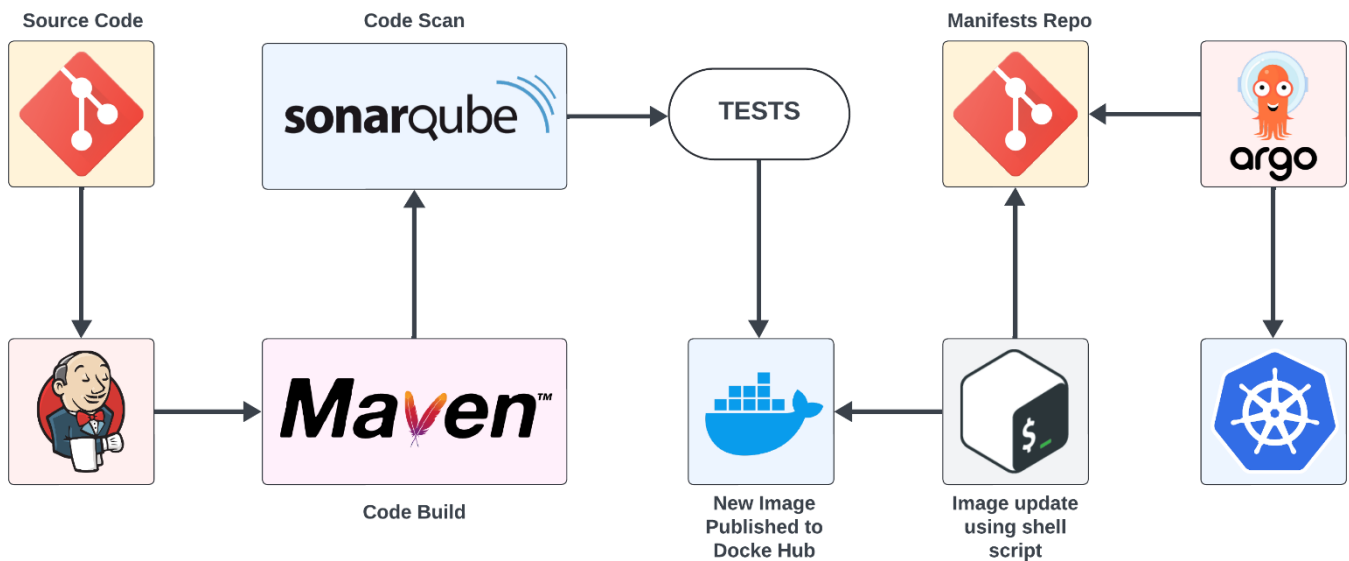


END TO END CICD IMPLEMENTATION OF SPRING BOOT APPLICATION ON AWS



1. Introduction

The CI/CD pipeline automates the process of building, testing, and deploying a Spring Boot application. By integrating tools like Jenkins, SonarQube, Maven, Docker, Argo CD, and AWS, the pipeline ensures code quality, quick builds, and seamless deployments to a Kubernetes cluster.

2. Tools and Technologies Used:

- **Jenkins:** For continuous integration and automation of the build process.
- **SonarQube:** For static code analysis and ensuring code quality.
- **Maven:** For managing dependencies and building the Spring Boot application.
- **Docker:** For containerizing the application.
- **Argo CD:** For continuous deployment to a Kubernetes cluster.
- **Git:** For version control and source code management.
- **AWS:** For provisioning servers (EC2 instances) to host Jenkins, SonarQube, and other services.

All The codes are available in

- <https://github.com/shehan404/CICD-project-1.git>
- <https://github.com/shehan404/CICD-Project-1-Manifest.git>

2. Pipeline Architecture

The pipeline is designed to automate the process from code commits to deployment. Below is the architecture:

Source Code Repository (Git):

The pipeline starts with a source code commit that is manually triggered or scheduled within Jenkins.

Jenkins:

Hosted on an AWS EC2 instance. Jenkins starts the pipeline based on manual triggers or schedules. Jenkins triggers SonarQube to perform a code scan. After a successful code scan, Jenkins initiates the build process using Maven.

SonarQube:

Hosted on an AWS EC2 instance. Analyzes the source code for bugs, vulnerabilities, and code smells. Generates a report that is used to decide whether to proceed with the build.

Maven:

Builds the Spring Boot application. Generates an artifact that is then containerized using Docker.

Docker:

The build artifact is containerized. A new Docker image is created and pushed to Docker Hub.

Manifest Repository (Git):

A separate repository for Kubernetes manifests. The manifest files are updated with the new Docker image using a shell script.

ArgoCD:

ArgoCD monitors the manifests repository. When changes are detected, ArgoCD automatically deploys the updated application to the Kubernetes cluster hosted on AWS.

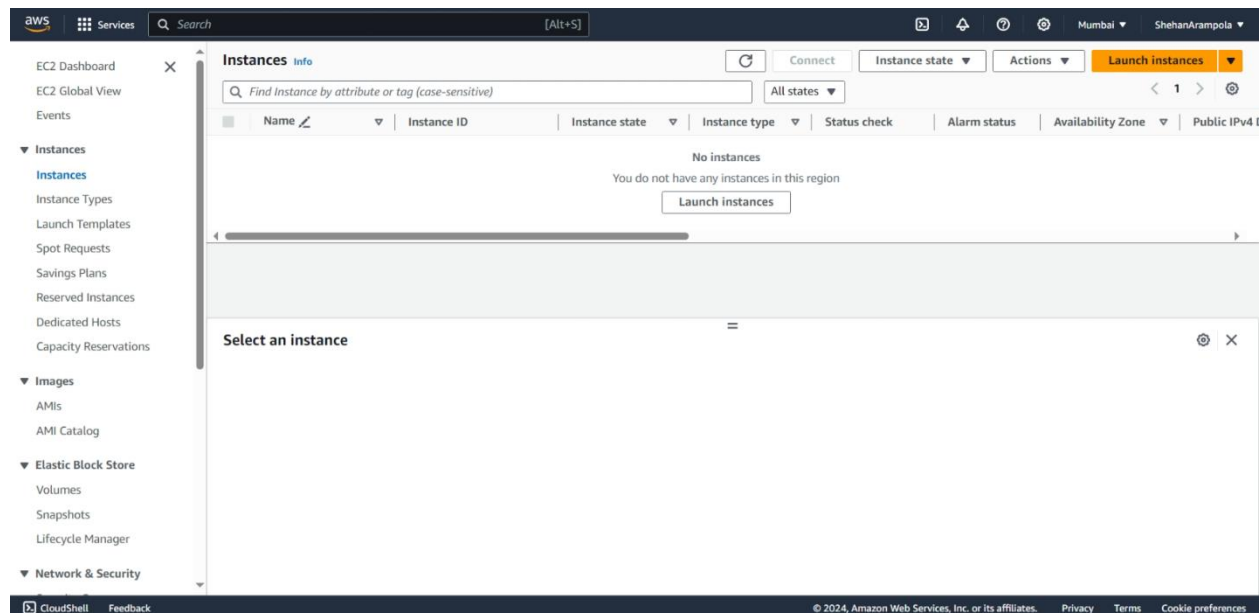
3. Setup and Configuration

1. AWS:

Provision EC2 instances. Follow these steps.

Go to AWS console and create two EC2 instance as follows with the same settings choose the name as you like (Mine: Jenkins-server, Deploy-server).

Click Launch instances



Setup following settings

▼ Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Recents

Quick Start

Amazon Linux

aws

macOS

Mac

Ubuntu

ubuntu

Windows

Microsoft

Red Hat

Red Hat

SUSE Li

SUSE

Browse more AMIs

Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Ubuntu Server 24.04 LTS (HVM), SSD Volume Type
ami-0ad21ae1d0696ad58 (64-bit (x86)) / ami-01f6c796d6dbc1e36 (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

Free tier eligible

Description

Ubuntu Server 24.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).

Architecture

AMI ID

Verified provider

64-bit (x86)

ami-0ad21ae1d0696ad58

▼ Summary

Number of instances Info
2
When launching more than 1 instance, consider EC2 Auto Scaling

Software Image (AMI)
Canonical, Ubuntu, 24.04 LTS, ...read more
ami-0ad21ae1d0696ad58

Virtual server type (instance type)
t2.large

Firewall (security group)
CICD Devops project

Storage (volumes)
1 volume(s) - 30 GiB

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 750 hours of public IPv4 address usage per month, 30 GiB

Cancel

Launch instance

Review commands

▼ Instance type Info | Get advice

Instance type

t2.large
Family: t2 2 vCPU 8 GiB Memory Current generation: true
On-Demand Windows base pricing: 0.1272 USD per Hour
On-Demand SUSE base pricing: 0.1992 USD per Hour
On-Demand Linux base pricing: 0.0992 USD per Hour
On-Demand RHEL base pricing: 0.128 USD per Hour

All generations

Compare instance types

Additional costs apply for AMIs with pre-installed software

Shehan Arampola | <https://www.linkedin.com/in/shehan-arampola> | <https://github.com/shehan404>

Choose key pair. If you don't have one create a new key pair. The .pem file created is used for authenticating when remote login into EC2 instances.

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

devopsprojects ▼

[Create new key pair](#)

Choose a security group. In the inbound rules of SG, we allow all inbound traffic to make our project easy. But in real situations it should be allowed only traffic that we needed. You can simply search for security group in search bar and setup it.

▼ Network settings [Info](#)

Edit

Network [Info](#)

vpc-07af9da7cd63546bb | project1-cicd-vpc

Subnet [Info](#)

No preference (Default subnet in any availability zone)

Auto-assign public IP [Info](#)

Enable

[Additional charges apply](#) when outside of [free tier allowance](#)

Firewall (security groups) [Info](#)

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

☐ Create security group ☒ Select existing security group

Common security groups [Info](#)

Select security groups ▼

CICD Devops project sg-085a2737042868d63 ✕

VPC: vpc-07af9da7cd63546bb

[Compare security group rules](#)

Security groups that you add or remove here will be added to or removed from all your network interfaces.

▼ Configure storage

Info

Advanced

1x

30

GiB

gp3

Root volume (Not encrypted)

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage

×

Add new volume

The selected AMI contains more instance store volumes than the instance allows. Only the first 0 instance store volumes from the AMI will be accessible from the instance

🕒 Click refresh to view backup information

The tags that you assign determine whether the instance will be backed up by any Data Lifecycle Manager policies.

↻

0 x File systems

Edit

Click Launch instance and instances will be created. Now come to instances menu and rename them.

Instances (2)

Info

↻

Connect

Instance state ▼

Acti

Find Instance by attribute or tag (case-sensitive)

All states ▼

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
<input type="checkbox"/>	Jenkins-server	i-0495c3d1ea86f01db	Running	t2.large	-	View alarms +	ap-south-1b
<input type="checkbox"/>	Deploy-server	i-0bd8462c24b5f8696	Running	t2.large	Initializing	View alarms +	ap-south-1b

Now instances are created.

Select each instance and copy the public IP address of both instances.

Instances (1/2)

Info

↻

Connect

Instance s

Find Instance by attribute or tag (case-sensitive)

All states ▼

<input checked="" type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Av
<input checked="" type="checkbox"/>	Jenkins-server	i-0495c3d1ea86f01db	Running	t2.large	-	View alarms +	ap-
<input type="checkbox"/>	Deploy-server	i-0bd8462c24b5f8696	Running	t2.large	Initializing	View alarms +	ap-

i-0495c3d1ea86f01db (Jenkins-server)

Details

Status and alarms

Monitoring

Security

Networking

Storage

Tags

▼ Instance summary

Info

Instance ID

i-0495c3d1ea86f01db (Jenkins-server)

IPv6 address

-

Public IPv4 address

13.233.116.84 | open address

Instance state

Running

Private IPv4 ac

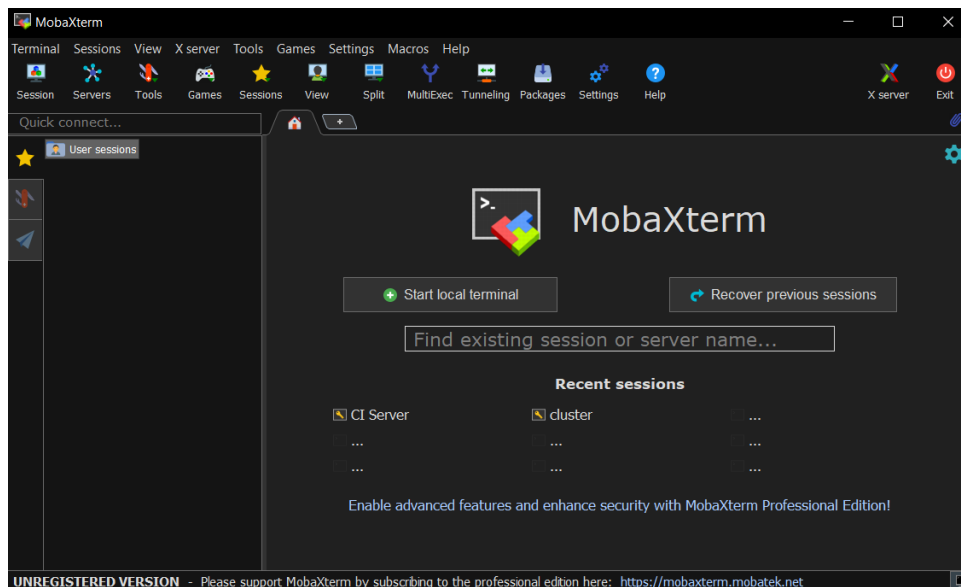
172.31.15

Public IPv4 DN

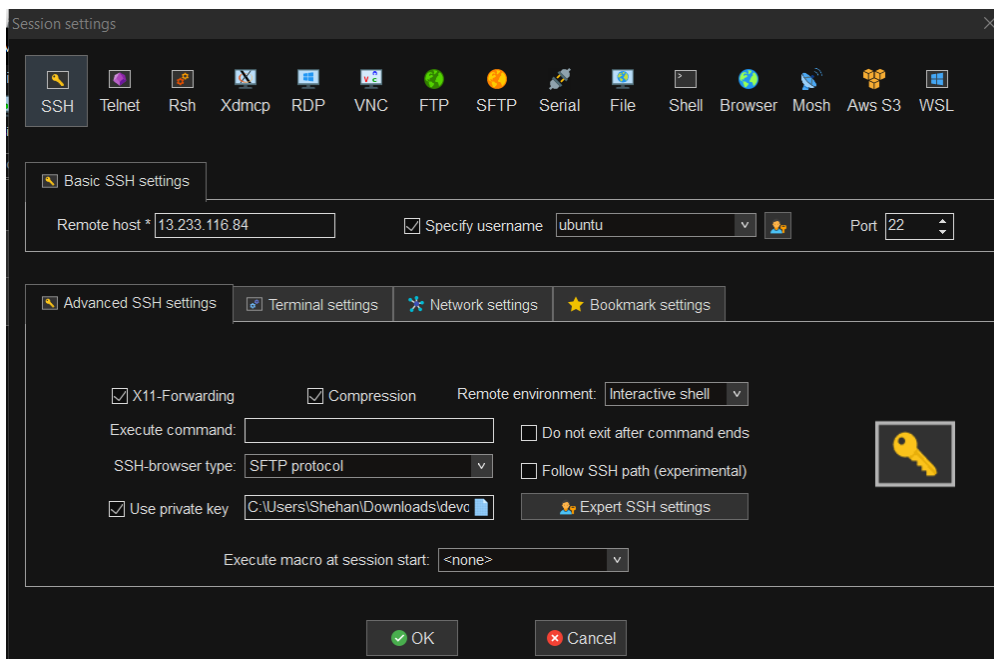
ec2-13-23

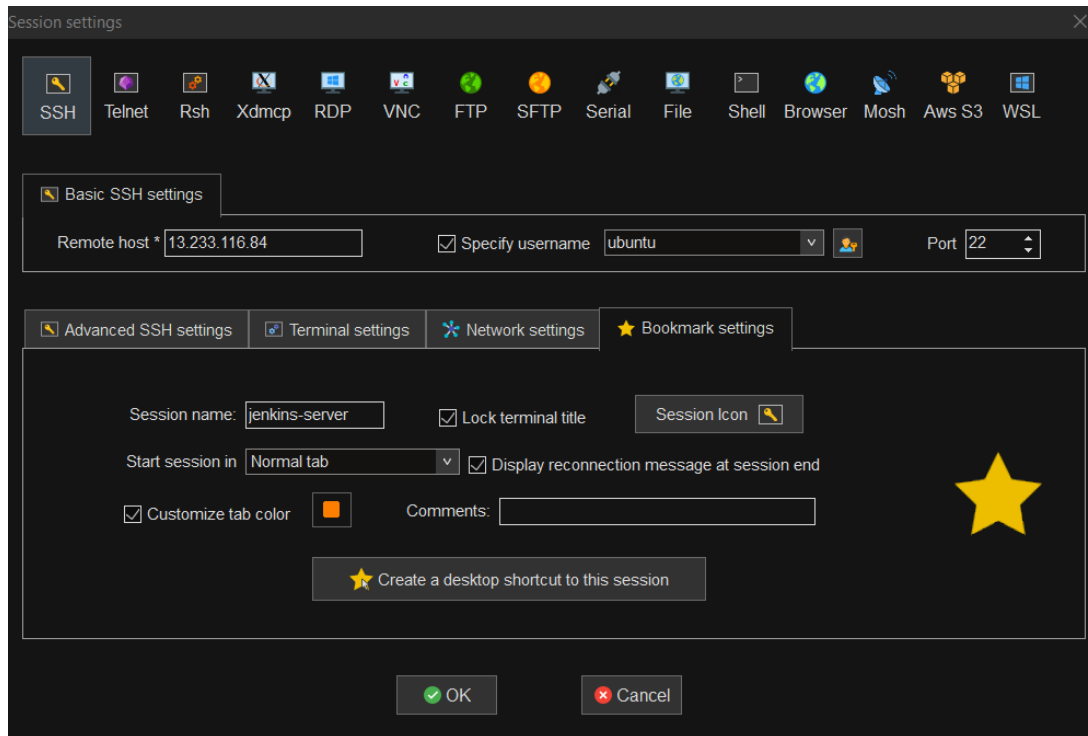
open address |

Let's get remote access to EC2 instances using MobaXterm. Download and install the MobaXterm.

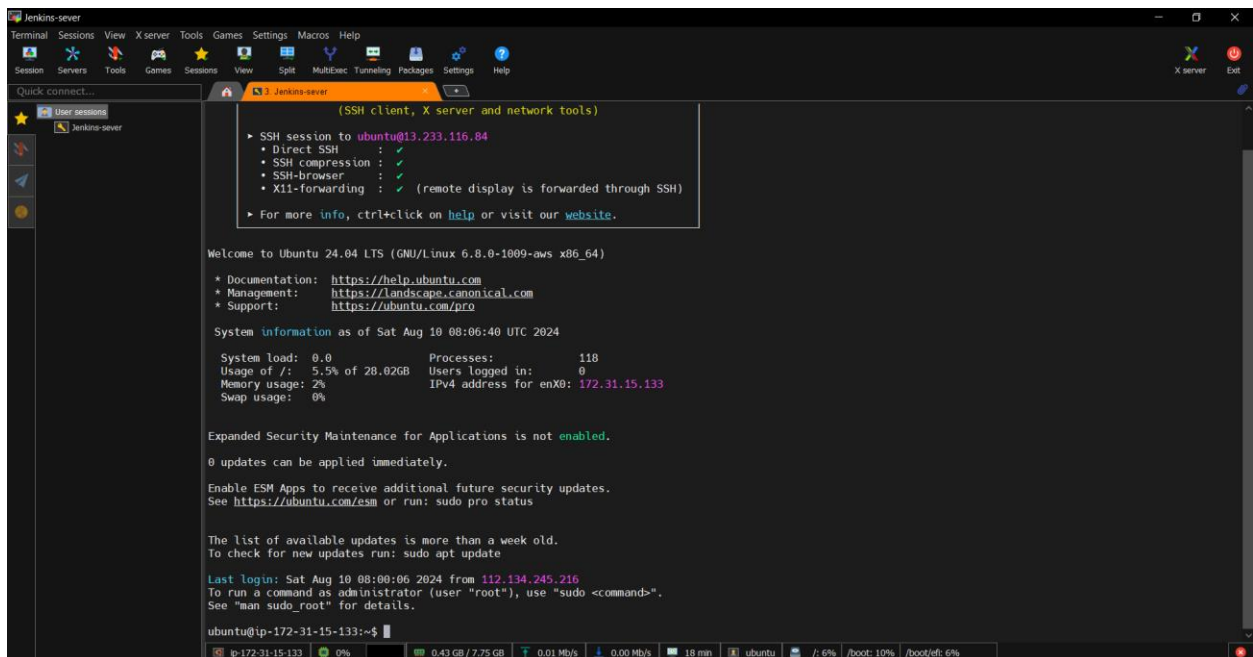


Go to session > SSH and enter the Public IP Address of EC2 instance and specify username as ubuntu in Basic SSH settings. In advanced settings, select use private key and brows for .pem file I mentioned previously. Click OK. In Bookmark settings set session name as Jenkins-server and choose a color for easy identification.

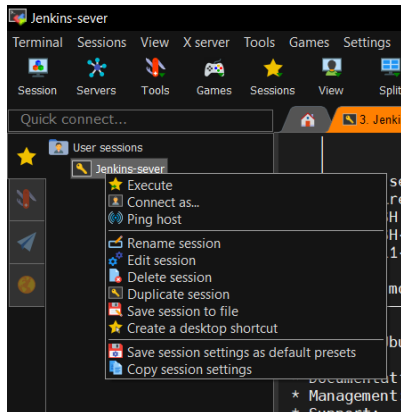




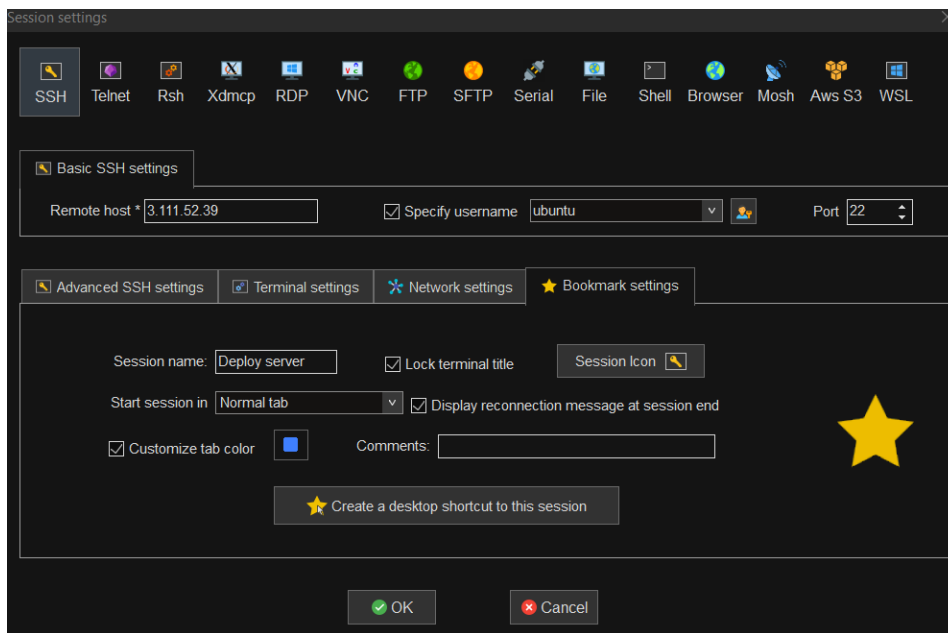
Great now you are remotely connected to the one EC2 instance. In my case I have connected to Jenkins-server.



Right click on Jenkins-server and Click Duplicate Session

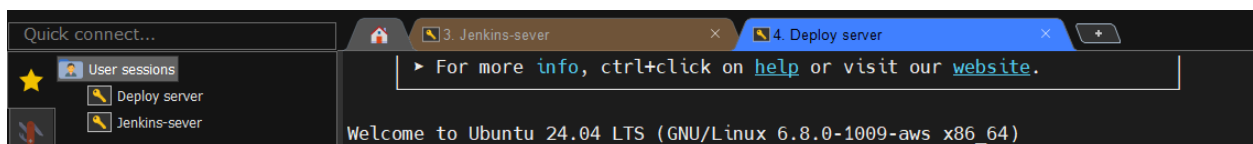


Now right click on duplicated session and click rename session. Insert IP address of other EC2 instance and change the session name and color. Click ok.



Run `sudo apt update` in both servers.

Great We have setup or servers 😊



2. Jenkins

Go to Jenkins server and install jdk. Because Jenkins requires java.

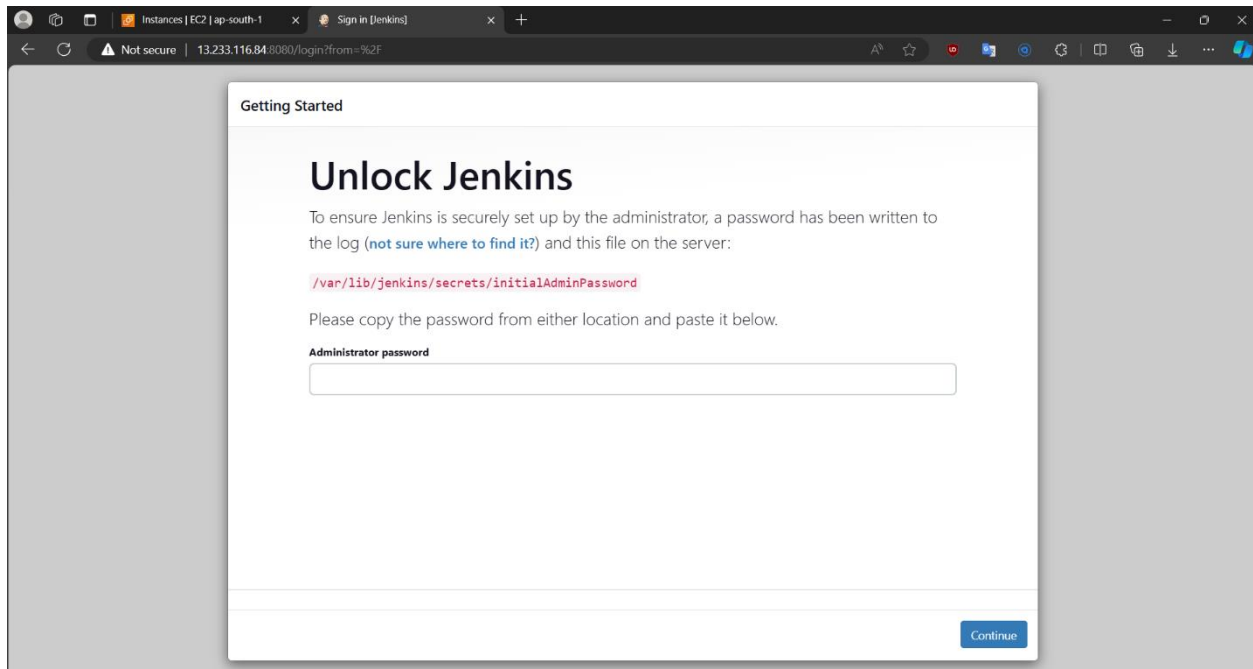
```
sudo apt install default-jre
```

Now install Jenkins using the following commands. (It is much easier if insert following commands in .sh file and run it)

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

Jenkins is default run in port 8080. Go to your browser and insert the following link

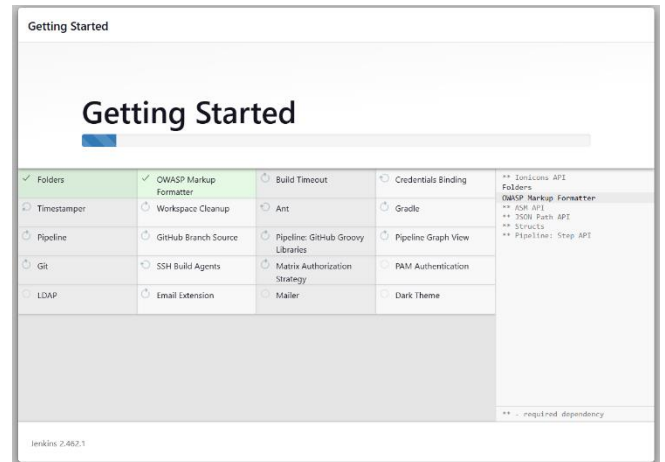
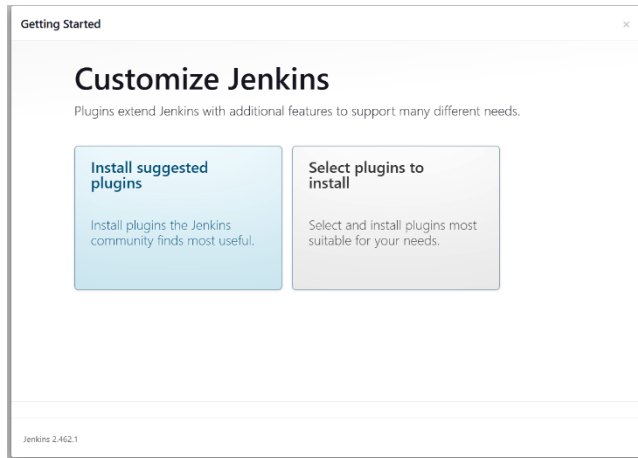
<http://<jenkins-server ec2 instance public ip address>:8080>



For get the administrator password run following command.

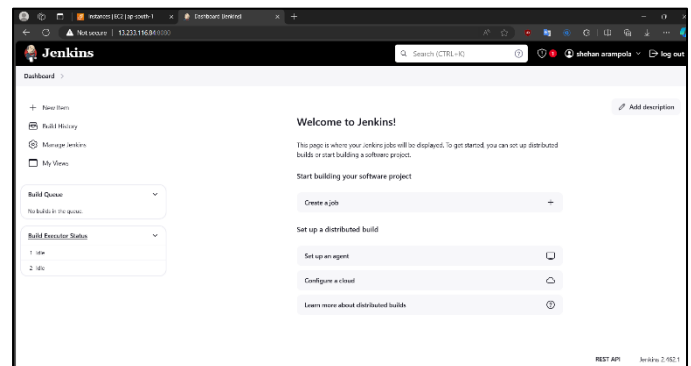
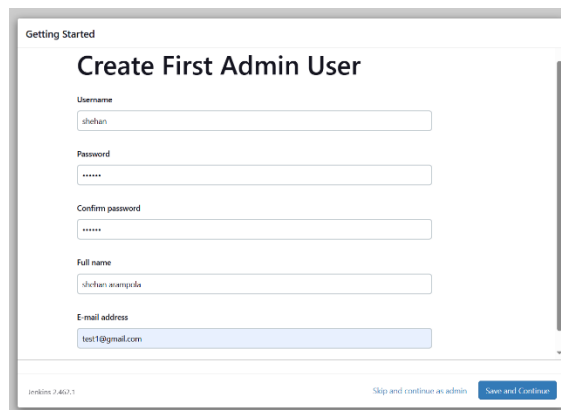
```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Copy the password given and insert it in the above textbox. In next page choose Install suggested plugins. Jenkins will install the plugins it needed.

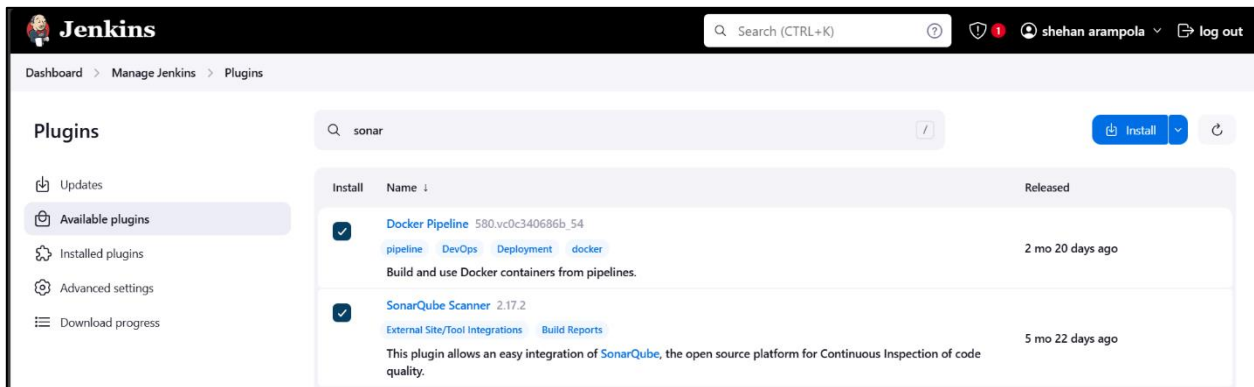


Create username and password

Cool We successfully installed Jenkins 😊



Go to Manage Jenkins > Plugins > Available plugins and search for docker pipeline and SonarQube Scanner and install them.



3. SonarQube

Go to Jenkins-server EC2 on MobaXterm and install docker using following commands. Because we use SonarQube in a docker container.

```
# 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

# 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

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

Run `sudo docker run hello-world` to confirm the docker installation.

Run `sudo chmod 666 /var/run/docker.sock` to give permission to all users to run docker commands

Now install SonarQube using following command

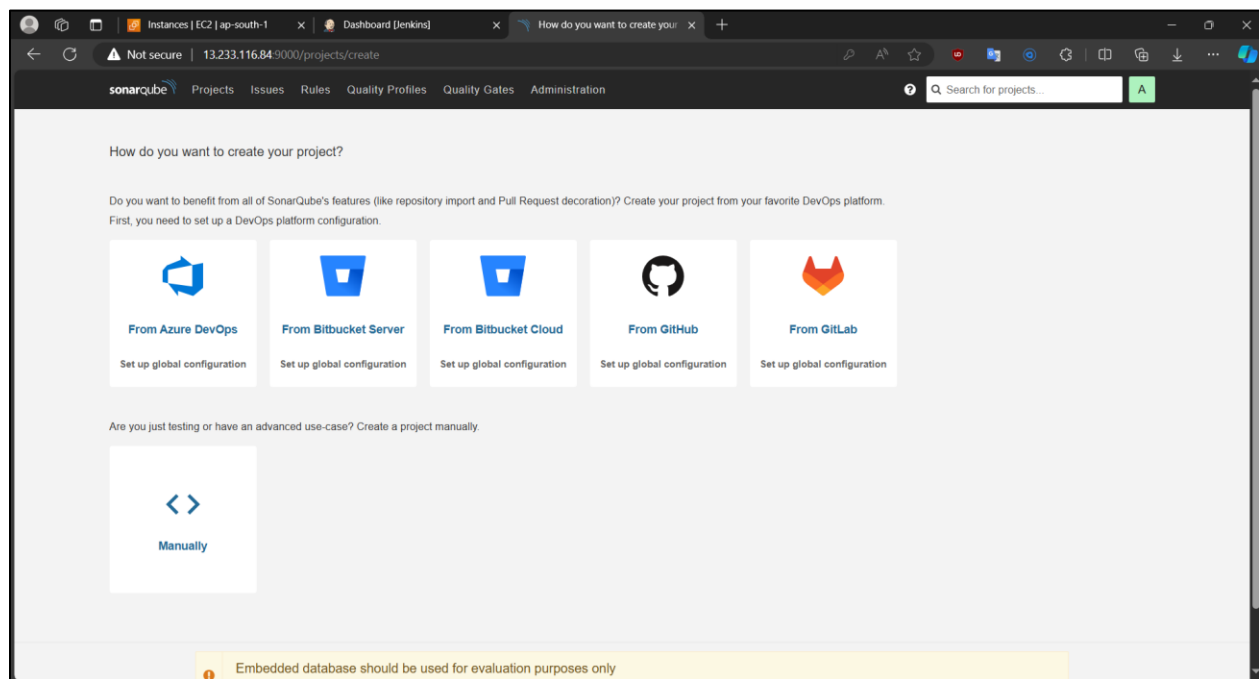
```
docker run -d --name sonar -p 9000:9000 sonarqube:lts-community
```

Cool 😊. Now SonarQube is running on port 9000.

Go to browser and insert following link.

<http://<jenkins-server ec2 instance public ip address>:9000>

Login to SonarQube. Default username: admin, Password: admin. Login and create a new password.



Great 😊.

4. Setup credentials

First, Let's configure credentials for different platforms which we need in upcoming steps.

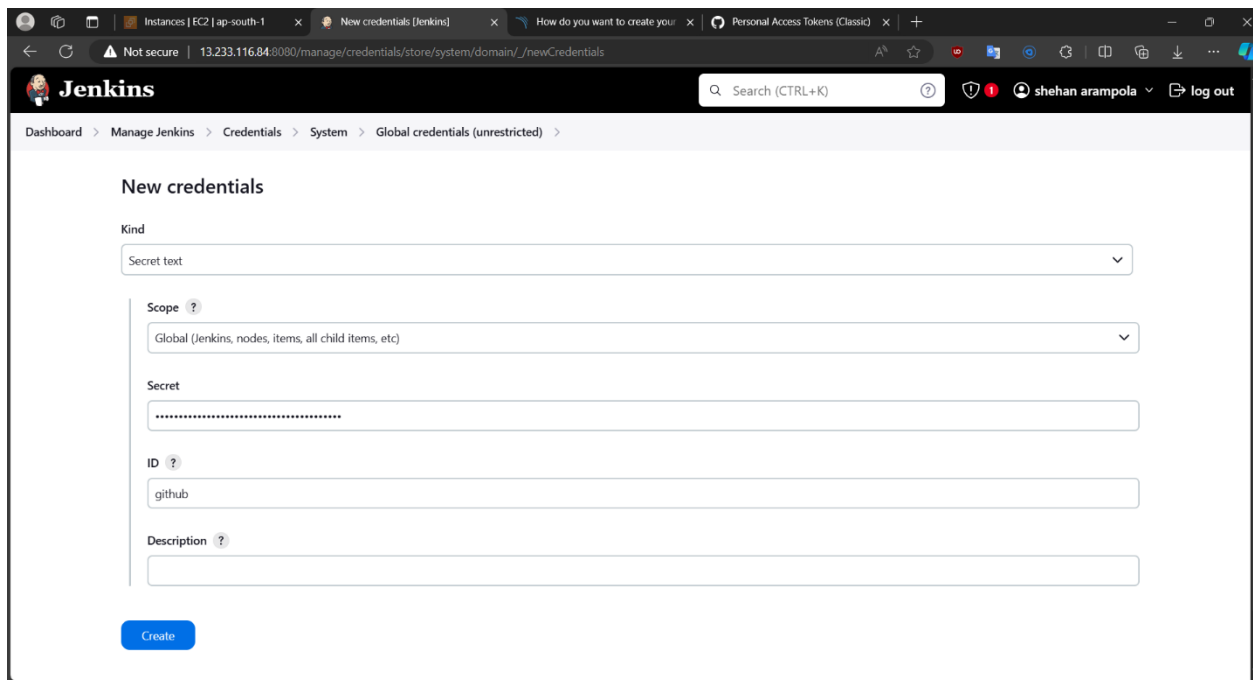
Go to Jenkins Dashboard. Manage Jenkins > Credentials. Click on global. Click on Add Credentials.

1. GitHub (If you use private repo)

Go to GitHub and generate a token in Settings > Developer Settings > Personal access tokens > Token (classic). Give required permissions and generate a token. Save it.

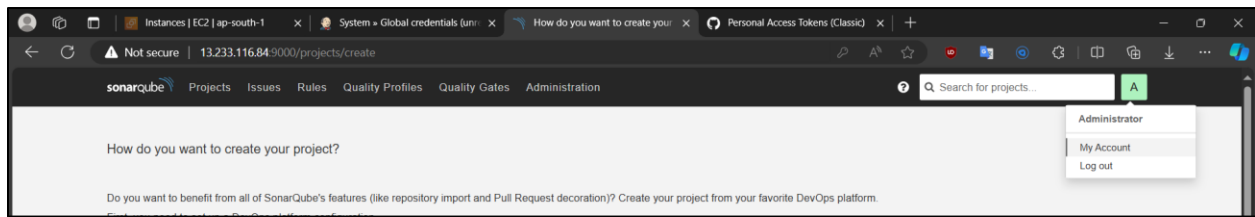
Now in Jenkins under Add Credentials give that token as follows:

Paste the token in Secret. Click create.

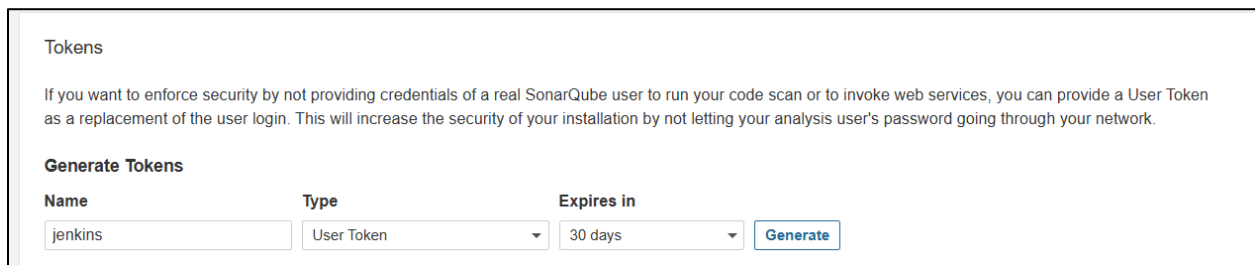
The screenshot shows the Jenkins web interface in a browser. The address bar indicates the URL is 13.233.116.84:8080/manage/credentials/store/system/domain/_/newCredentials. The page title is 'New credentials'. The breadcrumb navigation is 'Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted)'. The form has the following fields: 'Kind' is a dropdown menu with 'Secret text' selected; 'Scope' is a dropdown menu with 'Global (jenkins, nodes, items, all child items, etc)' selected; 'Secret' is a text input field containing a series of dots; 'ID' is a text input field with 'github' entered; and 'Description' is an empty text input field. A blue 'Create' button is at the bottom left of the form area. The top of the page shows the Jenkins logo, a search bar, and user information for 'shehan arampola' with a 'log out' link.

2. SonarQube

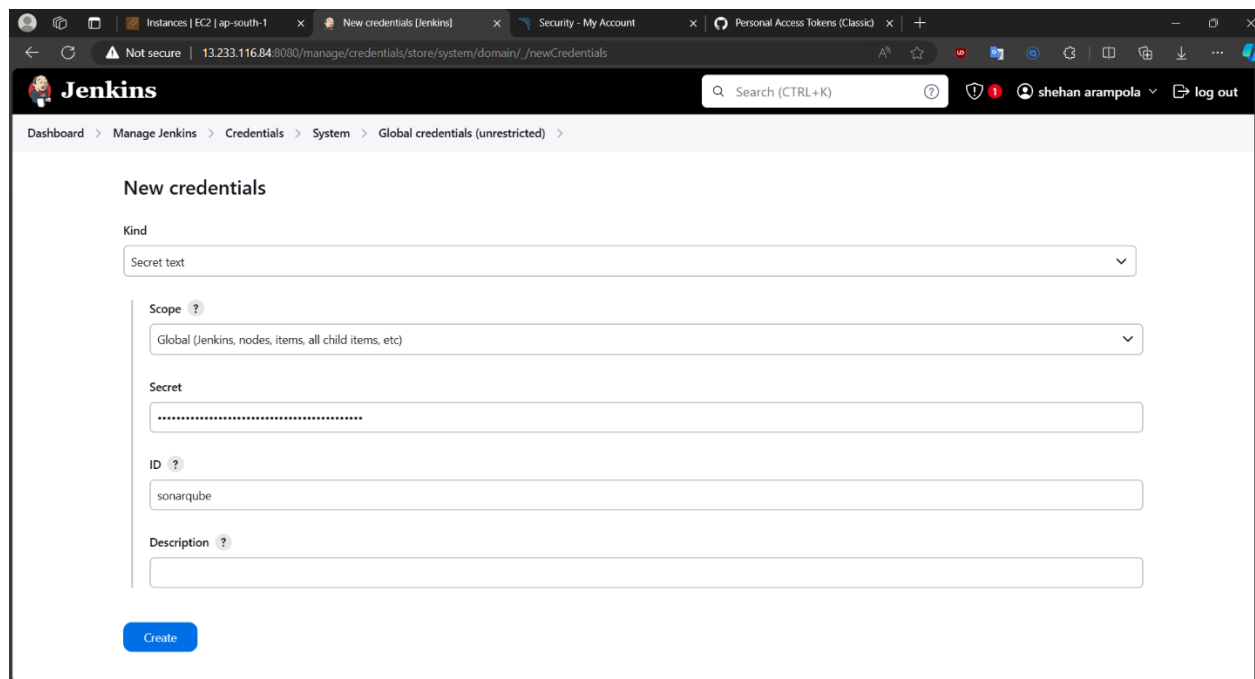
Go to SonarQube page and click on Profile > My account



Go to security and generate a token as follows

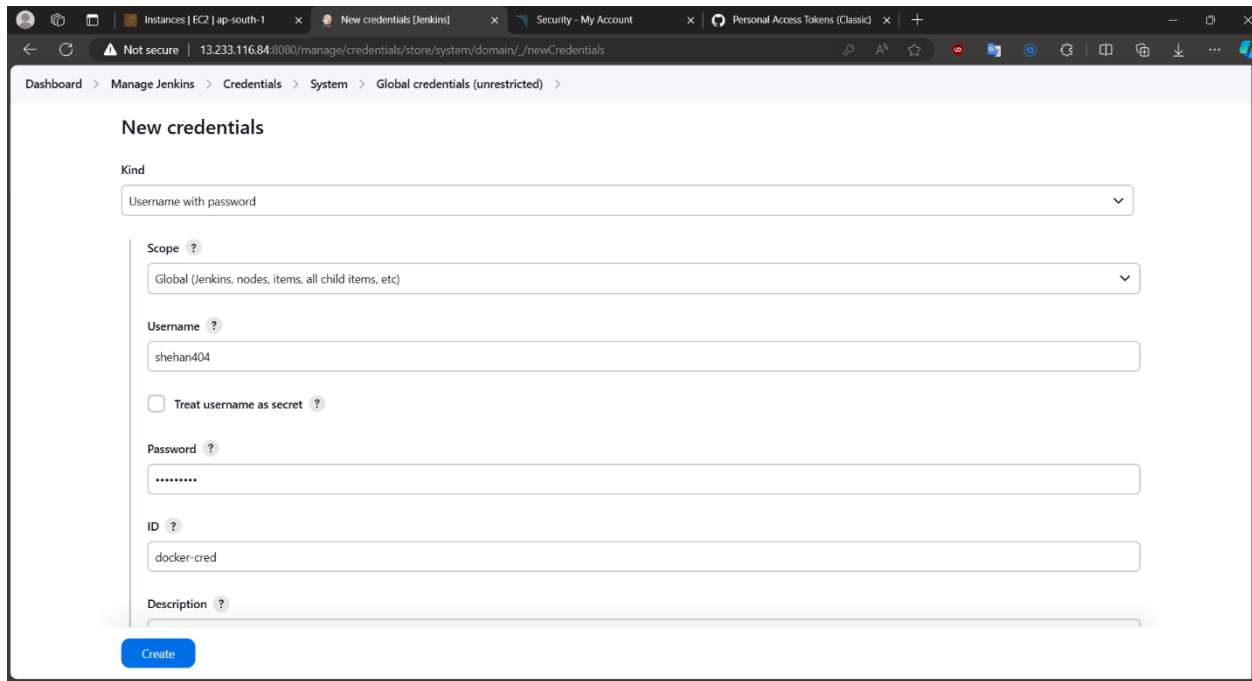


Now copy the token generated and go to Jenkins. Add credentials. Create credentials as follows.



3. Docker

Go to Jenkins add credentials and enter your Docker Hub credentials as follows.



The screenshot shows the 'New credentials' form in Jenkins. The 'Kind' is set to 'Username with password'. The 'Scope' is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' is 'shehan404'. The 'Password' is masked with asterisks. The 'ID' is 'docker-cred'. The 'Description' is empty. A 'Create' button is at the bottom.

Kind: Username with password

Scope: Global (Jenkins, nodes, items, all child items, etc)

Username: shehan404

☐ Treat username as secret

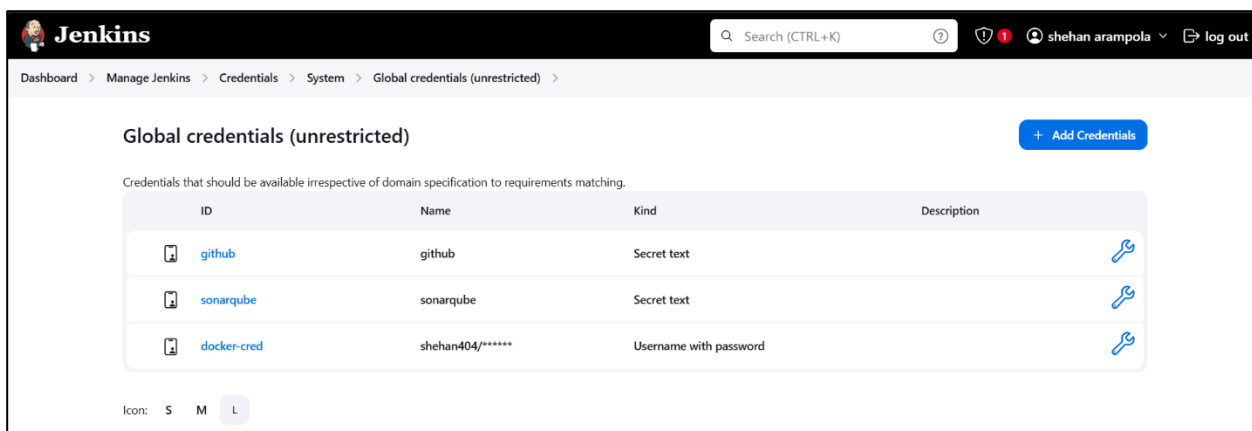
Password: *****

ID: docker-cred

Description:

Create

Great Now you have created following credentials



The screenshot shows the 'Global credentials (unrestricted)' page in Jenkins. It displays a table of credentials. The table has columns for ID, Name, Kind, and Description. There are three credentials listed: 'github', 'sonarqube', and 'docker-cred'. Each row has a blue key icon on the right. A '+ Add Credentials' button is at the top right. Below the table, there are filters for 'Icon: S M L'.

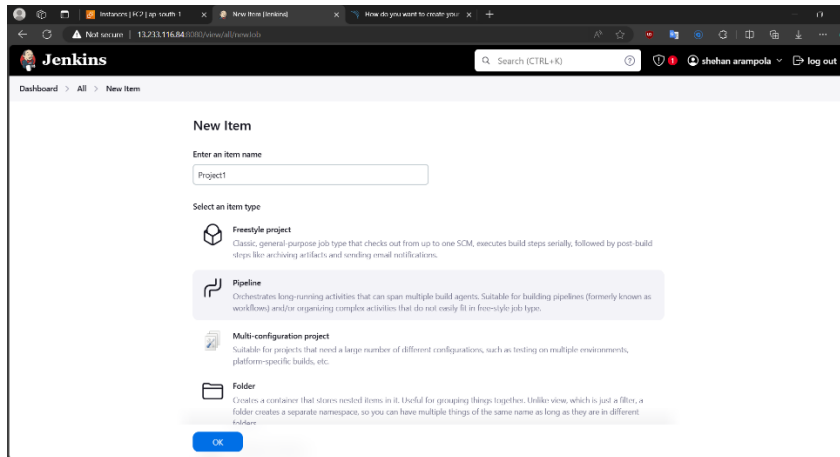
ID	Name	Kind	Description
github	github	Secret text	
sonarqube	sonarqube	Secret text	
docker-cred	shehan404/*****	Username with password	

Icon: S M L

5. CI/CD Pipeline

We use Jenkins for CI. We use ArgoCD for CD.

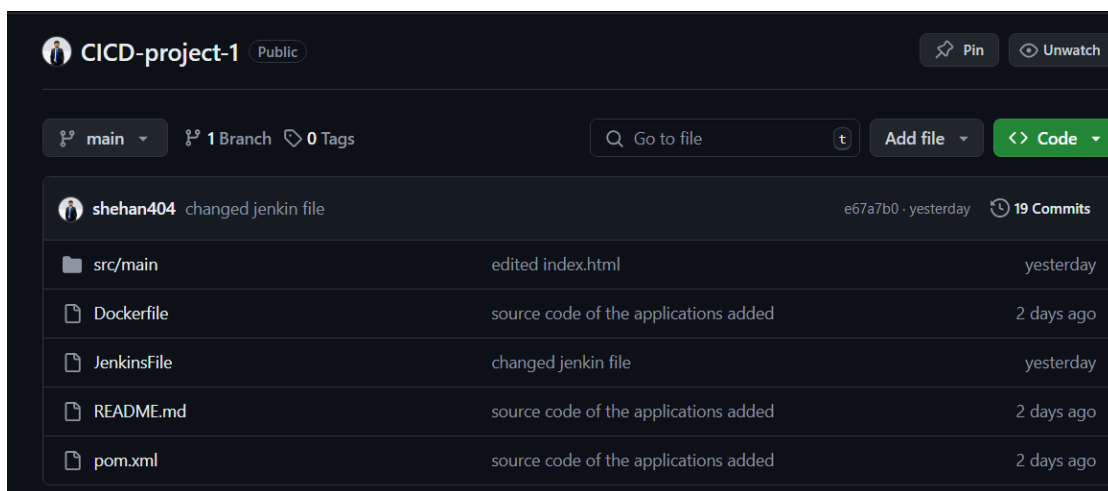
Go to Jenkins Dashboard and click on New Item. Enter a name and select Pipeline. Click Ok.



In configuration, under the pipeline choose “Pipeline script from SCM” in definition section. This allows us to get groovy pipeline script from GitHub instead of writing it directly on Jenkins.

I have added application source code and JenkinsFile in the following repository. Clone it into your own repository

<https://github.com/shehan404/CICD-project-1.git>



Now, in Jenkins after selecting “Pipeline script from SCM” setup it as follows and save.

Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/shehan404/CICD-project-1

Credentials ?

- none -

+ Add

Advanced

Branches to build ?

Branch Specifier (blank for 'any') ?

*/main

Add Branch

Repository browser ?

(Auto)

Additional Behaviours

Add

Script Path ?

JenkinsFile

☒ Lightweight checkout ?

Before build go to JenkinsFile in the github and do following changes.

1. Under 'Static Code Analysis' stage set

```
SONAR_URL = http://<jenkins-server ec2 instance public ip address>:9000
```

In my case <http://13.233.116.84:9000/>

2. Under 'Build and Push Docker Image' stage set

```
DOCKER_IMAGE = "<dockerhub user>/ultimate-cicd:${BUILD_NUMBER}"
```

In my case

```
DOCKER_IMAGE = "shehan404/ultimate-cicd:${BUILD_NUMBER}"
```

3. Under 'Update Deployment File' stage set

```
GIT_REPO_NAME = "<Manifest repo name>"
```

```
GIT_USER_NAME = "<github username>"
```

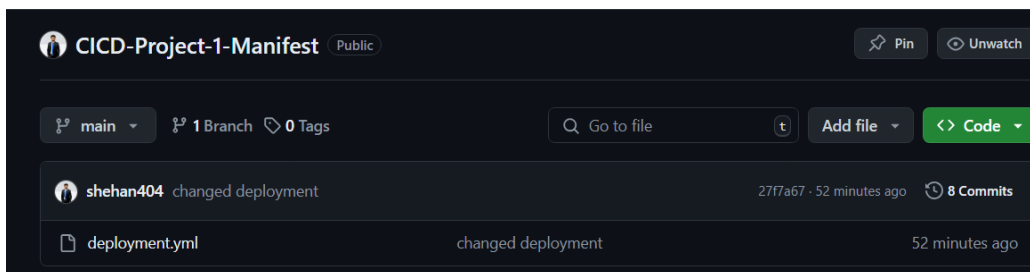
In my case

```
GIT_REPO_NAME = "CICD-Project-1-Manifest"
```

```
GIT_USER_NAME = "shehan404"
```

I have added manifest repo in following URL. Clone it into your GitHub repo

<https://github.com/shehan404/CICD-Project-1-Manifest>



Now commit changes.

Great. Now we are ready to Build 😊.

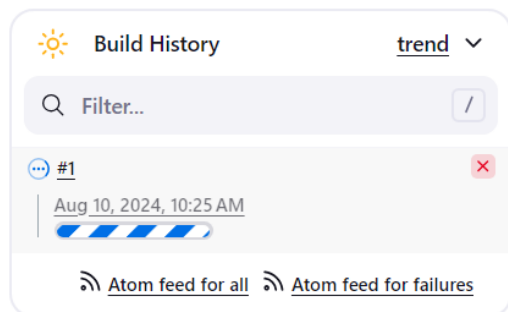
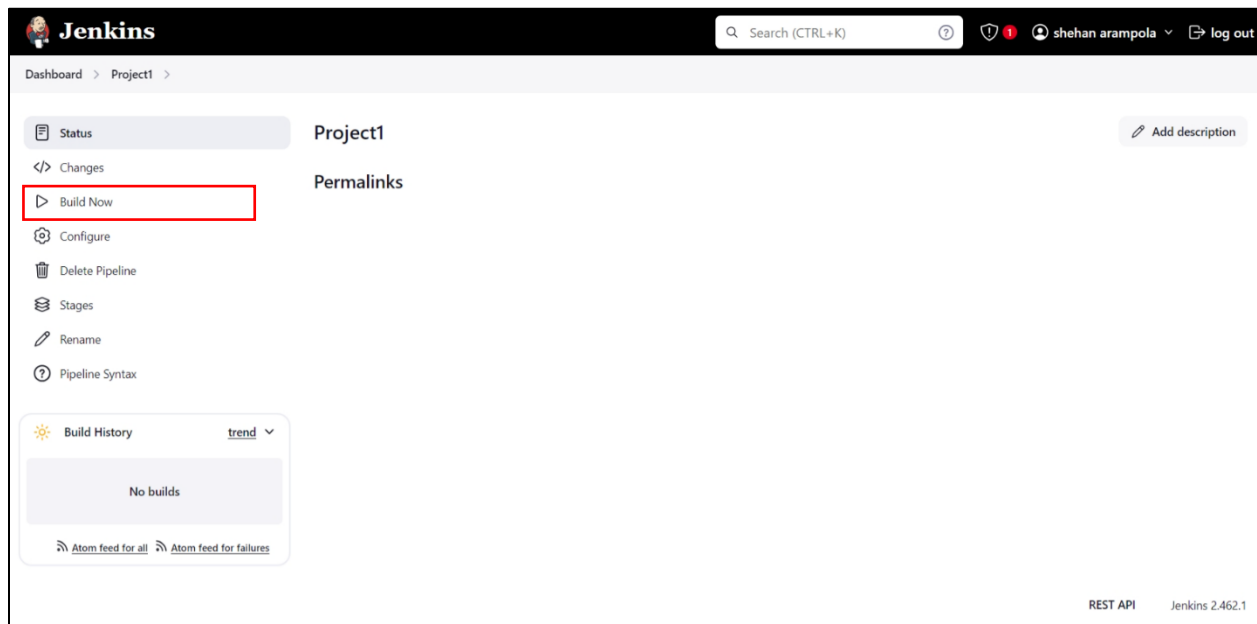
Before build it is a good practice to restart Jenkins.

Go to URL and add /restart to it and enter.

In my case <http://13.233.116.84:8080/restart>

Jenkins will restart. Then login again.

Now Go to Jenkins Dashboard > Project1 and click on Build Now. If everything done correctly no errors will occur



Click on build number and go to console output to find if there any error occurred.

Jenkins

Dashboard > Project1 > #1

Console Output

Download Copy View as plain text

Skipping 432 KB. [Full Log](#)

/106 kB

Progress (3): 258/527 kB | 14 kB | 78/106 kB

Progress (3): 258/527 kB | 14 kB | 82/106 kB

Progress (3): 258/527 kB | 14 kB | 86/106 kB

Progress (3): 258/527 kB | 14 kB | 90/106 kB

Progress (3): 258/527 kB | 14 kB | 94/106 kB

Progress (3): 258/527 kB | 14 kB | 98/106 kB

Progress (3): 258/527 kB | 14 kB | 102/106 kB

Progress (3): 258/527 kB | 14 kB | 106 kB

Progress (3): 262/527 kB | 14 kB | 106 kB

Progress (3): 266/527 kB | 14 kB | 106 kB

Progress (3): 270/527 kB | 14 kB | 106 kB

Progress (3): 274/527 kB | 14 kB | 106 kB

Progress (3): 279/527 kB | 14 kB | 106 kB

Progress (4): 279/527 kB | 14 kB | 106 kB | 4.1/108 kB

Progress (4): 279/527 kB | 14 kB | 106 kB | 8.2/108 kB

Progress (4): 279/527 kB | 14 kB | 106 kB | 12/108 kB

Progress (4): 279/527 kB | 14 kB | 106 kB | 16/108 kB

Progress (4): 279/527 kB | 14 kB | 106 kB | 20/108 kB

Progress (4): 279/527 kB | 14 kB | 106 kB | 25/108 kB

Progress (4): 279/527 kB | 14 kB | 106 kB | 29/108 kB

Progress (4): 279/527 kB | 14 kB | 106 kB | 33/108 kB

Progress (4): 279/527 kB | 14 kB | 106 kB | 37/108 kB

Progress (4): 279/527 kB | 14 kB | 106 kB | 41/108 kB

Progress (4): 279/527 kB | 14 kB | 106 kB | 45/108 kB

Progress (4): 279/527 kB | 14 kB | 106 kB | 49/108 kB

Downloaded from central: <https://repo.maven.apache.org/maven2/org/sonatype/aether/aether-spi/1.7/aether-spi-1.7.jar> (14 kB at 423 kB/s)

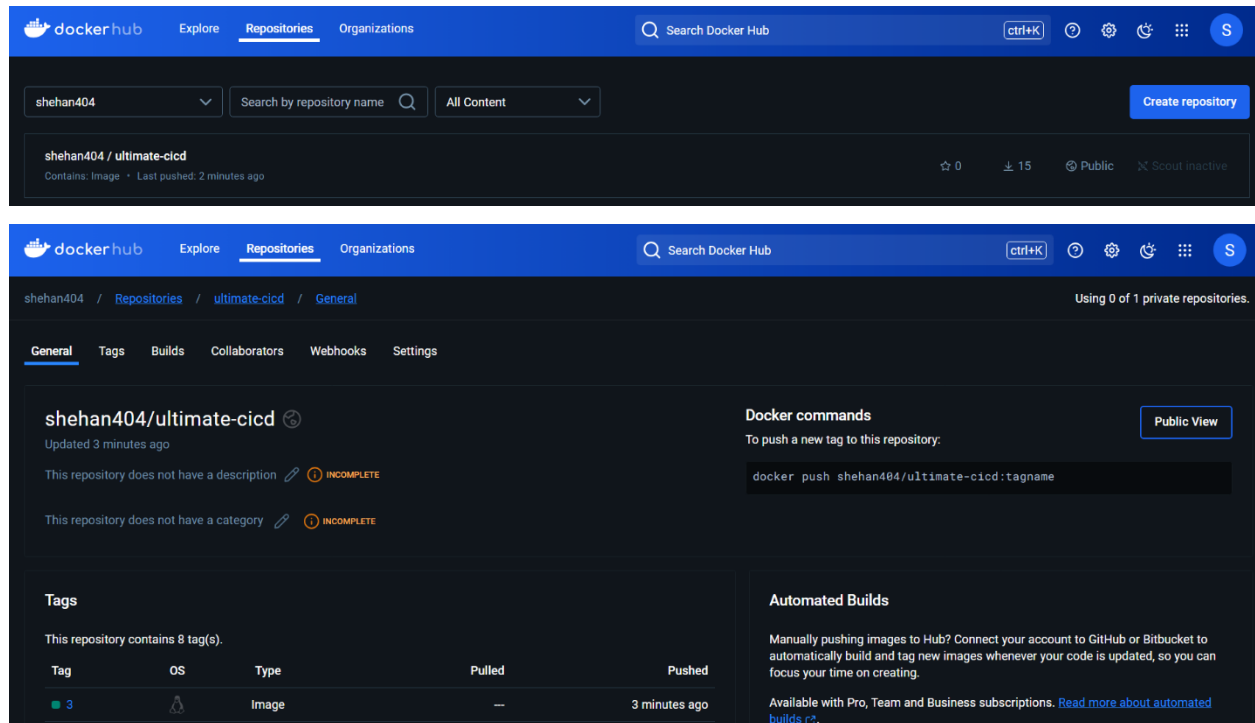
Downloading from central: <https://repo.maven.apache.org/maven2/org/sonatype/nexus/nexus-sec-dispatcher/1.3/nexus-sec-dispatcher-1.3.jar>

If an error occurs, refer the console output and try to troubleshoot the error. Then build again.

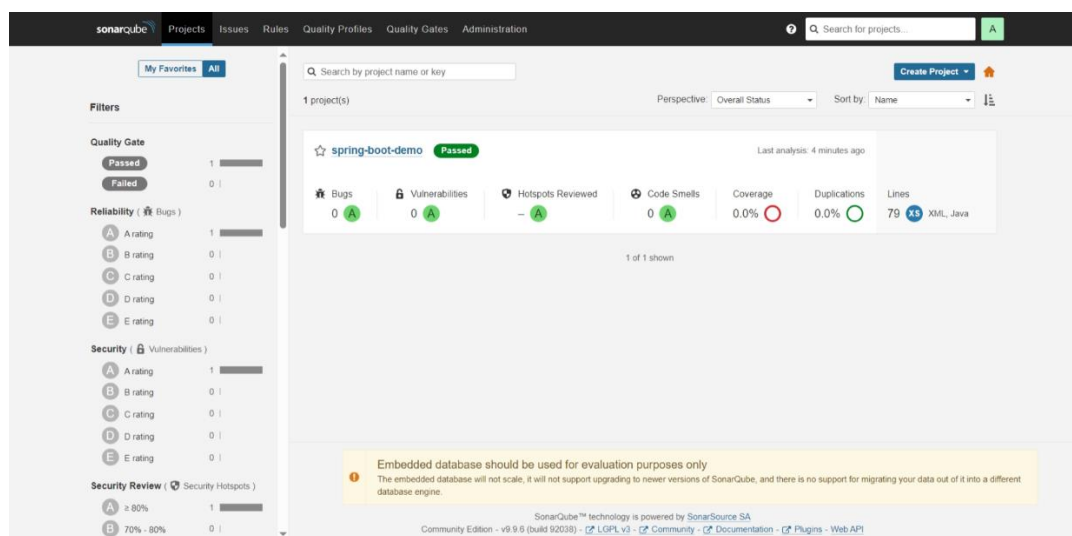
After all done well you will receive the following output.

```
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
$ docker stop --time=1 71f8b8cca3992d55a1318ec8a2f4db331b71443fefc32702e9cf58835a59eb3e
$ docker rm -f --volumes 71f8b8cca3992d55a1318ec8a2f4db331b71443fefc32702e9cf58835a59eb3e
[Pipeline] // withDockerContainer
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Now if you check your docker hub you can see a new docker image in a new repository.



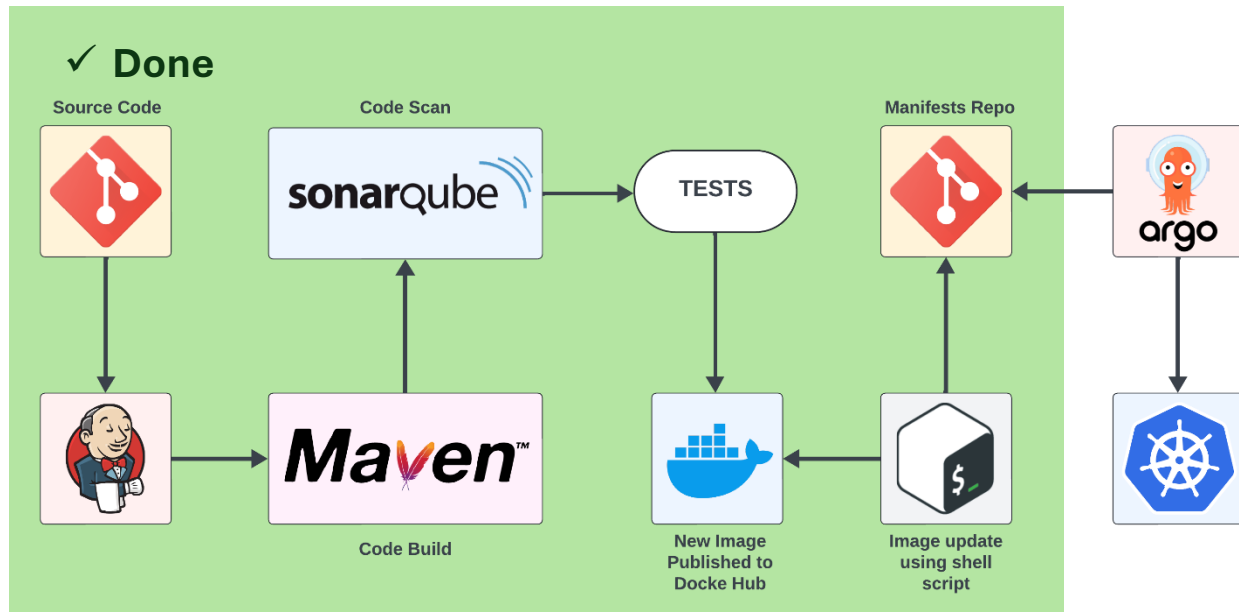
In SonarQube page under projects you can see like this



Which means no issues in the source code 😊.

Shehan Arampola | <https://www.linkedin.com/in/shehan-arampola> | <https://github.com/shehan404>

Good Job. We have implemented CI part successfully 😊



Now Let's move on to CD part with ArgoCD.

6. ArgoCD

Go to Deploy-server in MobaXterm

1. Install Kubectl

Run following commands

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
kubectl version --client
```

```
ubuntu@ip-172-31-0-143:~$ curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             %             Dload  Upload  Total   Spent    Left   Speed
100 138    100 138    0     0    312      0  0:00:00  0:00:00  0:00:00  312
100 49.0M  100 49.0M    0     0  53.7M      0  0:00:00  0:00:00  0:00:00  53.7M
ubuntu@ip-172-31-0-143:~$ sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
ubuntu@ip-172-31-0-143:~$ kubectl version --client
Client Version: v1.30.3
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
ubuntu@ip-172-31-0-143:~$
```

2. Install minikube

First install docker as we installed before. Because We use docker as driver for minikube.

Now run

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64
```

then run `minikube start --driver=docker`

```
ubuntu@ip-172-31-0-143:~$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 91.1M  100 91.1M    0     0  11.3M    0  0:00:07  0:00:07 --:--:-- 15.6M
ubuntu@ip-172-31-0-143:~$ minikube start --driver=docker
* minikube v1.33.1 on Ubuntu 24.04 (xen/amd64)
* Using the docker driver based on user configuration
* Using Docker driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.44 ...
* Downloading Kubernetes v1.30.0 preload ...
   > preloaded-images-k8s-v18-v1...: 342.90 MiB / 342.90 MiB 100.00% 15.49 M
   > gcr.io/k8s-minikube/kicbase...: 481.58 MiB / 481.58 MiB 100.00% 15.09 M
* Creating docker container (CPUs=2, Memory=2200MB) ...
* Preparing Kubernetes v1.30.0 on Docker 26.1.1 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring bridge CNI (Container Networking Interface) ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
ubuntu@ip-172-31-0-143:~$
```

Run `minikube status`

```
ubuntu@ip-172-31-0-143:~$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

ubuntu@ip-172-31-0-143:~$
```

Cool minikube is running!

3. Setup ArgoCD

Run following command

```
kubectl create namespace argocd
```

```
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

Now run `kubectl get pods -n argocd`

```
ubuntu@ip-172-31-0-143:~$ kubectl get pods -n argocd
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0           49s
argocd-applicationset-controller-86c8767989-gtw2t  1/1     Running   0           49s
argocd-dex-server-55c6d584b5-q6s76  1/1     Running   0           49s
argocd-notifications-controller-74965f8dc9-n7hvk  1/1     Running   0           49s
argocd-redis-5dcdbd87d95-tdfm9      1/1     Running   0           49s
argocd-repo-server-5b76df5967-5xgmn  1/1     Running   0           49s
argocd-server-77f69c9d6-lj49d       1/1     Running   0           49s
ubuntu@ip-172-31-0-143:~$
```

Cool pods are running.

Now run `kubectl get svc -n argocd`

```
ubuntu@ip-172-31-0-143:~$ kubectl get svc -n argocd
NAME                                TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)                                AGE
argocd-applicationset-controller    ClusterIP   10.96.58.58    <none>       7000/TCP,8080/TCP                     2m30s
argocd-dex-server                   ClusterIP   10.100.109.8   <none>       5556/TCP,5557/TCP,5558/TCP            2m30s
argocd-metrics                      ClusterIP   10.101.244.255 <none>       8082/TCP                              2m30s
argocd-notifications-controller-metrics ClusterIP   10.104.4.131   <none>       9001/TCP                              2m30s
argocd-redis                        ClusterIP   10.104.14.117  <none>       6379/TCP                              2m30s
argocd-repo-server                  ClusterIP   10.108.7.14    <none>       8081/TCP,8084/TCP                     2m30s
argocd-server                       ClusterIP   10.99.161.17   <none>       80/TCP,443/TCP                        2m30s
argocd-server-metrics               ClusterIP   10.107.237.252 <none>       8083/TCP                              2m30s
ubuntu@ip-172-31-0-143:~$
```

You can see that argocd-server is running at port 443

Let's forward it to port 9090

Run `kubectl port-forward svc/argocd-server -n argocd 9090:443 --address 0.0.0.0 &`

```
ubuntu@ip-172-31-0-143:~$ kubectl port-forward svc/argocd-server -n argocd 9090:443 --address 0.0.0.0 &
[1] 18563
ubuntu@ip-172-31-0-143:~$ Forwarding from 0.0.0.0:9090 -> 8080
```

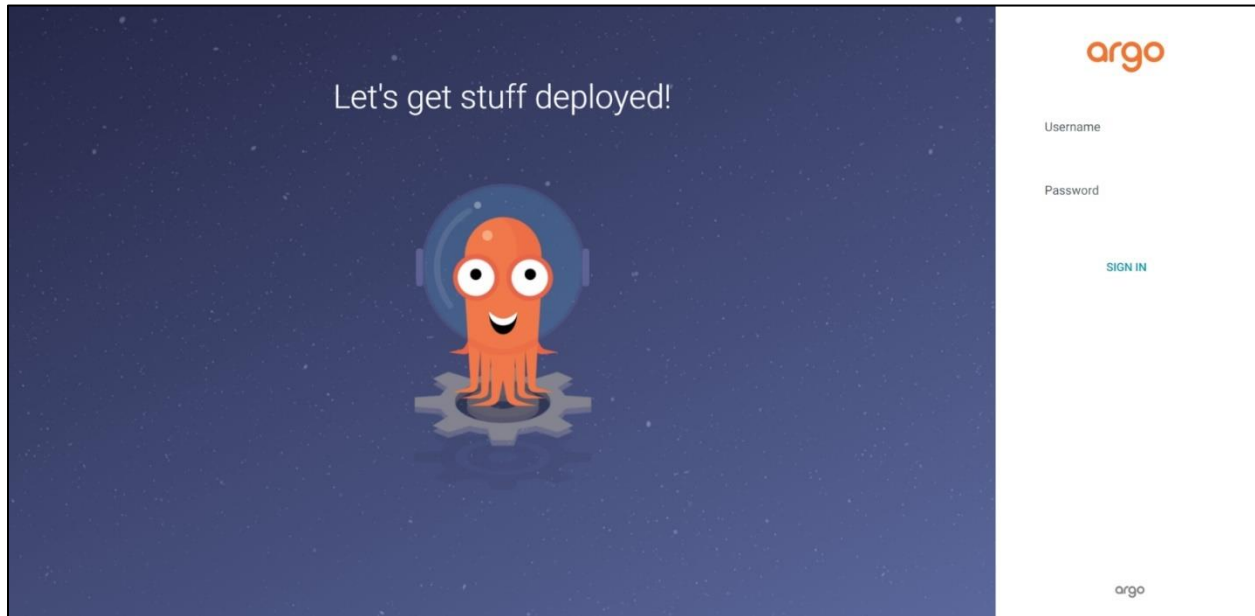
Now open the browser and enter following URL

<http://<Deploy-server ec2 instance public ip address>:9090>

In my case <https://3.111.52.39:9090/>

It will show that the page is not secure. Click advanced > [Continue to 3.111.52.39 \(unsafe\)](#)

Cool you entered to the ArgoCD Login page



Default username is admin.

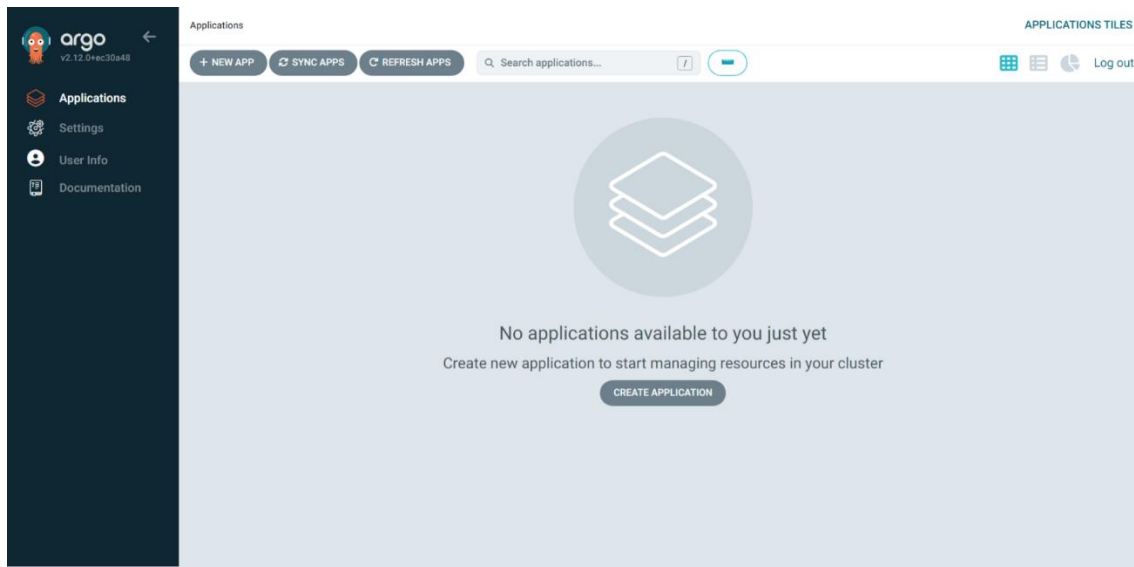
To obtain default password run following command

```
kubectl -n argocd get secret argocd-initial-admin-secret -o  
jsonpath="{.data.password}" | base64 -d
```

```
ubuntu@ip-172-31-0-143:~$ kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d  
oX-UE85gkp4n6YYQubuntu@ip-172-31-0-143:~$
```

and here is the password: oX-UE85gkp4n6YYQ

Login to ArgoCD using them



Go to NEW APP and insert following

This is a 'CREATE' dialog box for a new ArgoCD application. At the top are 'CREATE' and 'CANCEL' buttons. The dialog is divided into sections: 'GENERAL' with an 'EDIT AS YAML' button, 'SYNC POLICY', and 'SYNC OPTIONS'. In the 'GENERAL' section, 'Application Name' is 'testapp' and 'Project Name' is 'default'. The 'SYNC POLICY' section shows 'Automatic' selected. Under 'SYNC POLICY', there are three unchecked checkboxes: 'PRUNE RESOURCES', 'SELF HEAL', and 'SET DELETION FINALIZER'. The 'SYNC OPTIONS' section contains six unchecked checkboxes: 'SKIP SCHEMA VALIDATION', 'PRUNE LAST', 'RESPECT IGNORE DIFFERENCES', 'AUTO-CREATE NAMESPACE', 'APPLY OUT OF SYNC ONLY', and 'SERVER-SIDE APPLY'. At the bottom, 'PRUNE PROPAGATION POLICY' is set to 'foreground'.

SOURCE

Repository URL

https://github.com/shehan404/CICD-Project-1-Manifest

GIT ▼

Revision

HEAD

Branches ▼

Path

./

DESTINATION

Cluster URL

https://kubernetes.default.svc


URL ▼

Namespace

argocd

Click create

You will see following page



argo

v2.12.0+rec30a48

Applications

Settings

User info

Documentation

☐ ★ Favorites Only

SYNC STATUS

☐ Unknown 0
 ☒ Synced 1
 ☐ OutOfSync 0

HEALTH STATUS

☐ Unknown 0
 ☒ Progressing 1
 ☐ Suspended 0
 ☐ Healthy 0
 ☐ Degraded 0
 ☐ Missing 0

Applications

+ NEW APP

↻ SYNC APPS

↻ REFRESH APPS

🔍 Search applications...

🏠

☰

🌐

Log out

Sort: name ▼ Items per page: 10 ▼

testapp

Project: default

Labels:

Status: Progressing Synced

Reposito... https://github.com/shehan404/CICD-Pro...

Target R... HEAD

Path: ./

Destinati... in-cluster

Namesp... argocd

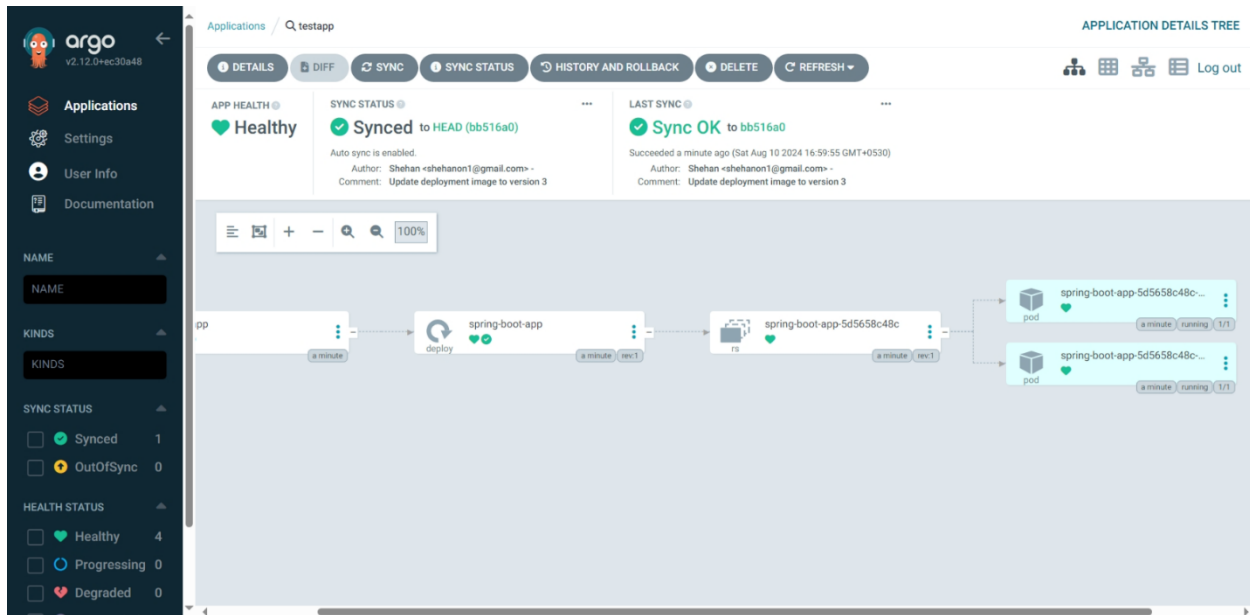
Created ... 08/10/2024 16:59:54 (a few seconds ag...

Last Sync: 08/10/2024 16:59:55 (a few seconds ag...

↻ SYNC

↻ REFRESH

🗑 DELETE



You can see that everything is **Synced** and **Healthy**.

CD part is completed.

Congratulation!!! You have successfully implemented End to End CI/CD Pipeline



Now let's try to access the application using browser

Run `kubectl get pods -n argocd`

```
ubuntu@ip-172-31-0-143:~$ kubectl get pods -n argocd
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0           28m
argocd-applicationset-controller-86c8767989-gtw2t  1/1     Running   0           28m
argocd-dex-server-55c6d584b5-q6s76  1/1     Running   0           28m
argocd-notifications-controller-74965f8dc9-n7hwk  1/1     Running   0           28m
argocd-redis-5dcdbd87d95-tdfm9      1/1     Running   0           28m
argocd-repo-server-5b76df5967-5xgmn  1/1     Running   0           28m
argocd-server-77f69c9d6-lj49d       1/1     Running   0           28m
spring-boot-app-5d5658c48c-d62qr    1/1     Running   0           7m45s
spring-boot-app-5d5658c48c-vw74g    1/1     Running   0           7m45s
```

Copy name of one pod

And run following command

`kubectl expose pod spring-boot-app-5d5658c48c-d62qr --type=NodePort --port=8080 -n argocd`

```
ubuntu@ip-172-31-0-143:~$ kubectl expose pod spring-boot-app-5d5658c48c-d62qr --type=NodePort --port=8080 -n argocd
service/spring-boot-app-5d5658c48c-d62qr exposed
```

Run `kubectl get svc -n argocd`

```
ubuntu@ip-172-31-0-143:~$ kubectl get svc -n argocd
NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)                                AGE
argocd-applicationset-controller    ClusterIP           10.96.58.58     <none>           7000/TCP,8080/TCP                     31m
argocd-dex-server                   ClusterIP           10.100.109.8    <none>           5556/TCP,5557/TCP,5558/TCP            31m
argocd-metrics                      ClusterIP           10.101.244.255  <none>           8082/TCP                               31m
argocd-notifications-controller-metrics ClusterIP           10.104.4.131    <none>           9001/TCP                               31m
argocd-redis                       ClusterIP           10.104.14.117   <none>           6379/TCP                               31m
argocd-repo-server                  ClusterIP           10.108.7.14     <none>           8081/TCP,8084/TCP                     31m
argocd-server                       ClusterIP           10.99.161.17    <none>           80/TCP,443/TCP                        31m
argocd-server-metrics               ClusterIP           10.107.237.252  <none>           8083/TCP                               31m
spring-boot-app-5d5658c48c-d62qr    NodePort            10.98.225.195   <none>           8080:30177/TCP                        3s
```

Now spring-boot-app-5d5658c48c-d62qr run on port 8080

Run following command for port forwarding

`kubectl port-forward svc/spring-boot-app-5d5658c48c-d62qr -n argocd 8080:8080 --address 0.0.0.0 &`

```
ubuntu@ip-172-31-0-143:~$ kubectl port-forward svc/spring-boot-app-5d5658c48c-d62qr -n argocd 8080:8080 --address 0.0.0.0 &
[2] 33661
ubuntu@ip-172-31-0-143:~$ Forwarding from 0.0.0.0:8080 -> 8080
ubuntu@ip-172-31-0-143:~$ curl 3.111.52.39:8080
```

Now run `curl <deploy-server public ip address>:8080`

In my case `curl 3.111.52.39:8080`

If it gives following result, we can access it using browser

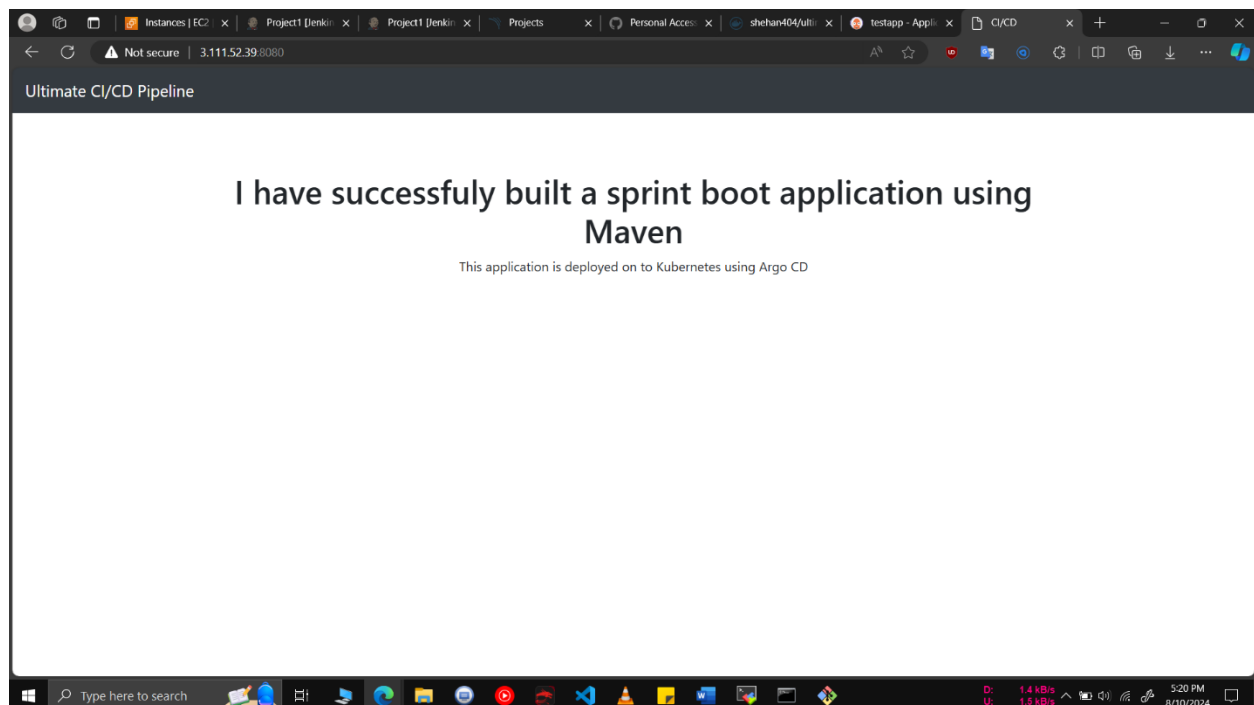
```
ubuntu@ip-172-31-0-143:~$ curl 3.111.52.39:8080
Handling connection for 8080
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"/>
  <link href="/css/main.css?" rel="stylesheet">
  <title>CI/CD</title>
</head>
<body>

<nav class="navbar navbar-expand-md navbar-dark bg-dark fixed-top">
  <a class="navbar-brand" href="#">Ultimate CI/CD Pipeline</div></a>
</nav>

<main role="main" class="container">
  <div class="starter-template">
    <h1>I have successfully built a sprint boot application using Maven</h1>
    <p>This application is deployed on to Kubernetes using Argo CD</p>
  </div>
</main>

<script src="/js/main.js"></script>
</body>
</html>
```

Go to browser and paste URL used in curl command



Here is our application 😊😊😊

Shehan Arampola | <https://www.linkedin.com/in/shehan-arampola> | <https://github.com/shehan404>

Note :

After completing the demonstration, shut down and terminate the EC2 instance; otherwise, you will incur additional costs.

KEEP LEARNING 😊😊😊