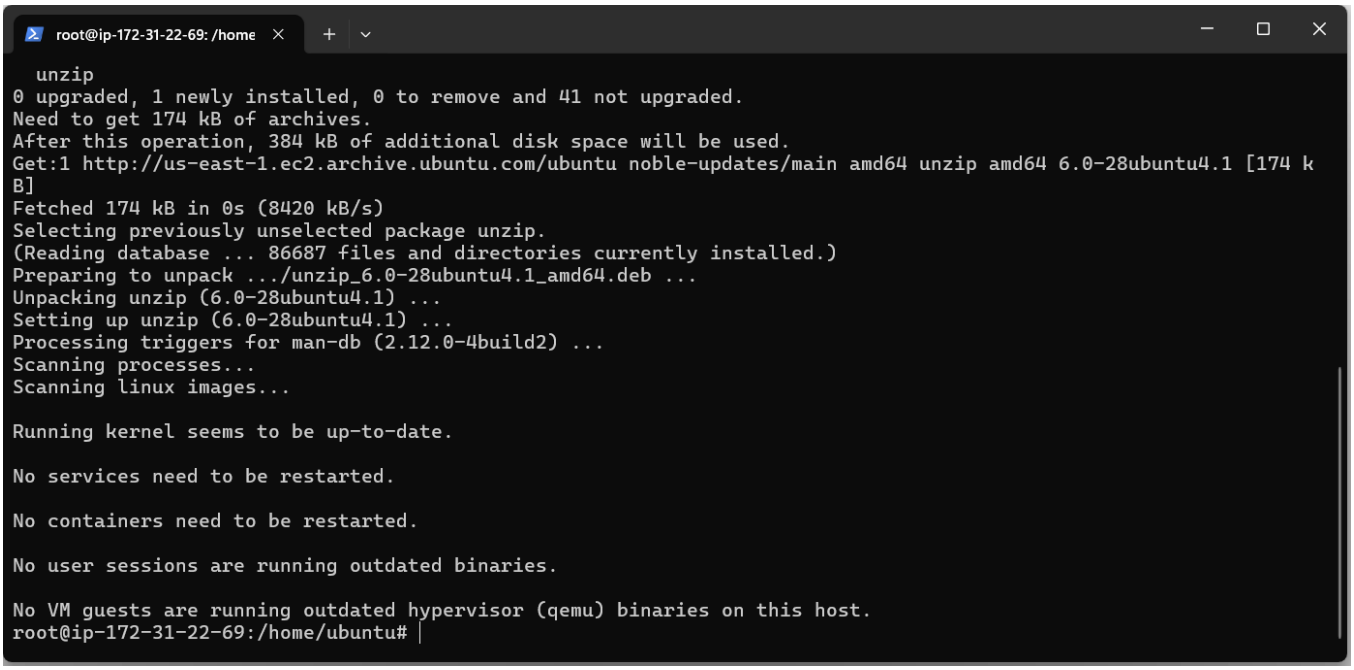


The user has been added

Part 2: Install unzip

We have to install unzip. This will be used to unzip the SonarQube zip file we are going to download.

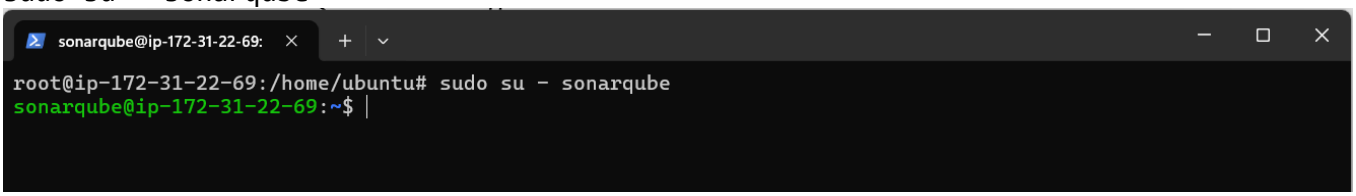
`apt install unzip`



Part 3: Switch to the user

Switch to the SonarQube user using the command:

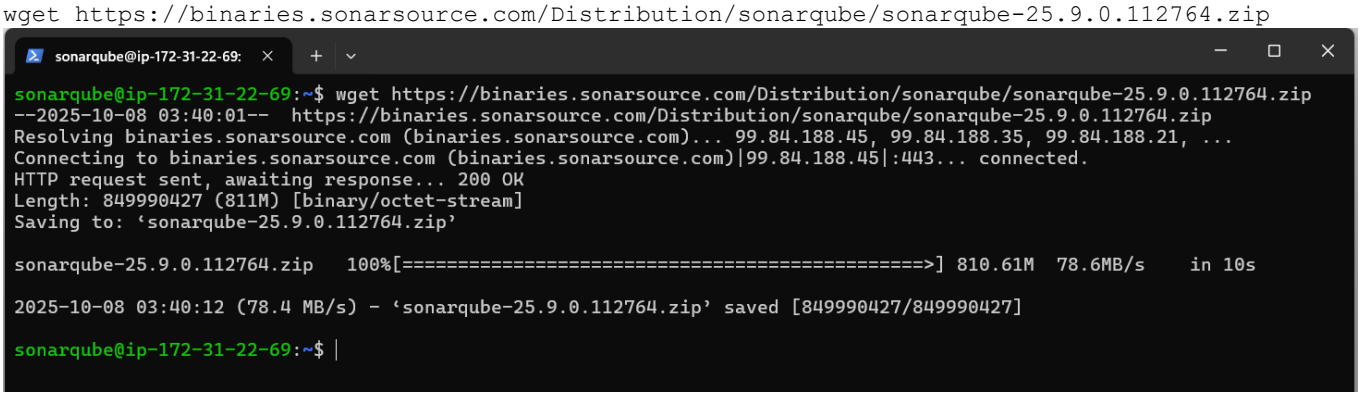
`sudo su - sonarqube`



Part 4: Download SonarQube:

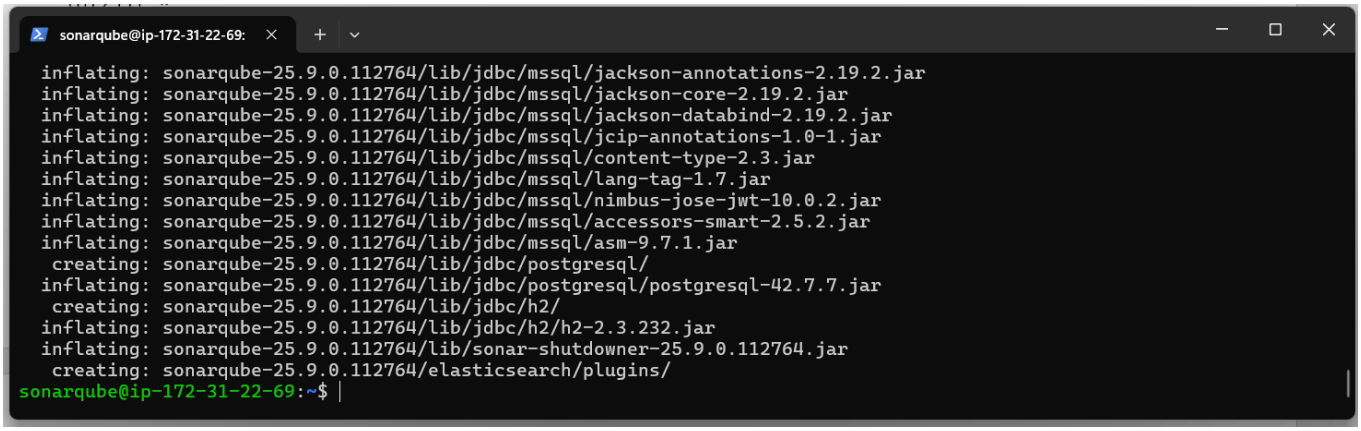
Download SonarQube using the command

`wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-25.9.0.112764.zip`



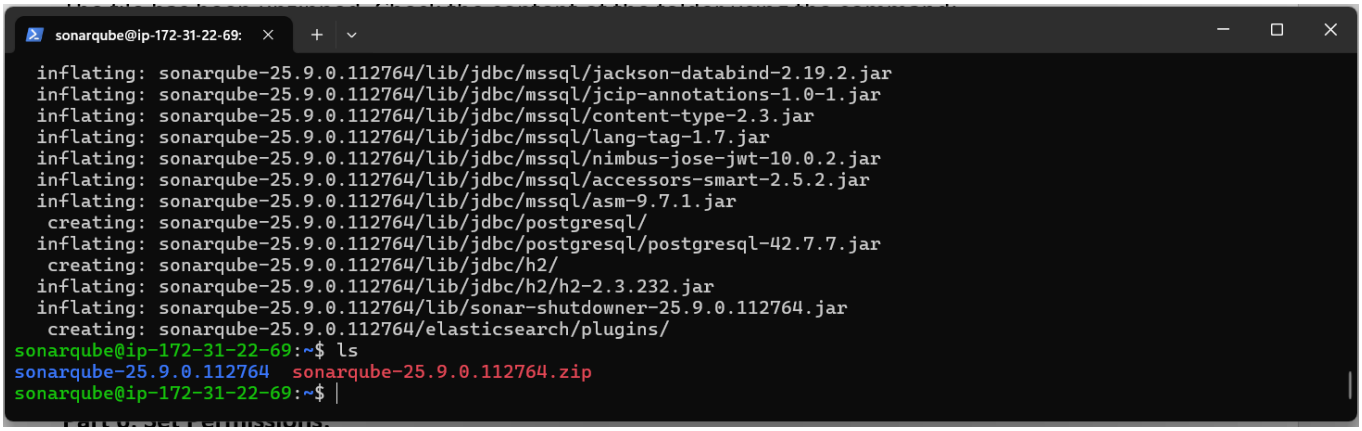
Part 5: Extract Downloaded SonarQube file

`unzip *`



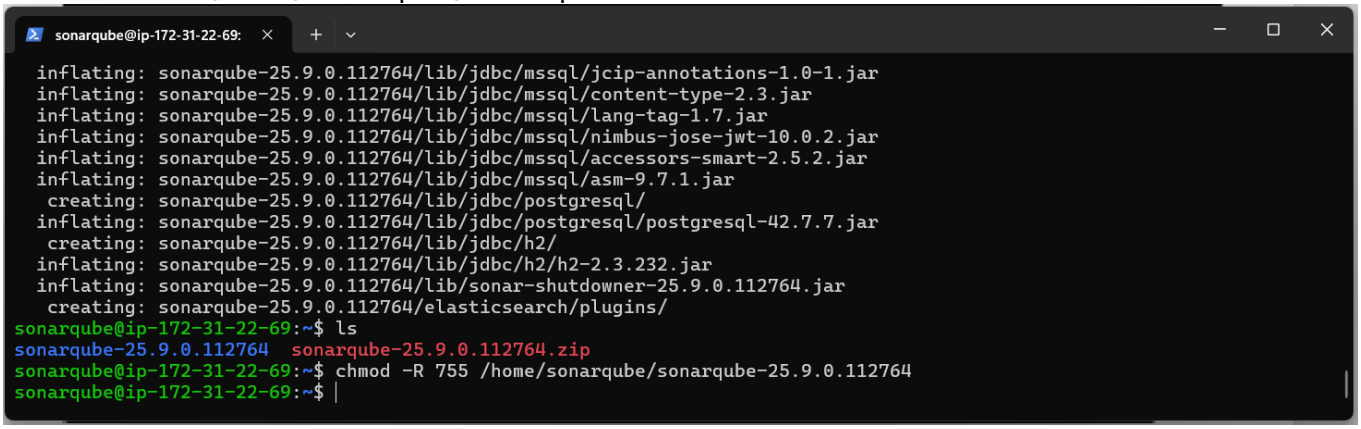
The file has been unzipped. Check the content of the folder using the command:

```
ls
```



Part 6: Set Permissions:

```
chmod -R 755 /home/sonarqube/sonarqube-25.9.0.112764
```



Then run the command:

```
chown -R sonarqube:sonarqube /home/sonarqube/sonarqube-25.9.0.112764
```

```
sonarqube@ip-172-31-22-69:
  inflating: sonarqube-25.9.0.112764/lib/jdbc/mssql/content-type-2.3.jar
  inflating: sonarqube-25.9.0.112764/lib/jdbc/mssql/lang-tag-1.7.jar
  inflating: sonarqube-25.9.0.112764/lib/jdbc/mssql/nimbus-jose-jwt-10.0.2.jar
  inflating: sonarqube-25.9.0.112764/lib/jdbc/mssql/accessors-smart-2.5.2.jar
  inflating: sonarqube-25.9.0.112764/lib/jdbc/mssql/asm-9.7.1.jar
   creating: sonarqube-25.9.0.112764/lib/jdbc/postgresql/
  inflating: sonarqube-25.9.0.112764/lib/jdbc/postgresql/postgresql-42.7.7.jar
   creating: sonarqube-25.9.0.112764/lib/jdbc/h2/
  inflating: sonarqube-25.9.0.112764/lib/jdbc/h2/h2-2.3.232.jar
  inflating: sonarqube-25.9.0.112764/lib/sonar-shutdowner-25.9.0.112764.jar
   creating: sonarqube-25.9.0.112764/elasticsearch/plugins/
sonarqube@ip-172-31-22-69:~$ ls
sonarqube-25.9.0.112764  sonarqube-25.9.0.112764.zip
sonarqube@ip-172-31-22-69:~$ chmod -R 755 /home/sonarqube/sonarqube-25.9.0.112764
sonarqube@ip-172-31-22-69:~$ chown -R sonarqube:sonarqube /home/sonarqube/sonarqube-25.9.0.112764
sonarqube@ip-172-31-22-69:~$
```

Part 7: Start SonarQube:

Navigate to the directory linux-x86-64 using the command:

`cd sonarqube-25.9.0.112764/bin/linux-x86-64/`

```
sonarqube@ip-172-31-22-69:
  inflating: sonarqube-25.9.0.112764/lib/jdbc/mssql/lang-tag-1.7.jar
  inflating: sonarqube-25.9.0.112764/lib/jdbc/mssql/nimbus-jose-jwt-10.0.2.jar
  inflating: sonarqube-25.9.0.112764/lib/jdbc/mssql/accessors-smart-2.5.2.jar
  inflating: sonarqube-25.9.0.112764/lib/jdbc/mssql/asm-9.7.1.jar
   creating: sonarqube-25.9.0.112764/lib/jdbc/postgresql/
  inflating: sonarqube-25.9.0.112764/lib/jdbc/postgresql/postgresql-42.7.7.jar
   creating: sonarqube-25.9.0.112764/lib/jdbc/h2/
  inflating: sonarqube-25.9.0.112764/lib/jdbc/h2/h2-2.3.232.jar
  inflating: sonarqube-25.9.0.112764/lib/sonar-shutdowner-25.9.0.112764.jar
   creating: sonarqube-25.9.0.112764/elasticsearch/plugins/
sonarqube@ip-172-31-22-69:~$ ls
sonarqube-25.9.0.112764  sonarqube-25.9.0.112764.zip
sonarqube@ip-172-31-22-69:~$ chmod -R 755 /home/sonarqube/sonarqube-25.9.0.112764
sonarqube@ip-172-31-22-69:~$ chown -R sonarqube:sonarqube /home/sonarqube/sonarqube-25.9.0.112764
sonarqube@ip-172-31-22-69:~$ cd sonarqube-25.9.0.112764/bin/linux-x86-64/
sonarqube@ip-172-31-22-69:~/sonarqube-25.9.0.112764/bin/linux-x86-64$
```

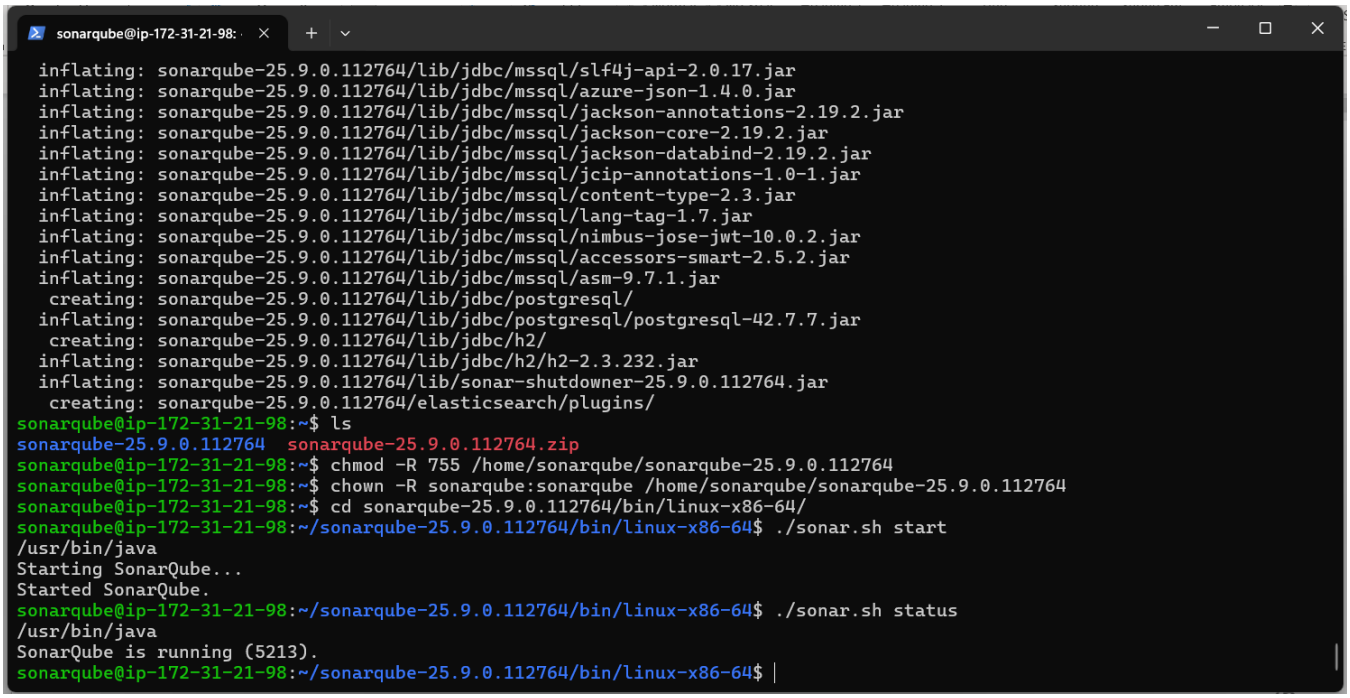
We will start SonarQube by running the command:

`./sonar.sh start`

```
sonarqube@ip-172-31-22-69:
   creating: sonarqube-25.9.0.112764/lib/jdbc/postgresql/
  inflating: sonarqube-25.9.0.112764/lib/jdbc/postgresql/postgresql-42.7.7.jar
   creating: sonarqube-25.9.0.112764/lib/jdbc/h2/
  inflating: sonarqube-25.9.0.112764/lib/jdbc/h2/h2-2.3.232.jar
  inflating: sonarqube-25.9.0.112764/lib/sonar-shutdowner-25.9.0.112764.jar
   creating: sonarqube-25.9.0.112764/elasticsearch/plugins/
sonarqube@ip-172-31-22-69:~$ ls
sonarqube-25.9.0.112764  sonarqube-25.9.0.112764.zip
sonarqube@ip-172-31-22-69:~$ chmod -R 755 /home/sonarqube/sonarqube-25.9.0.112764
sonarqube@ip-172-31-22-69:~$ chown -R sonarqube:sonarqube /home/sonarqube/sonarqube-25.9.0.112764
sonarqube@ip-172-31-22-69:~$ cd sonarqube-25.9.0.112764/bin/linux-x86-64/
sonarqube@ip-172-31-22-69:~/sonarqube-25.9.0.112764/bin/linux-x86-64$ ./sonar.sh start
/usr/bin/java
Starting SonarQube...
Started SonarQube.
sonarqube@ip-172-31-22-69:~/sonarqube-25.9.0.112764/bin/linux-x86-64$
```

Check Sonarqube status using the command:

`./sonar.sh status`



You can access it by navigating to the IP address of your server followed by port 9000 in a web browser.

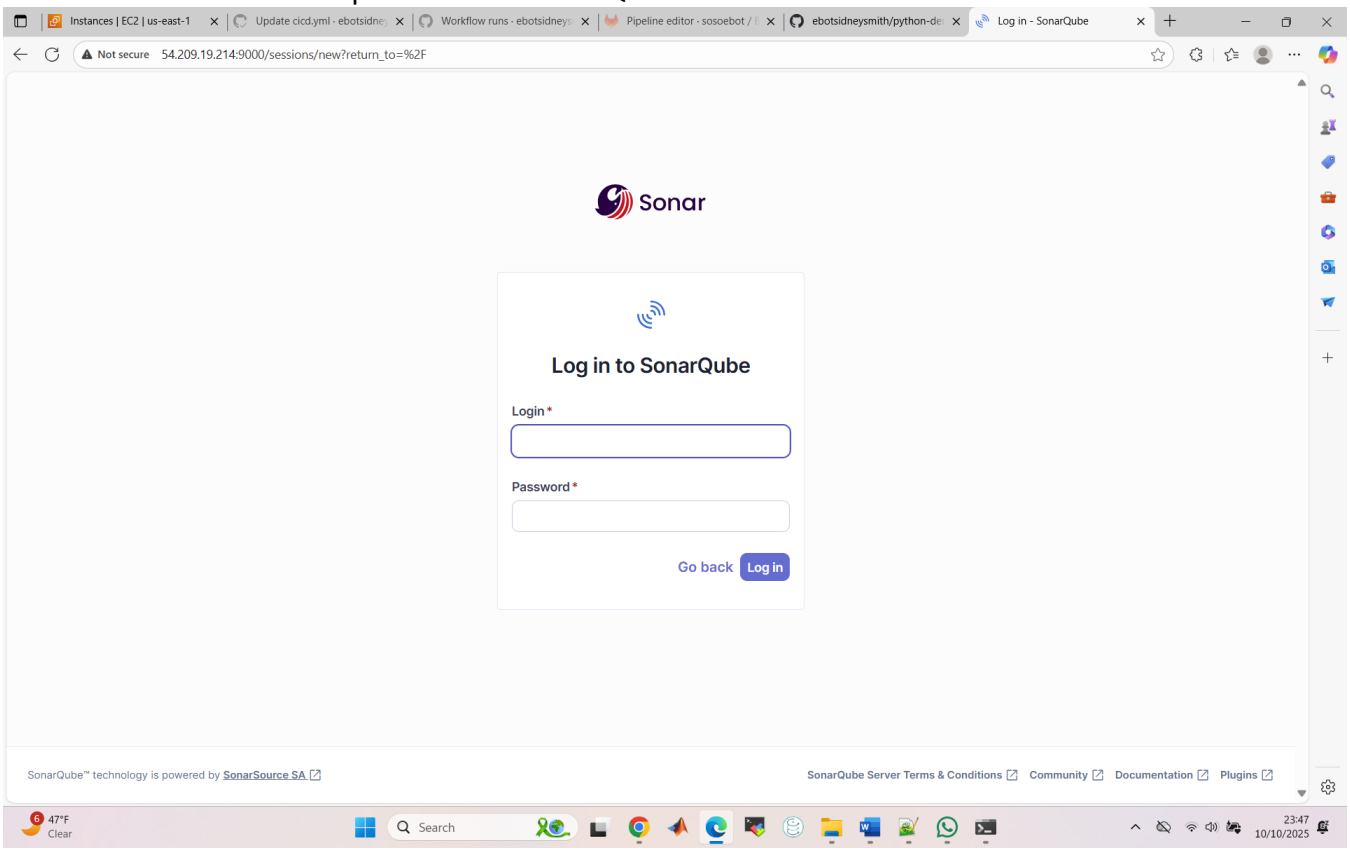
STEP 6: Access SonarQube on Browser

We will now access SonarQube on the browser using

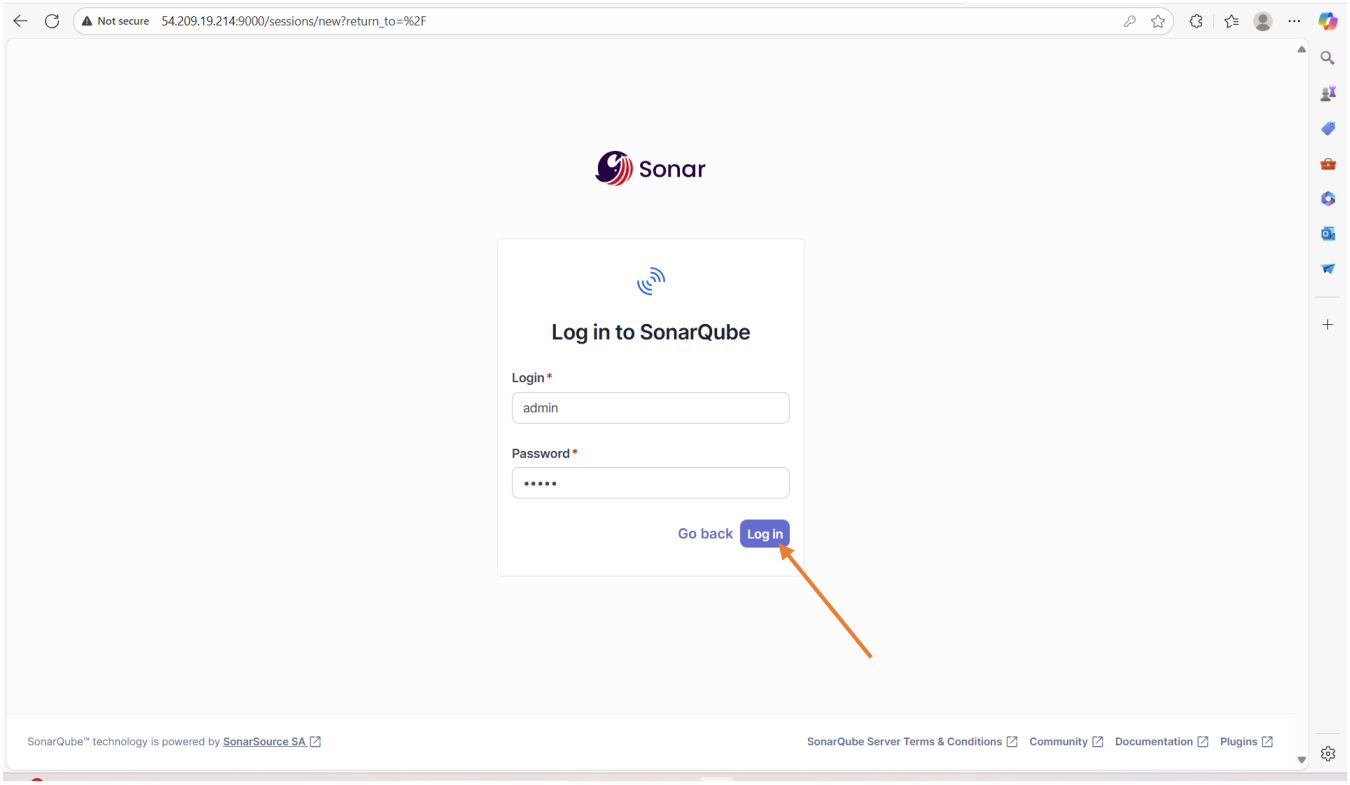
http://<Public IPv4 address>:9000

That is **http://54.209.19.214:9000**

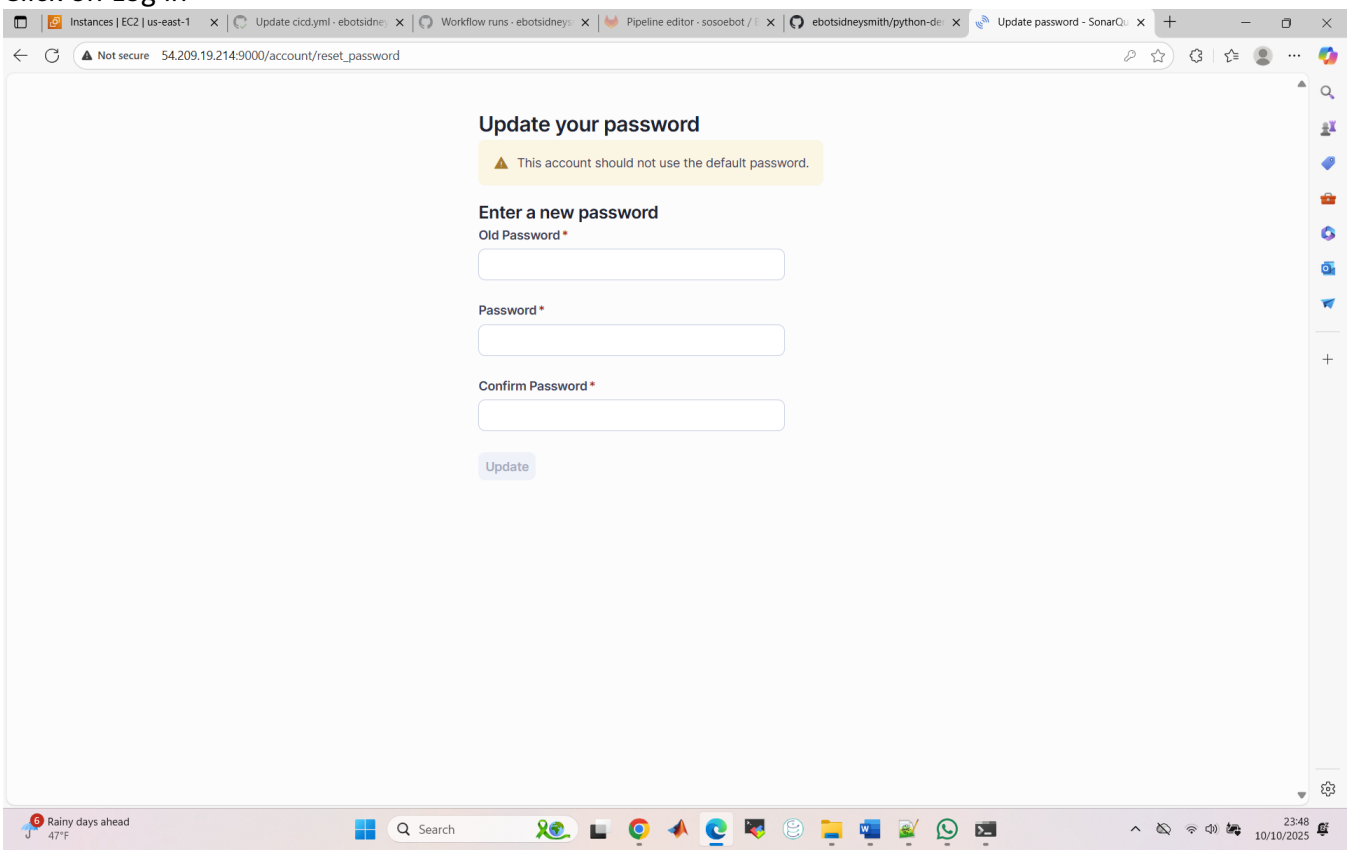
The default username and password for SonarQube are both "**admin**".



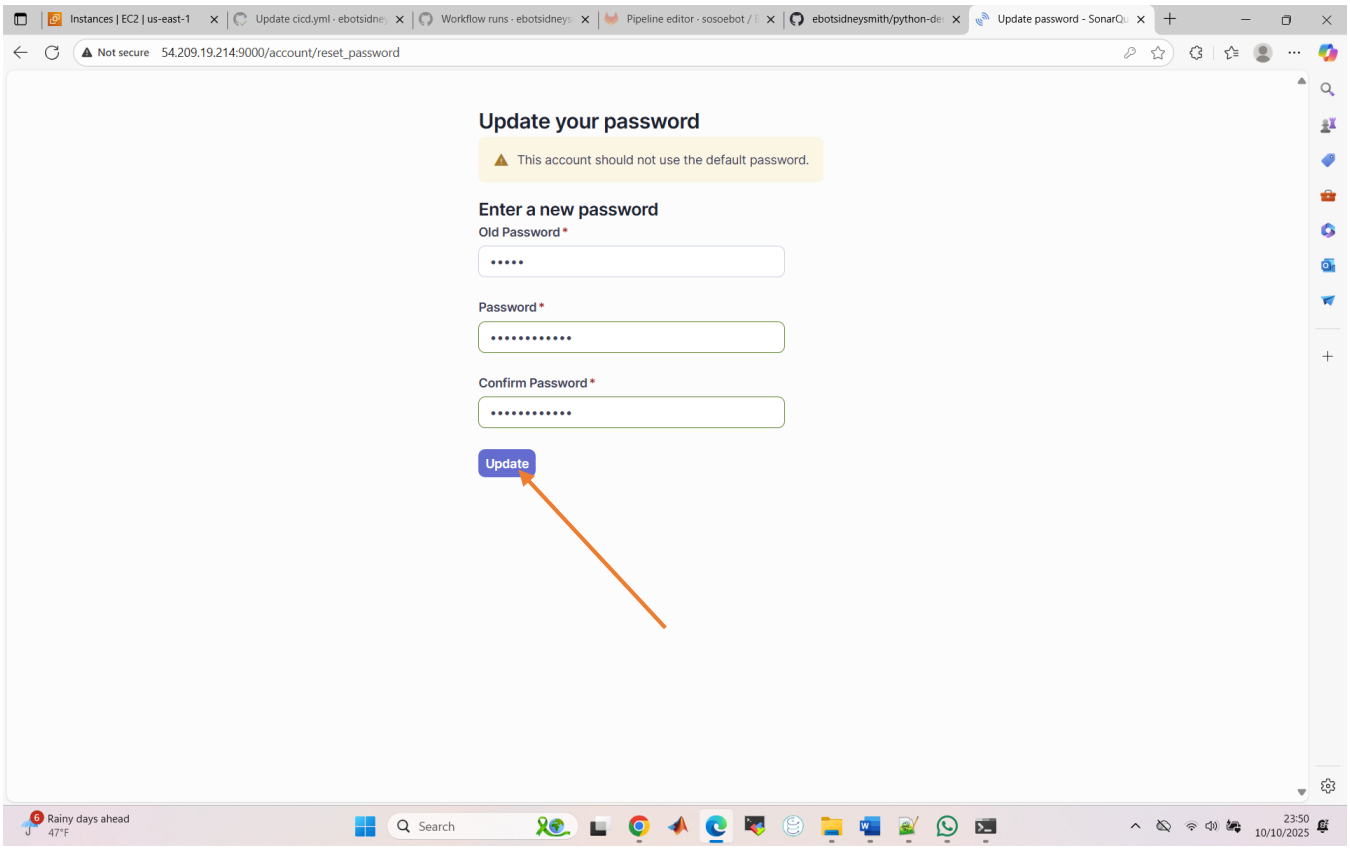
Enter the default username and password "**admin**"



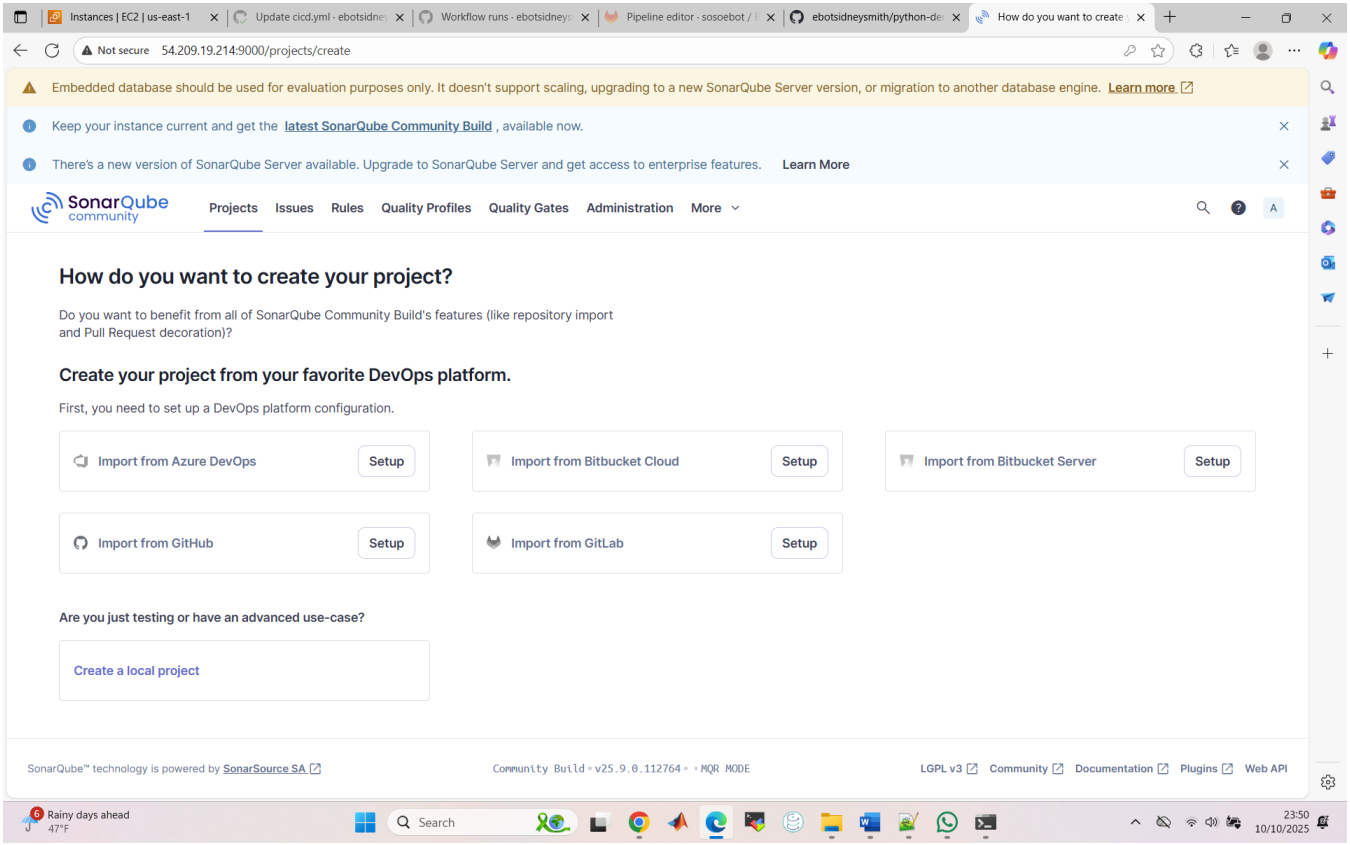
Click on Log in



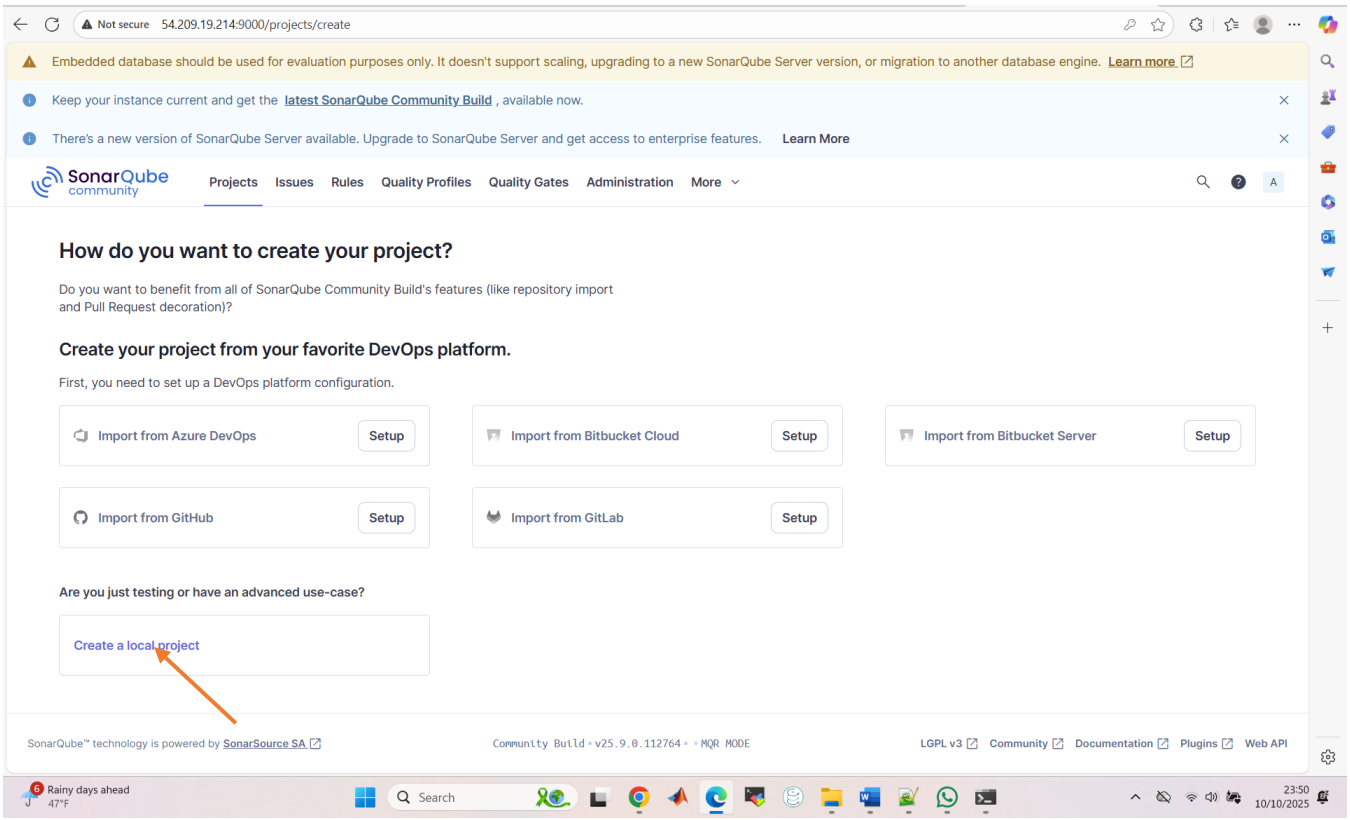
Then modify the password by entering the old password and your new password. Then click on "**Update**"



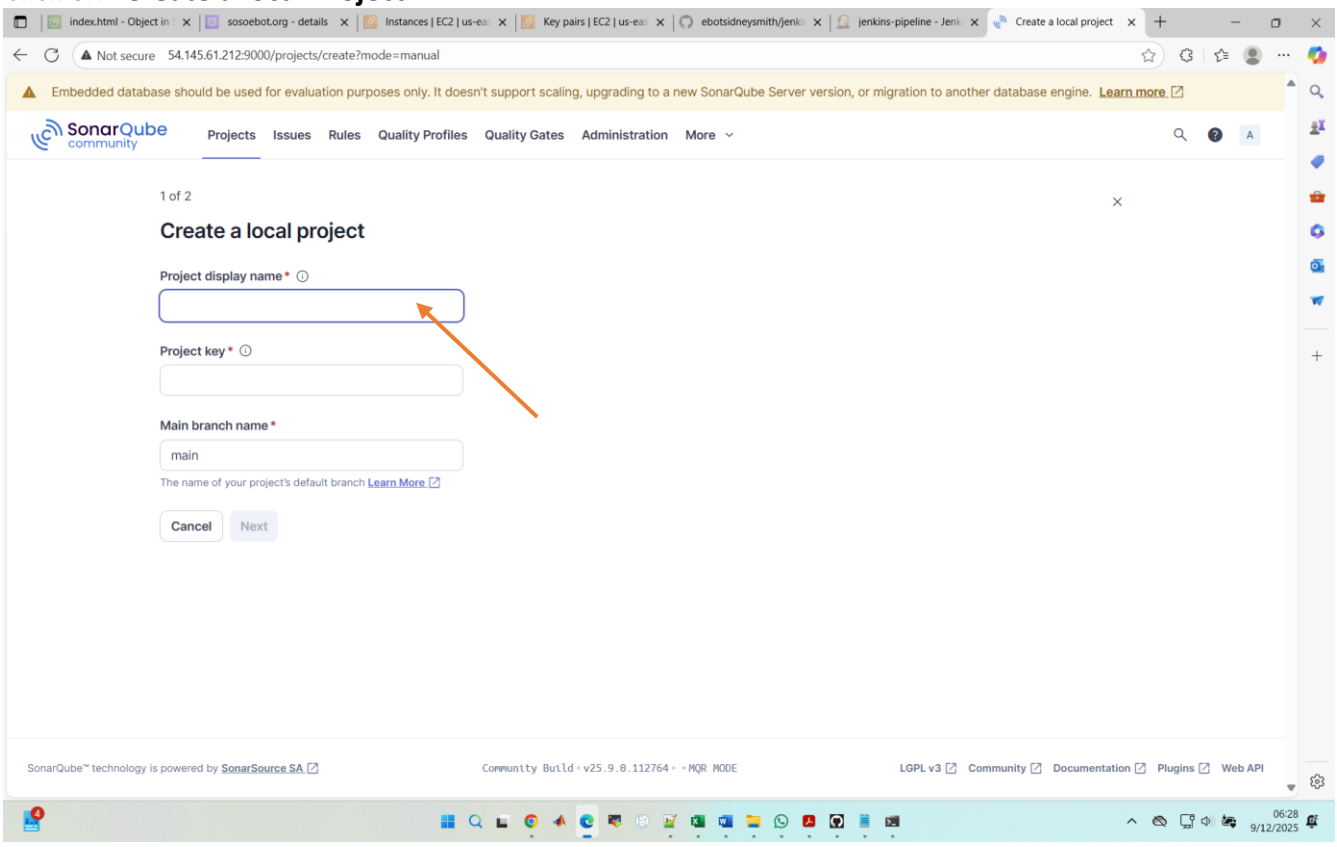
Click on “**Update**”



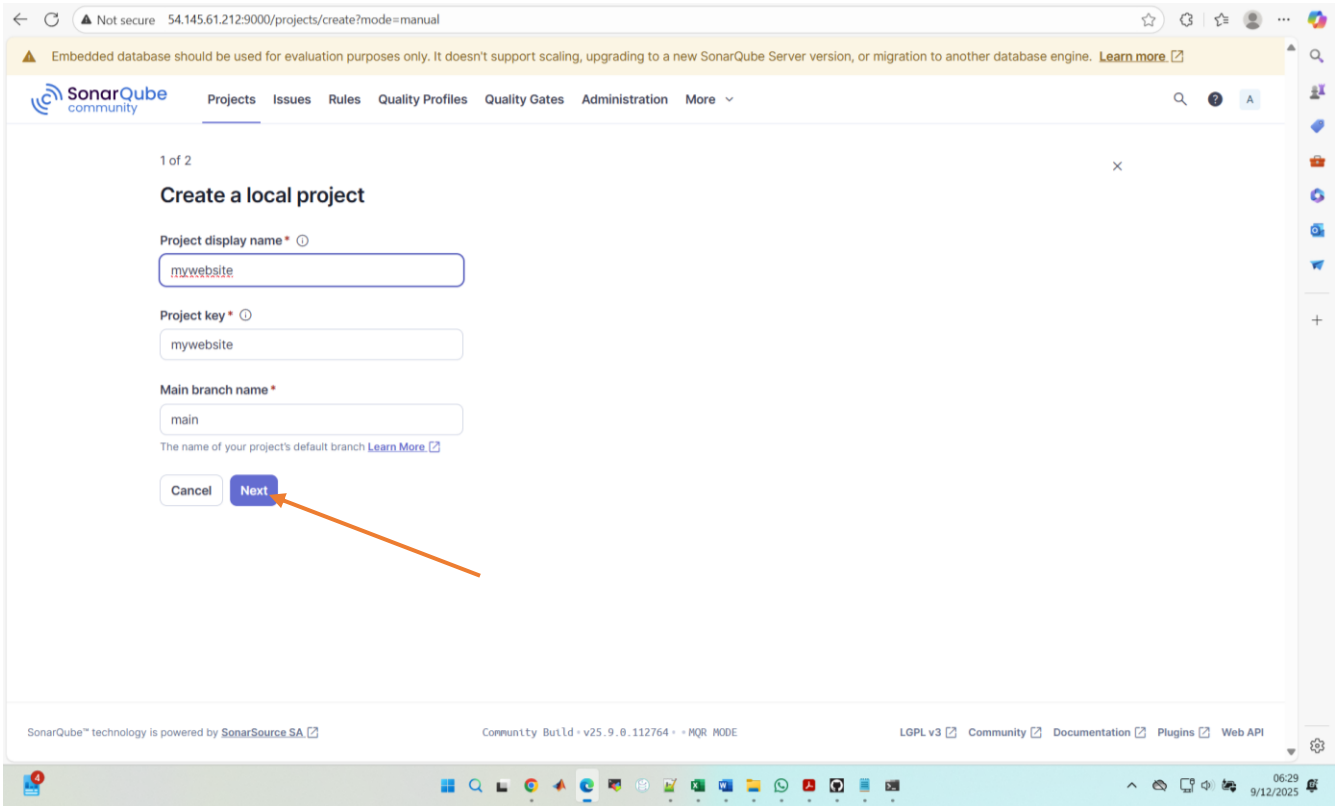
Part 1: Create a Local Project



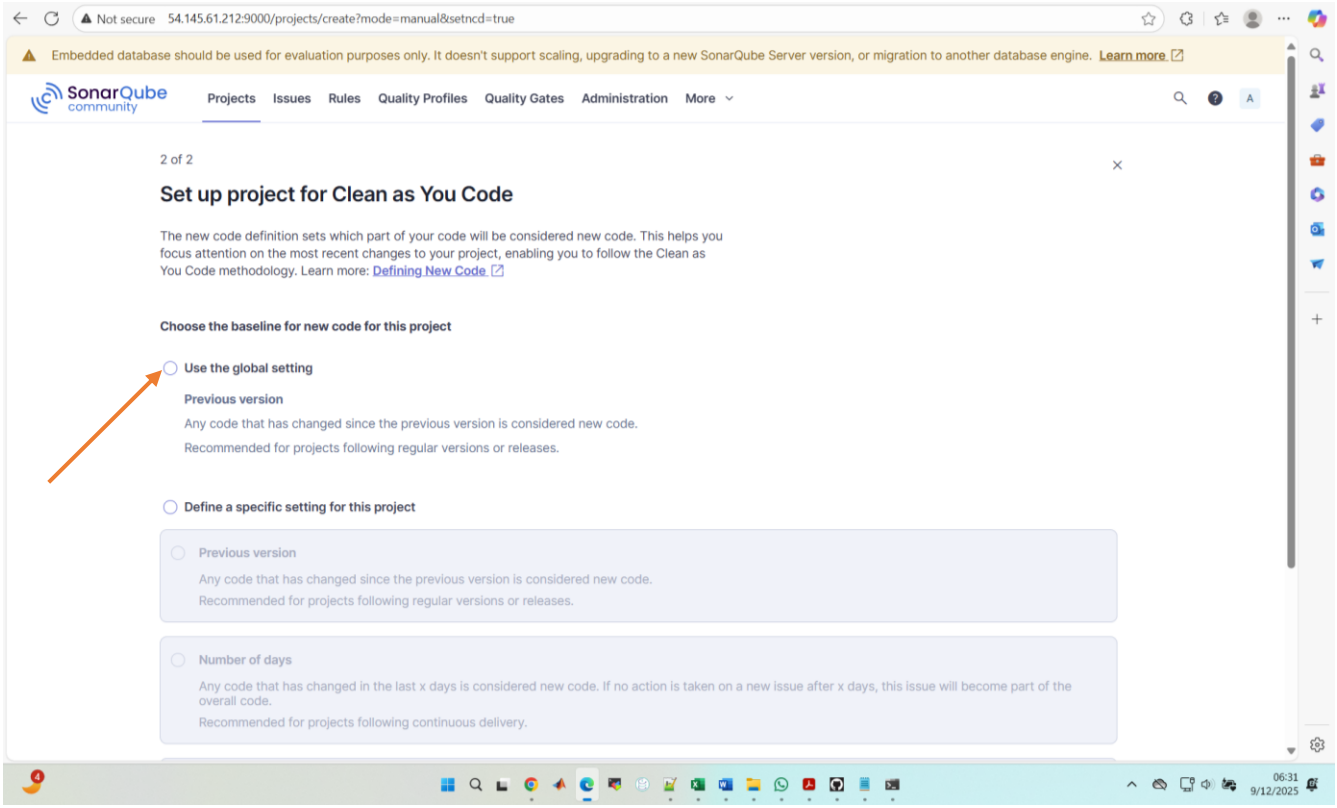
Click on "**Create a local Project**"



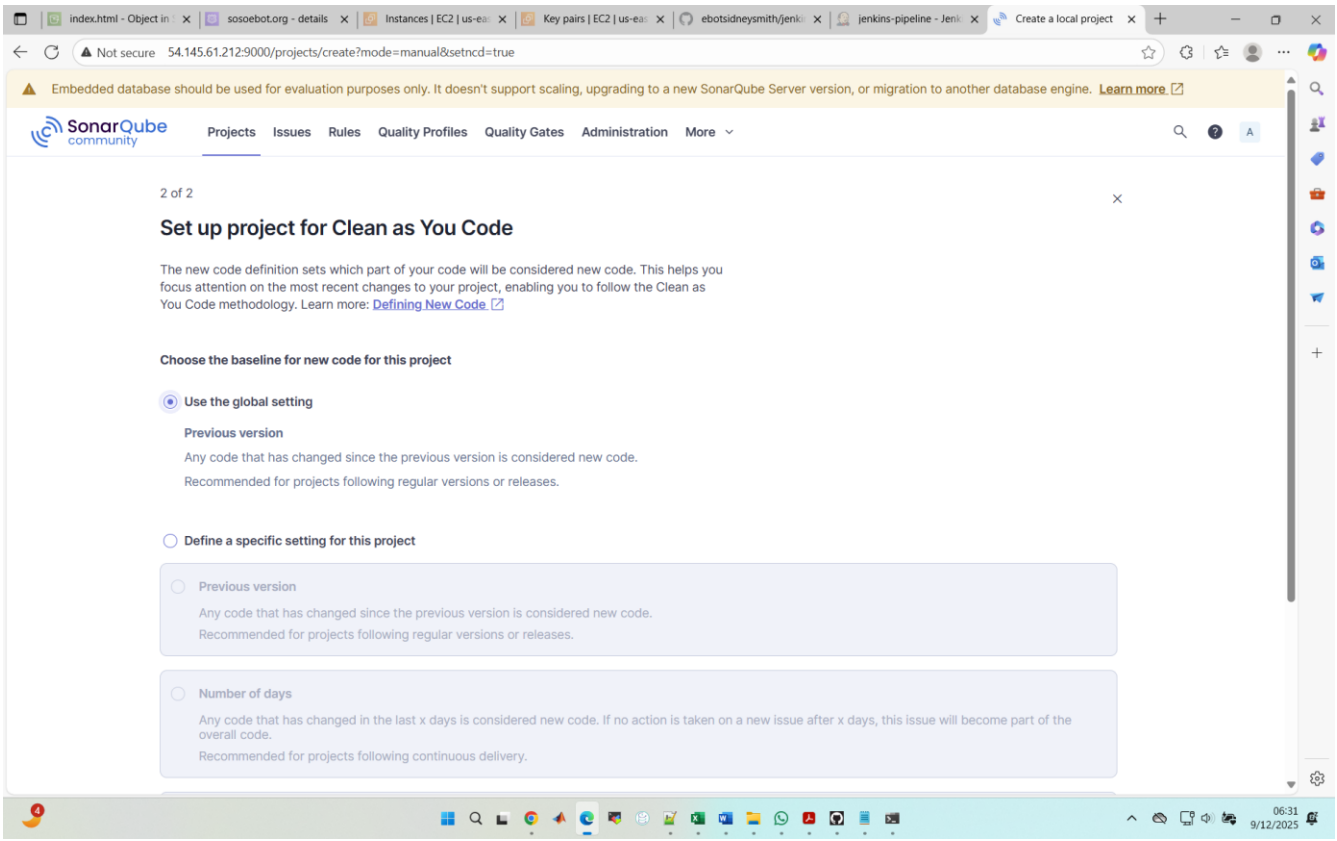
Give the project a name, I will call it "**mywebsite**"



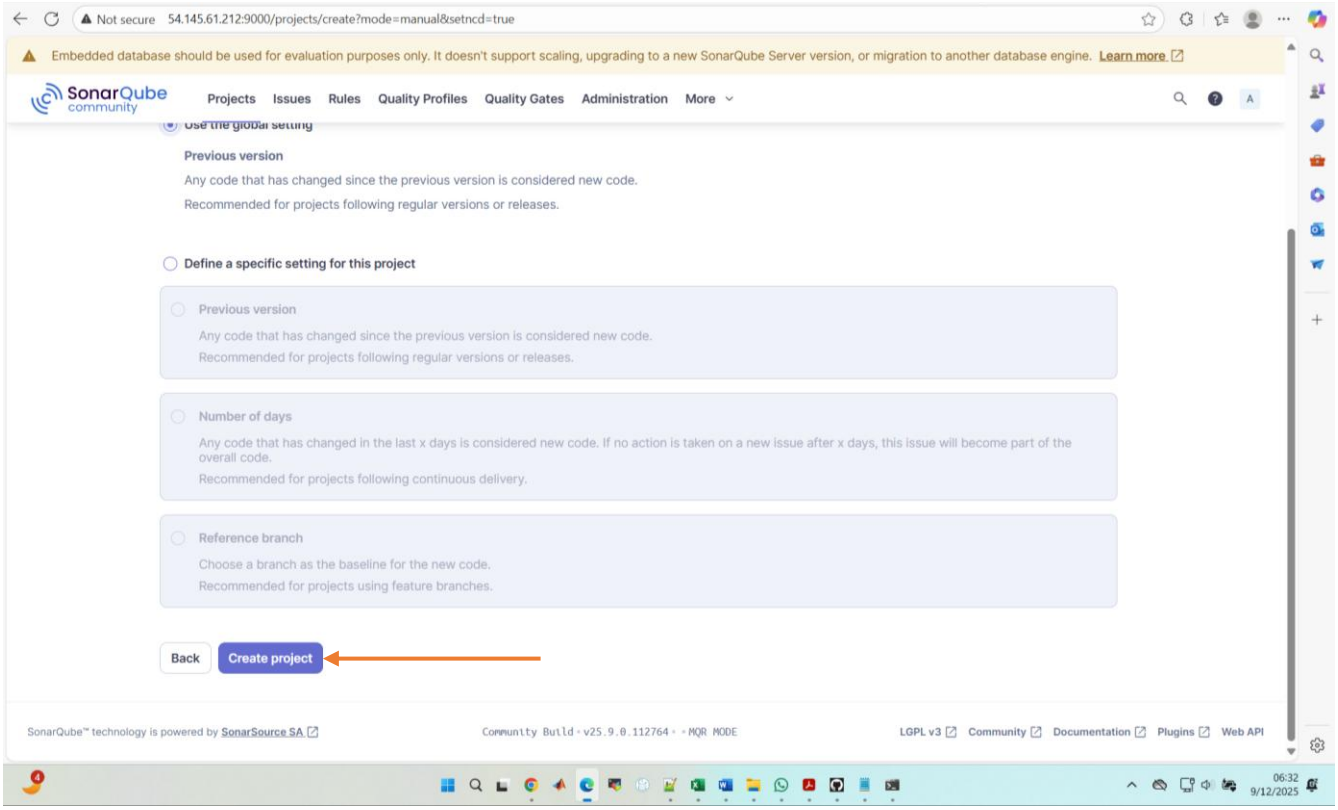
Click on "**Next**"



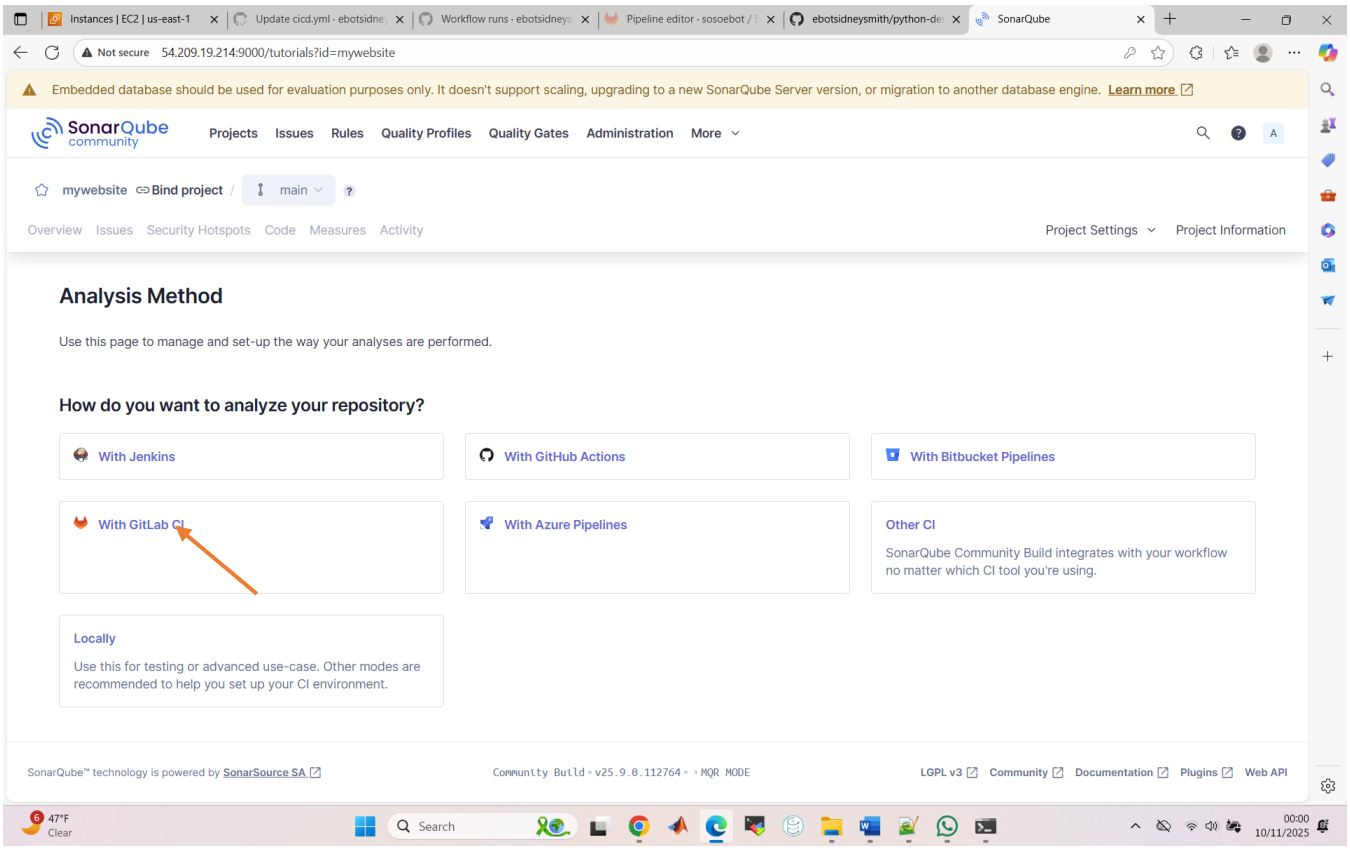
Select "**Use the global setting**"



Scroll down



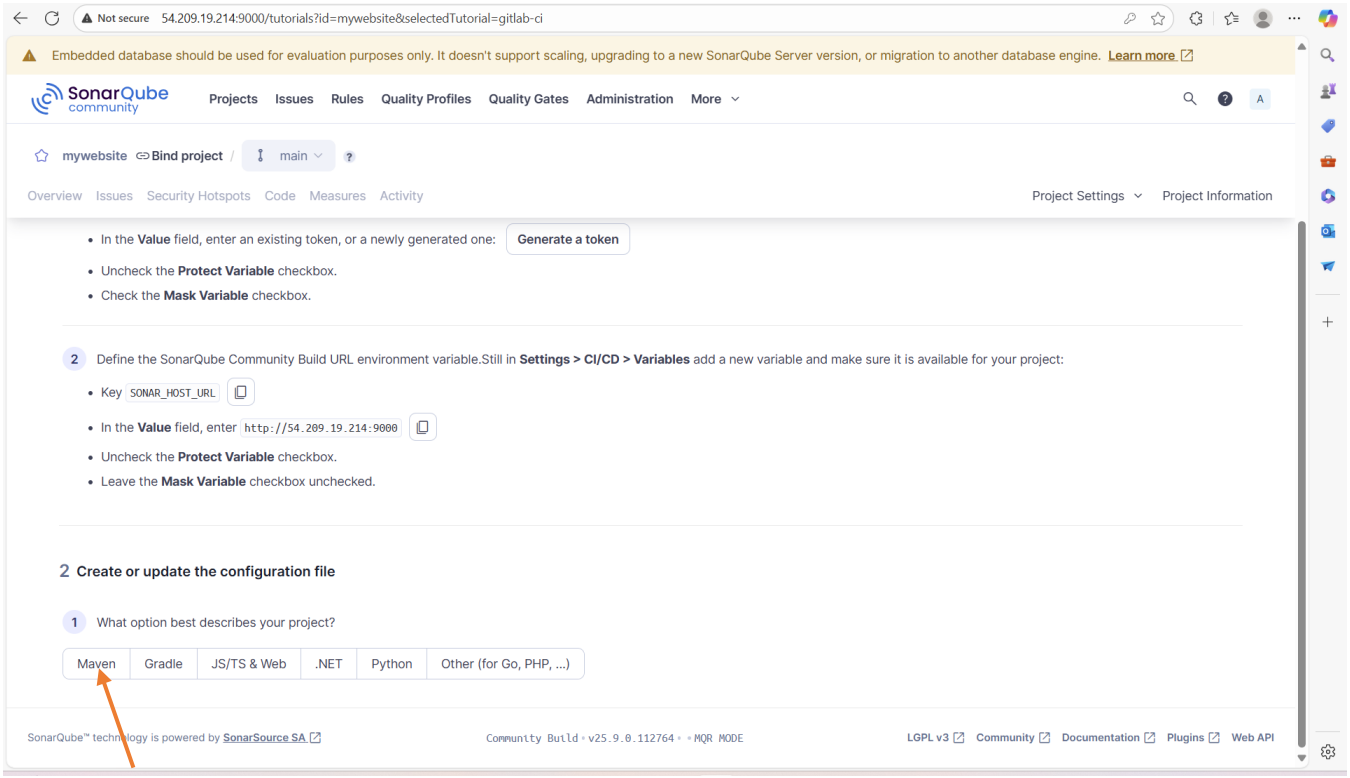
Click on “**create project**”



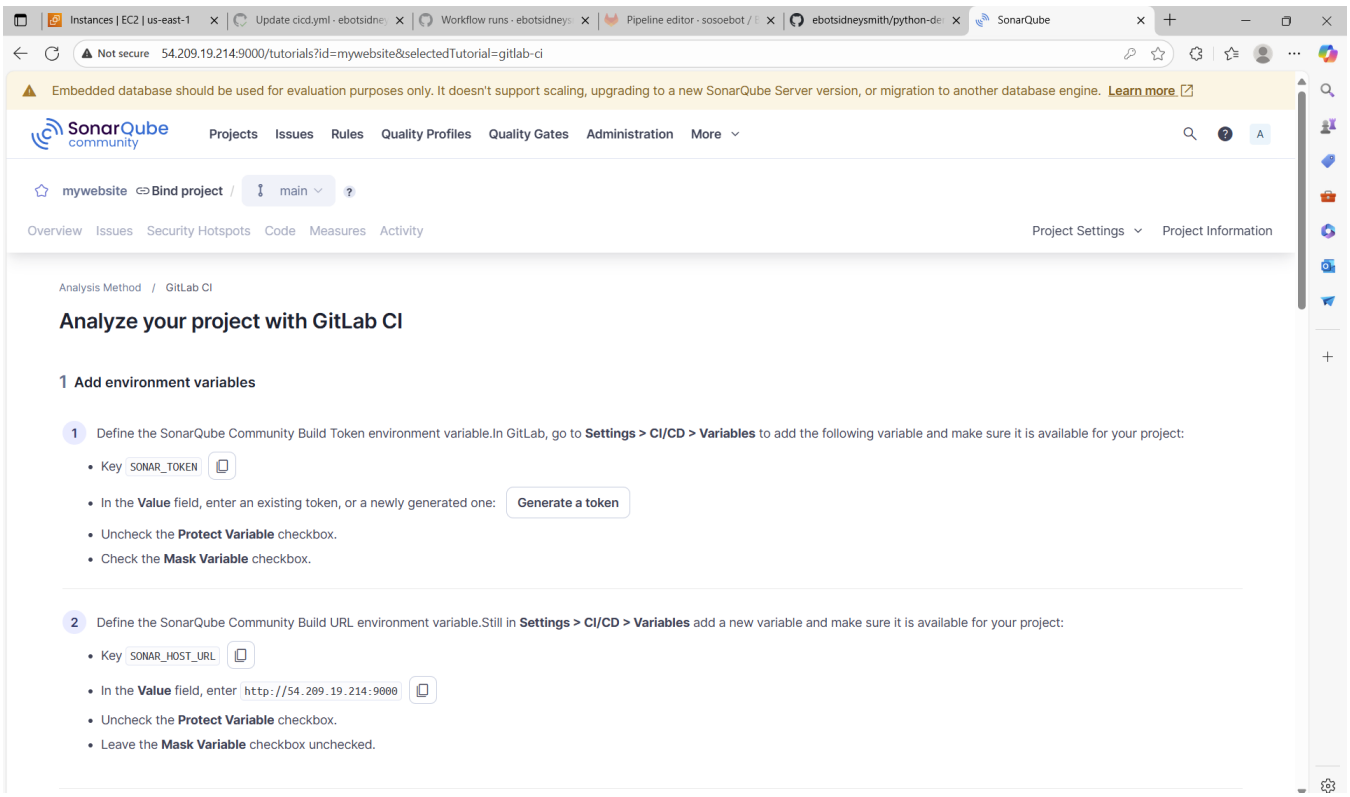
Our project has been created.

Part 3: Setup the Repository

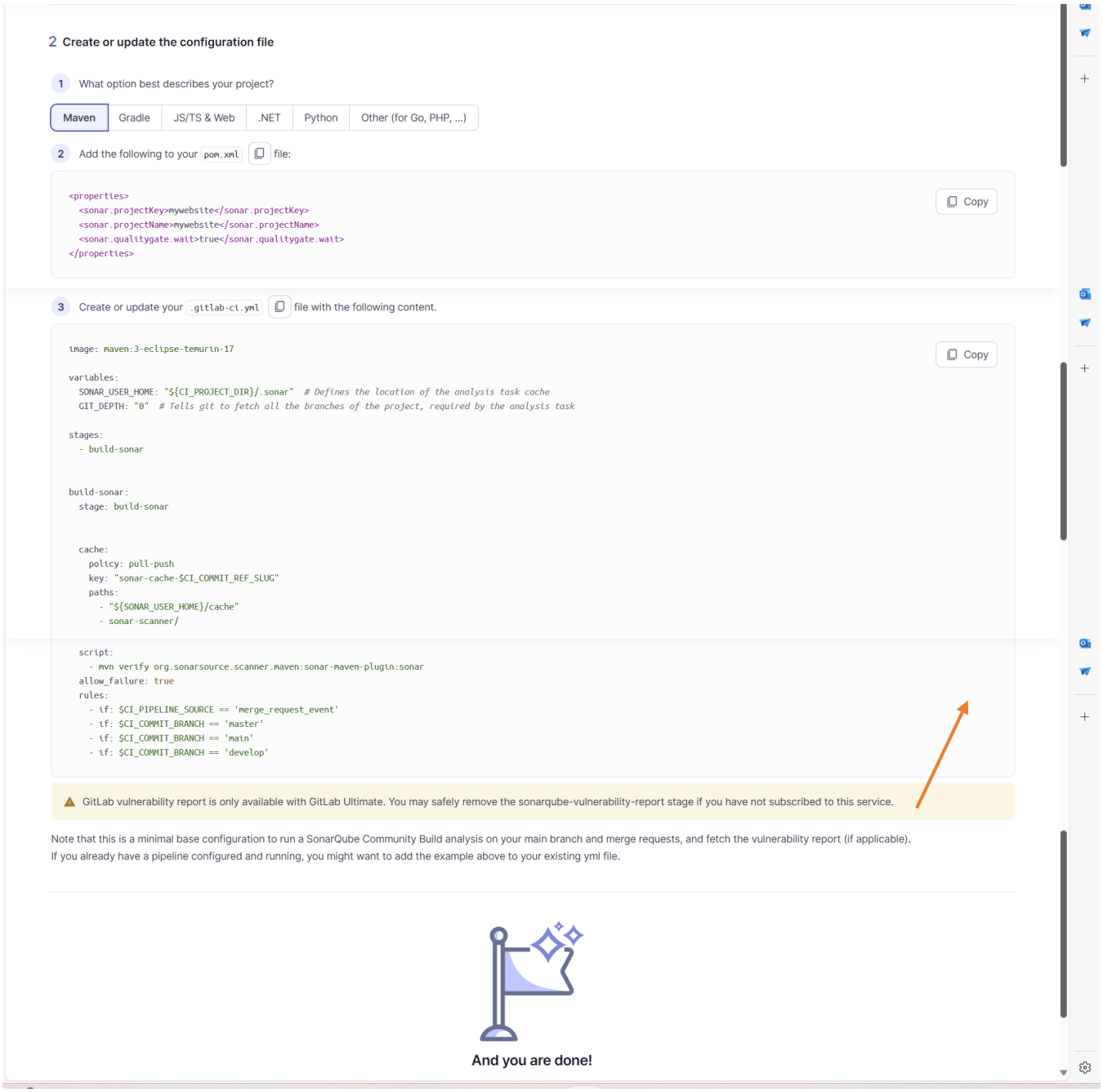
Now, let us setup the repository that SonarQube can analyze. We will use Jenkins, so click on "**With Gitlab**"



Here you can run analysis of your project, so specify the code that you are using on your project. Since we are using a Java-based code, select "**Maven**"



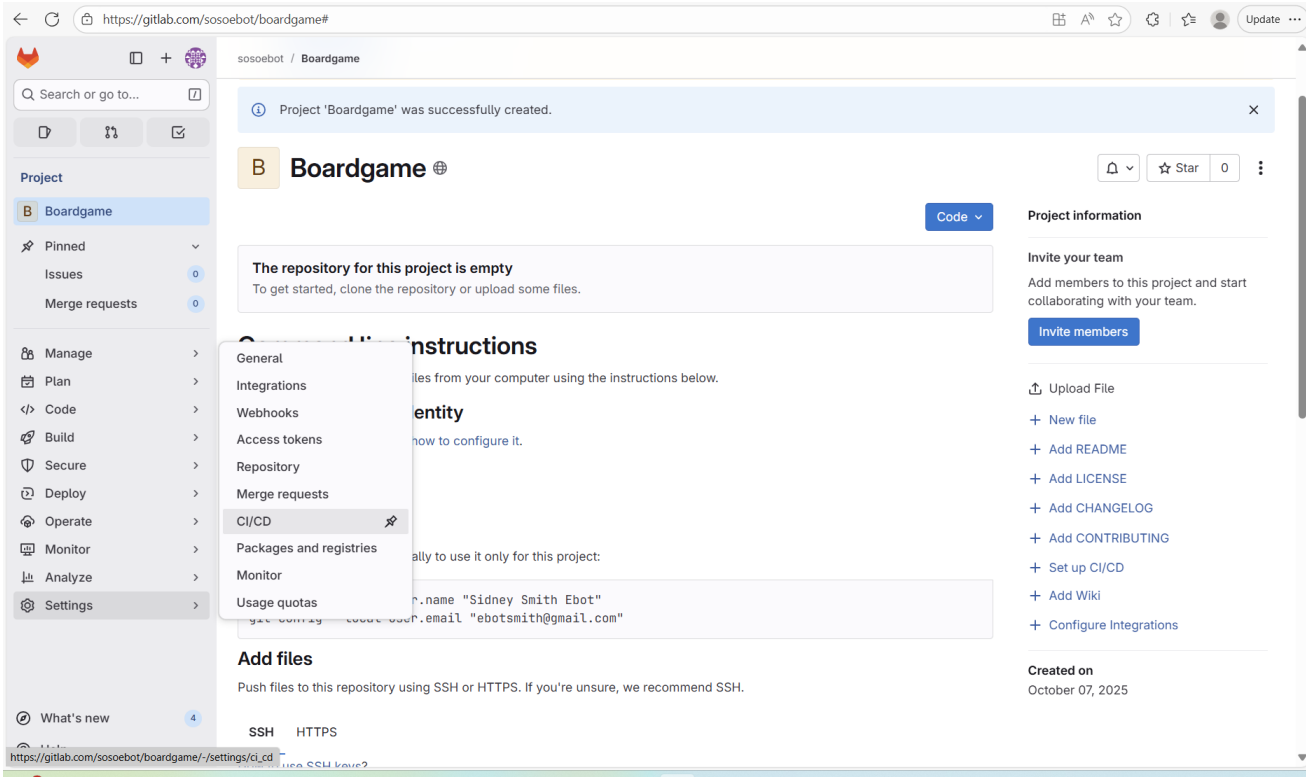
Scroll down



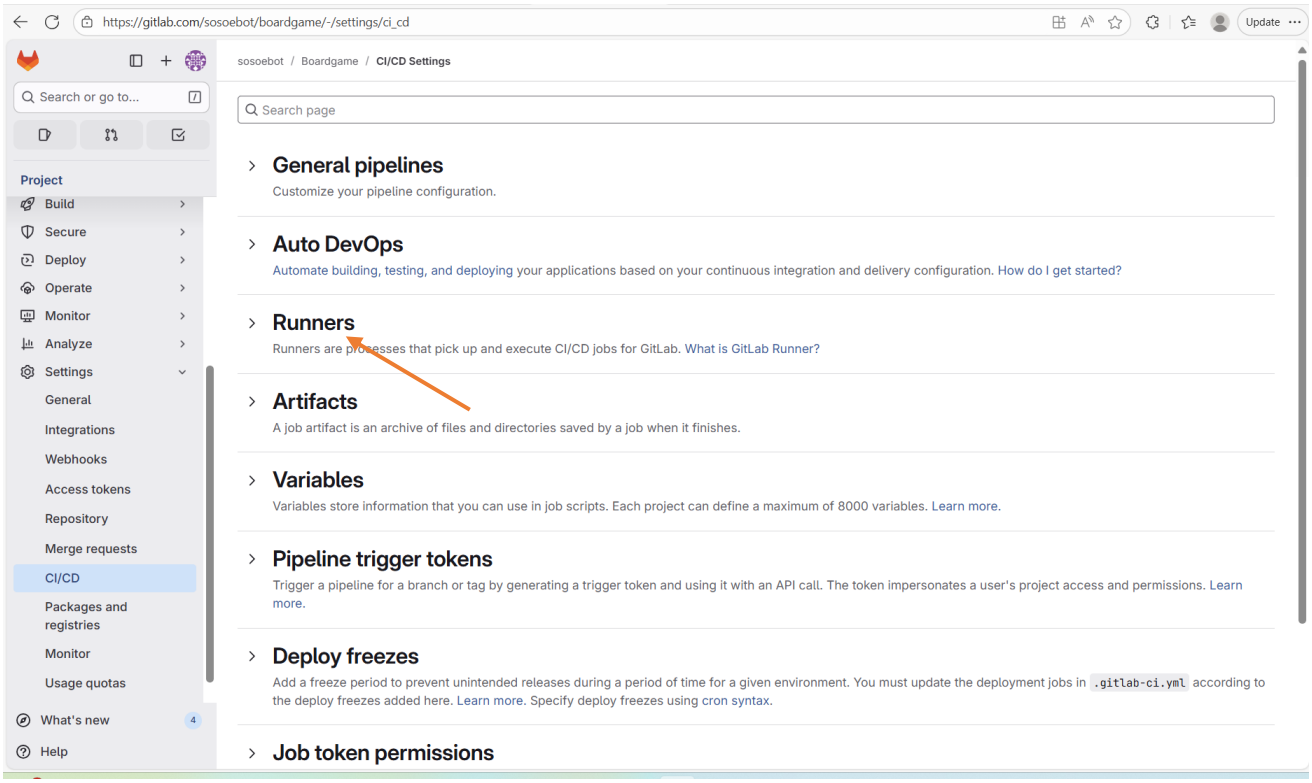
We will use the above information when writing the job for SonarQube.

STEP 7: Installing Gitlab Runner on Ubuntu EC2 Instance

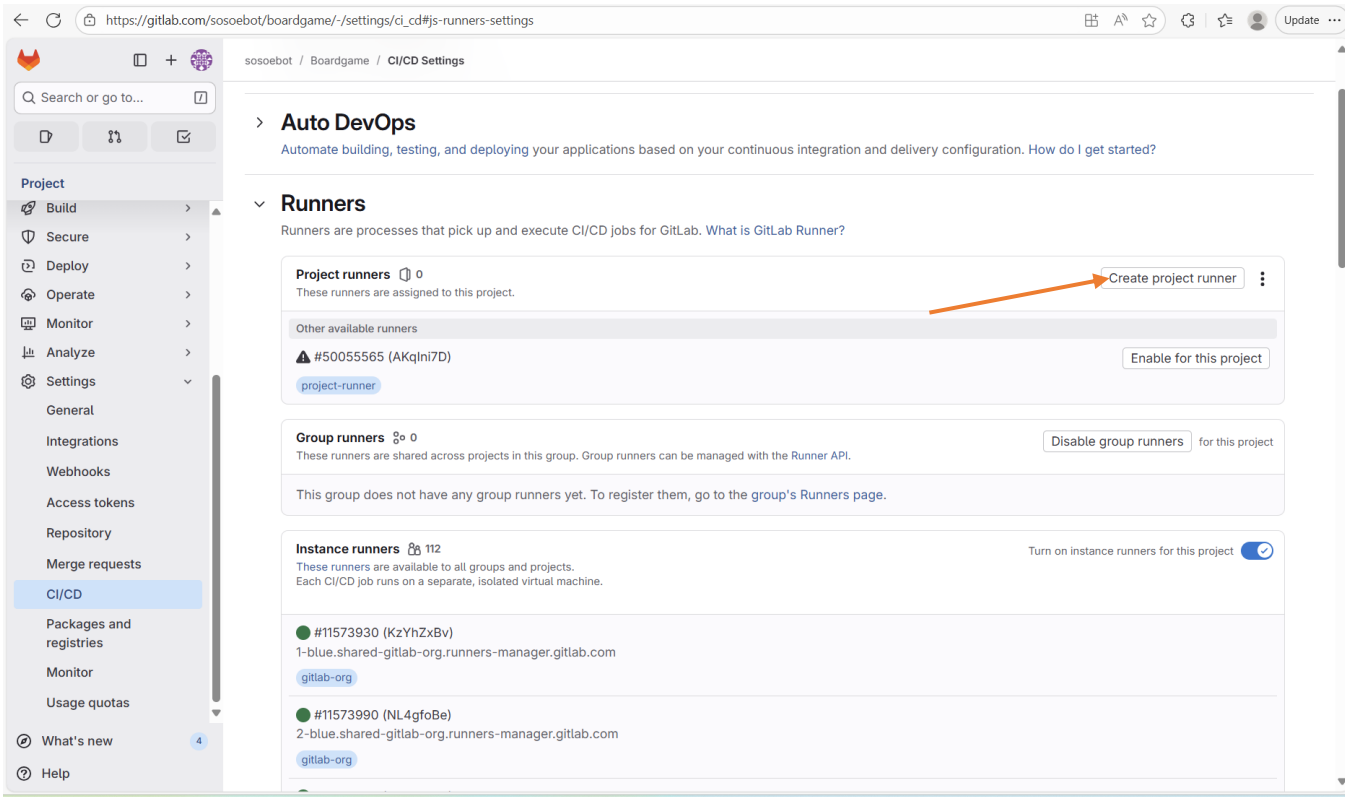
Part 1: Create Gitlab runner
Go to the Gitlab Project



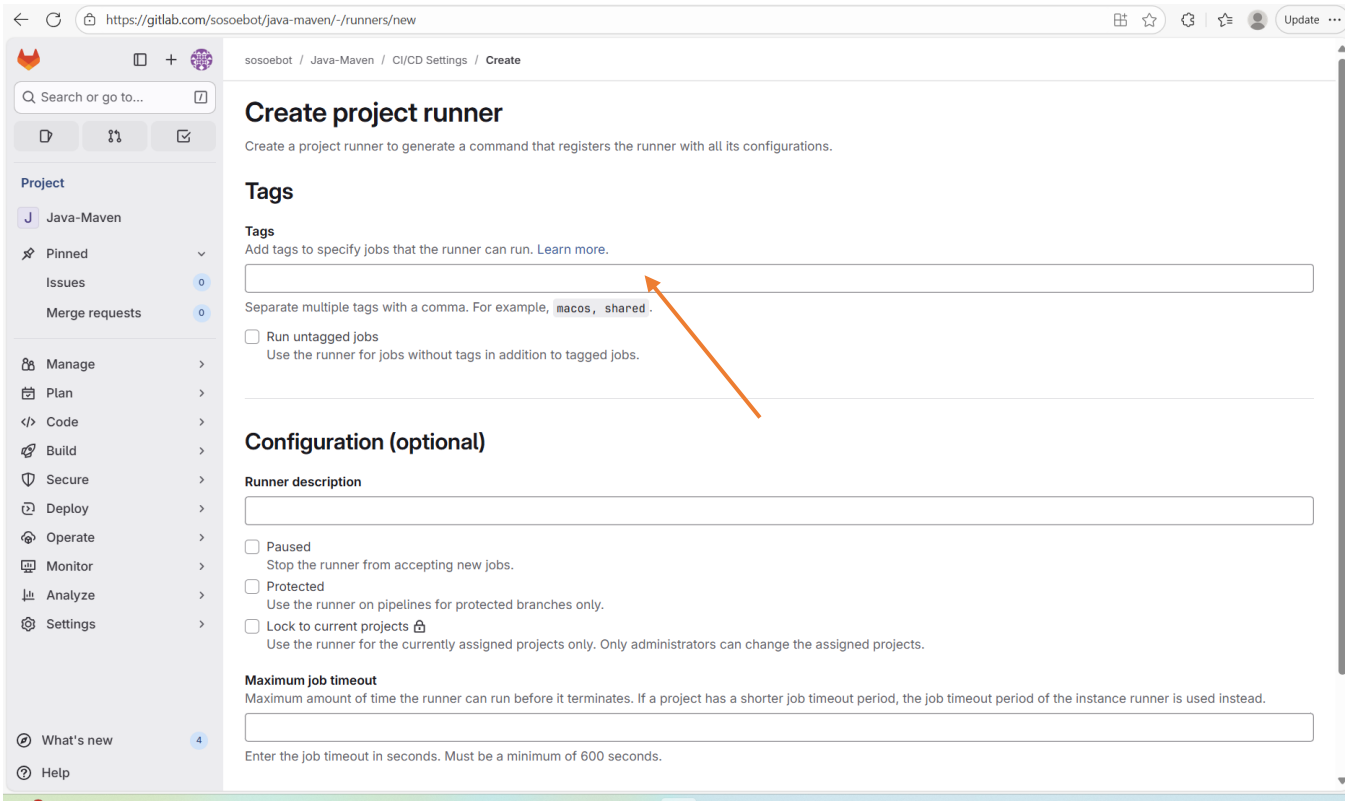
Click on "**Settings**" and select "**CI/CD**"



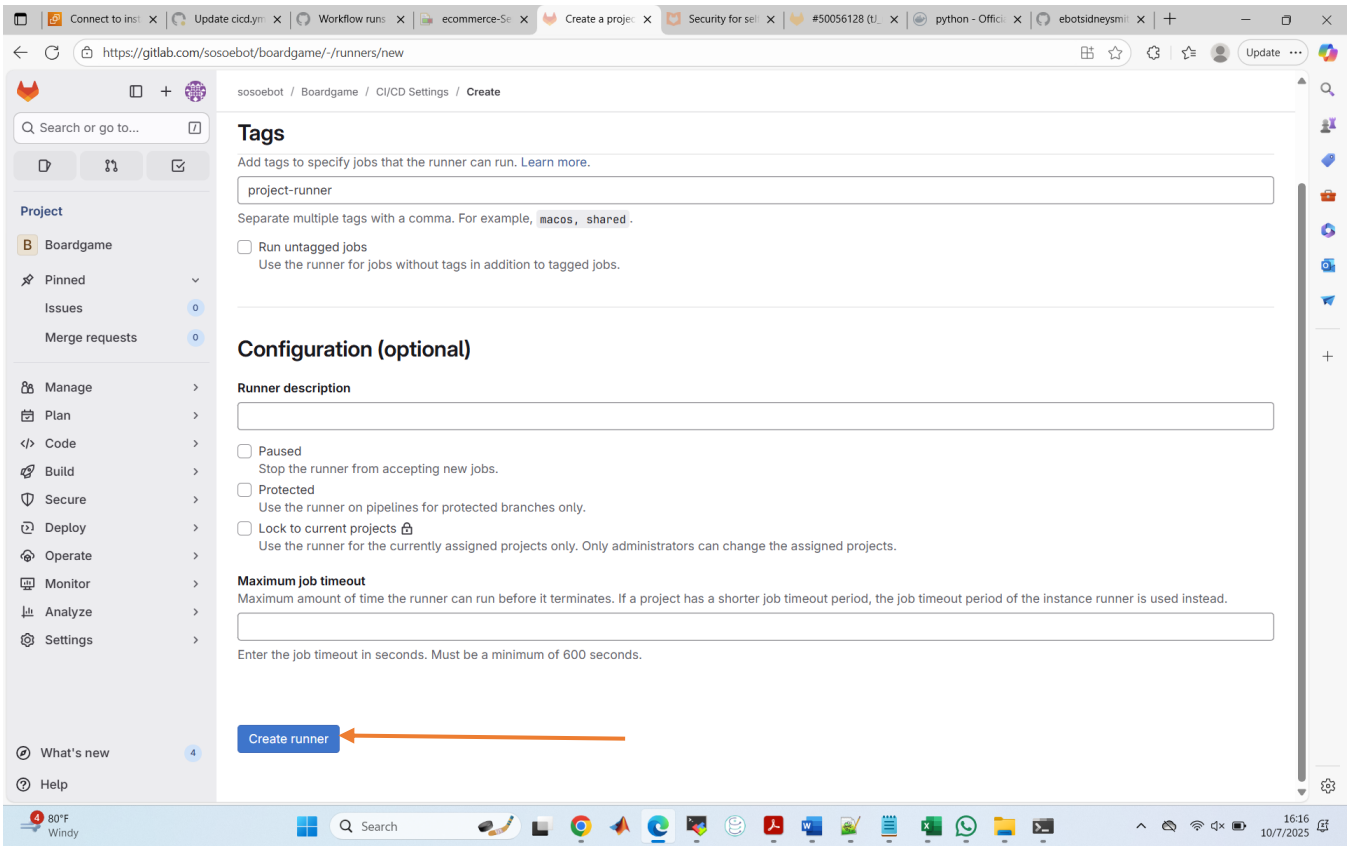
Click on “**Runners**”



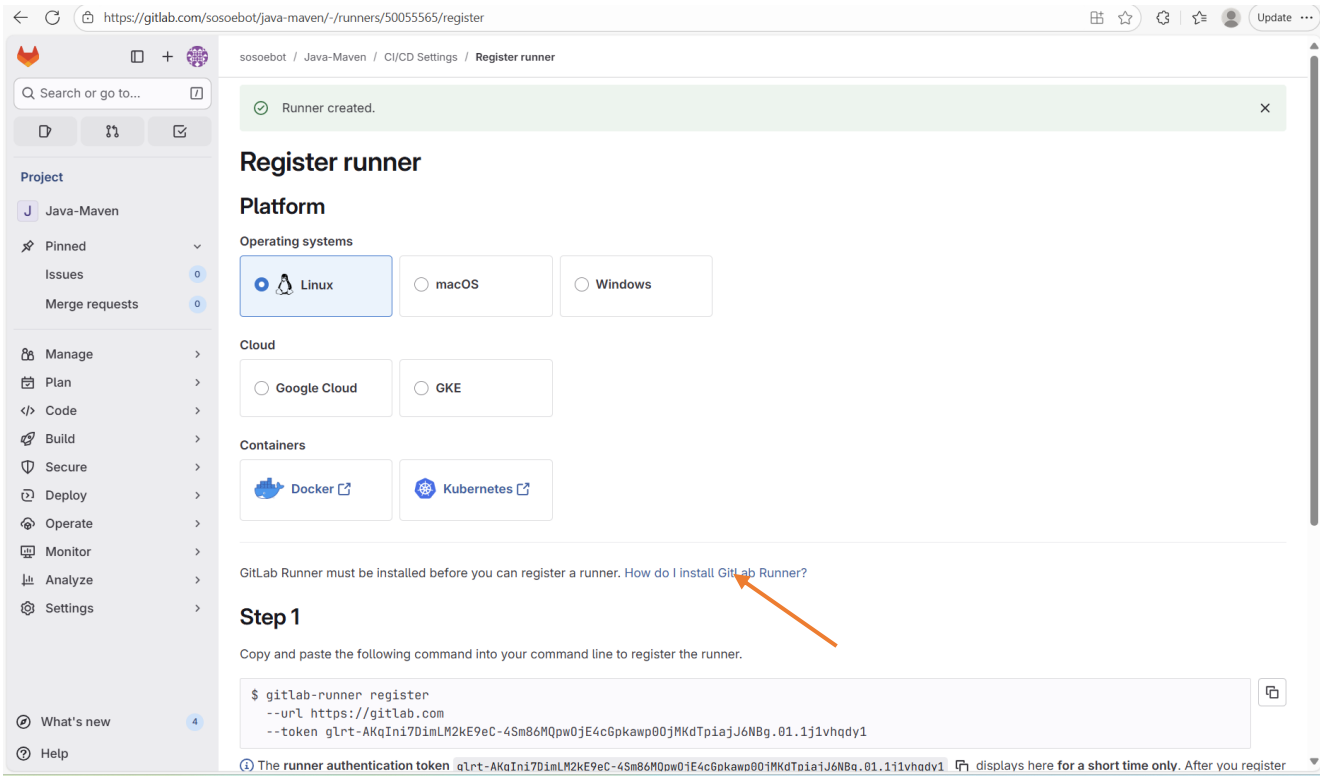
Click on “**Create project runner**”



Give it a name. I will call it “**project-runner**”



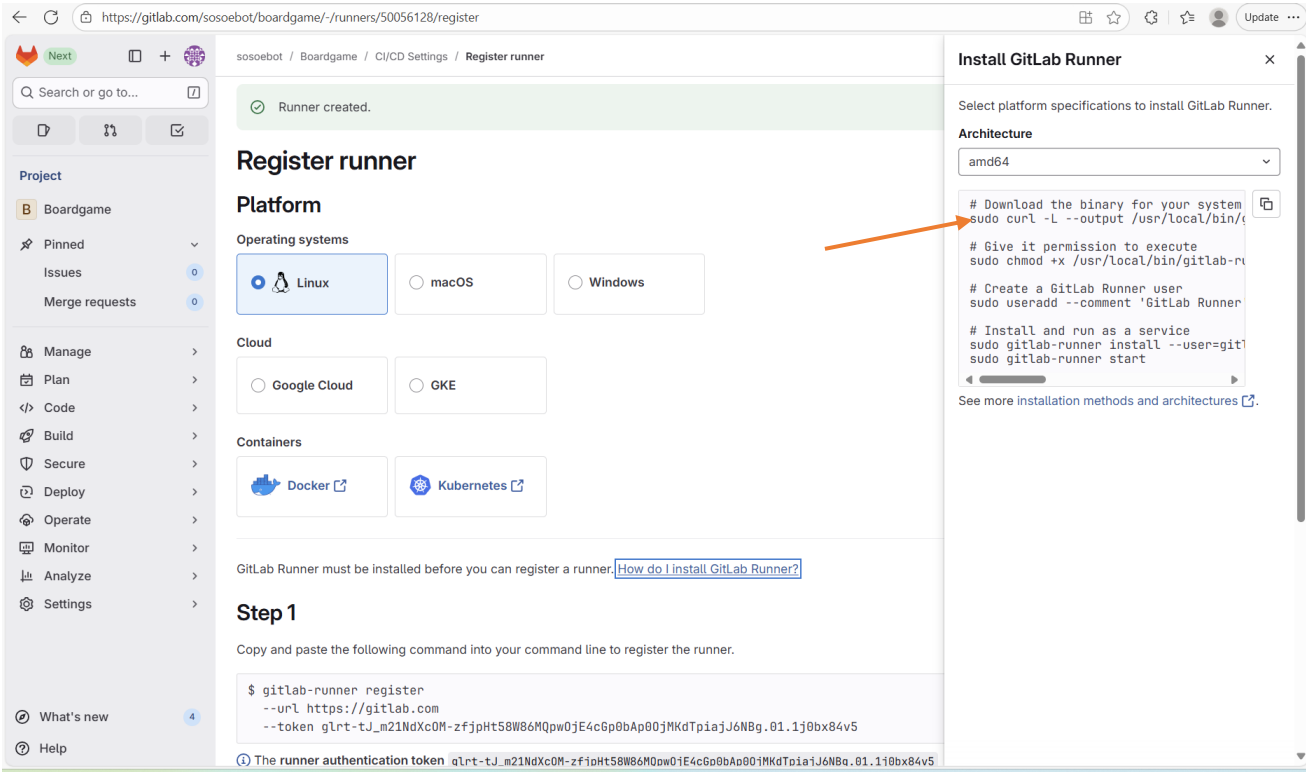
Click on "**Create Runner**"



My EC2 instance is ubuntu, so I will select "**Linux**".

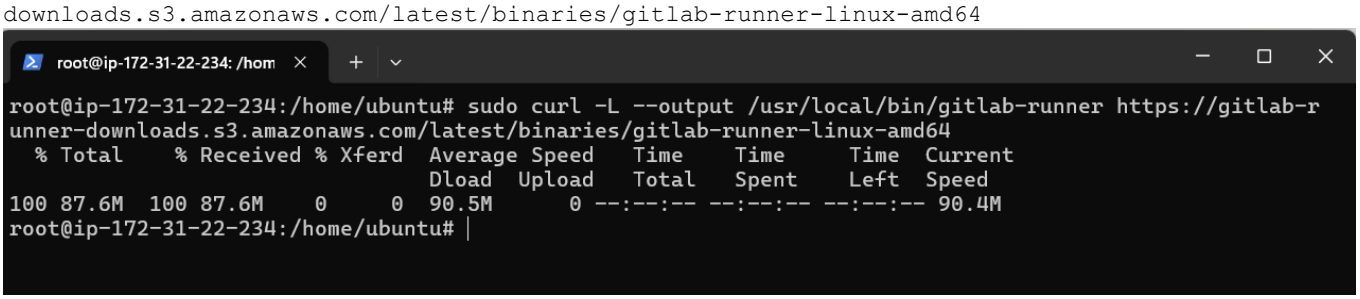
Part 2: Install Gitlab Runner

Click on "**How do I install Gitlab Runner**"



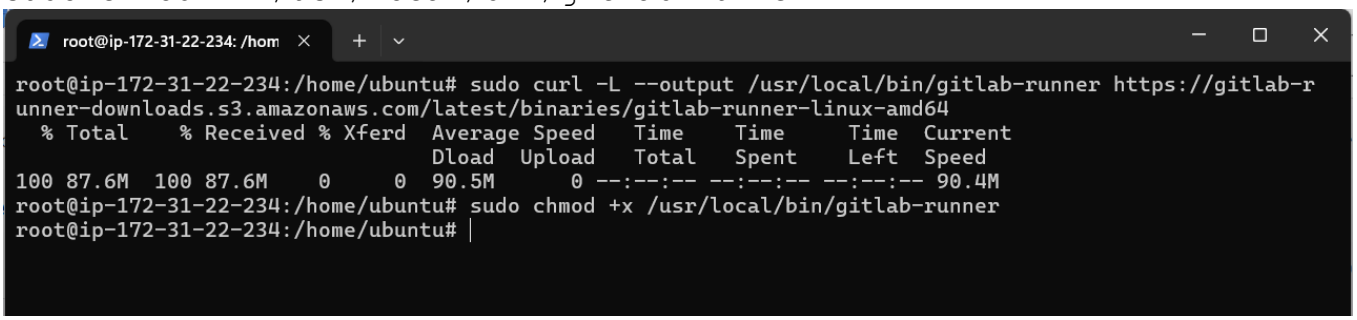
Copy the first line of code and paste on the terminal to Download the binary for your system

```
sudo curl -L --output /usr/local/bin/gitlab-runner https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-amd64
```



Give it permission to execute

```
sudo chmod +x /usr/local/bin/gitlab-runner
```



Create a GitLab Runner user

```
sudo useradd --comment 'GitLab Runner' --create-home gitlab-runner --shell /bin/bash
```

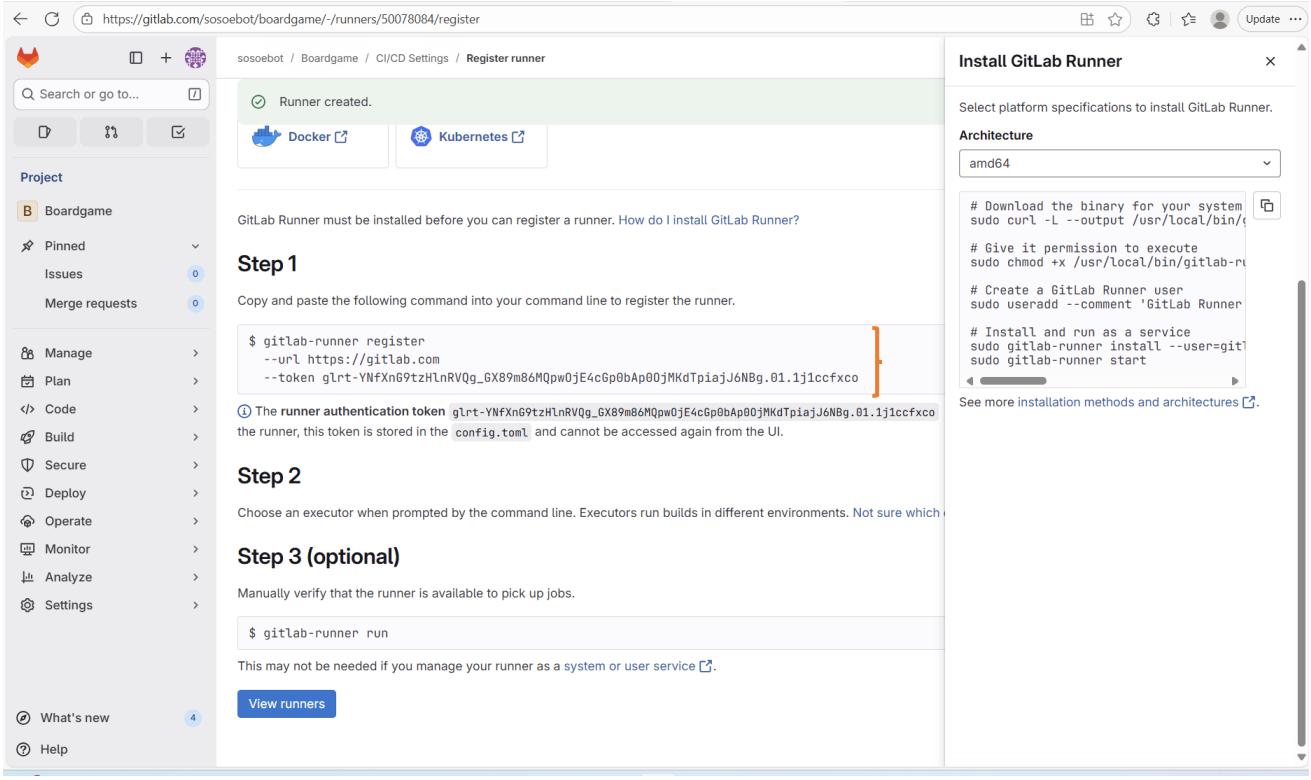
```
root@ip-172-31-22-234:/home/ubuntu# sudo curl -L --output /usr/local/bin/gitlab-runner https://gitlab-r
unner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 87.6M  100 87.6M    0     0  90.5M      0 --:--:-- --:--:-- --:--:-- 90.4M
root@ip-172-31-22-234:/home/ubuntu# sudo chmod +x /usr/local/bin/gitlab-runner
root@ip-172-31-22-234:/home/ubuntu# sudo useradd --comment 'GitLab Runner' --create-home gitlab-runner
--shell /bin/bash
root@ip-172-31-22-234:/home/ubuntu#
```

Install and run as a service

```
sudo gitlab-runner install --user=gitlab-runner --working-directory=/home/gitlab-runner
sudo gitlab-runner start
```

```
root@ip-172-31-22-234:/home/ubuntu# sudo curl -L --output /usr/local/bin/gitlab-runner https://gitlab-runner-downloads.
s3.amazonaws.com/latest/binaries/gitlab-runner-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 87.6M  100 87.6M    0     0  90.5M      0 --:--:-- --:--:-- --:--:-- 90.4M
root@ip-172-31-22-234:/home/ubuntu# sudo chmod +x /usr/local/bin/gitlab-runner
root@ip-172-31-22-234:/home/ubuntu# sudo useradd --comment 'GitLab Runner' --create-home gitlab-runner --shell /bin/bas
h
root@ip-172-31-22-234:/home/ubuntu# sudo gitlab-runner install --user=gitlab-runner --working-directory=/home/gitlab-ru
nner
sudo gitlab-runner start
Runtime platform                                    arch=amd64 os=linux pid=1341 revision=139a0ac0 version=18.4.0
Runtime platform                                    arch=amd64 os=linux pid=1437 revision=139a0ac0 version=18.4.0
root@ip-172-31-22-234:/home/ubuntu#
```

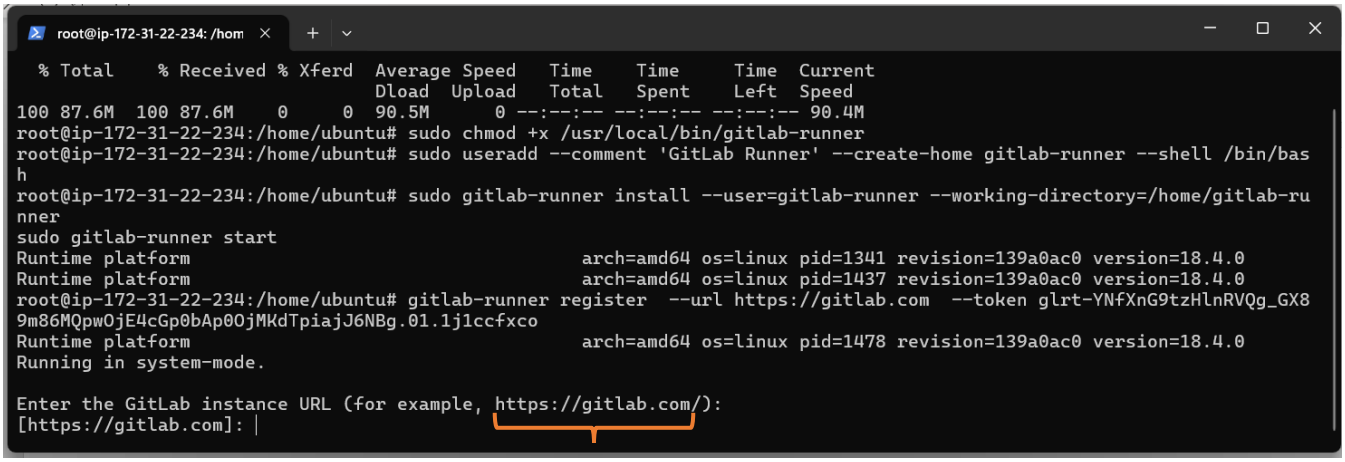
Head back to our Gitlab repository



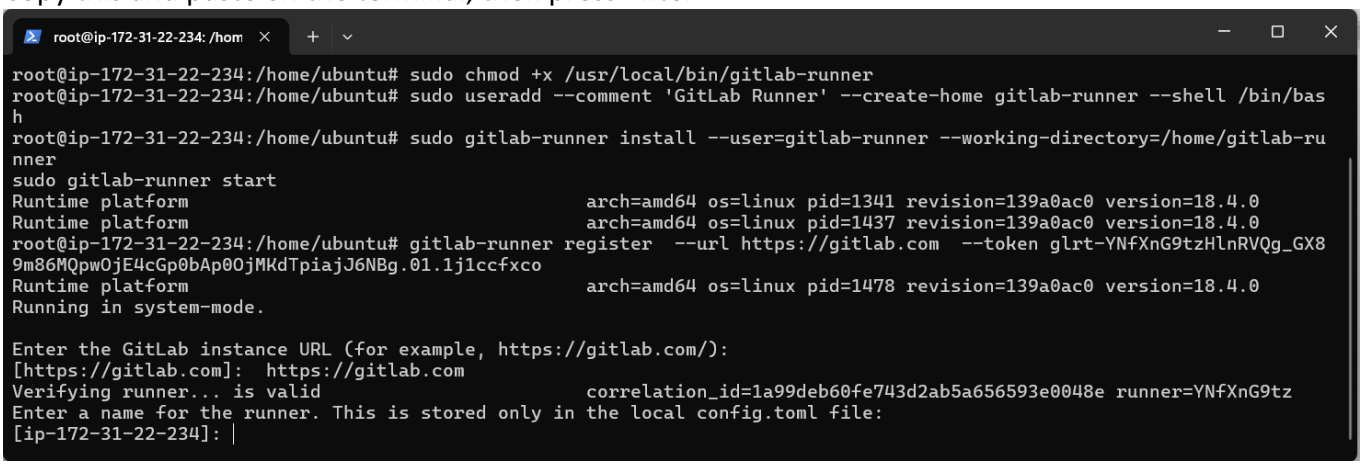
Then follow the steps

Copy and paste the following command into your command line to register the runner.

```
gitlab-runner register  --url https://gitlab.com  --token glrt-
YNfXnG9tzHlnRVQg_GX89m86MQpwOjE4cGp0bAp0OjMKdTpiajJ6NBg.01.1j1ccfxco
```

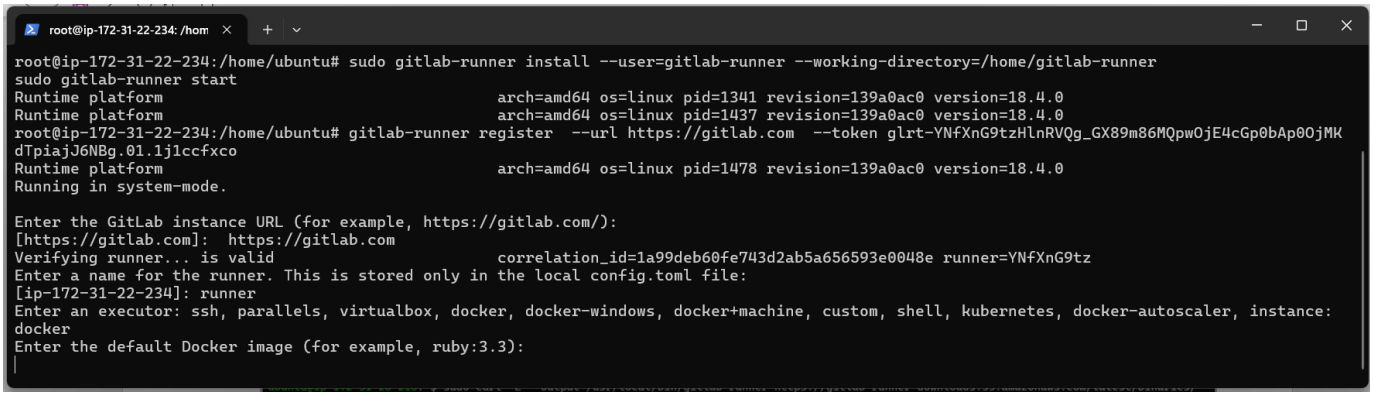


Copy this and paste on the terminal, then press Enter



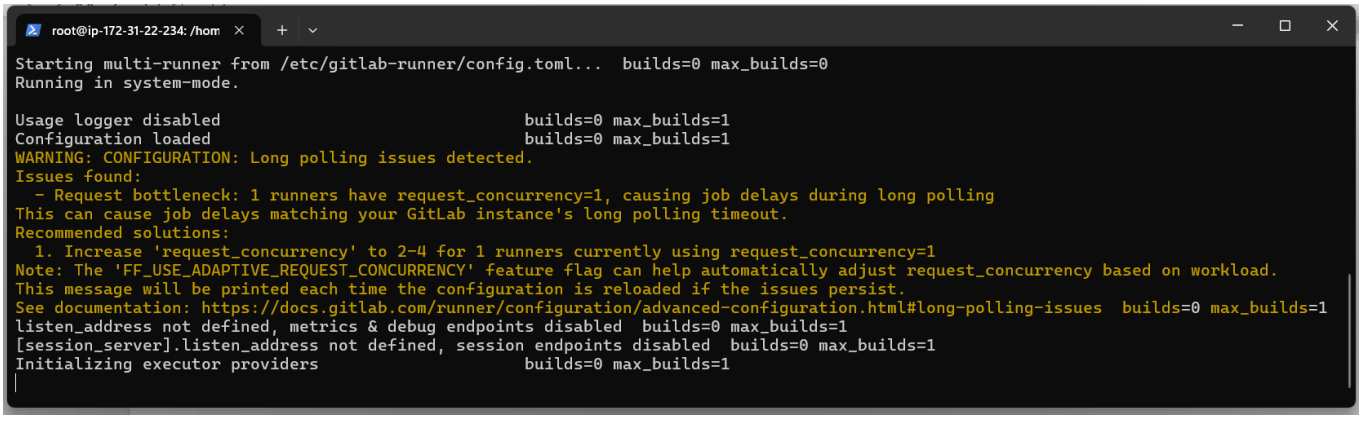
Let us use the name “**runner**”, so enter “**runner**”

Our executor is “**docker**”. So, enter “docker”

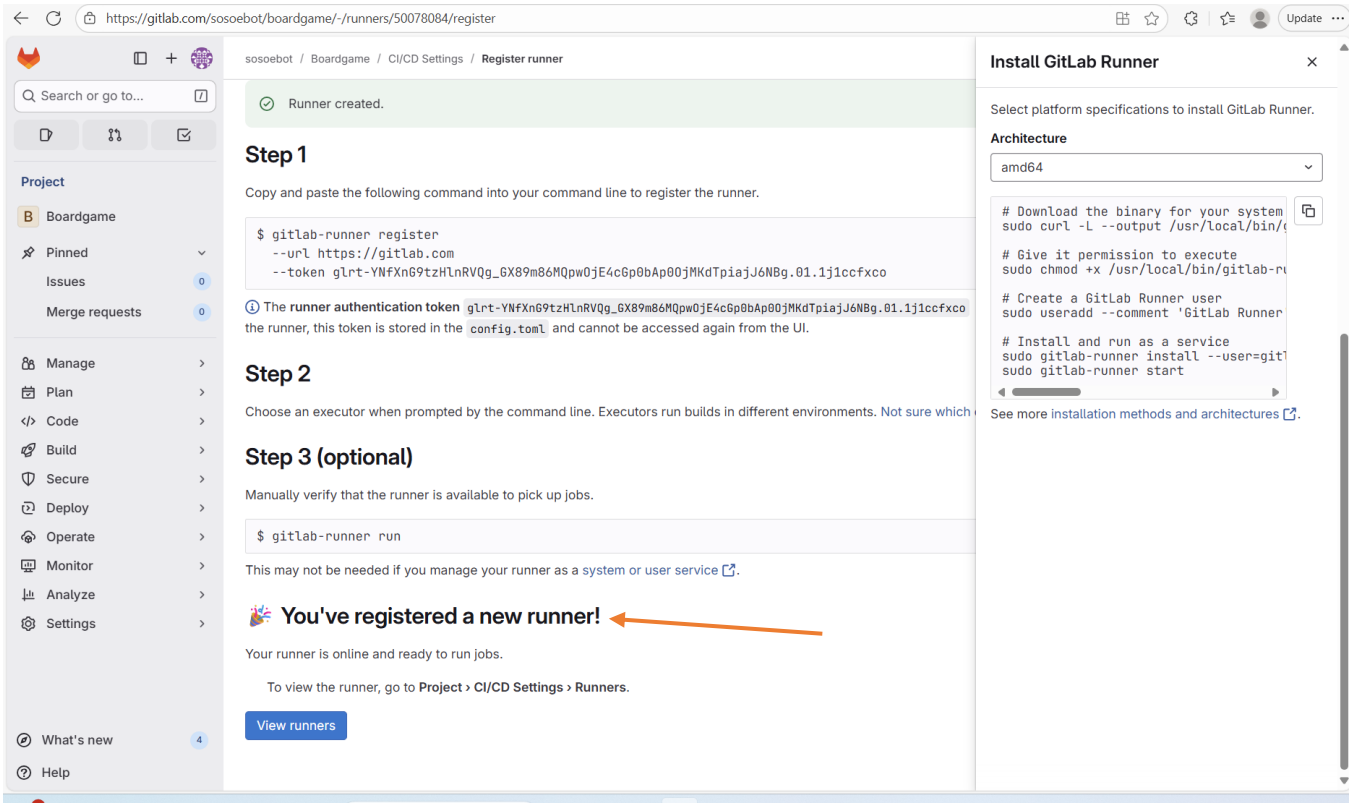


Our project is a Java-based application. So, enter the command:
`maven:3.8.4-openjdk-17`

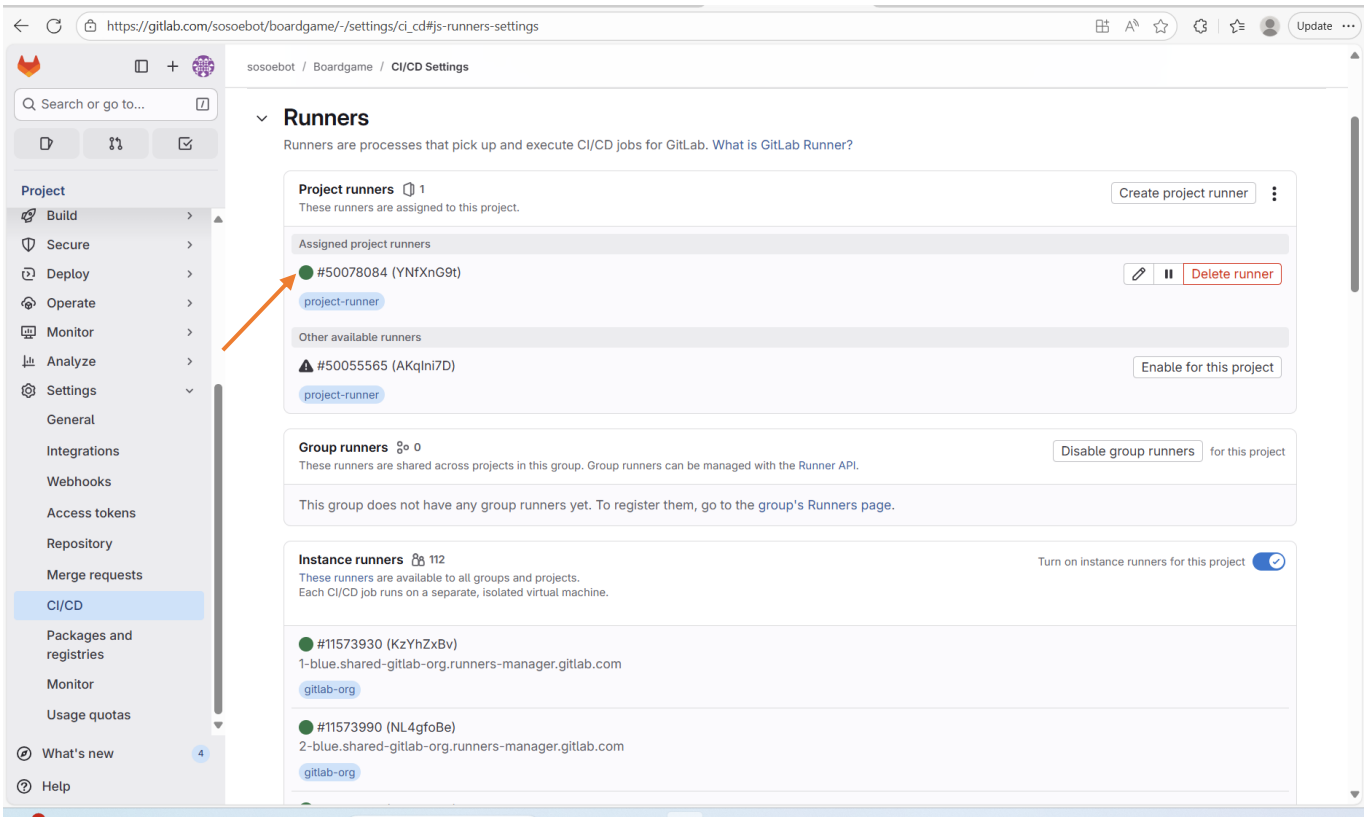
We have successfully registered the runner with Gitlab instance. Manually verify that the runner is available to pick up jobs using the command:
`gitlab-runner run`



Head back to the Gitlab project



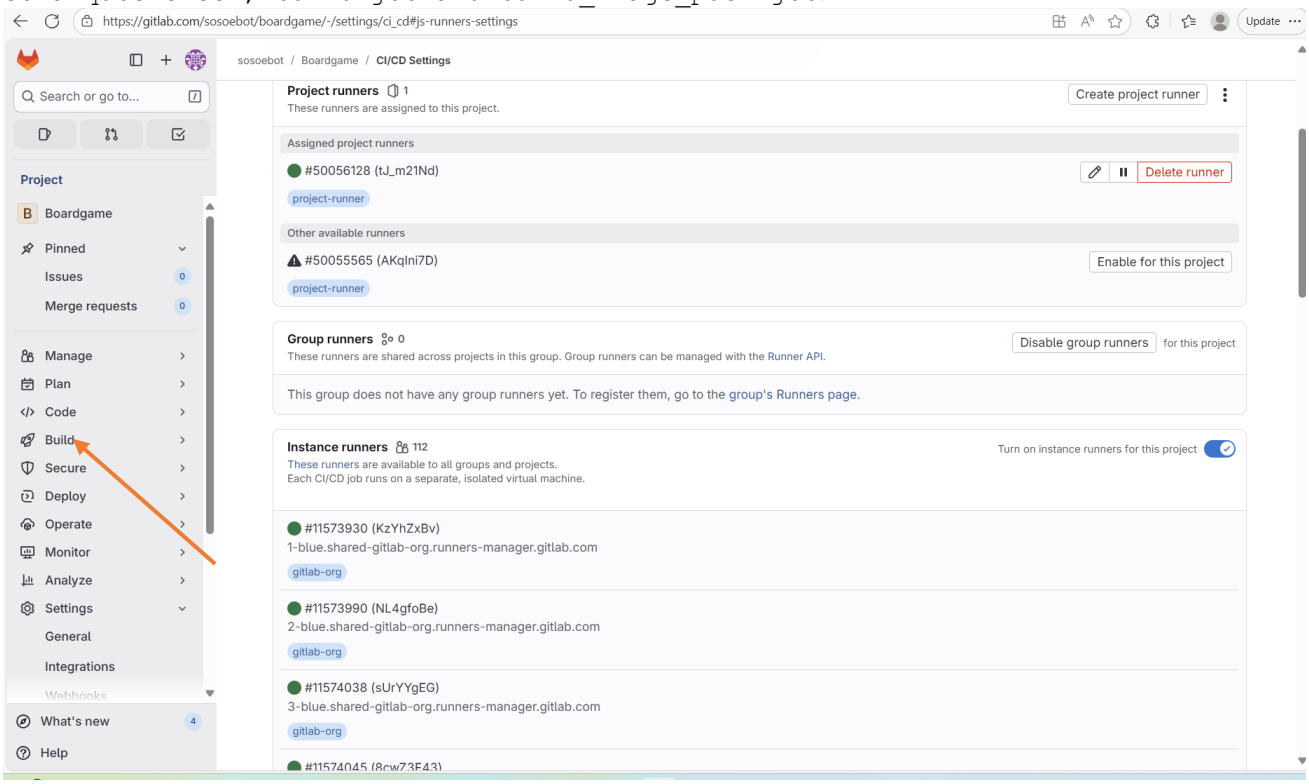
You can see that the runner has been registered. Click on "**View Runners**"



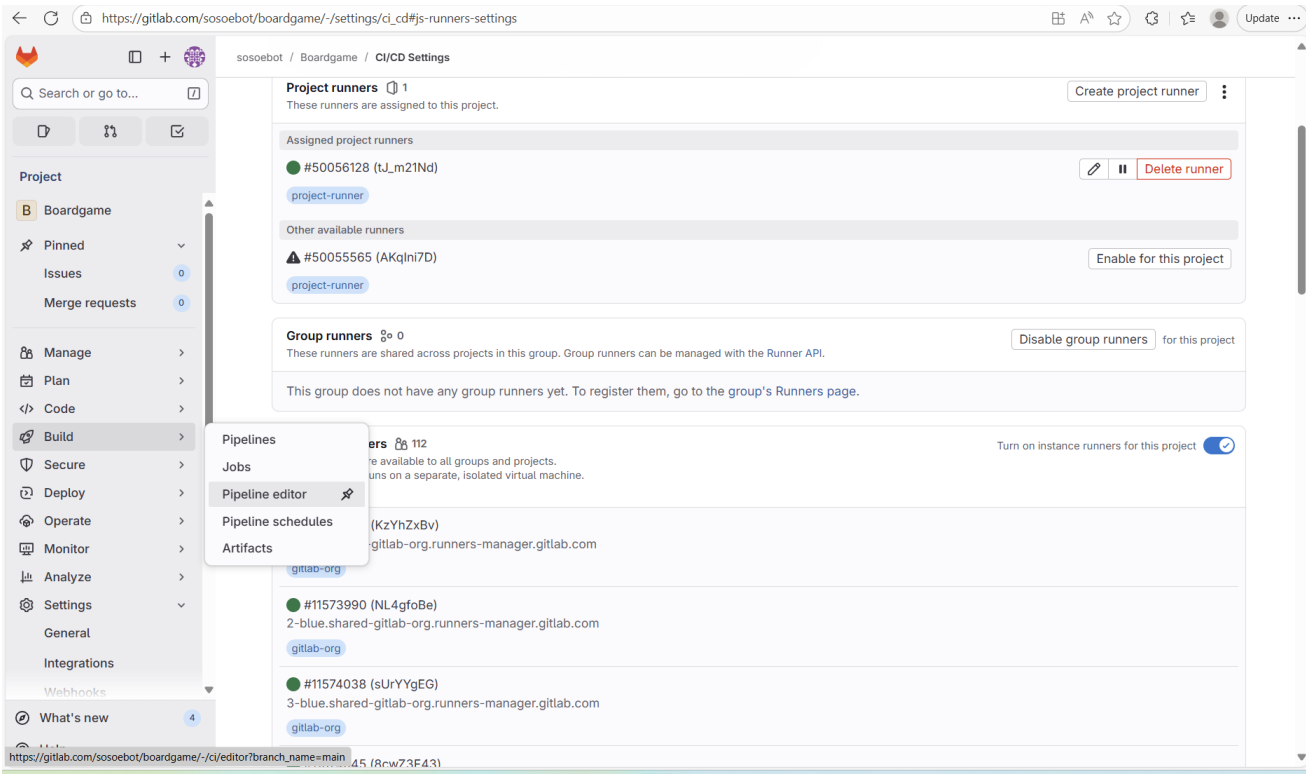
You can see the green signal which means it is available to run jobs. The Gitlab runner is set up.

STEP 8: Adding Jobs

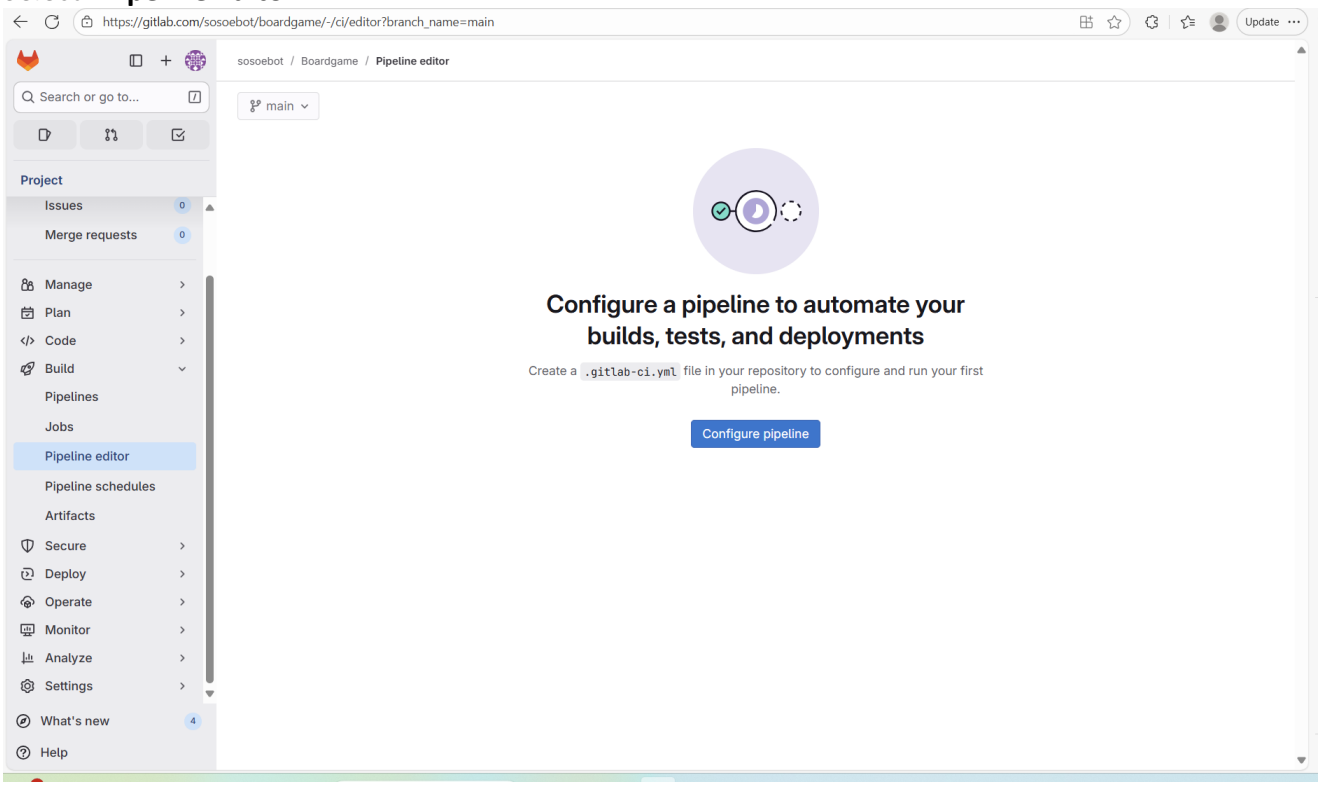
Let us proceed now with the adding of jobs for testing, building and analysis of the code. Followed by building and pushing the Docker image to Docker hub by Docker. We will name the jobs: `test-job, sonarqube-check, build-job and build_image_push-job`.



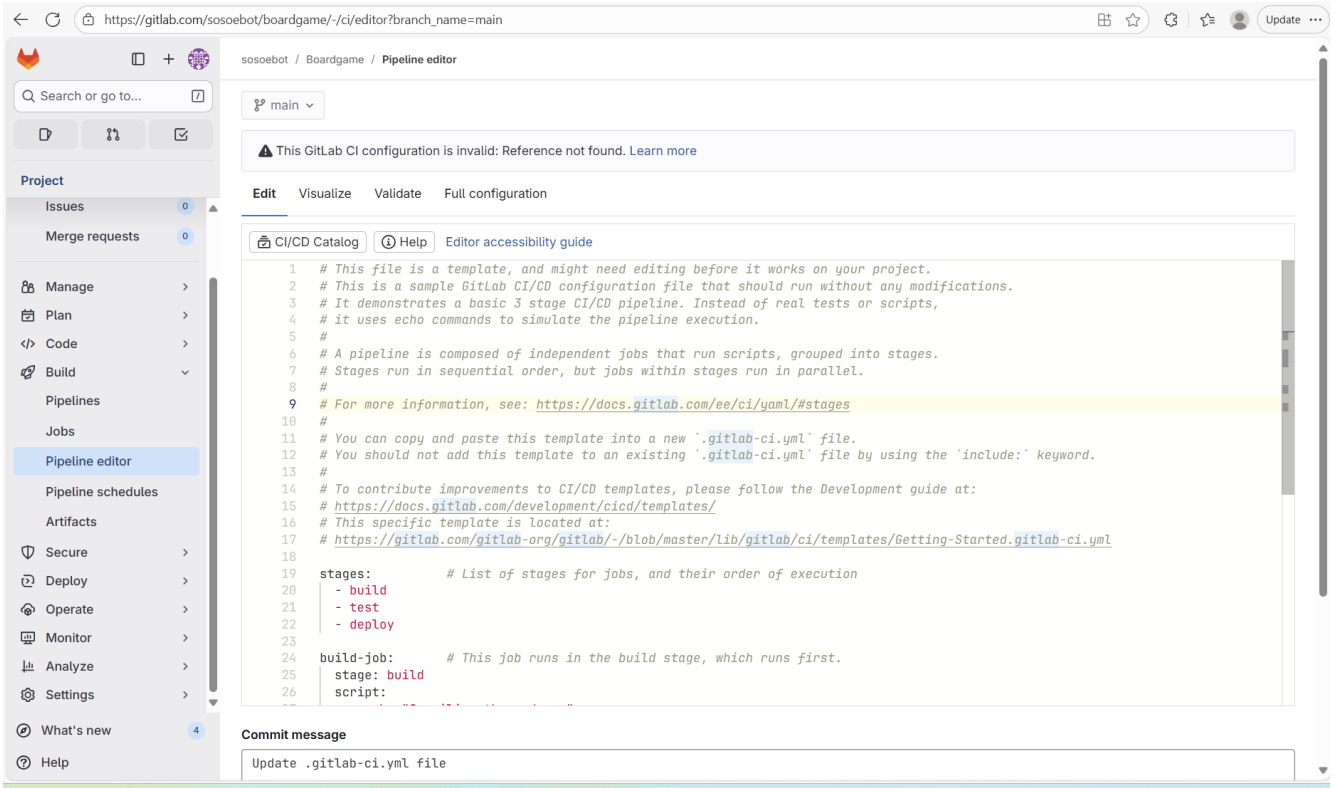
Click on "**Build**"



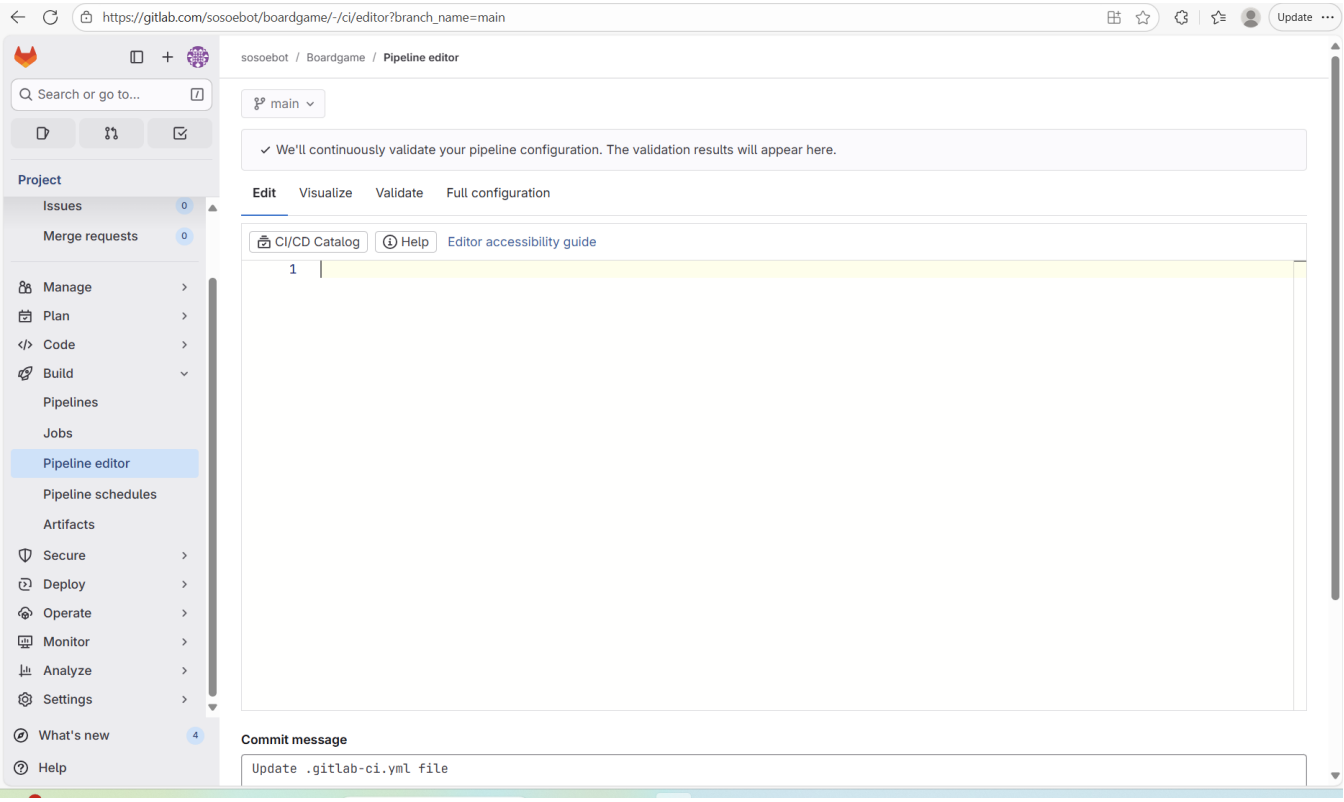
Select “**Pipeline Editor**”



Click on “**Configure Pipeline**”



Delete the existing code



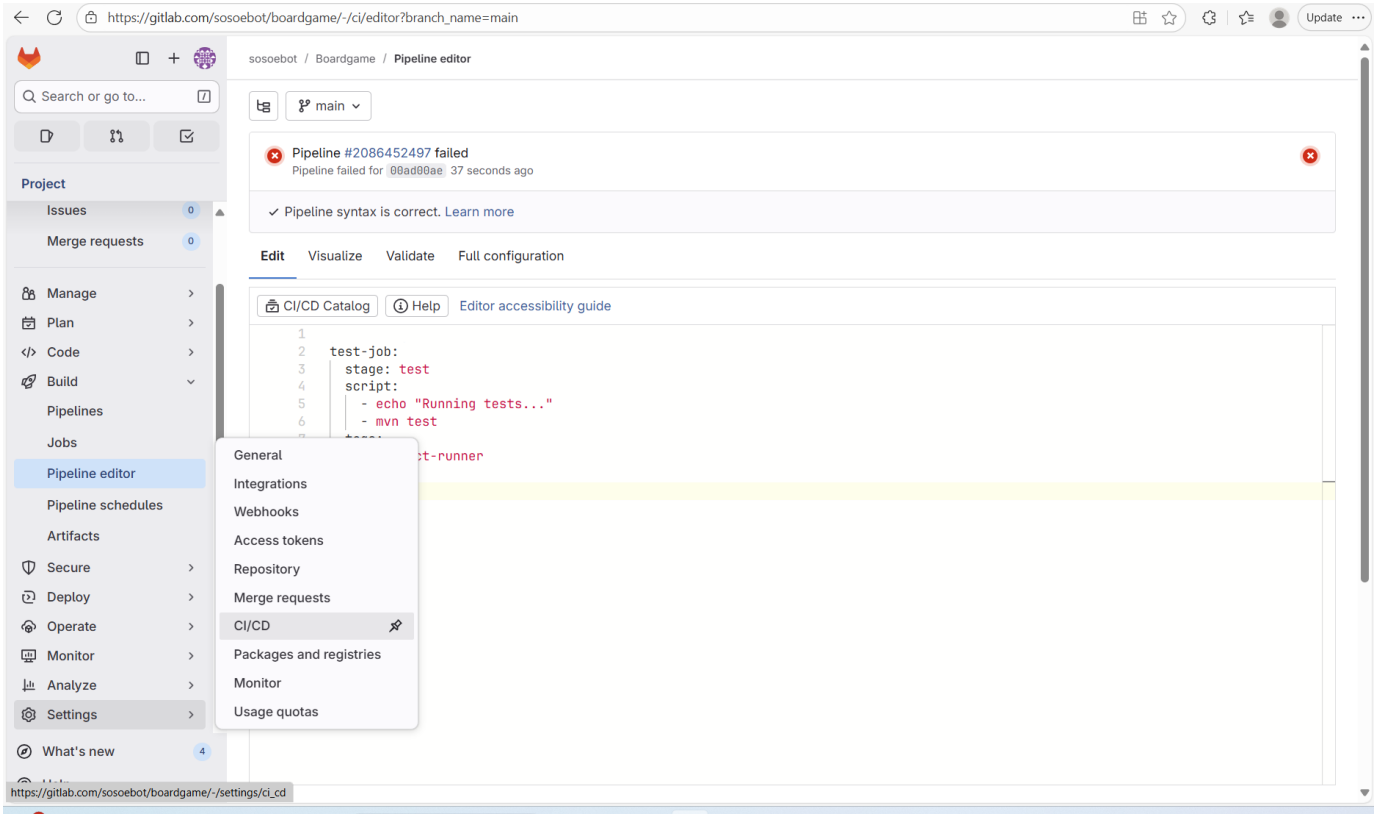
Now, we can start writing code for our jobs.

Part 1: Adding "test-job"
Our first job is the test with Maven

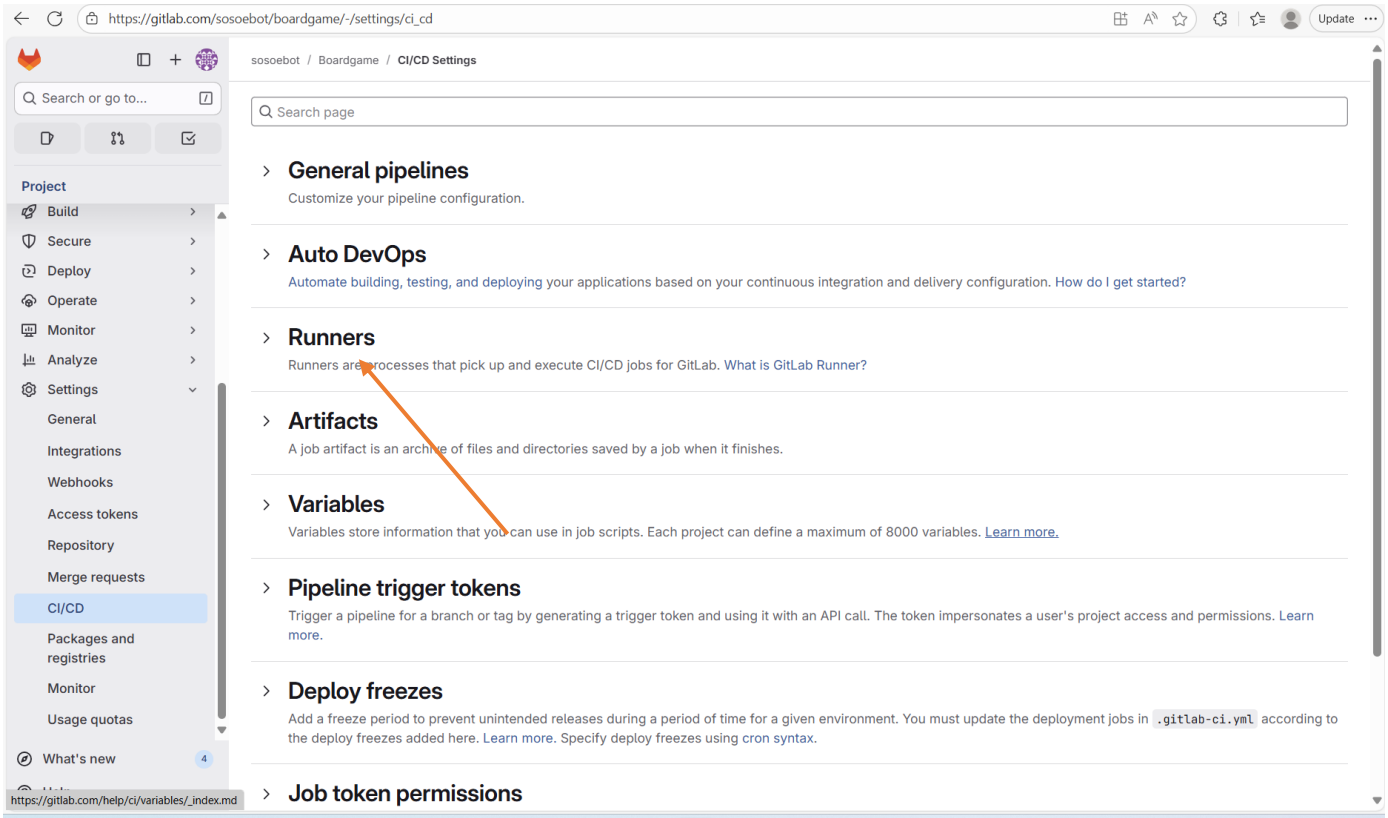
Adding the job

```
test-job:
  stage: test
  script:
    - echo "Running tests..."
    - mvn test
  tags:
    - project-runner
```

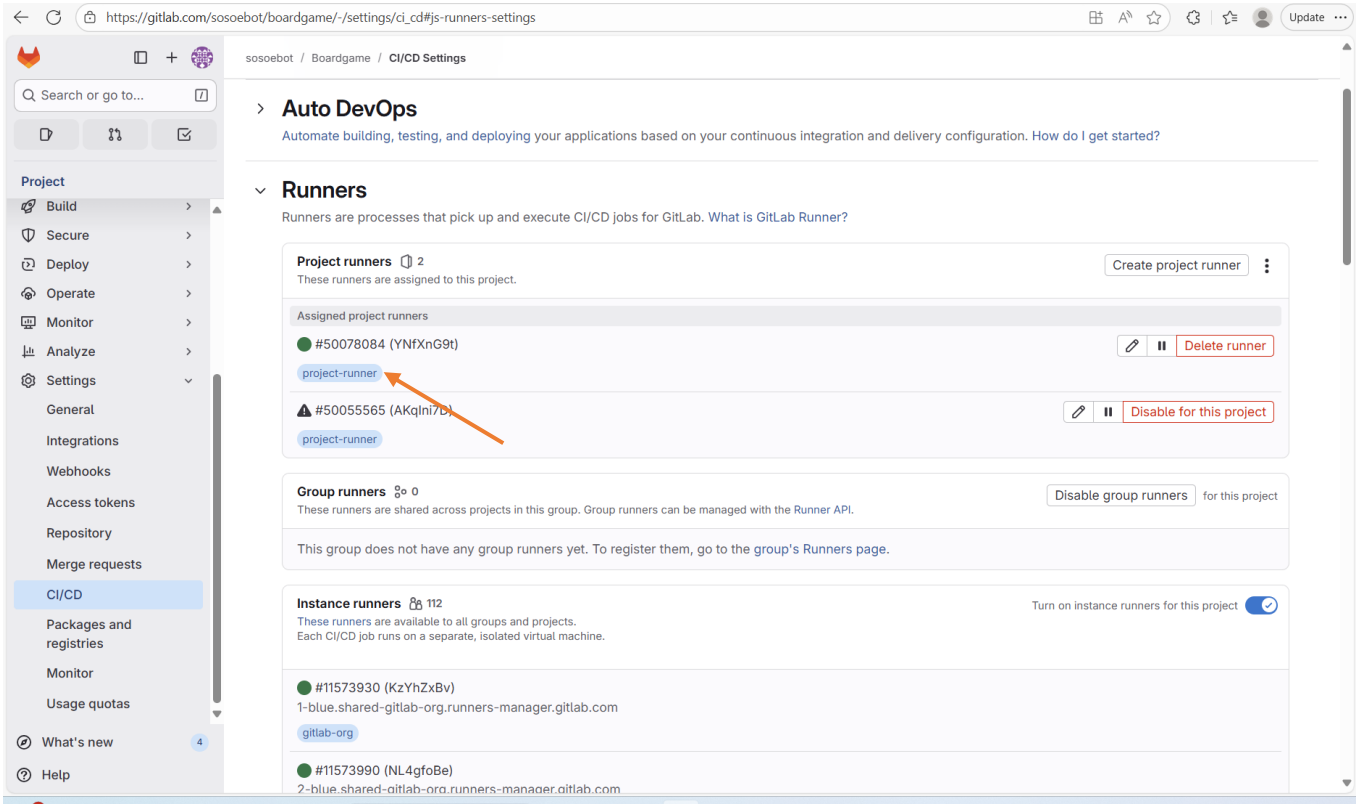
The "**tags**" is obtained as follows



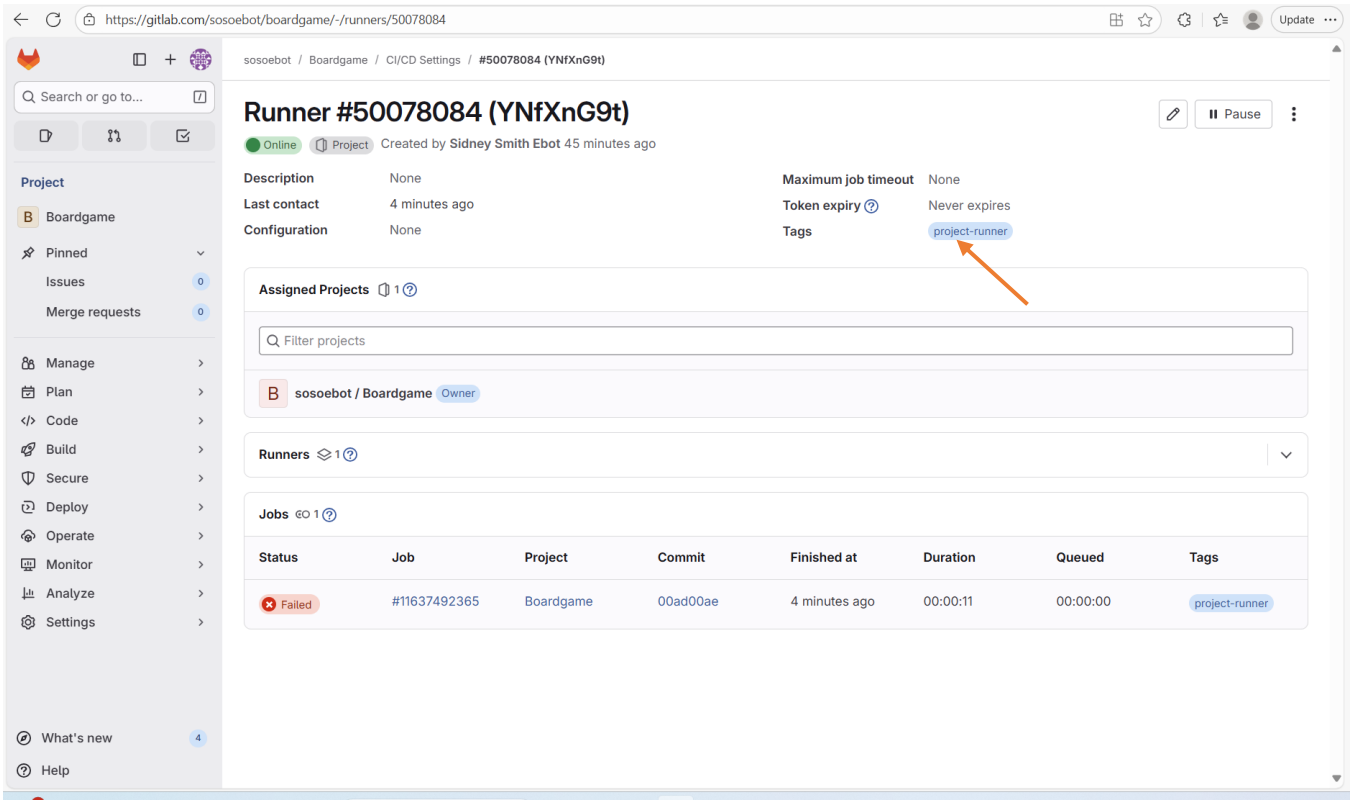
Click on CI/CD



Click on “**Runners**”



Click on the Project runner



You can see the tag = “**project-runner**”

Part 2: Adding “sonarqube-check”

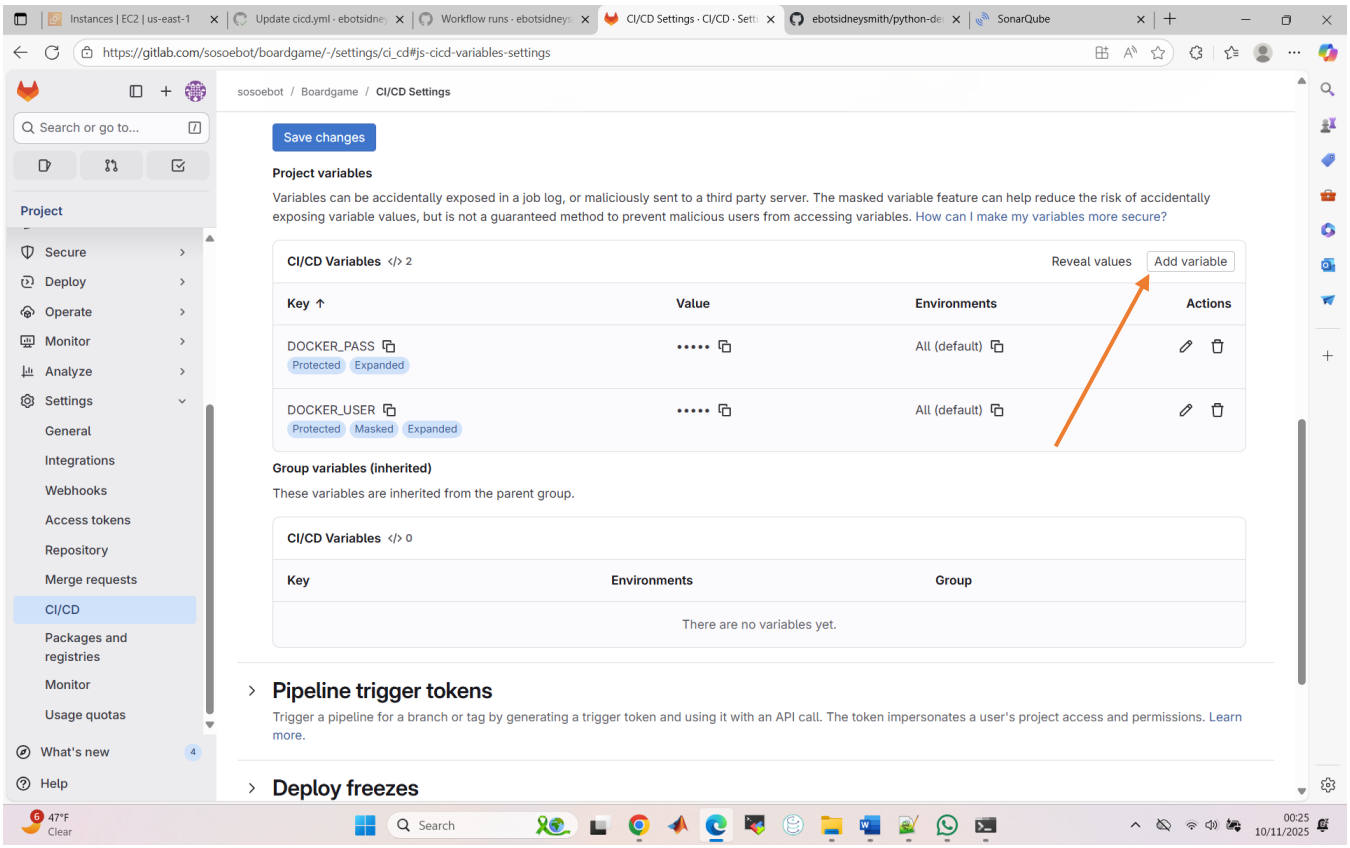
Now we will add the job to do the code analysis. We will follow these steps:

Adding Environment Variables

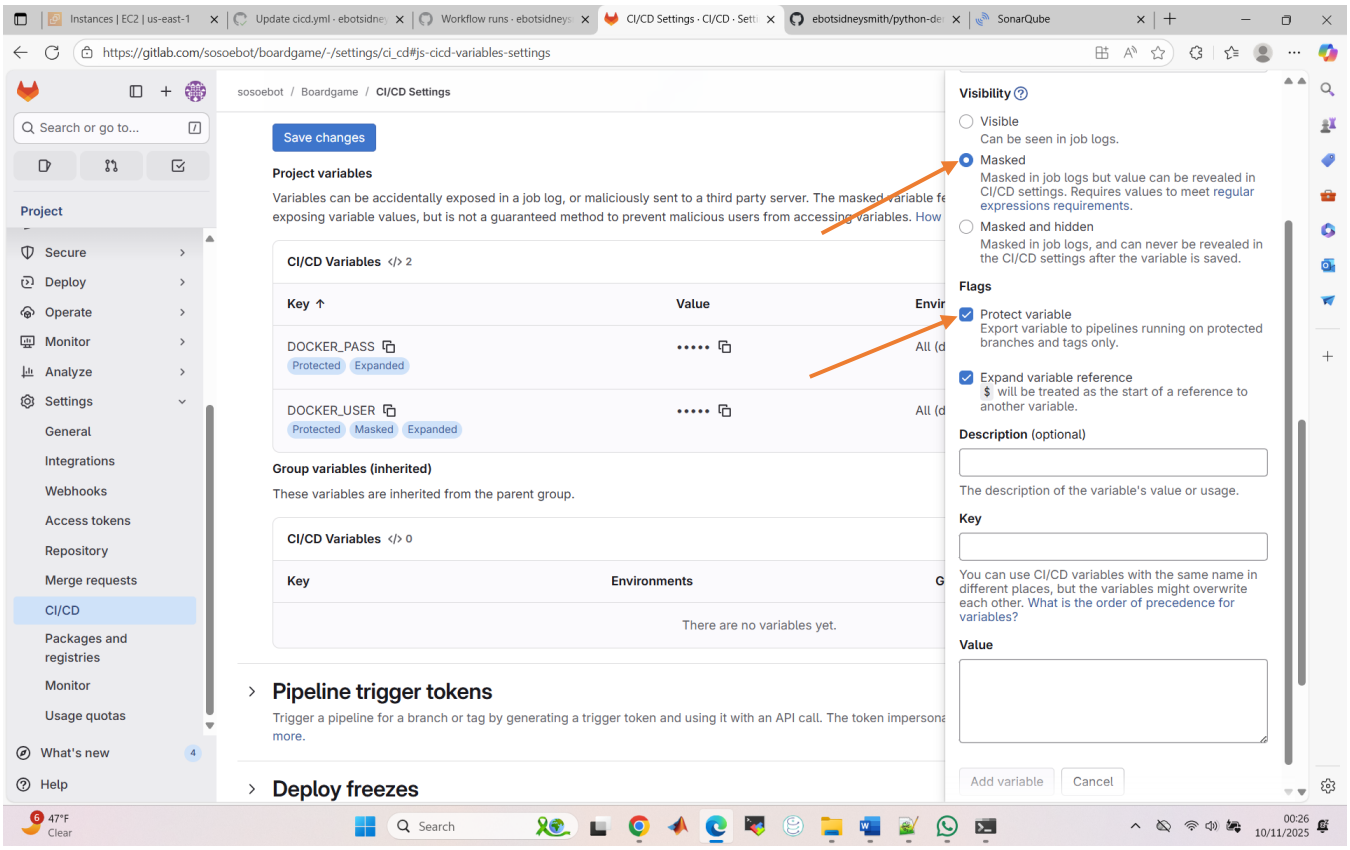
1 Add environment variables

1 Define the SonarQube Community Build Token environment variable.In GitLab, go to **Settings > CI/CD > Variables** to add the following variable and make sure it is available for your project:

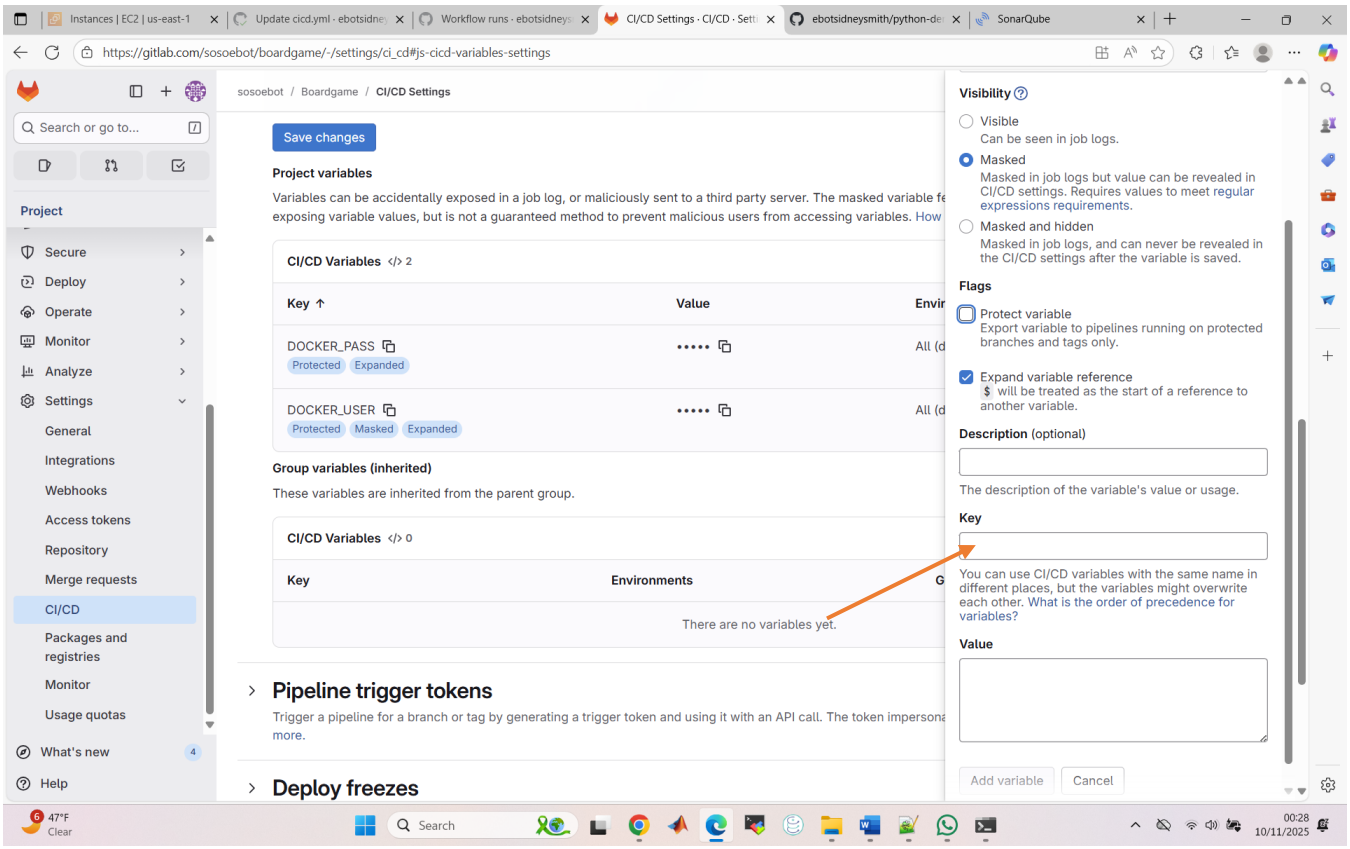
- Key `SONAR_TOKEN`
- In the **Value** field, enter an existing token, or a newly generated one: Generate a token
- Uncheck the **Protect Variable** checkbox.
- Check the **Mask Variable** checkbox.



Click on “Add Variable”

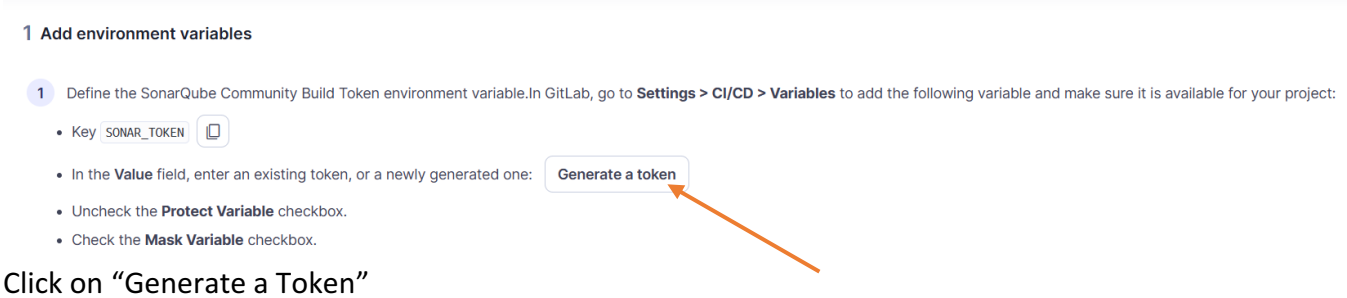


Uncheck “**Protect Variable**” and check “**Masked**”

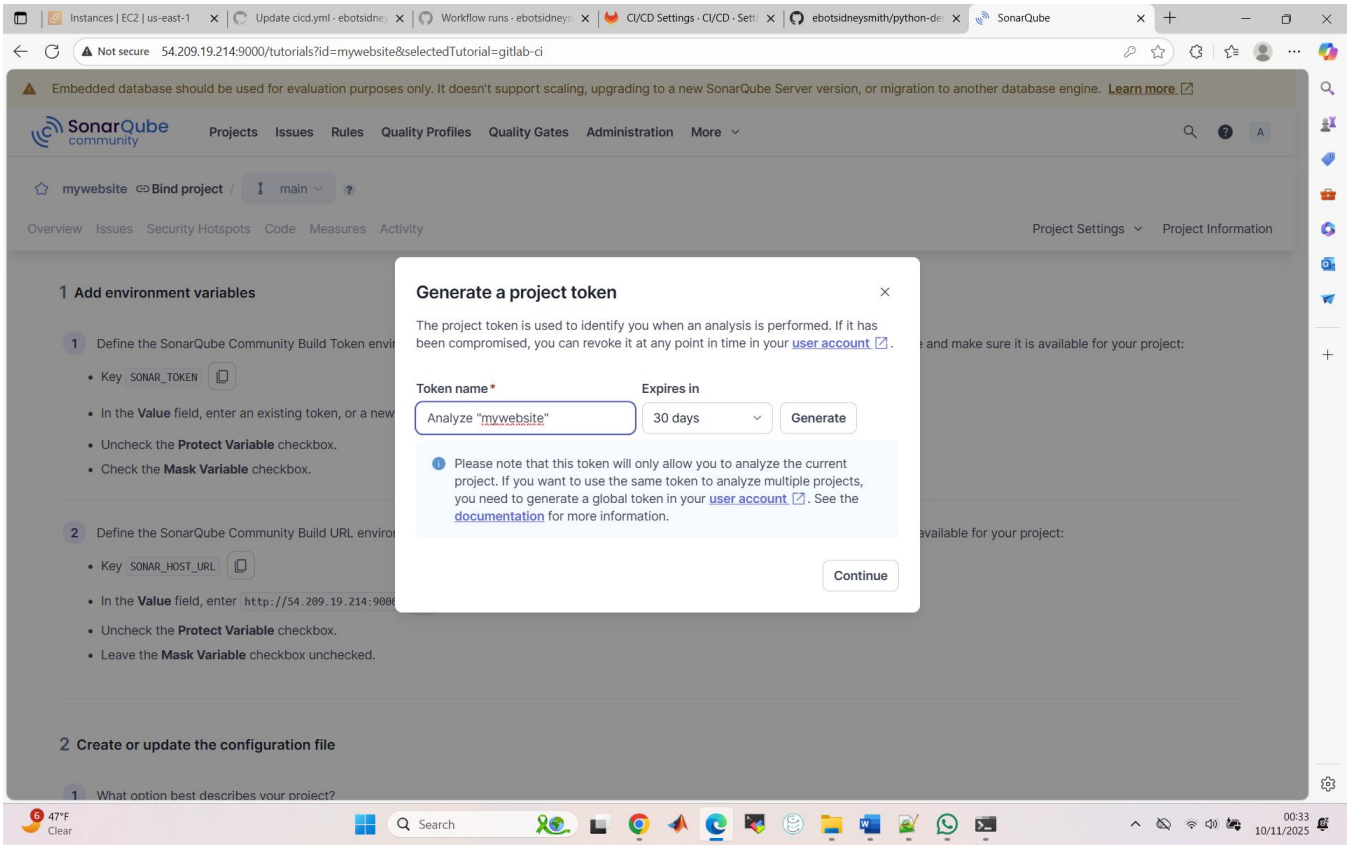


For “Key”, enter “SONAR_TOKEN”

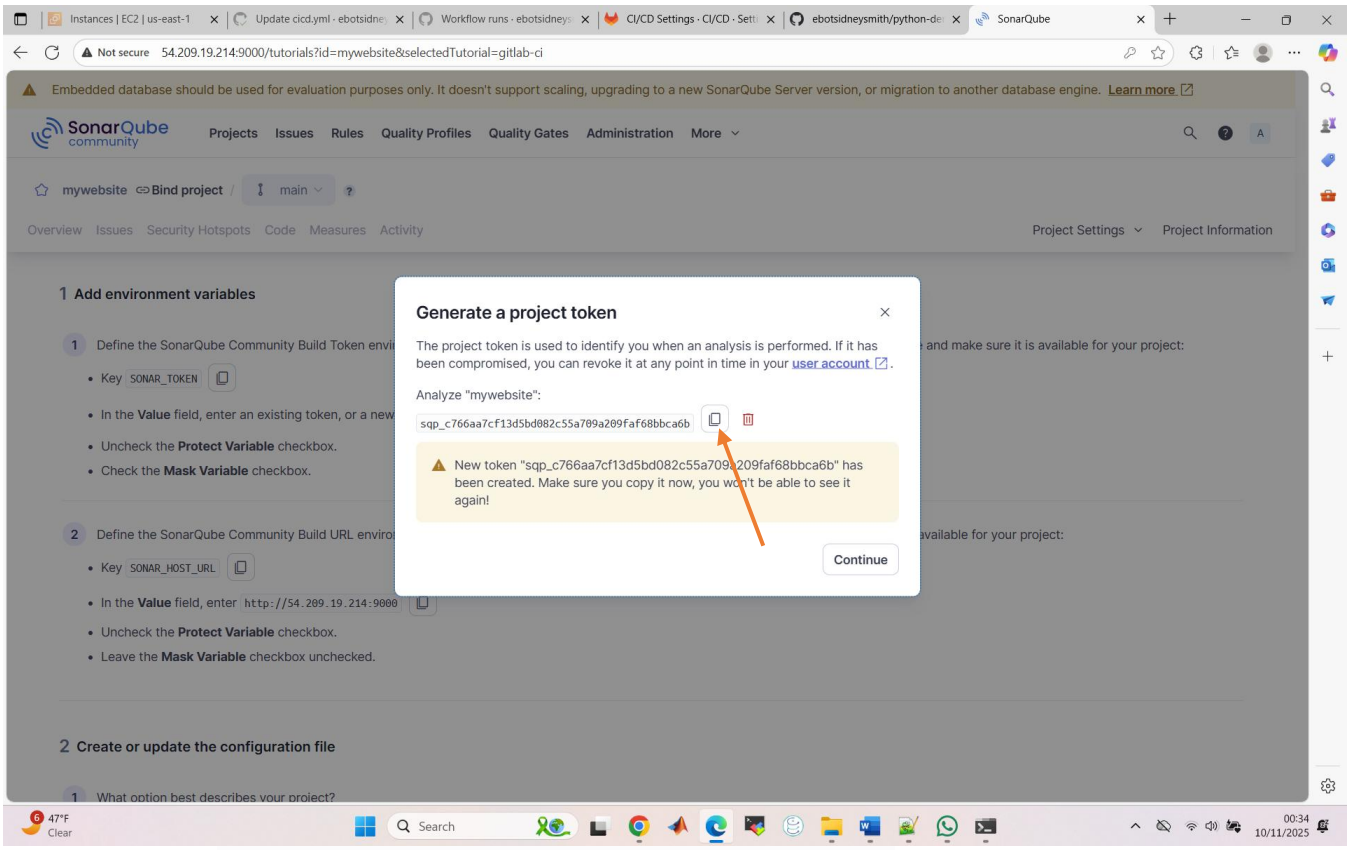
For the “Value”, go to the steps in the Sonarqube browser



Click on “Generate a Token”

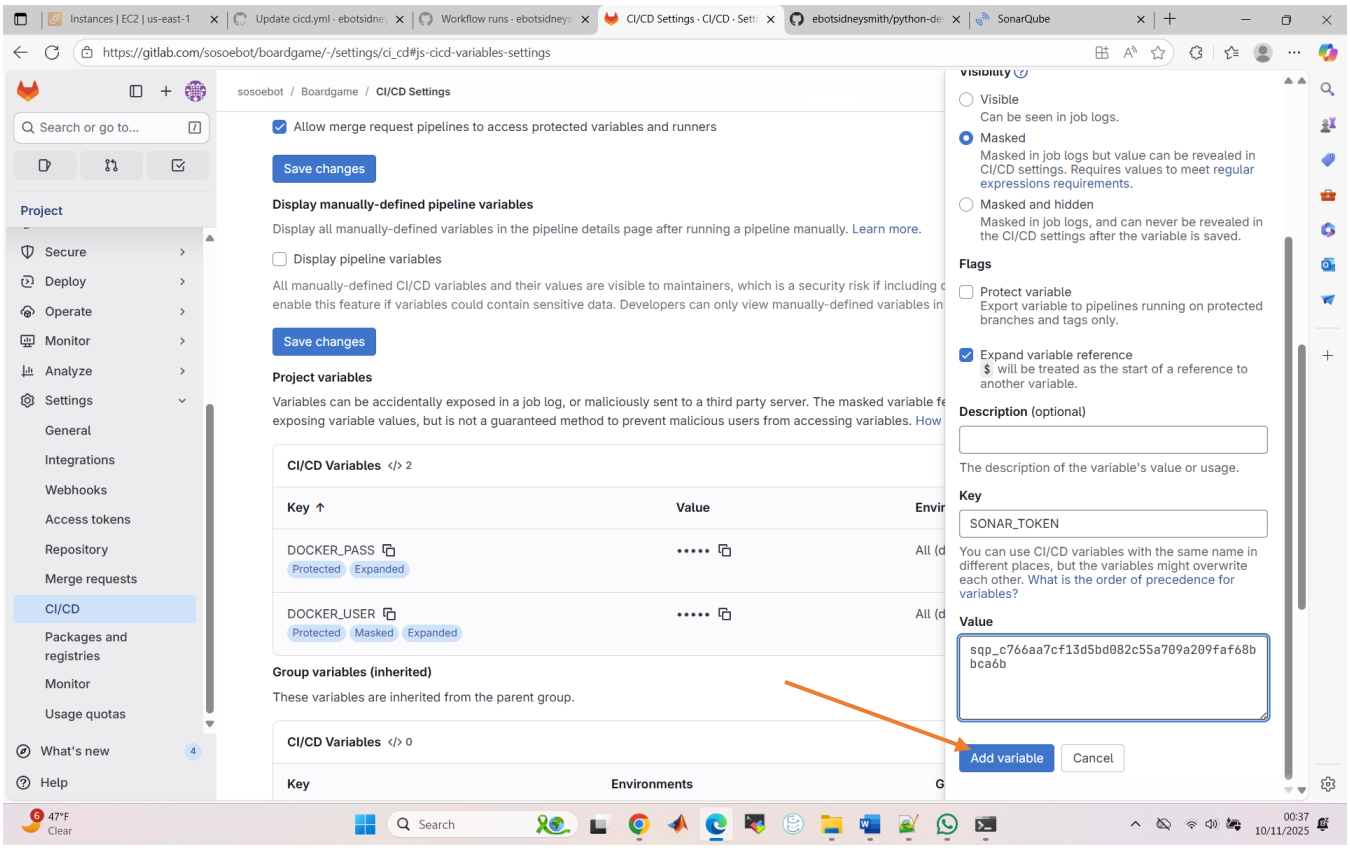


Click on “Generate”



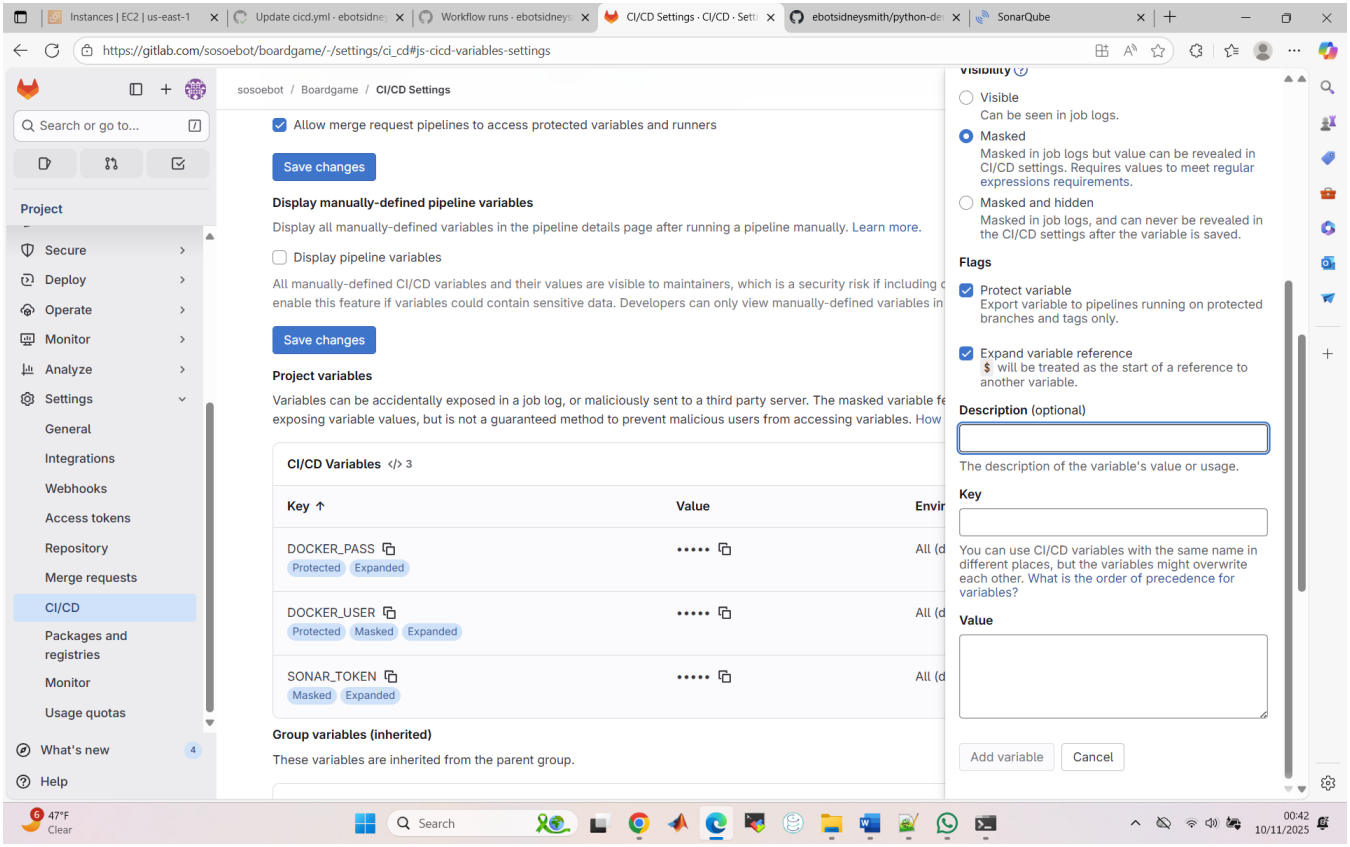
Copy the token: `sqp_c766aa7cf13d5bd082c55a709a209faf68bbca6b`

Enter this on the “Value” field on the Gitlab browser

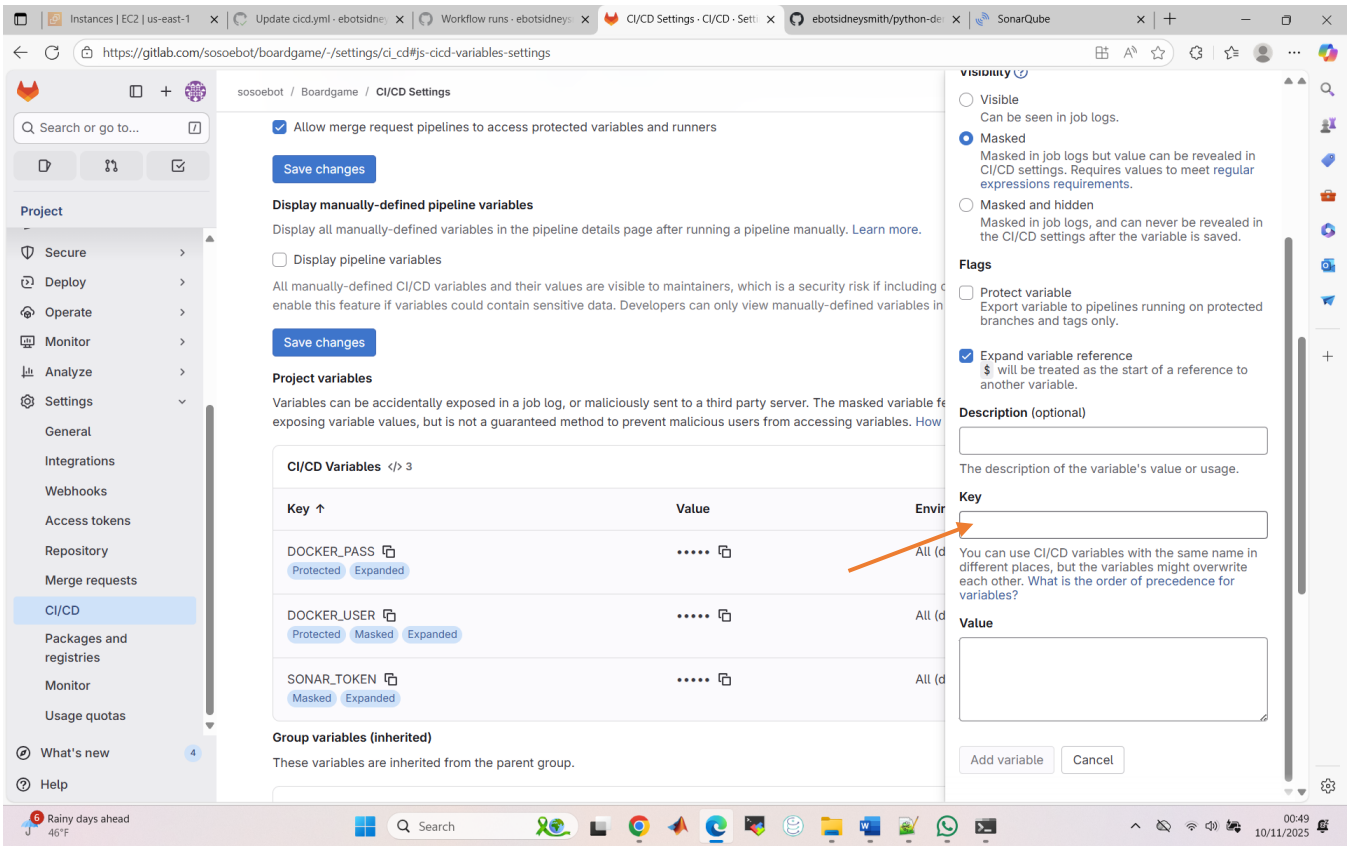


Click on “Add Variable”

Now, we have to the URL



Uncheck the “**Protect Variable**” and leave the “**Mask Variable**” checkbox unchecked

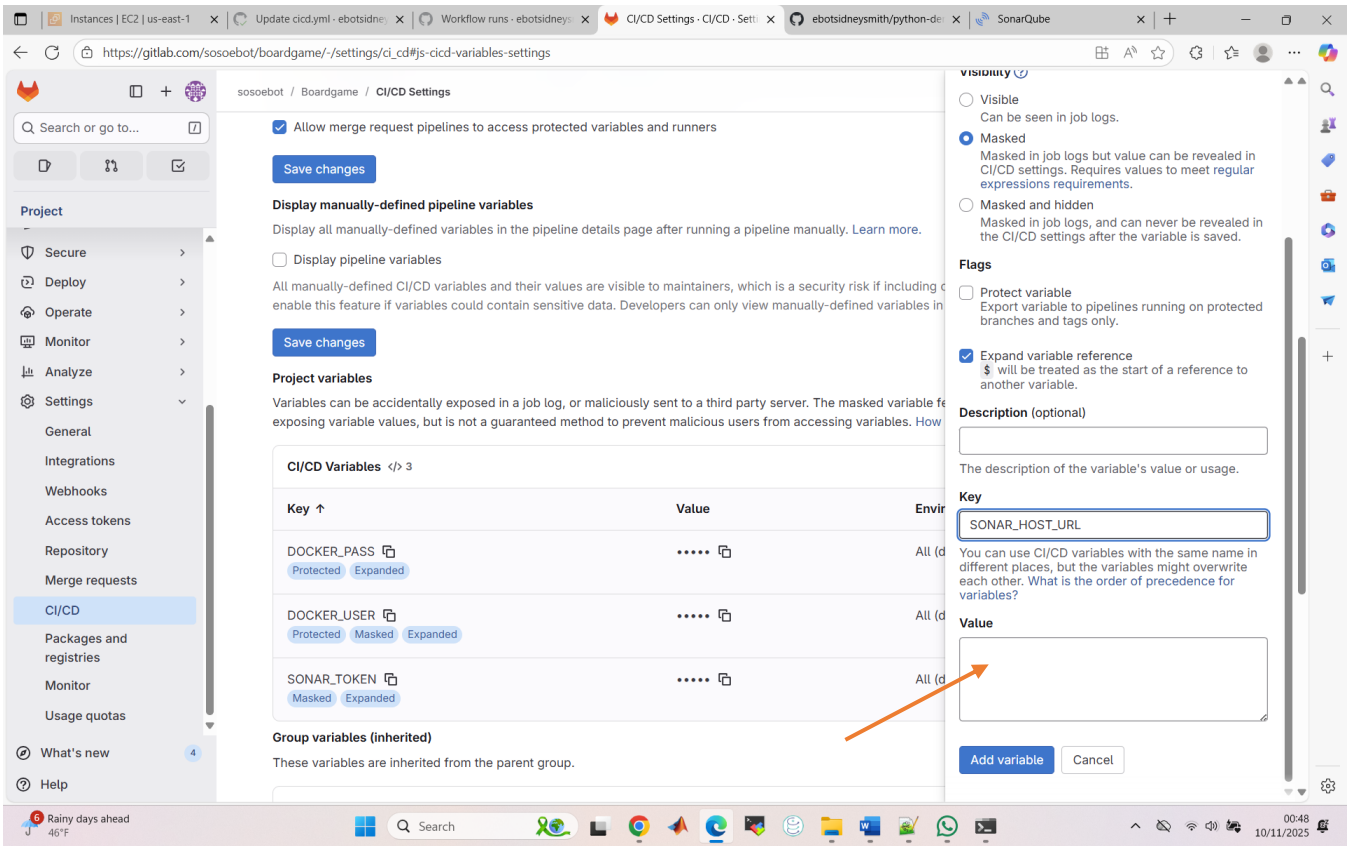


For “Key”, head back to SonarQube browser

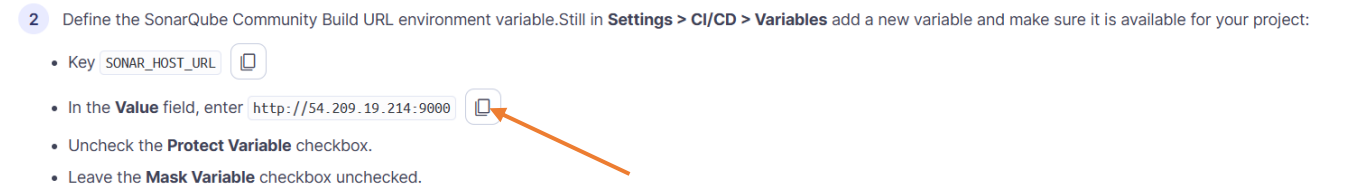
2 Define the SonarQube Community Build URL environment variable.Still in **Settings > CI/CD > Variables** add a new variable and make sure it is available for your project:

- Key `SONAR_HOST_URL`
- In the **Value** field, enter `http://54.209.19.214:9000`
- Uncheck the **Protect Variable** checkbox.
- Leave the **Mask Variable** checkbox unchecked.

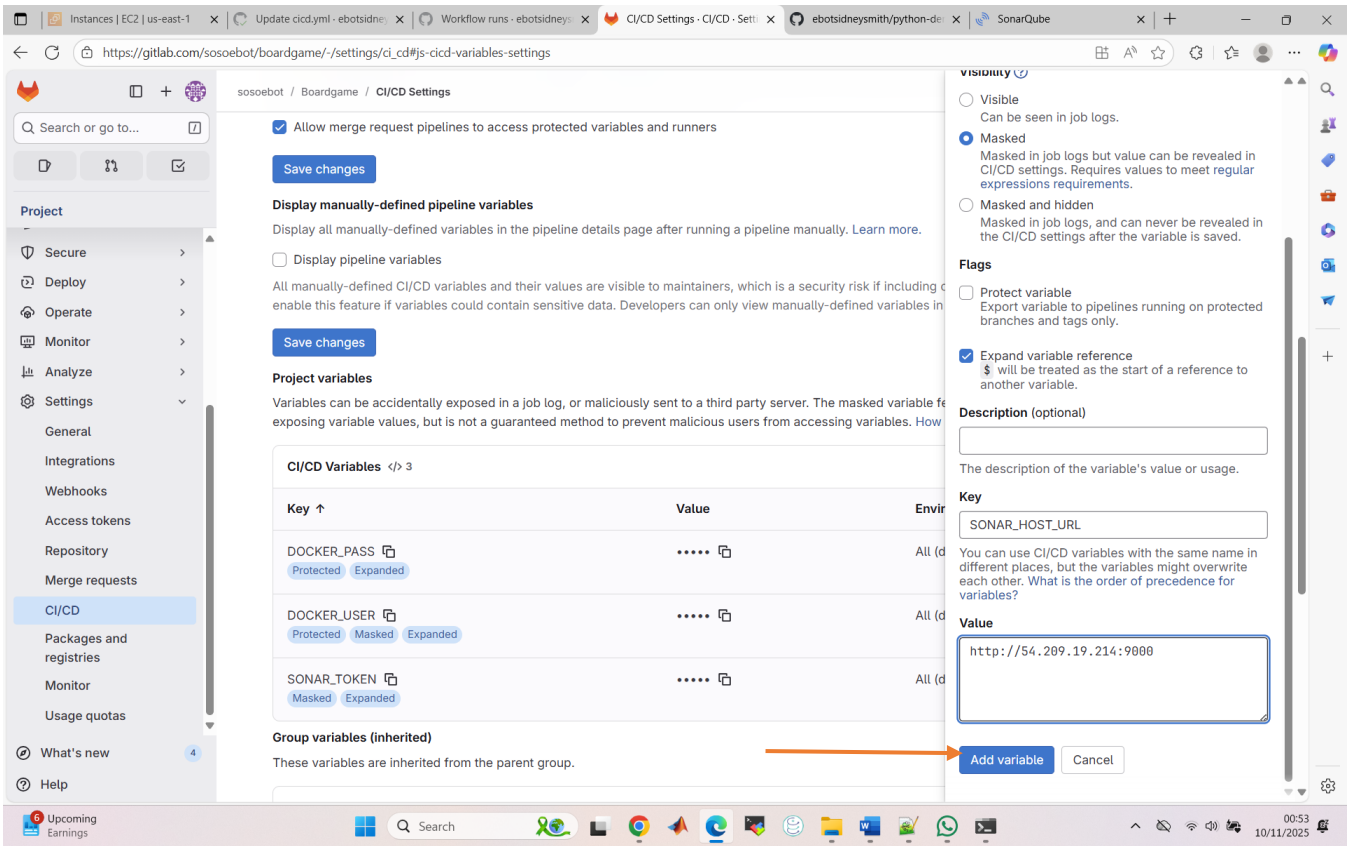
Copy the “Key” and paste it on the “Key” field on the Gitlab browser



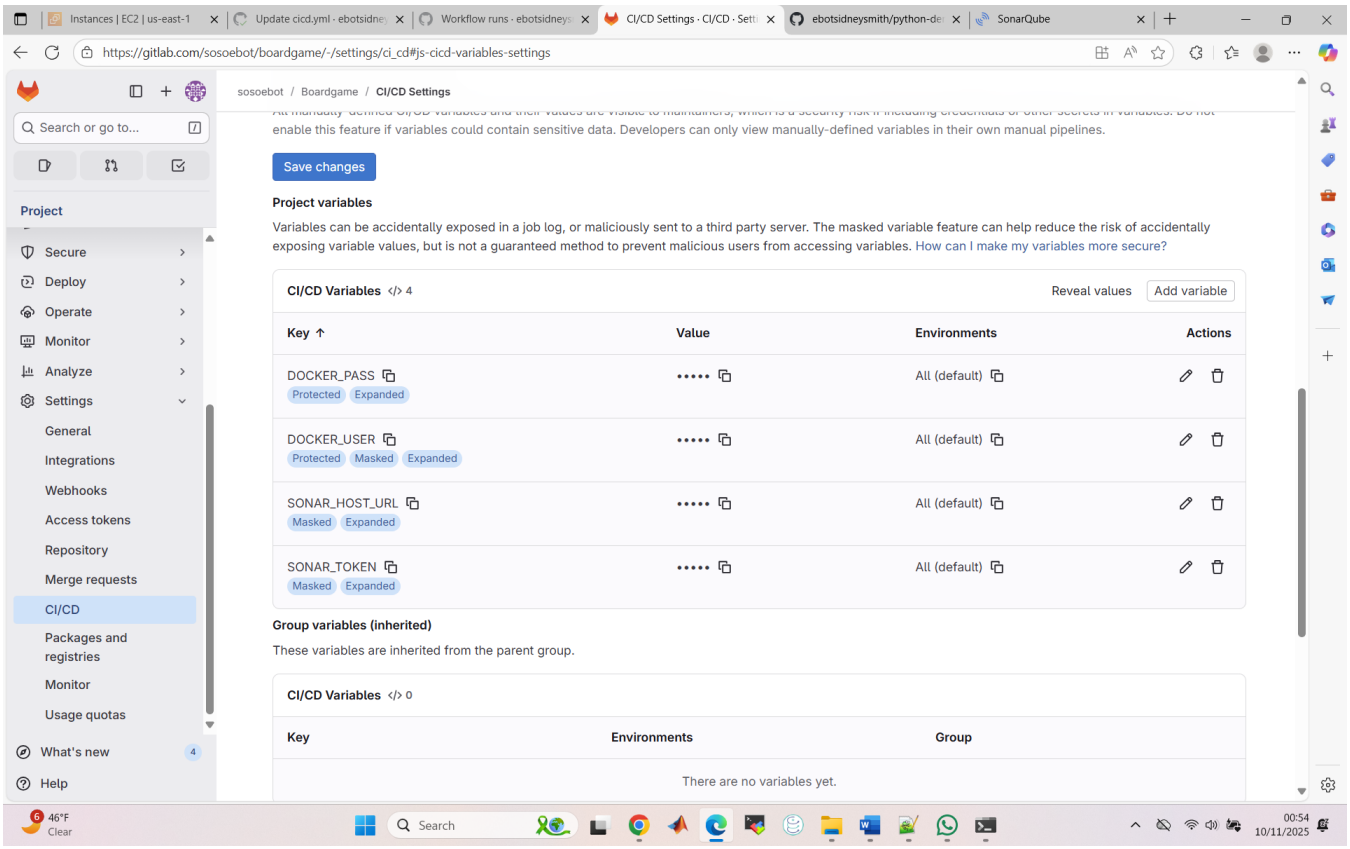
For the “Value”, head back to SonarQube browser



Copy the “Value” and paste it on the “Value” field on the Gitlab browser

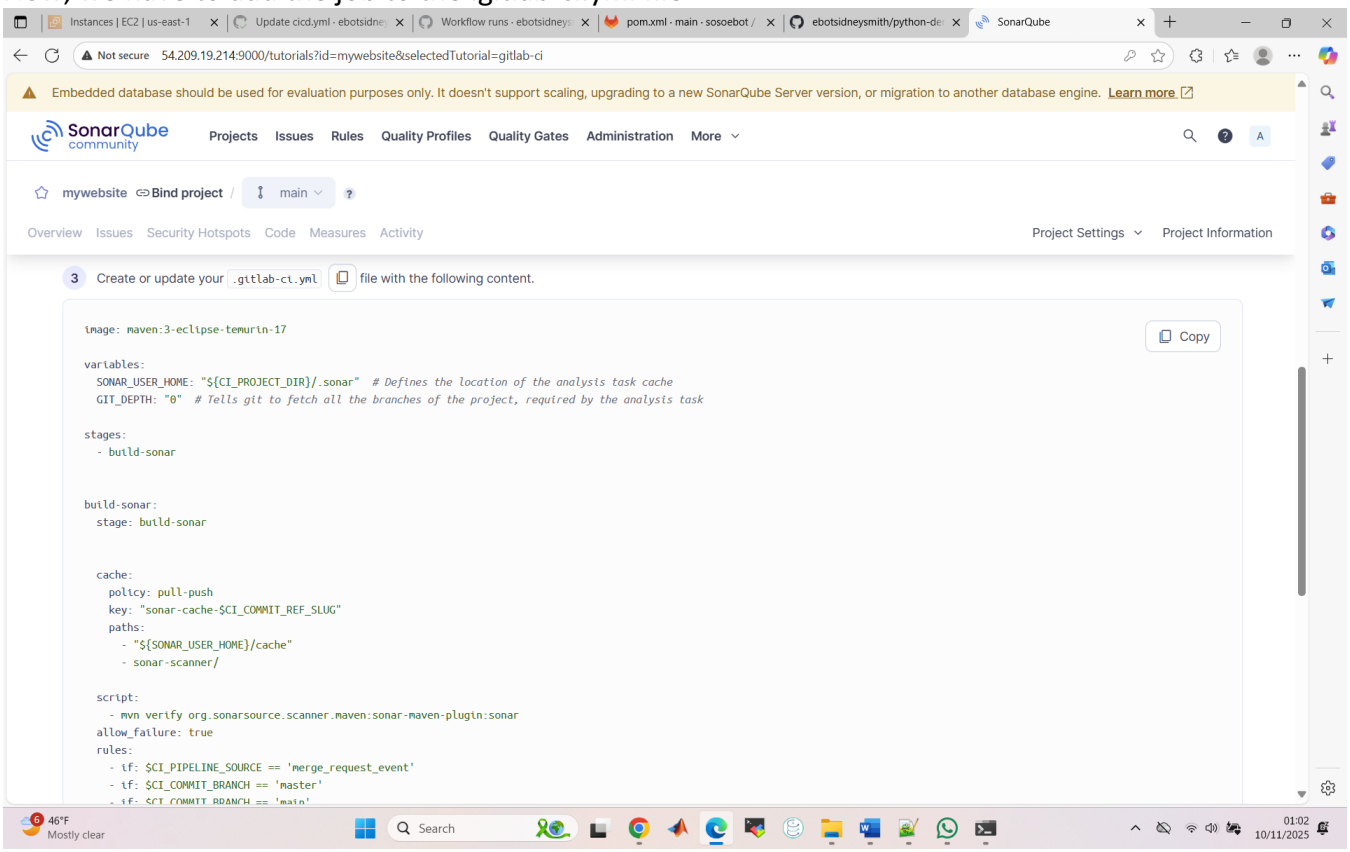


Then click on “Add Variable”

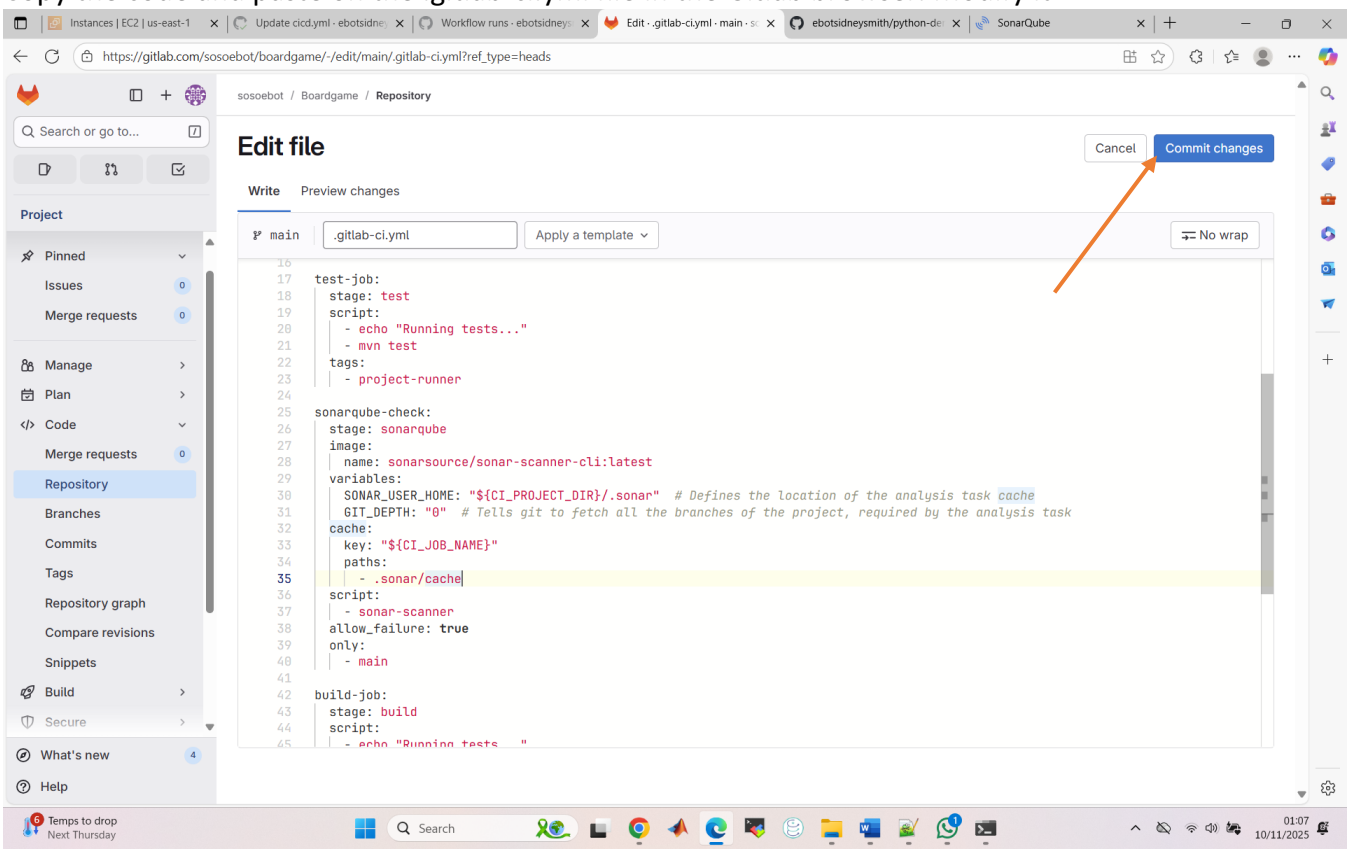


The variables have been added

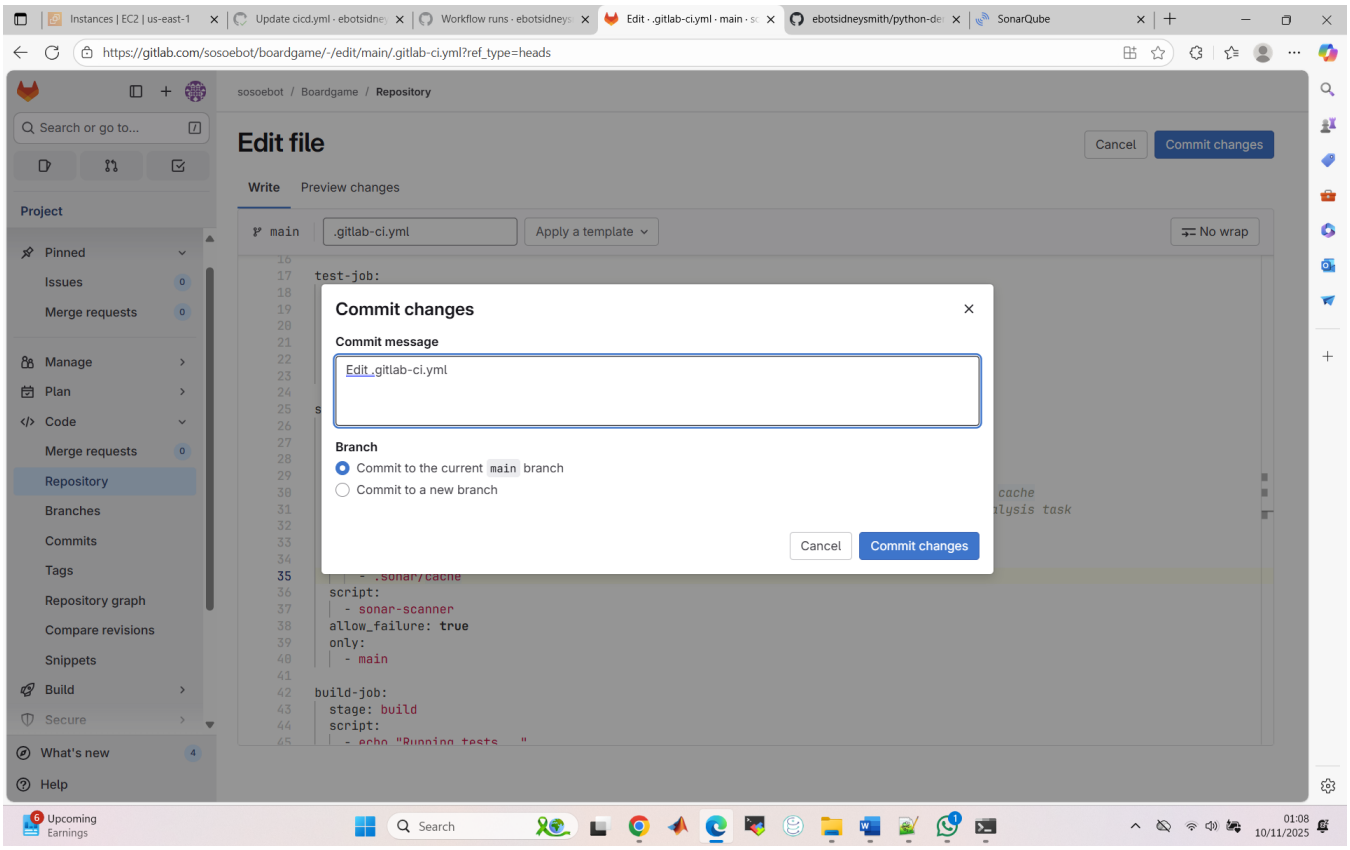
Now, we have to add the job to the .gitlab-ci.yml file



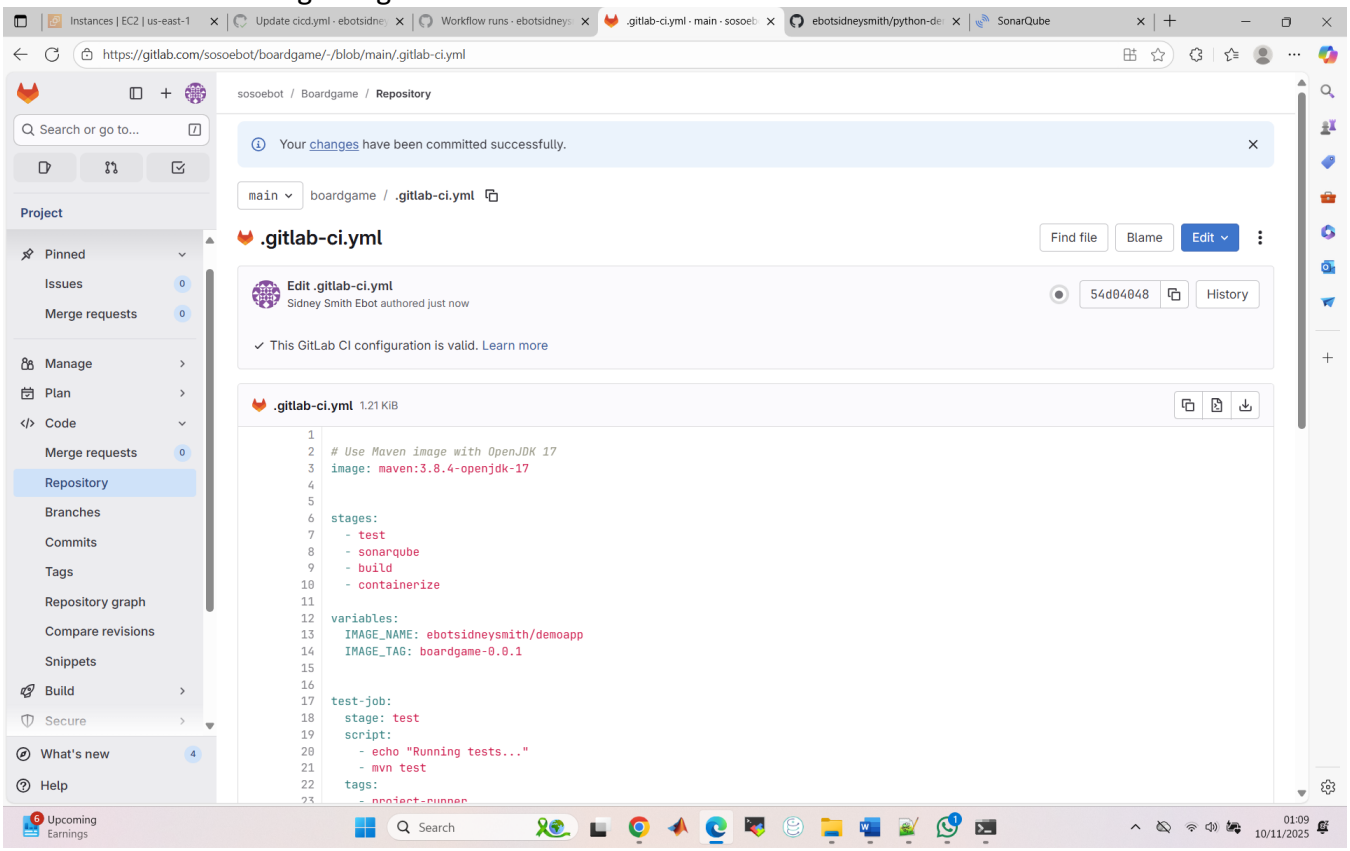
Copy the code and paste on the .gitlab-ci.yml file in the Gitlab browser. Modify it



Then click on “Commit Changes”



Click on “Commit changes” again



Adding the “sonarqube-check” job

```
sonarqube-check:
  stage: sonarqube
  image:
    name: sonarsource/sonar-scanner-cli:latest
  variables:
    SONAR_USER_HOME: "${CI_PROJECT_DIR}/.sonar"  # Defines the location of the
analysis task cache
    GIT_DEPTH: "0"  # Tells git to fetch all the branches of the project, required
by the analysis task
  cache:
    policy: pull-push
    key: "sonar-cache-$CI_COMMIT_REF_SLUG"
    paths:
    - "${SONAR_USER_HOME}/cache"
    - sonar-scanner/
  script:
    - sonar-scanner
  allow_failure: true
  only:
    - main
```

Finally, make sure you change the code on the “sonar-project.properties” file to match the project name you used on SonarQube.

```
sonar.projectKey=mywebsite
sonar.qualitygate.wait=true
sonar.projectName=mywebsite
sonar.java.binaries=.
```

Part 3: Adding “build-job”

Now we will add the job to do the code analysis.

Adding the “build-job”

```
build-job:
  stage: build
  script:
    - echo "Running tests..."
    - mvn package
  artifacts:
     paths:
       - target/*.jar
  tags:
    - project-runner
```

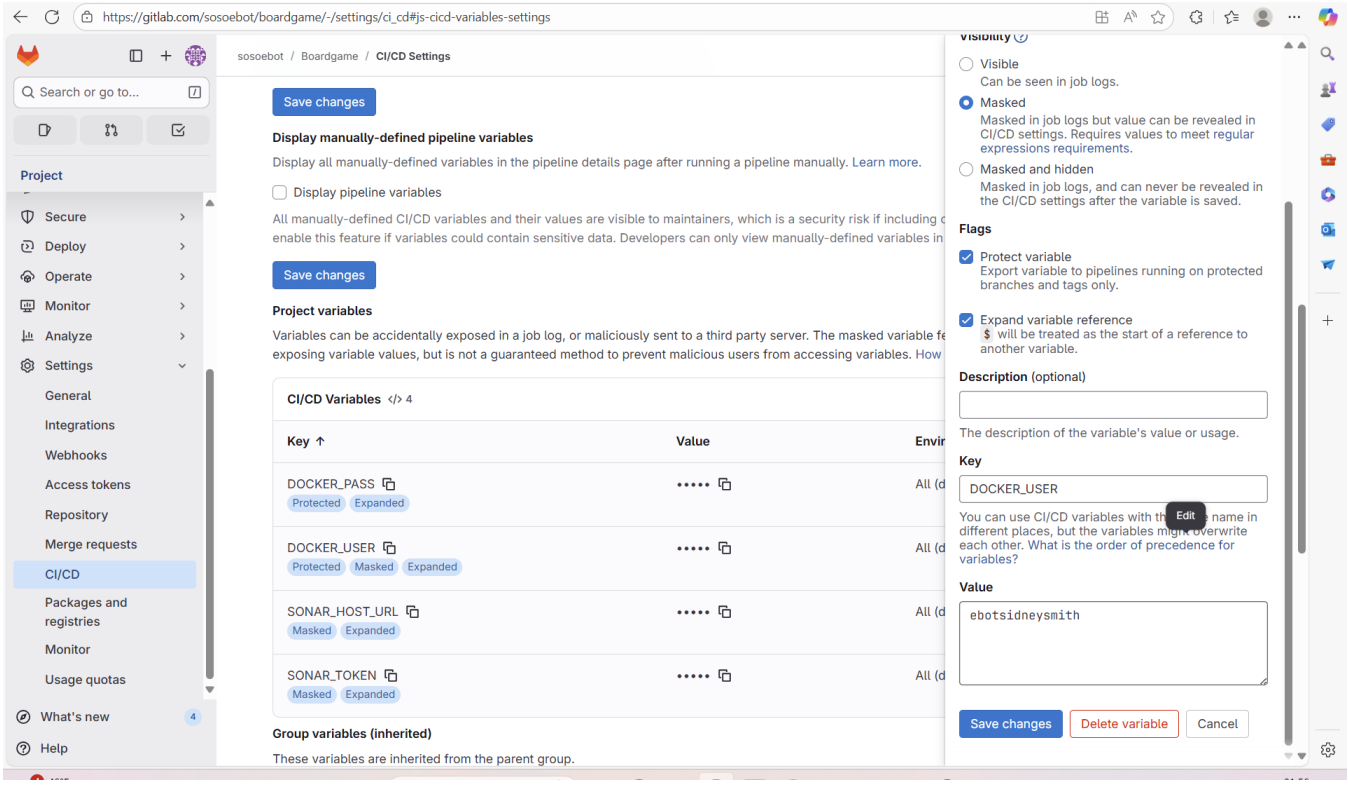
Part 4: Adding "build_image_push-job"
Now we will add the job to do the code analysis.

Adding the "build_image_push-job"

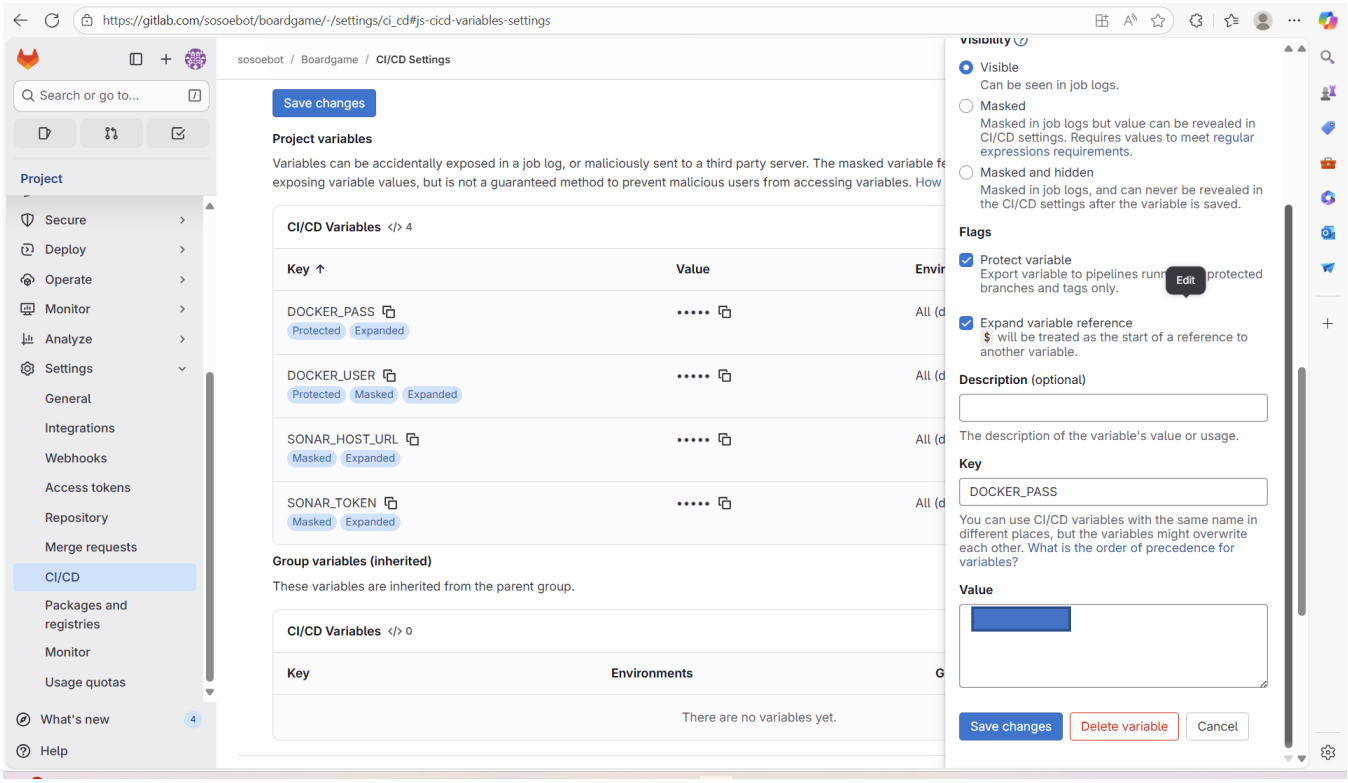
```
build_image_push-job:
  image: docker:27.1.1
  services:
    - docker:27.1.1-dind
  variables:
    DOCKER_TLS_CERTDIR: ""
  stage: containerize
  before_script:
    - docker login -u $DOCKER_USER -p $DOCKER_PASS
  script:
    - docker build -t $IMAGE_NAME:$IMAGE_TAG .
    - docker push $IMAGE_NAME:$IMAGE_TAG
```

Adding Variables
Here we have some variables, namely DOCKER_USER and DOCKER_PASS

Adding Docker user



Adding the docker password


https://gitlab.com/sosoebot/boardgame/-/settings/ci_cd#js-cicd-variables-settings
Search or go to...
Project
Secure
Deploy
Operate
Monitor
Analyze
Settings
General
Integrations
Webhooks
Access tokens
Repository
Merge requests
CI/CD
Packages and registries
Monitor
Usage quotas
What's new
Help
sosoebot / Boardgame / CI/CD Settings
Save changes
Project variables
Variables can be accidentally exposed in a job log, or maliciously sent to a third party server. The masked variable fe
exposing variable values, but is not a guaranteed method to prevent malicious users from accessing variables. How
CI/CD Variables 4
Key
Value
DOCKER_PASS
Protected Expanded
DOCKER_USER
Protected Masked Expanded
SONAR_HOST_URL
Masked Expanded
SONAR_TOKEN
Masked Expanded
Group variables (inherited)
These variables are inherited from the parent group.
CI/CD Variables 0
Key
Environments
There are no variables yet.
Visibility
Visible
Can be seen in job logs.
Masked
Masked in job logs but value can be revealed in CI/CD settings. Requires values to meet regular expressions requirements.
Masked and hidden
Masked in job logs, and can never be revealed in the CI/CD settings after the variable is saved.
Flags
Protect variable
Edit
Expand variable reference
$ will be treated as the start of a reference to another variable.
Description (optional)
The description of the variable's value or usage.
Key
DOCKER_PASS
You can use CI/CD variables with the same name in different places, but the variables might overwrite each other. What is the order of precedence for variables?
Value
Save changes
Delete variable
Cancel

STEP 9: Enabling Jobs to run sequentially

We have to make the code sequentially since some jobs/stages depends on each other.
To do this we have to add stages as follows:

```
stages:
  - test
  - sonarqube
  - build
  - containerize
```

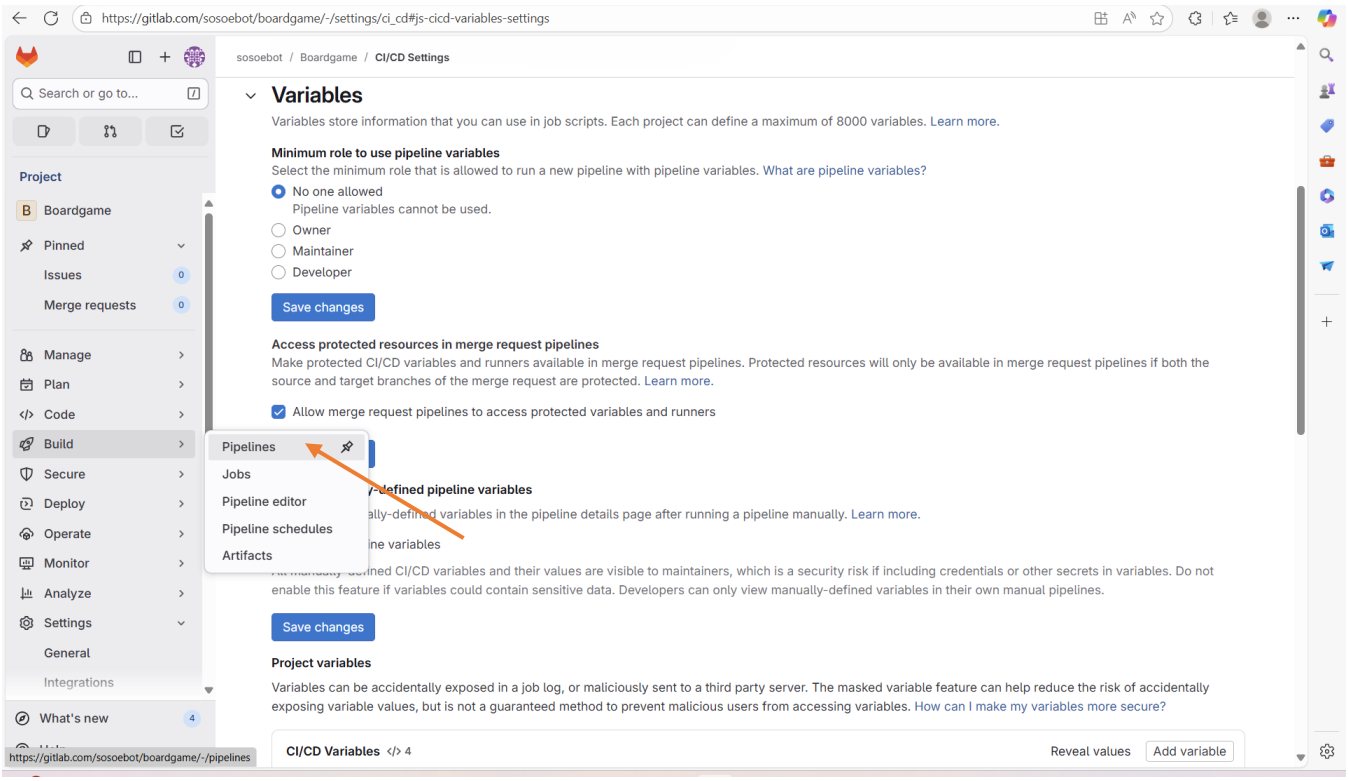
Final Code

```
# Use Maven image with OpenJDK 17
image: maven:3.8.4-openjdk-17


stages:
  - test
  - sonarqube
  - build
  - containerize

variables:
  IMAGE_NAME: ebotsidneysmith/demoapp
  IMAGE_TAG: boardgame-0.0.1


test-job:
  stage: test
  script:
    - echo "Running tests..."
    - mvn test
  tags:
    - project-runner

sonarqube-check:
  stage: sonarqube
  image:
    name: sonarsource/sonar-scanner-cli:latest
  variables:
    SONAR_USER_HOME: "${CI_PROJECT_DIR}/.sonar"  # Defines the location of the analysis
task cache
    GIT_DEPTH: "0"  # Tells git to fetch all the branches of the project, required by the
analysis task
  cache:
    policy: pull-push
    key: "sonar-cache-$CI_COMMIT_REF_SLUG"
    paths:
```

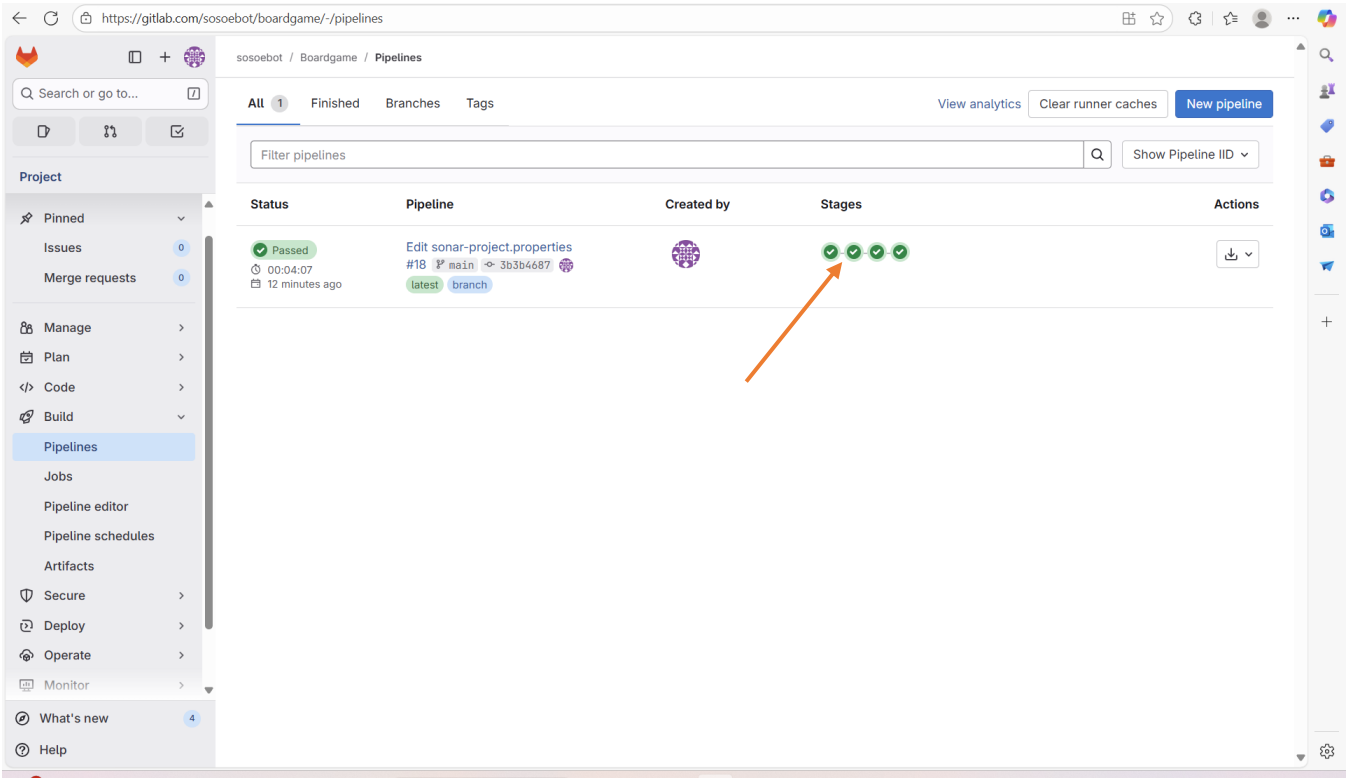
```
    - "${SONAR_USER_HOME}/cache"
    - sonar-scanner/
  script:
    - sonar-scanner
  allow_failure: true
  only:
    - main

build-job:
  stage: build
  script:
    - echo "Running tests..."
    - mvn package
  artifacts:
     paths:
       - target/*.jar
  tags:
    - project-runner

build_image_push-job:
  image: docker:27.1.1
  services:
    - docker:27.1.1-dind
  variables:
    DOCKER_TLS_CERTDIR: ""
  stage: containerize
  before_script:
    - docker login -u $DOCKER_USER -p $DOCKER_PASS
  script:
    - docker build -t $IMAGE_NAME:$IMAGE_TAG .
    - docker push $IMAGE_NAME:$IMAGE_TAG
```

STEP 10: Result

Click on "**Build**"

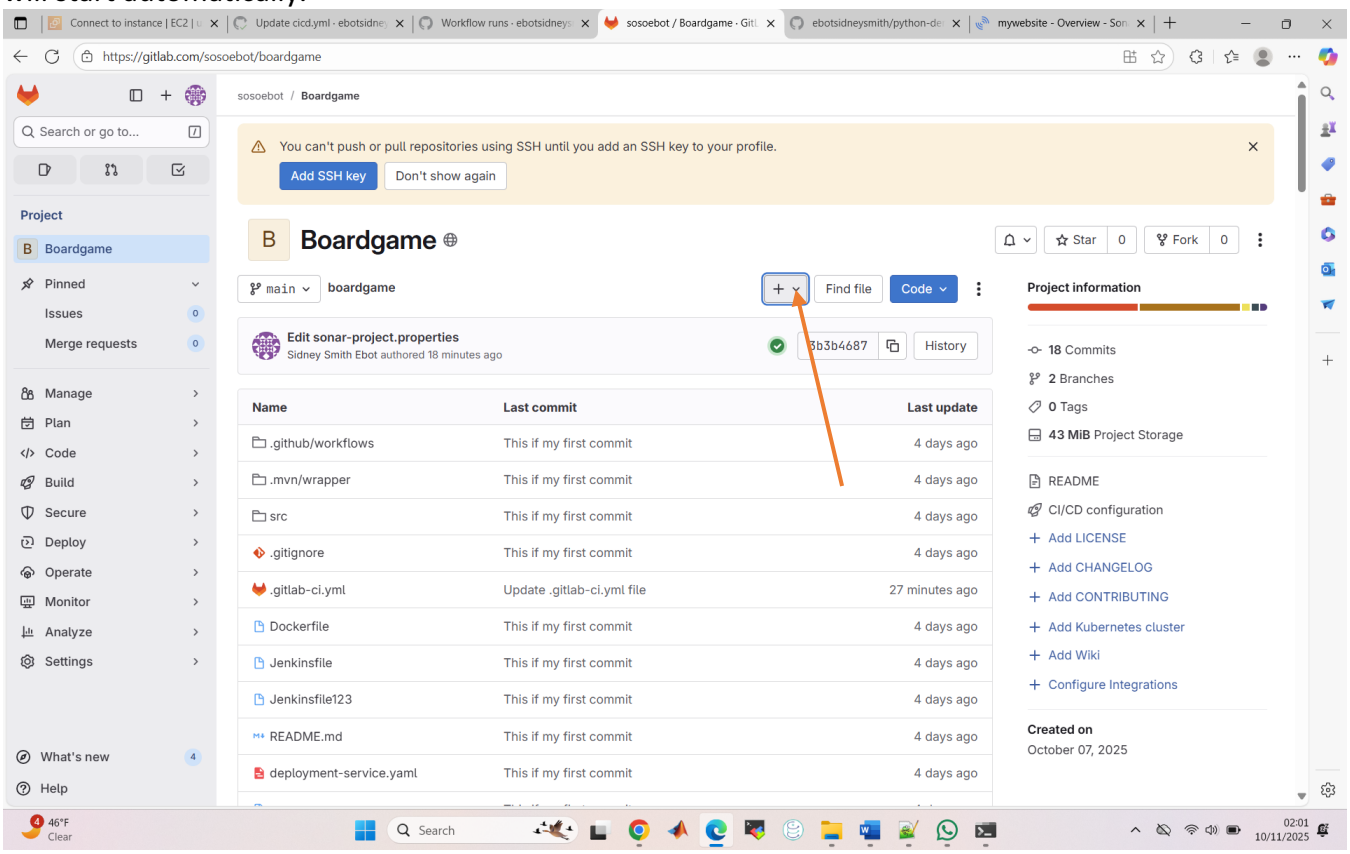


Select "**Pipeline**"

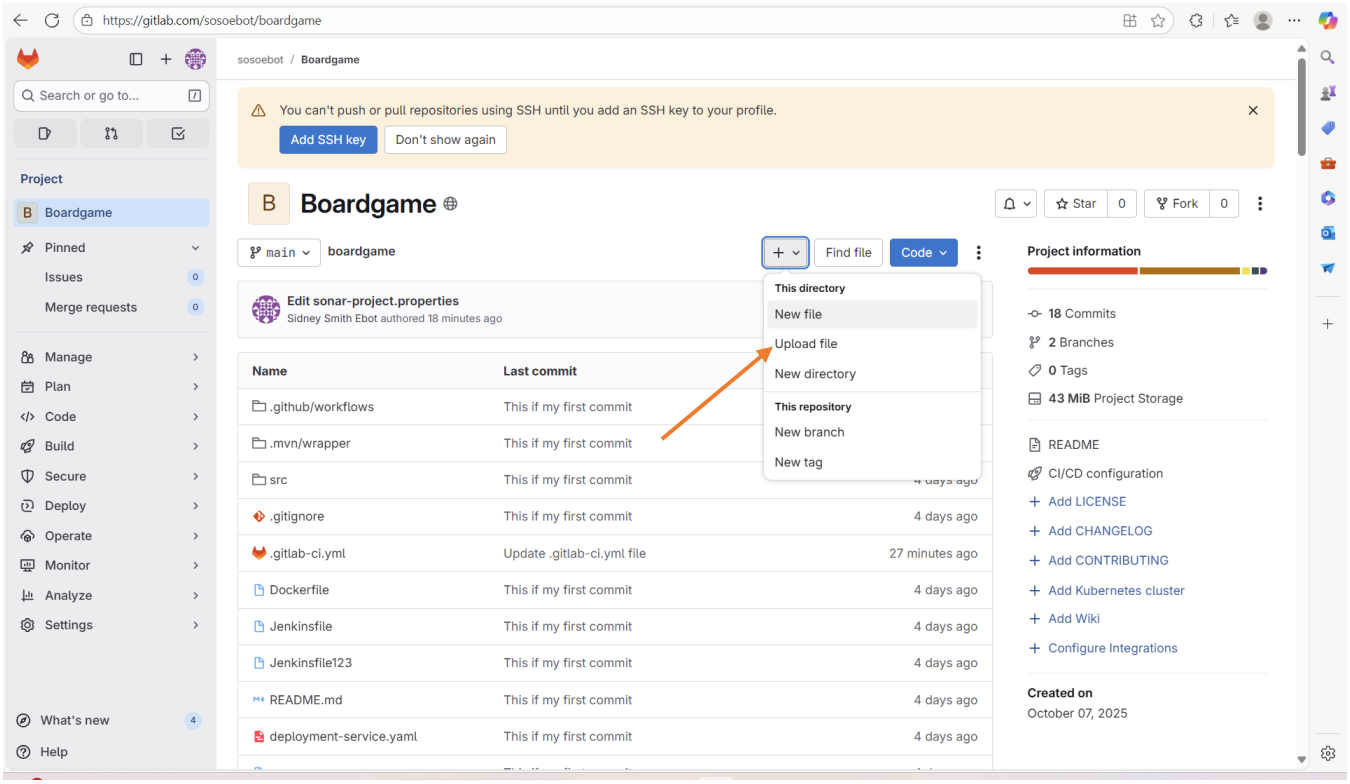


You can see that the build is successful.

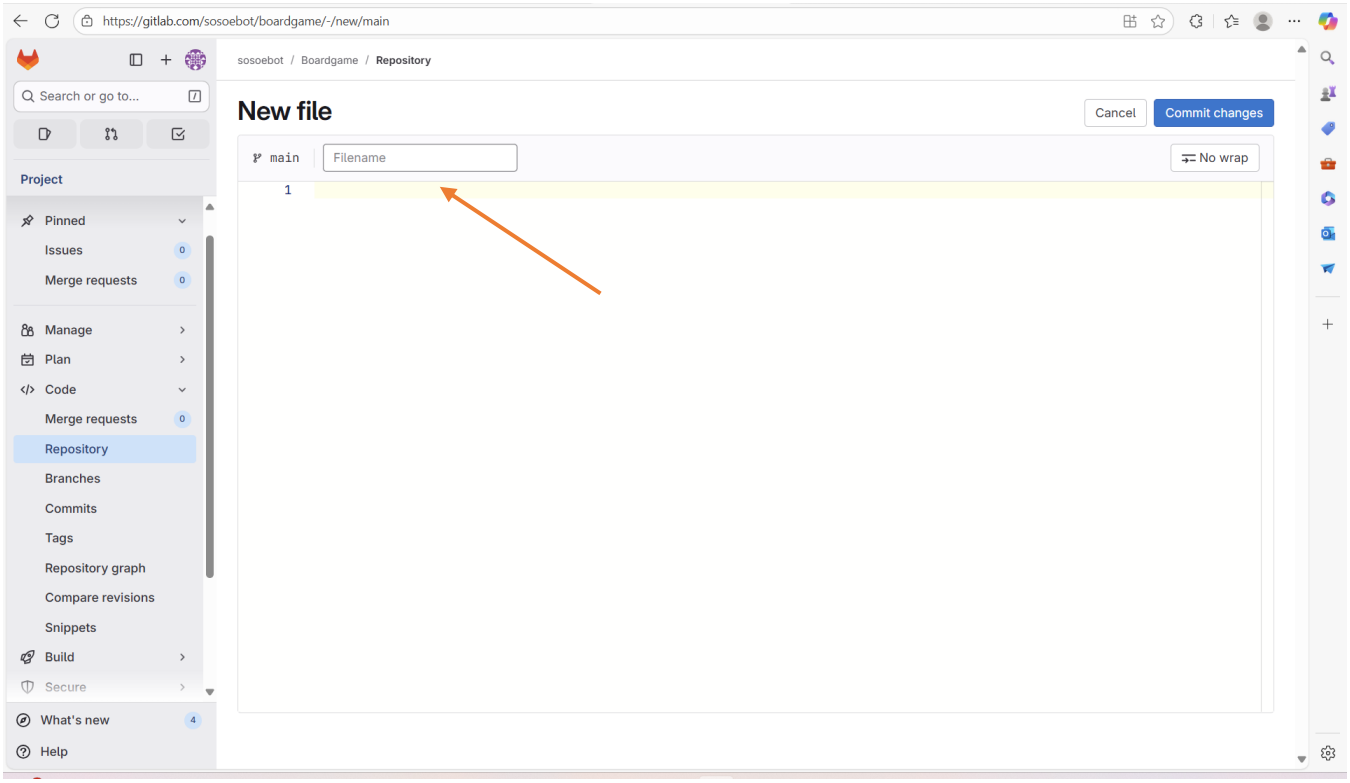
Now, let us try to make some changes in the Gitlab repo and commit the changes. Then see if the Pipeline will start automatically.



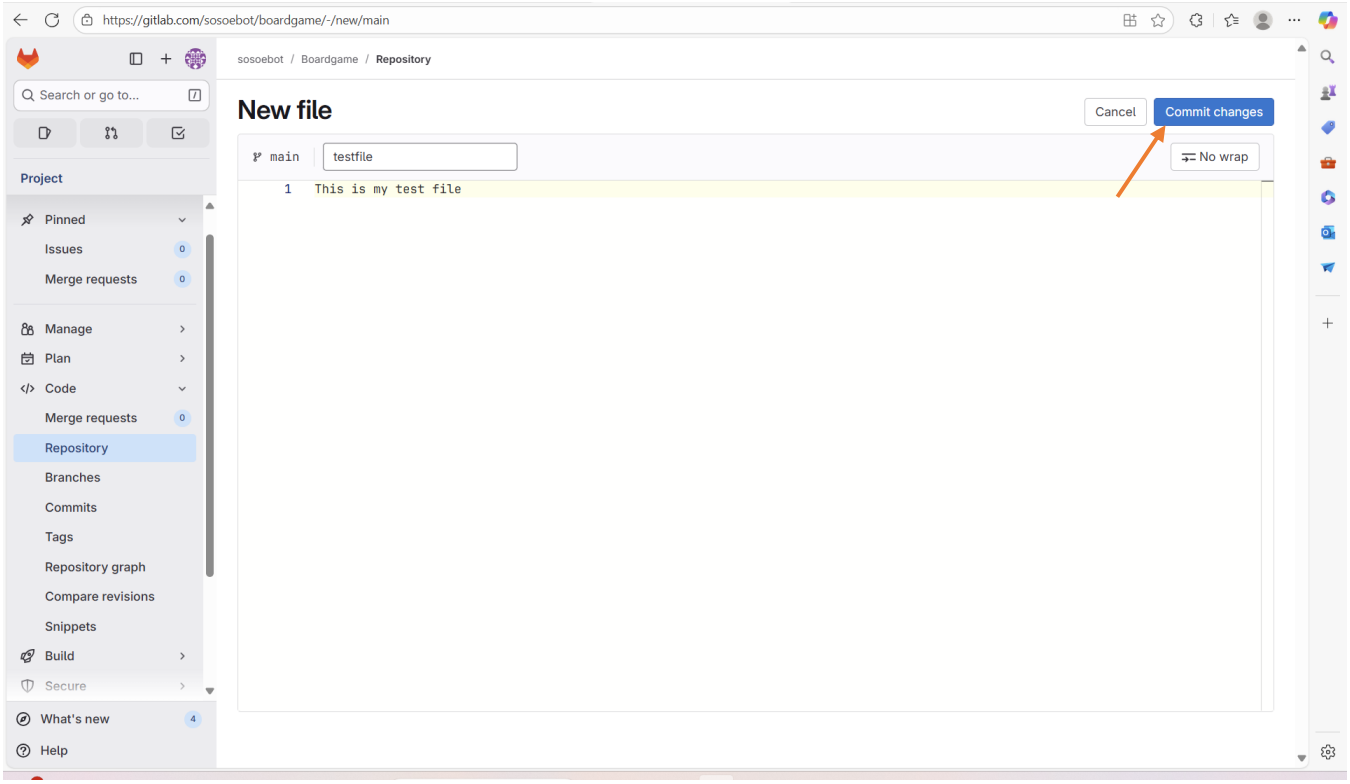
We will add a text file. Click on "**+**"



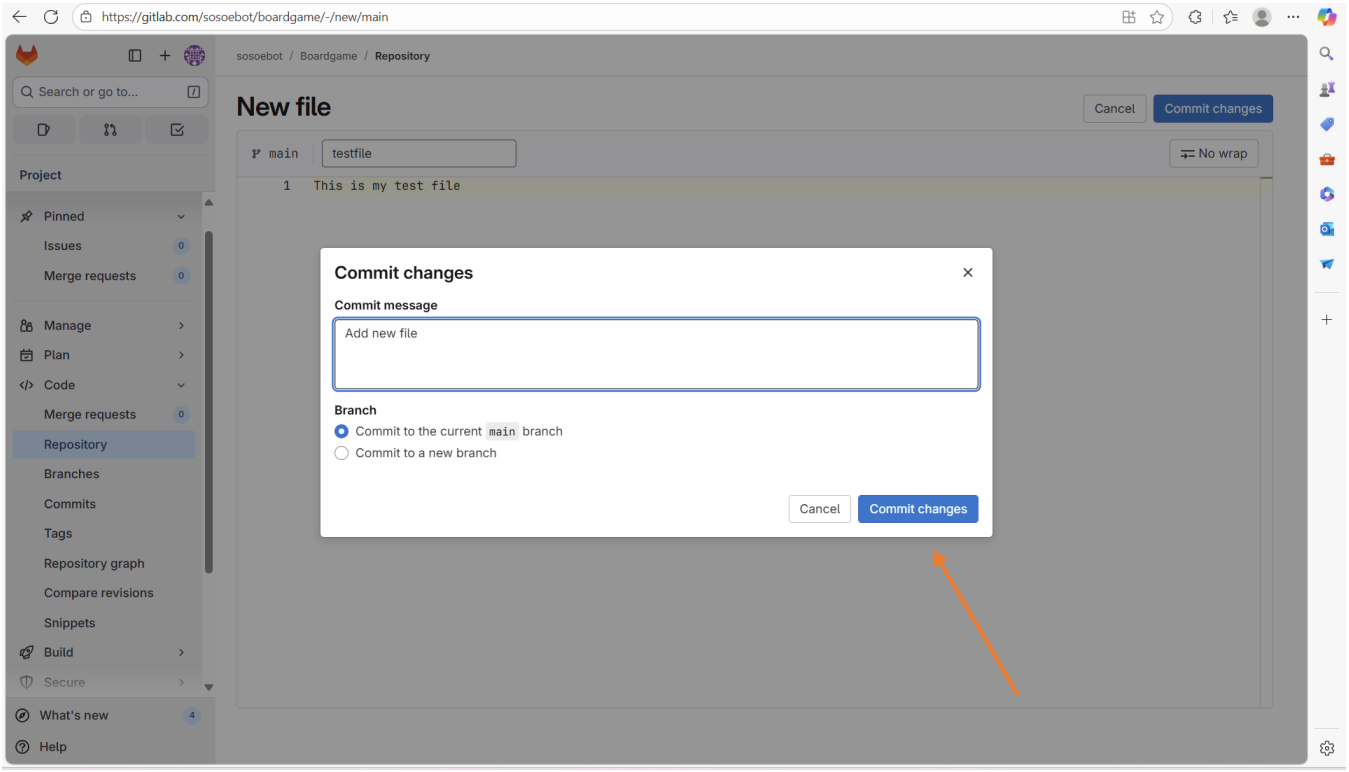
Select "**New File**"



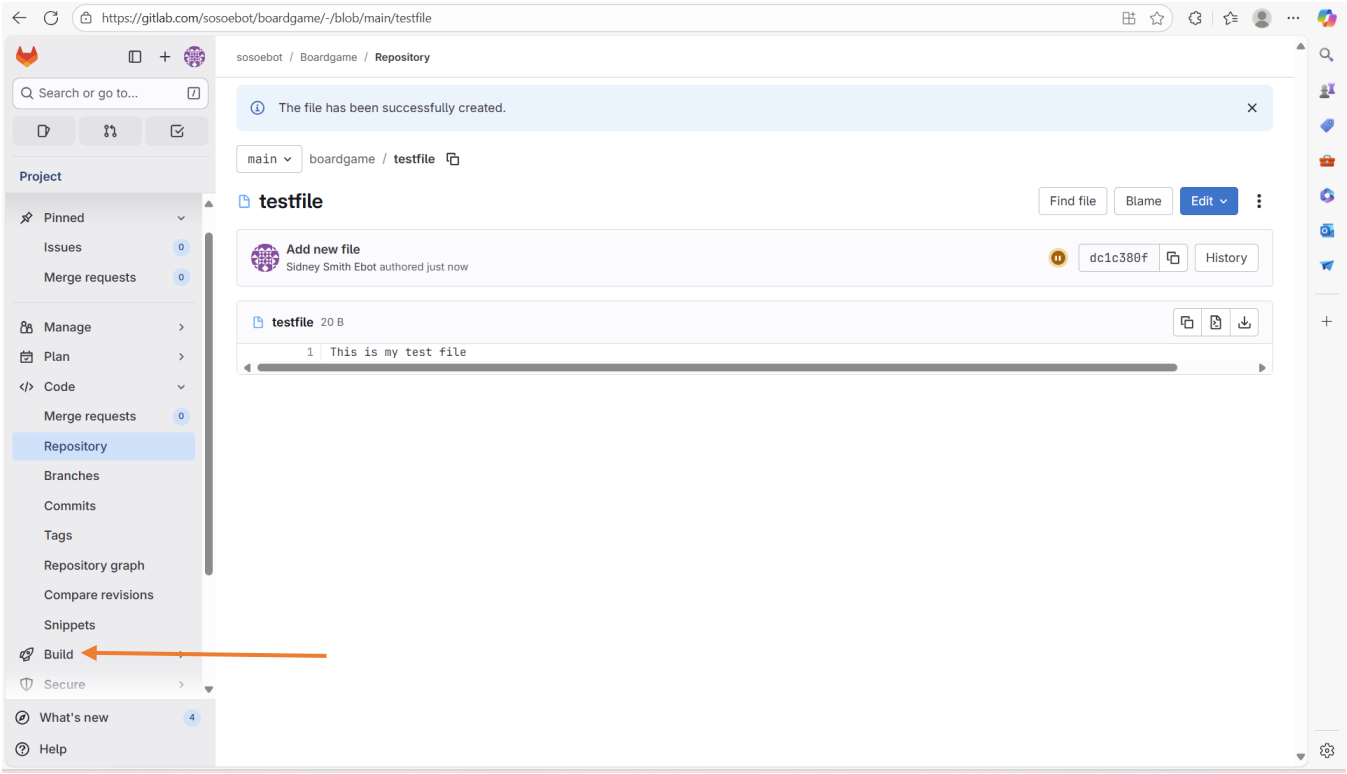
Give the file a name and add some text on the file



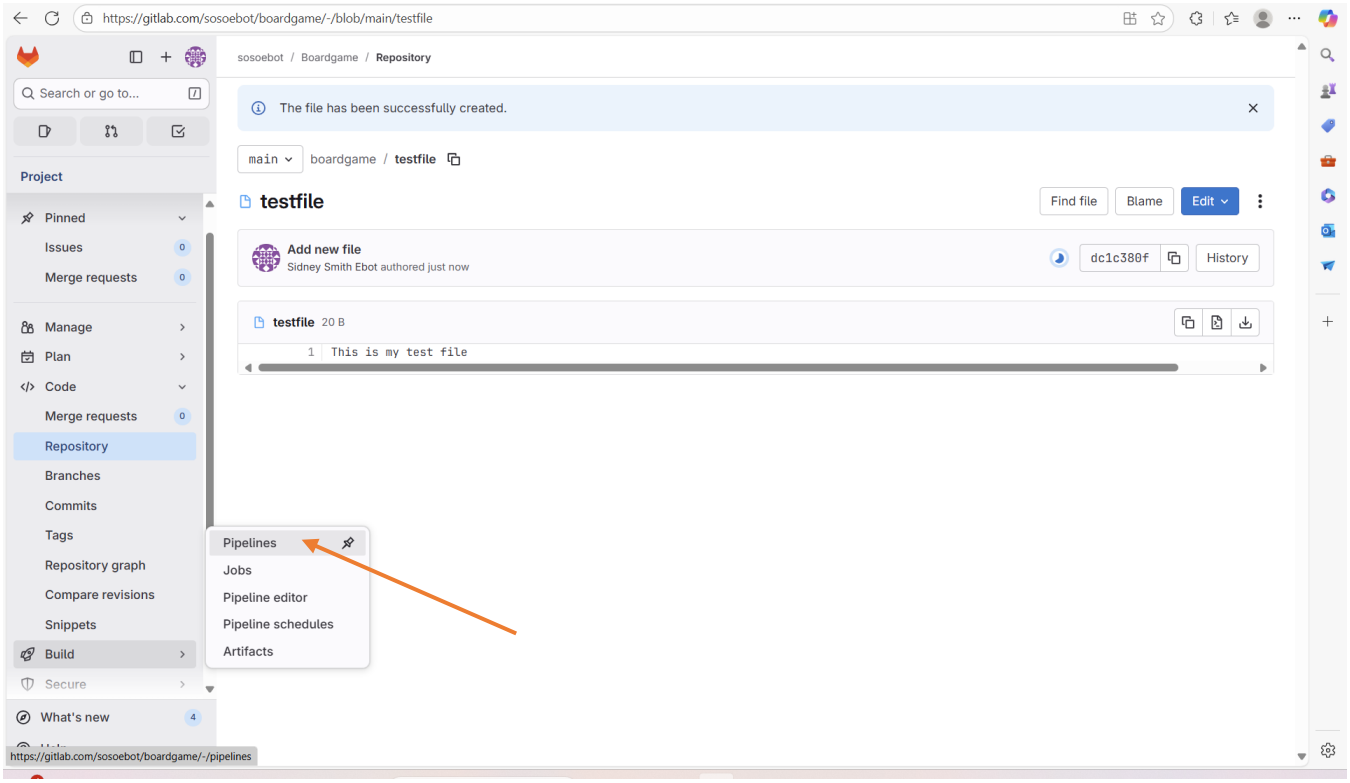
Then commit the changes by clicking on "**Commit changes**"



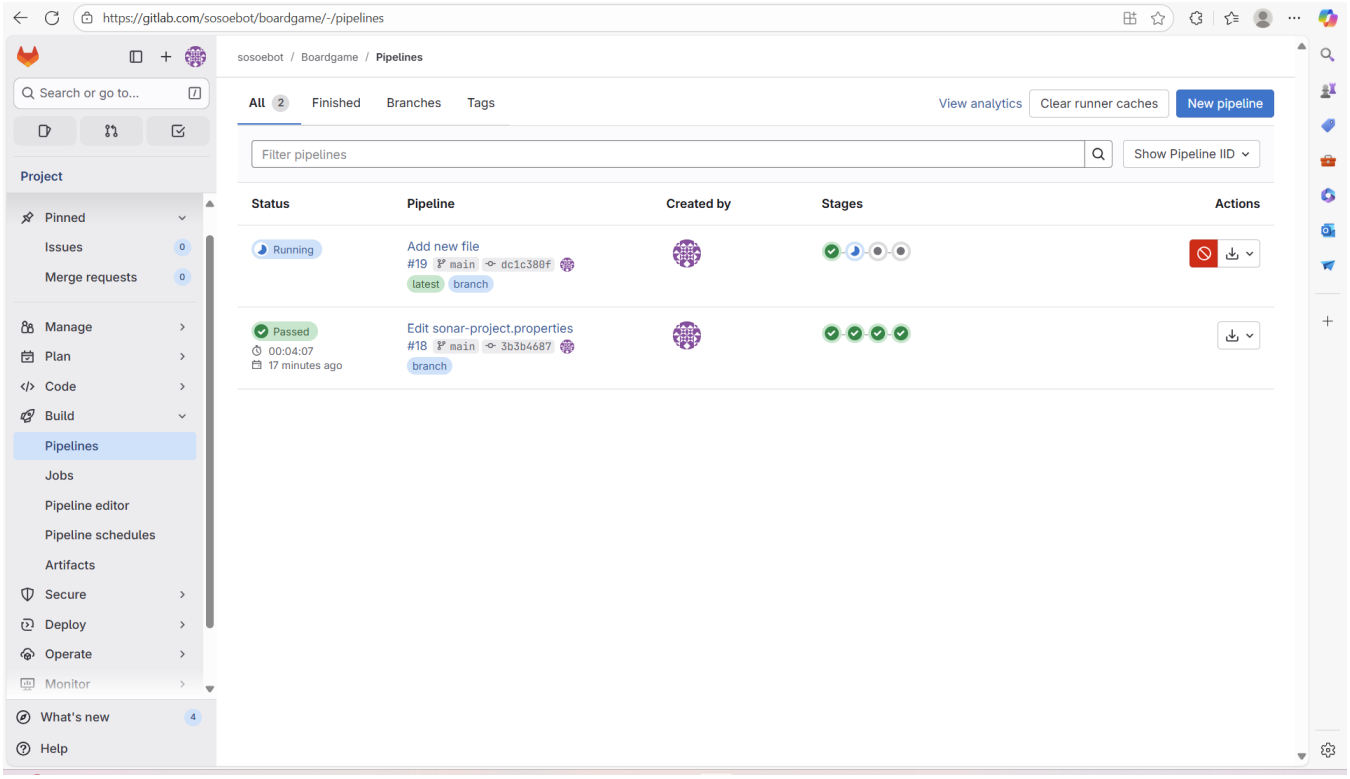
Then click on "**Commit changes**" again



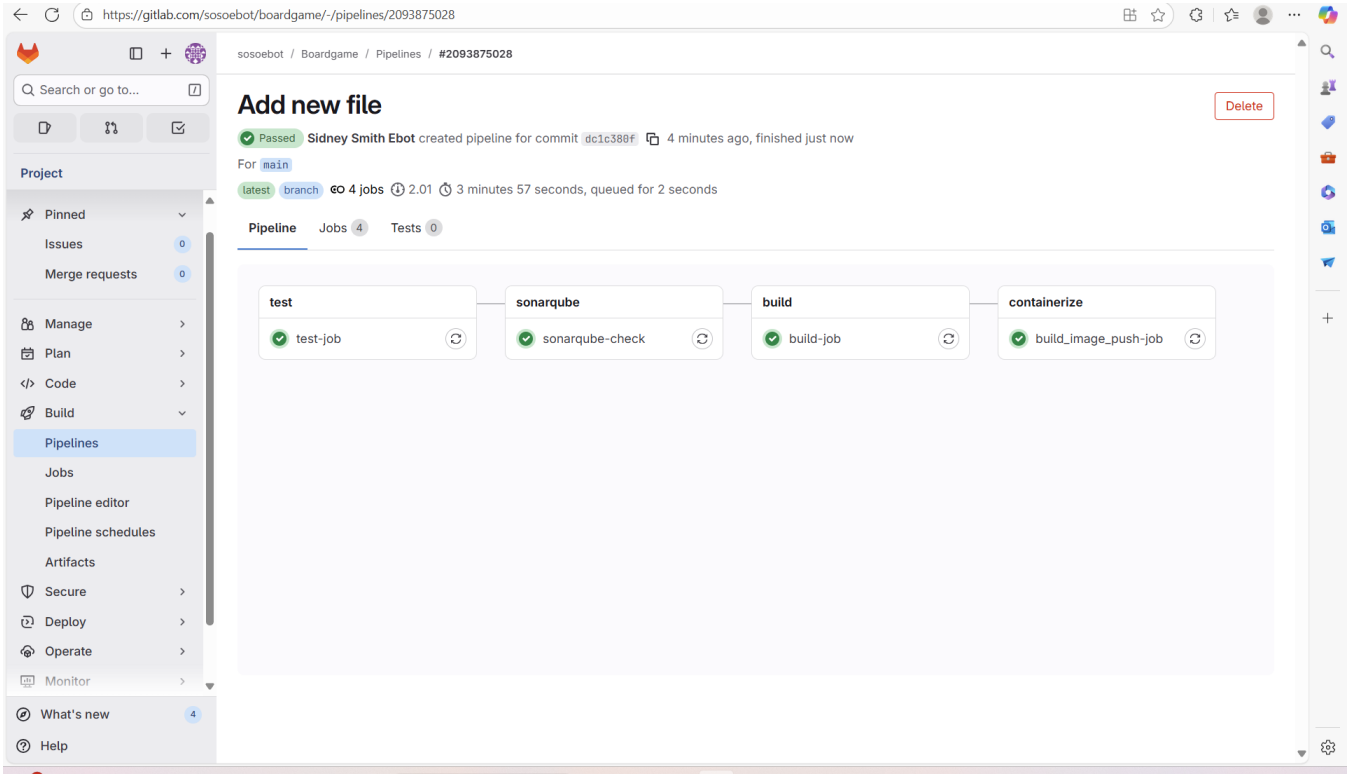
Then click on "**Build**"



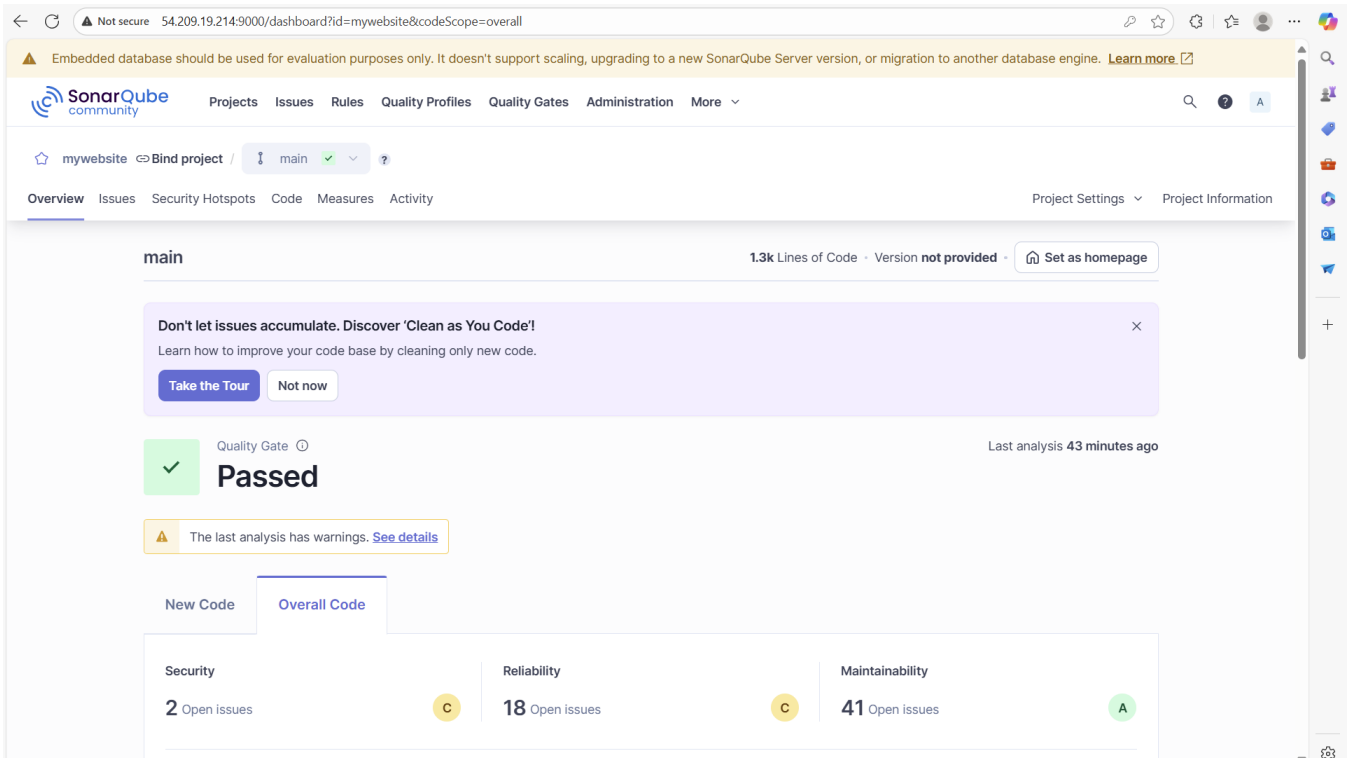
And select “**Pipeline**”

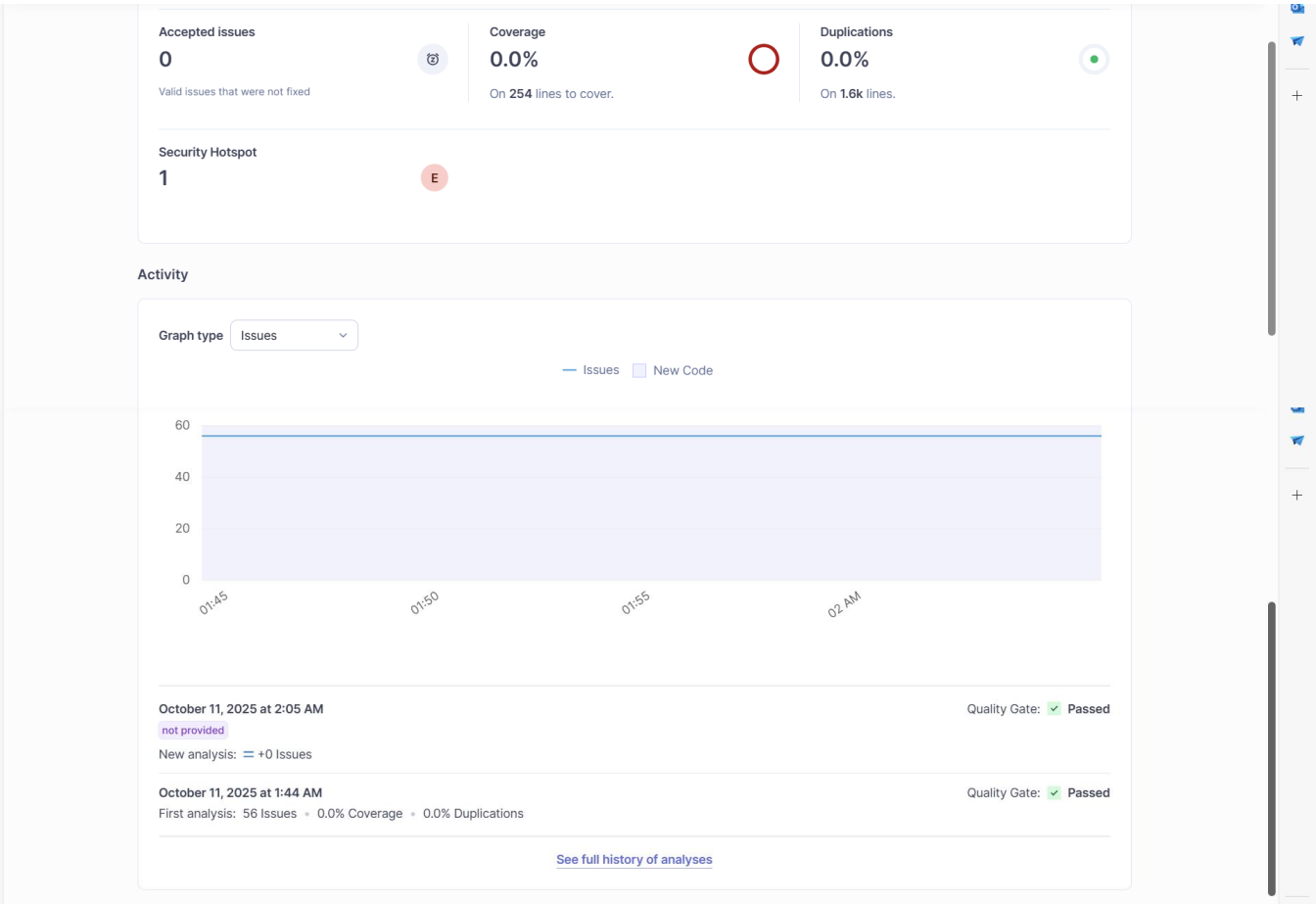


You can see that the pipeline has started running automatically



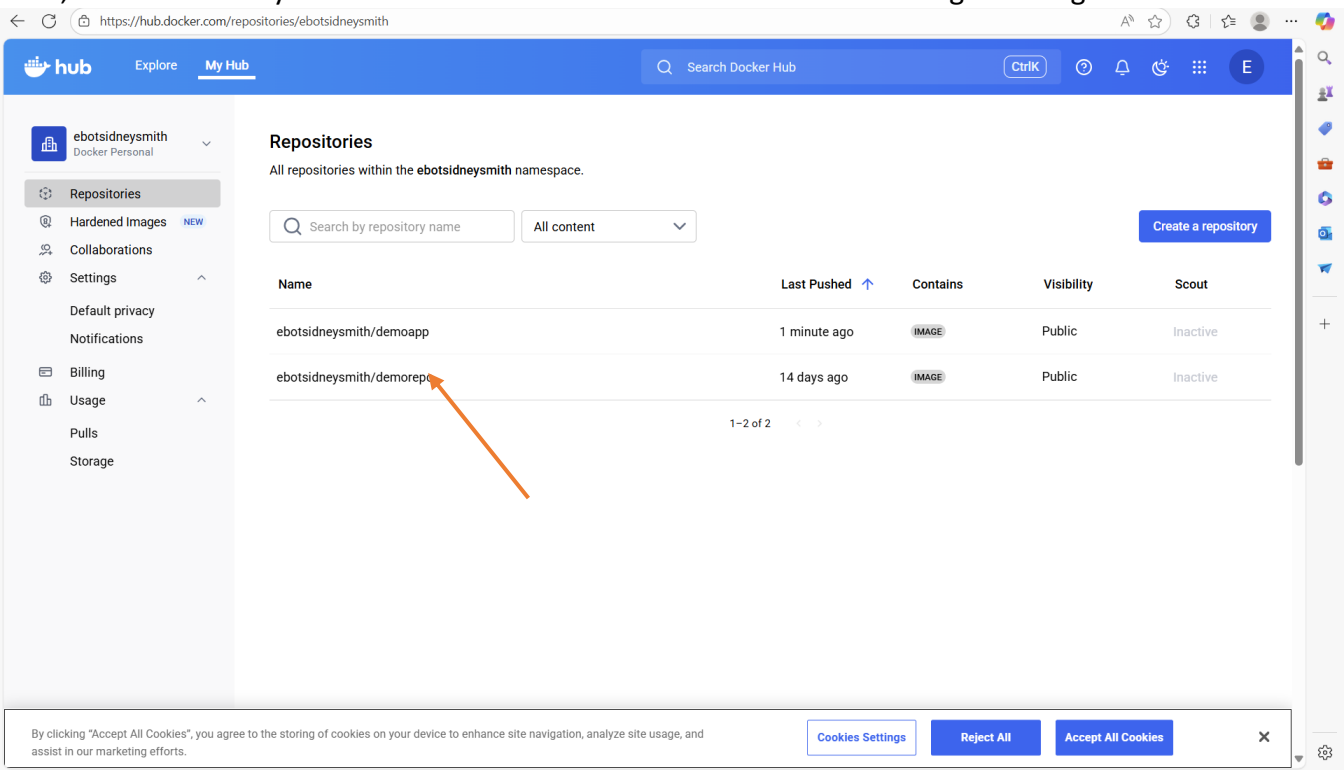
You can see that the Pipeline is successful. Then check the SonarQube results



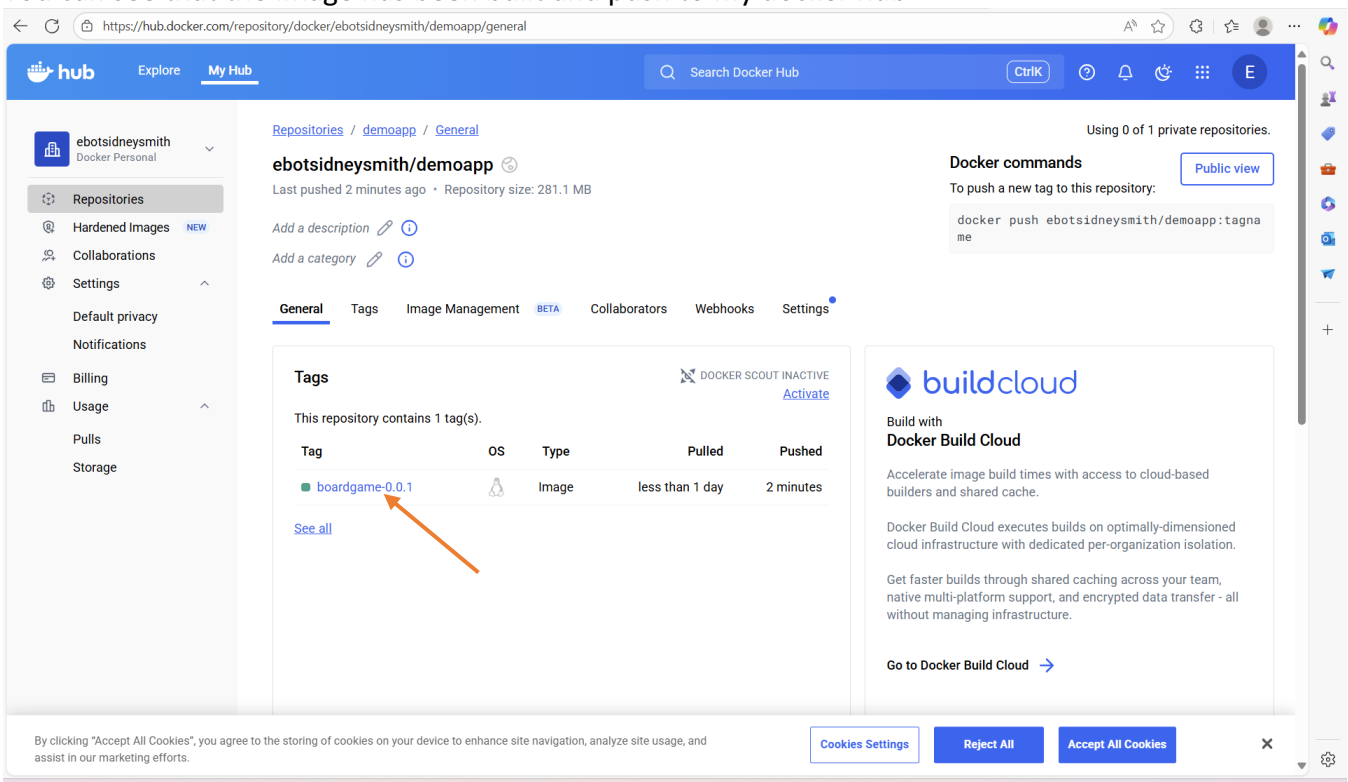


You can see that the code passed the Quality Gate analysis.

Now, let me check my Docker hub if there is a new folder there containing the image



You can see that the image has been built and push to my docker hub



This is the built image.