BootCamp Project -3
Jenkins-CICD Pipeline

Services Used:

Jenkins

Jenkins is the tool that runs everything for us. It watches our code repo, and when we push new code, Jenkins takes over — it builds the project, runs tests, and pushes our app Docker image to Docker Hub.

Docker

Docker lets us put our app and everything it needs into a container. This way, no matter where we run it, the app works the same without any surprises.

Docker Hub

Docker Hub is like a storage space for our container images. Once Jenkins builds the image, it uploads it here so our Kubernetes cluster can download and run it.

SonarQube

SonarQube helps us check our code quality automatically. It finds bugs, vulnerabilities, or bad coding practices so we can fix them early before they cause issues.

GitHub

GitHub stores all our source code. It’s the place where Jenkins checks for changes and triggers the whole pipeline when we push new commits.

Kubernetes (Minikube)

Kubernetes is the platform where we run our containerized app. Minikube lets us run Kubernetes locally on our machine for testing before deploying to real clusters.

ArgoCD

ArgoCD automates the deployment of our app to Kubernetes. It watches our Git repo for deployment file changes and syncs those changes to the cluster automatically.

Helm

Helm helps us manage Kubernetes apps easily by using charts, which are like app templates. We can install, upgrade, or rollback with simple commands.

Email Notifications

We set up email notifications so the team gets informed when builds succeed or fail, helping us stay updated on the project status.

AWS EC2

We use EC2 instances as servers to run Jenkins agents and host parts of our infrastructure in the cloud, so we don't have to manage physical hardware.

Project Flow

When you push new code to **GitHub**, Jenkins detects the change.

Jenkins triggers the **pipeline**: it builds, tests, and analyzes the code with **SonarQube**.

Jenkins then builds a new Docker image and pushes it to **Docker Hub**.

Using **ArgoCD**, the deployment manifests in GitHub get synced to Kubernetes, updating the app pods with the new image.

Kubernetes spins up the updated pods, ready to serve new user requests.

→Clone the git repo to local machine

```
jyoshnabonagiri@Macbook ~ % git clone https://github.com/iam-veeramalla/Jenk
ins-Zero-To-Hero.git
Cloning into 'Jenkins-Zero-To-Hero'...
remote: Enumerating objects: 311, done.
remote: Total 311 (delta 0), reused 0 (delta 0), pack-reused 311 (from 1)
Receiving objects: 100% (311/311), 156.56 KiB | 649.00 KiB/s, done.
Resolving deltas: 100% (107/107), done.
jyoshnabonagiri@Macbook ~ % cd Jenkins-Zero-To-Hero
jyoshnabonagiri@Macbook Jenkins-Zero-To-Hero % ls
CODE_OF_CONDUCT.md                      my-first-pipeline
Interview_Questions.md                  python-jenkins-argocd-k8s
java-maven-sonar-argocd-helm-k8s        README.md
LICENSE                                 shared-libraries
multi-stage-multi-agent                 vars
jyoshnabonagiri@Macbook Jenkins-Zero-To-Hero % git branch
* main
jyoshnabonagiri@Macbook Jenkins-Zero-To-Hero % cd java-maven-sonar-argocd-he
lm-k8s
jyoshnabonagiri@Macbook java-maven-sonar-argocd-helm-k8s % ls
Argo CD                         spring-boot-app
README.md                       spring-boot-app-manifests
jyoshnabonagiri@Macbook java-maven-sonar-argocd-helm-k8s % cd spring-boot-ap
p
jyoshnabonagiri@Macbook spring-boot-app % mvn clean package
```

→Go to the spring app and install maven and give command
Mavenclean package and now installing all the dependencies and run the test and build jar in the /target file for the spring-boot-app

```
==> Installing maven dependency: little-cms2
==> Downloading https://ghcr.io/v2/homebrew/core/little-cms2/manifests/2.17
Already downloaded: /Users/jyoshnabonagiri/Library/Caches/Homebrew/downloads
/0e5e5ac9e1df07ae001662a85a70b344b31345897c284172f14874c9d329cb85--little-cm
s2-2.17.bottle_manifest.json
==> Pouring little-cms2--2.17.arm64_sequoia.bottle.tar.gz
🍺  /opt/homebrew/Cellar/little-cms2/2.17: 23 files, 1.4MB
==> Installing maven dependency: openjdk
==> Downloading https://ghcr.io/v2/homebrew/core/openjdk/manifests/24.0.1
Already downloaded: /Users/jyoshnabonagiri/Library/Caches/Homebrew/downloads
/e3046ed51e076d2a1ab3b47d40fc5b0383a4cbe4ebc6638b44f4a1b4bd8316e1--openjdk-2
4.0.1.bottle_manifest.json
==> Pouring openjdk--24.0.1.arm64_sequoia.bottle.tar.gz
🍺  /opt/homebrew/Cellar/openjdk/24.0.1: 556 files, 368.9MB
==> Installing maven
==> Pouring maven--3.9.11.arm64_sequoia.bottle.tar.gz
🍺  /opt/homebrew/Cellar/maven/3.9.11: 100 files, 10.3MB
==> Running `brew cleanup maven`...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
jyoshnabonagiri@Macbook spring-boot-app %
```

→build and run the docker image
Build command : docker build -t ultimate-cicd-pipeline:v1 .
Run command : docker run -d -p 8010:8080 -t ultimate-cicd-pipeline:v1
Here the port order
-p <host-port>:<container-port>
we are mapping container port 8080 to host port 8010.
So, we can access our app using : localhost:8010

```
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  56.031 s
[INFO] Finished at: 2025-07-19T11:29:34-07:00
[INFO] ------------------------------------------------------------------------
jyoshnabonagiri@Macbook spring-boot-app % docker build -t ultimate-cicd-pipeline:v1 .
[+] Building 16.1s (8/8) FINISHED
     docker:desktop-linux
 => [internal] load build definition from Dockerfile
                    0.0s
 => => transferring dockerfile: 347B
                    0.0s
 => [internal] load metadata for docker.io/library/eclipse-temurin:11-jre
                    0.7s
 => [internal] load .dockerignore
                    0.0s
 => => transferring context: 2B
                    0.0s
 => [1/3] FROM docker.io/library/eclipse-temurin:11-jre@sha256:176adb5b3504ef4f6c31b4019e690129381fa2d
6f43f63b01138ddb4b  14.9s
 => => resolve docker.io/library/eclipse-temurin:11-jre@sha256:176adb5b3504ef4f6c31b4019e690129381fa2d
6f43f63b01138ddb4bc  0.0s
 => => sha256:7e43f102e48aae3113862830b699486abece9e61bda977b712f77d824d042bb7 2.28kB / 2.28kB
                    0.7s
 => => sha256:c9742e79fa175b920e620dfaba5d9457398b1131620c7afaf5bb98973cc6d23b 157B / 157B
                    1.3s
 => => sha256:a08d455f6861b8efda923d9f884c65c2f36d00aaffe36ece39e377eceeeec966 45.59MB / 45.59MB
                    6.0s
 => => sha256:cb929026e9ed5221e25d172e6dd03b0c34e46fcc9573a82858a653e6b5bce94d 16.99MB / 16.99MB
                    4.9s
 => => sha256:e3bd89a9dac501ff564b39359113adad7c3d2813d5e04eab53ee10e20a6793a7 28.86MB / 28.86MB
                   13.2s
 => => extracting sha256:e3bd89a9dac501ff564b39359113adad7c3d2813d5e04eab53ee10e20a6793a7
                    0.3s
 => => extracting sha256:cb929026e9ed5221e25d172e6dd03b0c34e46fcc9573a82858a653e6b5bce94d
                    0.2s
 => => extracting sha256:a08d455f6861b8efda923d9f884c65c2f36d00aaffe36ece39e377eceeeec966
                    0.4s
 => => extracting sha256:c9742e79fa175b920e620dfaba5d9457398b1131620c7afaf5bb98973cc6d23b
                    0.0s
 => => extracting sha256:7e43f102e48aae3113862830b699486abece9e61bda977b712f77d824d042bb7
                    0.0s
```

Running it…

```
                    0.1s
jyoshnabonagiri@Macbook spring-boot-app % docker run -d -p 8010:8080 -t ultimate-cicd-pipeline:v1
556aa73b05d0122b0fd2e02fd4b714425367705de4d03f935c85d20e10e2493d
jyoshnabonagiri@Macbook spring-boot-app % vim Dockerfile
```

Accessing it through browser using local host

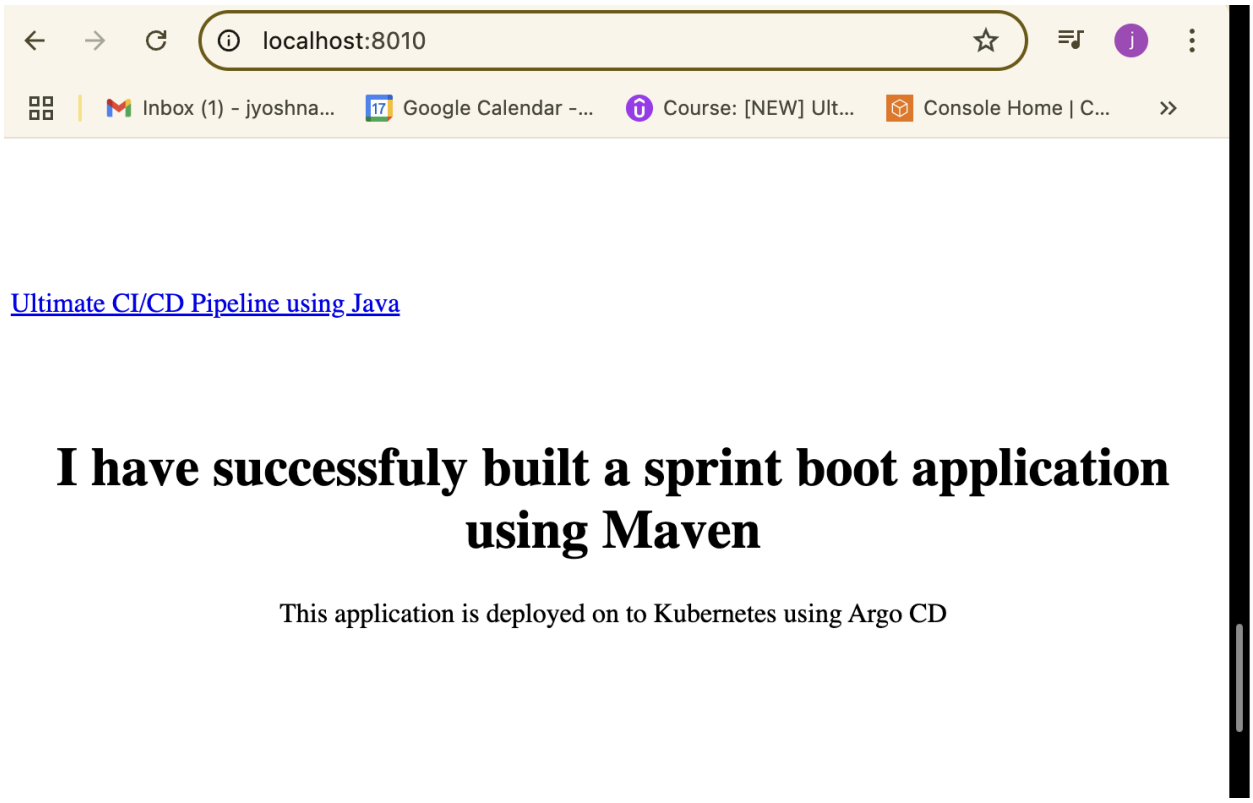
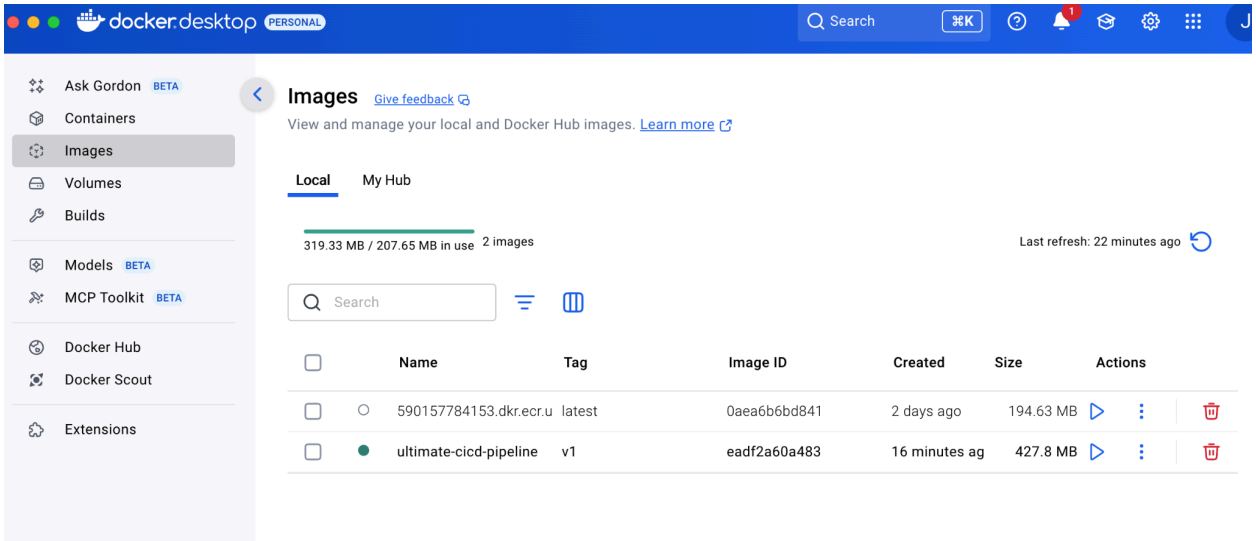
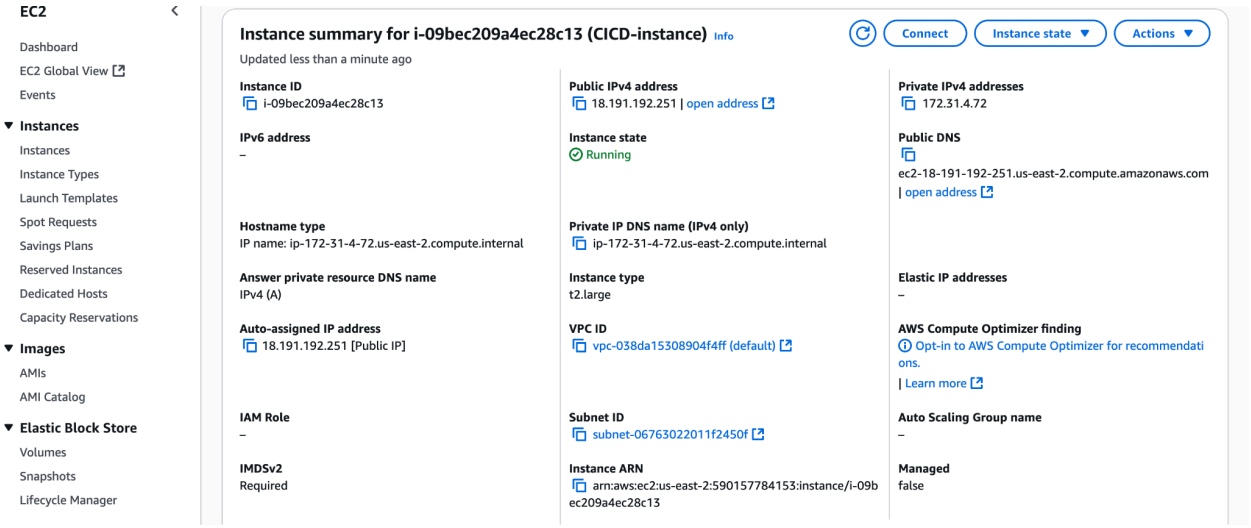


Image from the docker hub



→ Next step Launch an Ubuntu instance > t2.large >Ubuntu OS >sg-allowing ssh



→ Connect using SSH and install java using
command > sudo apt update
sudo apt install openjdk-17-jre

```
Adding debian:TWCA_Global_Root_CA.pem
Adding debian:TWCA_Root_Certification_Authority.pem
Adding debian:TeliaSonera_Root_CA_v1.pem
Adding debian:Telia_Root_CA_v2.pem
Adding debian:TrustAsia_Global_Root_CA_G3.pem
Adding debian:TrustAsia_Global_Root_CA_G4.pem
Adding debian:Trustwave_Global_Certification_Authority.pem
Adding debian:Trustwave_Global_ECC_P256_Certification_Authority.pem
Adding debian:Trustwave_Global_ECC_P384_Certification_Authority.pem
Adding debian:TunTrust_Root_CA.pem
Adding debian:UCA_Extended_Validation_Root.pem
Adding debian:UCA_Global_G2_Root.pem
Adding debian:USERTrust_ECC_Certification_Authority.pem
Adding debian:USERTrust_RSA_Certification_Authority.pem
Adding debian:XRamp_Global_CA_Root.pem
Adding debian:certSIGN_ROOT_CA.pem
Adding debian:certSIGN_Root_CA_G2.pem
Adding debian:e-Szigno_Root_CA_2017.pem
Adding debian:ePKI_Root_Certification_Authority.pem
Adding debian:emSign_ECC_Root_CA_-_C3.pem
Adding debian:emSign_ECC_Root_CA_-_G3.pem
Adding debian:emSign_Root_CA_-_C1.pem
Adding debian:emSign_Root_CA_-_G1.pem
Adding debian:vTrus_ECC_Root_CA.pem
Adding debian:vTrus_Root_CA.pem
done.
Setting up openjdk-17-jre:amd64 (17.0.15+6~us1-0ubuntu1~24.04) ...
Processing triggers for libc-bin (2.39-0ubuntu8.4) ...
Processing triggers for libgdk-pixbuf-2.0-0:amd64 (2.42.10+dfsg-3ubuntu3.1) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-4-72:~$
```

```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-4-72:~$ java -version
openjdk version "17.0.15" 2025-04-15
OpenJDK Runtime Environment (build 17.0.15+6-Ubuntu-0ubuntu124.04)
OpenJDK 64-Bit Server VM (build 17.0.15+6-Ubuntu-0ubuntu124.04, mixed mode, sharing)
```

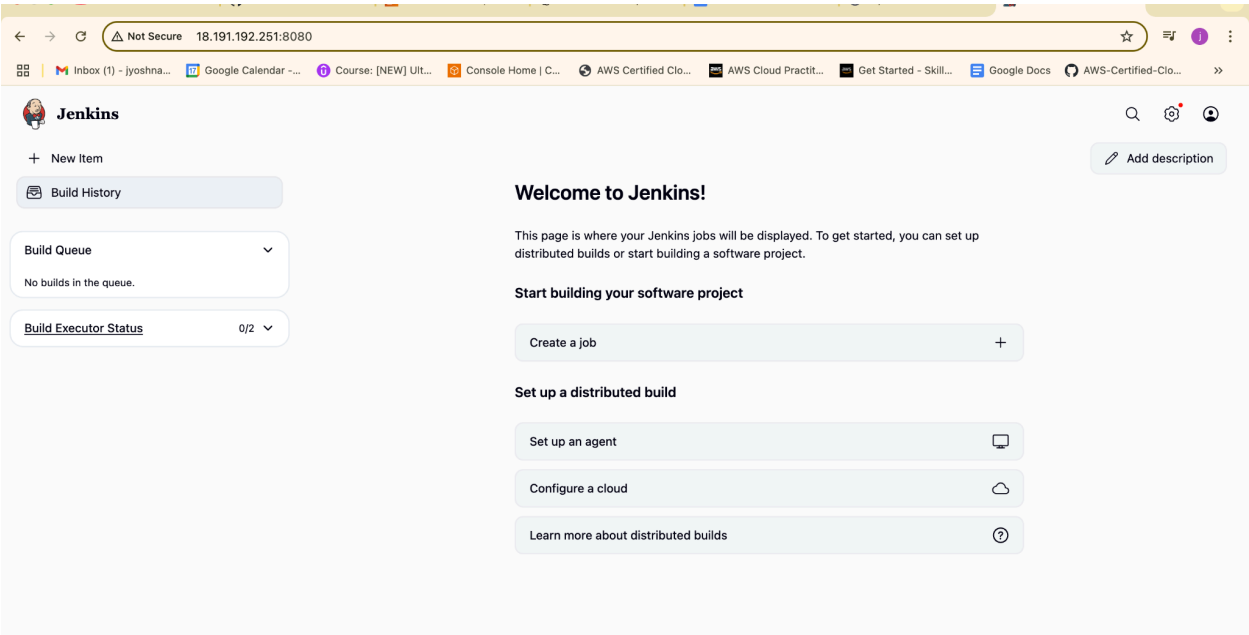
Install Jenkins >default port for running jenkins is port :8080
Can see if jenkins is running are not

```
ubuntu@ip-172-31-4-72:~$ ps -ef | grep jenkins
jenkins     4100       1 19 18:55 ?        00:00:21 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080
ubuntu      4249    1245  0 18:57 pts/0    00:00:00 grep --color=auto jenkins
```

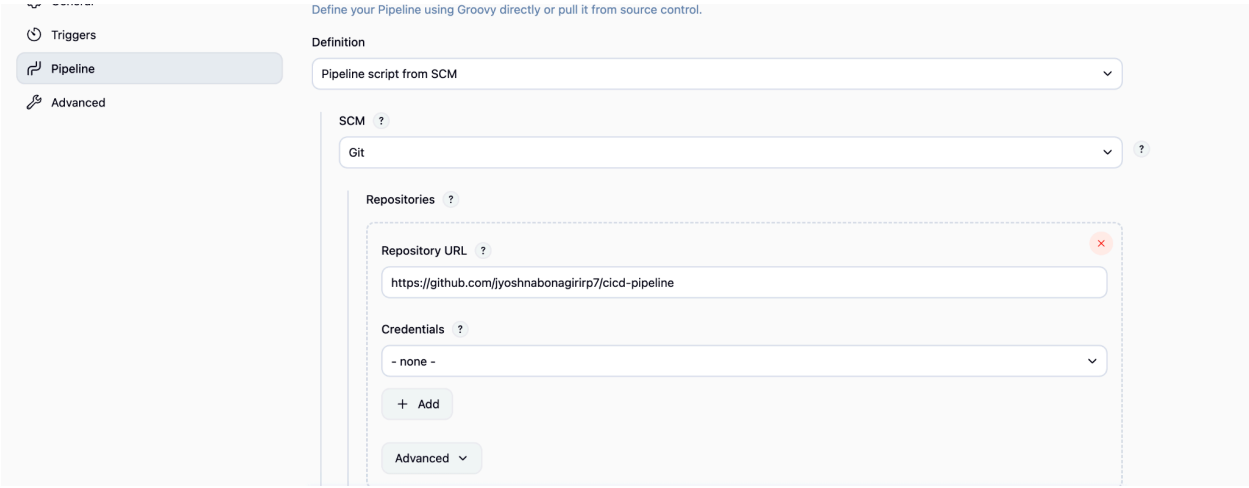
And to know the password apply this command and first and create creds and login to jenkins

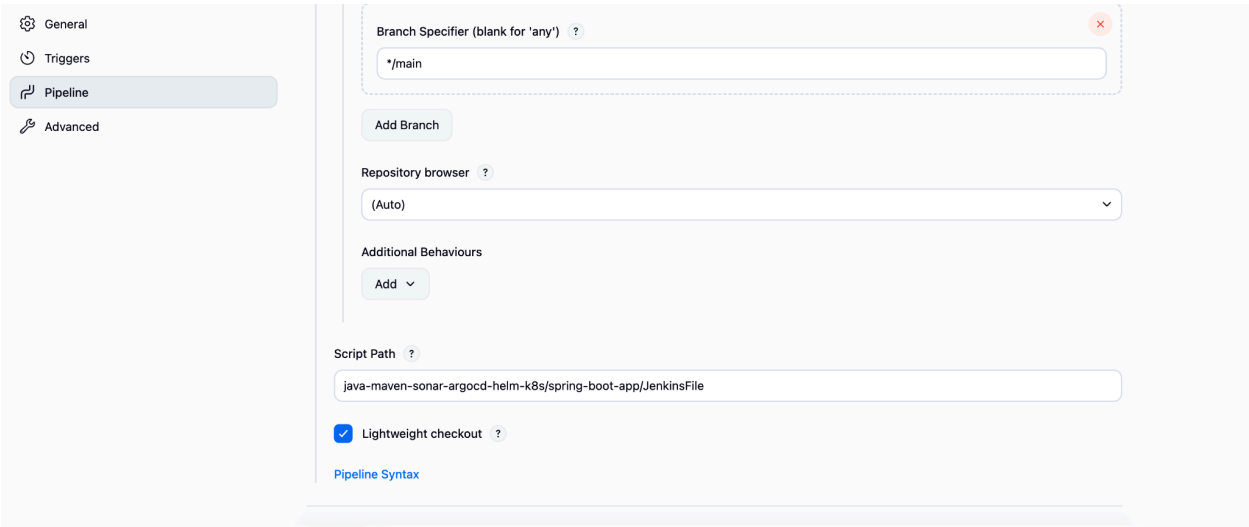
```
ubuntu@ip-172-31-4-72:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword

b1c9aecffd9846c98e28227e87a57284
```

Jenkins is ready



Create a pipeline :ultimate-demo
Add jenkins file path from git repo





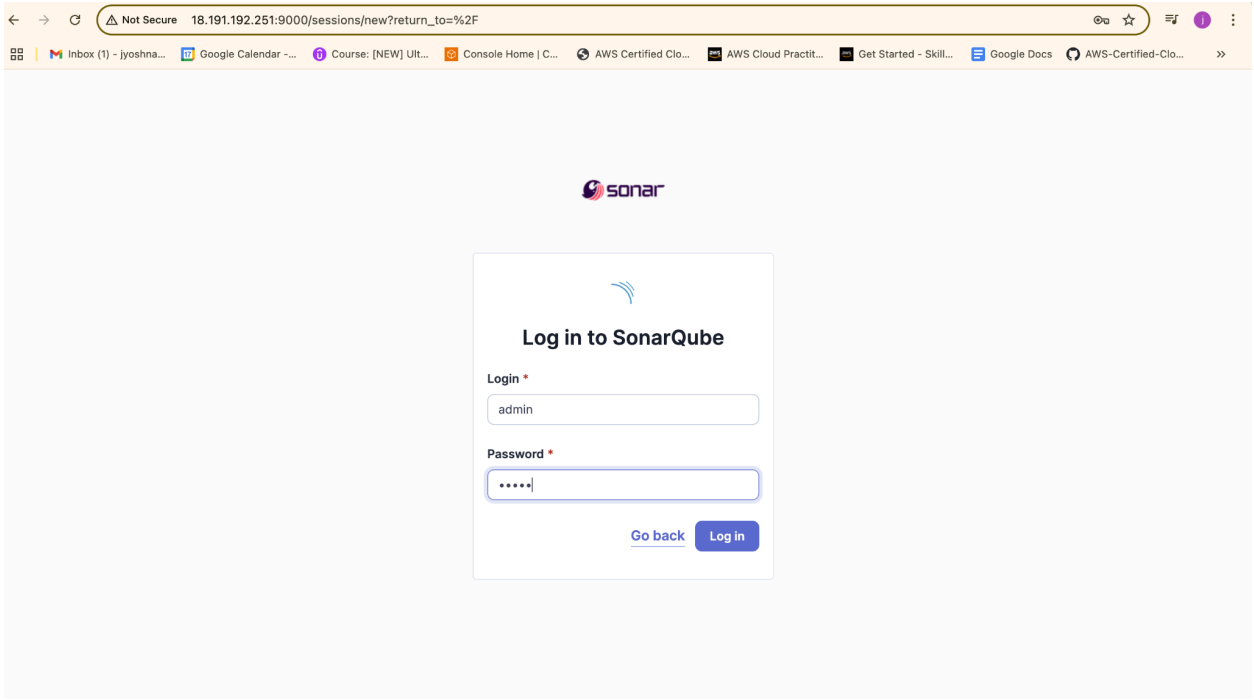
→ Now install plugins >docker pipeline plugin (since im using docker agent in my jenkins file)
>SonarQube Scanner
Now lets Install Sonar Server in my EC2

>create a user called sonarcube
>switch to that user and install dependencies and
>unzip it
>give permissions and start sonar server

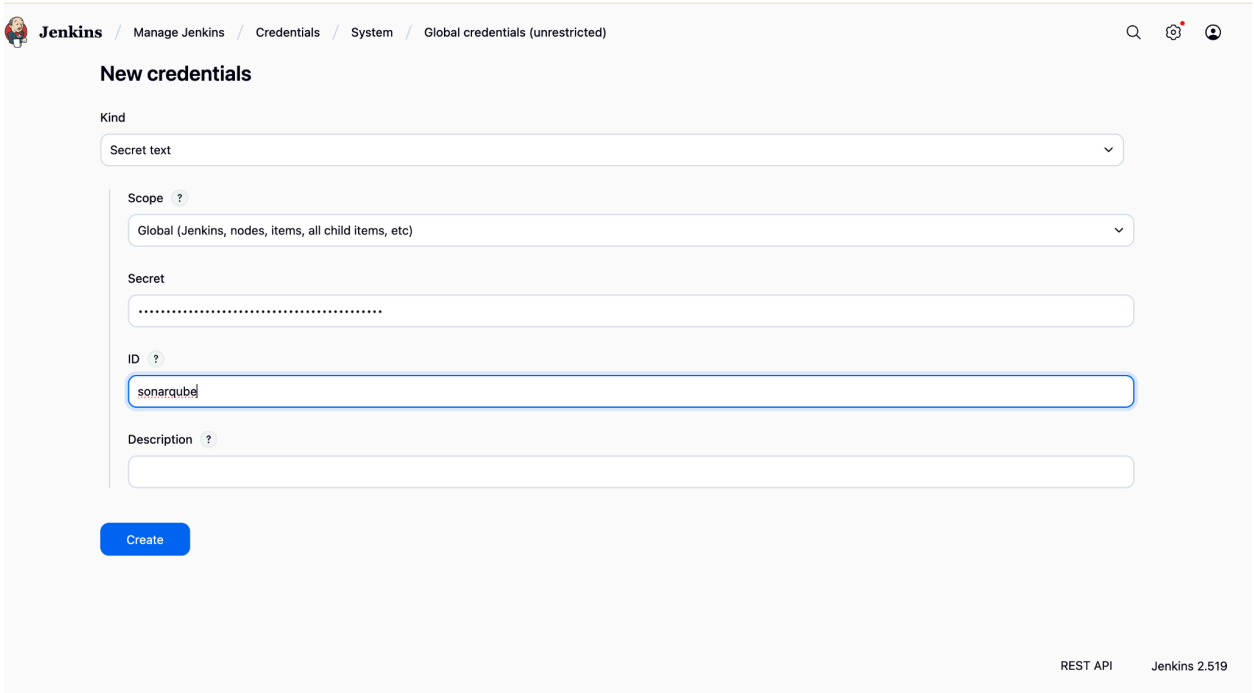
```
ubuntu@ip-172-31-4-72:~$ sudo su -
root@ip-172-31-4-72:~# adduser sonarqube
fatal: The user `sonarqube' already exists.
root@ip-172-31-4-72:~# sudo su - sonarqube
sonarqube@ip-172-31-4-72:~$ wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-10.4.1.88267.zip
--2025-07-19 20:48:49--  https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-10.4.1.88267.zip
Resolving binaries.sonarsource.com (binaries.sonarsource.com)... 108.156.184.16, 108.156.184.23, 108.156.184.47, ...
Connecting to binaries.sonarsource.com (binaries.sonarsource.com)|108.156.184.16|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 455156530 (434M) [binary/octet-stream]
Saving to: 'sonarqube-10.4.1.88267.zip.1'

sonarqube-10.4.1.88267.zip.1                    100%[=============================================================================================================>] 434.07M  54.0MB/s    in 20s

2025-07-19 20:49:09 (21.5 MB/s) - 'sonarqube-10.4.1.88267.zip.1' saved [455156530/455156530]

sonarqube@ip-172-31-4-72:~$ unzip *
Archive:  sonarqube-10.4.1.88267.zip
caution: filename not matched:  sonarqube-10.4.1.88267.zip.1
sonarqube@ip-172-31-4-72:~$ rm sonarqube-10.4.1.88267.zip.1
sonarqube@ip-172-31-4-72:~$ ls -l
total 444496
-rw-rw-r-- 1 sonarqube sonarqube 455156530 Feb 26  2024 sonarqube-10.4.1.88267.zip
sonarqube@ip-172-31-4-72:~$ unzip sonarqube-10.4.1.88267.zip
Archive:  sonarqube-10.4.1.88267.zip
   creating: sonarqube-10.4.1.88267/
  inflating: sonarqube-10.4.1.88267/dependency-license.json
  inflating: sonarqube-10.4.1.88267/COPYING
   creating: sonarqube-10.4.1.88267/bin/
   creating: sonarqube-10.4.1.88267/bin/windows-x86-64/
```

```
chmod: changing permissions of '/opt/sonarqube/temp/conf/es/jvm.options': Operation not permitted
chmod: changing permissions of '/opt/sonarqube/temp/conf/es/elasticsearch.yml': Operation not permitted
chmod: changing permissions of '/opt/sonarqube/bin/linux-x86-64/SonarQube.pid': Operation not permitted
sonarqube@ip-172-31-4-72:~$ exit
logout
root@ip-172-31-4-72:~# sudo chown -R sonarqube:sonarqube /opt/sonarqube
root@ip-172-31-4-72:~# sudo chmod -R 775 /opt/sonarqube
root@ip-172-31-4-72:~# sudo su - sonarqube
sonarqube@ip-172-31-4-72:~$ cd /opt/sonarqube/bin/linux-x86-64
sonarqube@ip-172-31-4-72:/opt/sonarqube/bin/linux-x86-64$ ./sonar.sh start
/usr/bin/java
Starting SonarQube...
Removed stale pid file: ./SonarQube.pid
Started SonarQube.
sonarqube@ip-172-31-4-72:/opt/sonarqube/bin/linux-x86-64$ ls
SonarQube.pid  sonar.sh
sonarqube@ip-172-31-4-72:/opt/sonarqube/bin/linux-x86-64$
```

> try accessing through browser and add credentials



> now login and generate a token for jenkins to access this sonar
and add it in jenkins as global creds



>now install docker in to the ec2 instance

```
sonarqube@ip-172-31-4-72:/opt/sonarqube/bin/linux-x86-64$ exit
logout
root@ip-172-31-4-72:~# sudo apt install docker.io
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-buildx docker-compose-v2 docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io pigz runc ubuntu-fan
0 upgraded, 8 newly installed, 0 to remove and 84 not upgraded.
Need to get 79.2 MB of archives.
After this operation, 300 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 pigz amd64 2.8-1 [65.6 kB]
Get:2 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble/main amd64 bridge-utils amd64 1.7.1-1ubuntu2 [33.9 kB]
Get:3 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 runc amd64 1.2.5-0ubuntu1~24.04.1 [8043 kB]
Get:4 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 containerd amd64 1.7.27-0ubuntu1~24.04.1 [37.7 MB]
Get:5 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 dns-root-data all 2024071801~ubuntu0.24.04.1 [5918 B]
Get:6 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 dnsmasq-base amd64 2.90-2ubuntu0.1 [376 kB]
Get:7 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 docker.io amd64 27.5.1-0ubuntu3~24.04.2 [33.0 MB]
Get:8 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 ubuntu-fan all 0.12.16+24.04.1 [34.2 kB]
Fetched 79.2 MB in 1s (85.7 MB/s)
```

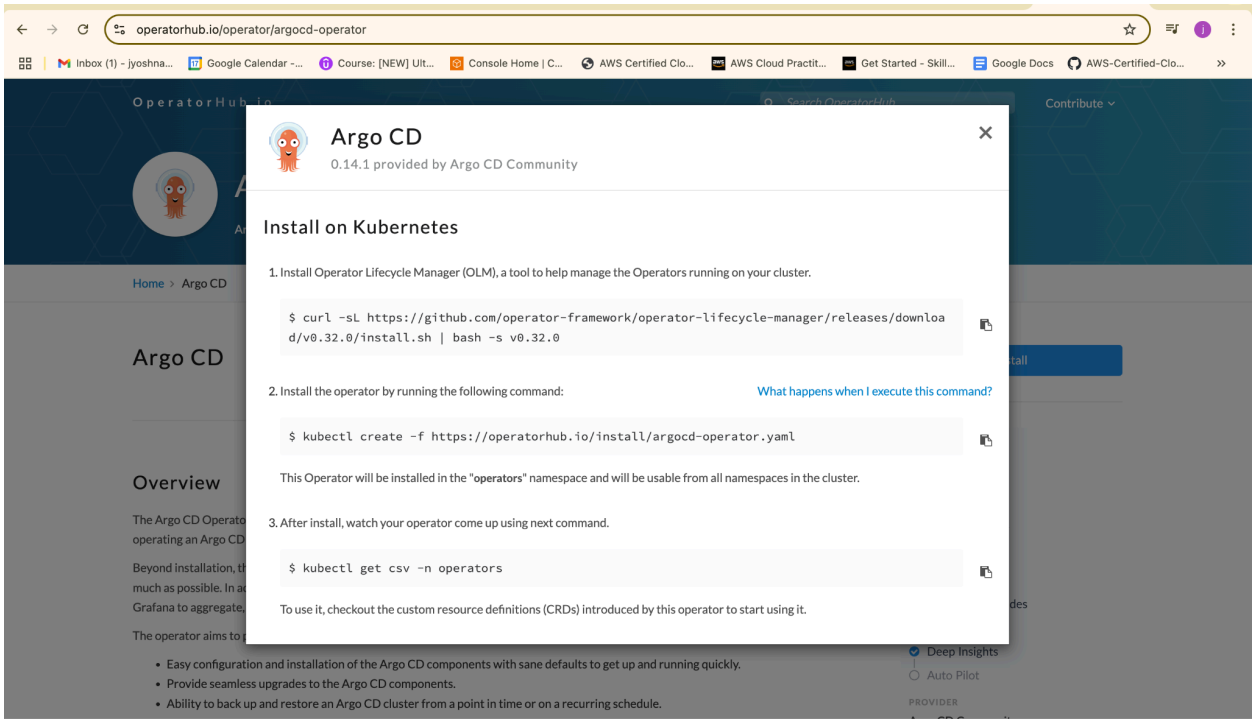
```
Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

User sessions running outdated binaries:
 ubuntu @ session #21: java[6449]

No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@ip-172-31-4-72:~# usermod -aG docker jenkins
usermod -aG docker ubuntu
root@ip-172-31-4-72:~# usermod -aG docker jenkins
root@ip-172-31-4-72:~# usermod -aG docker ubuntu
root@ip-172-31-4-72:~# systemctl restart docker
root@ip-172-31-4-72:~#
```

→now its time to instal k8s in my local machine (i dont need a hyperkit since i have docker desktop running)

```
jyoshnabonagiri@Macbook ~ % brew install minikube

==> Downloading https://formulae.brew.sh/api/formula.jws.json
==> Downloading https://formulae.brew.sh/api/cask.jws.json
==> Downloading https://ghcr.io/v2/homebrew/core/minikube/manifests/1.36.0
######################################################################################################################################################################################## 100.0%
==> Fetching minikube
==> Downloading https://ghcr.io/v2/homebrew/core/minikube/blobs/sha256:e9a0f9e6e9218387985c3dd006b85fbfd8a83f76578902da2bc15a3753bd2839
######################################################################################################################################################################################## 100.0%
==> Pouring minikube--1.36.0.arm64_sequoia.bottle.tar.gz
🍺  /opt/homebrew/Cellar/minikube/1.36.0: 10 files, 124.6MB
==> Running `brew cleanup minikube`...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
==> Caveats
zsh completions have been installed to:
  /opt/homebrew/share/zsh/site-functions
jyoshnabonagiri@Macbook ~ % minikube version

minikube version: v1.36.0
commit: f8f52f5de11fc6ad8244afac475e1d0f96841df1
```

```
jyoshnabonagiri@Macbook ~ % minikube start --driver=docker --memory=4096

😄  minikube v1.36.0 on Darwin 15.5 (arm64)
✨  Using the docker driver based on user configuration
📌  Using Docker Desktop driver with root privileges
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.47 ...
💾  Downloading Kubernetes v1.33.1 preload ...
    > preloaded-images-k8s-v18-v1...:  327.15 MiB / 327.15 MiB  100.00% 7.47 Mi
    > gcr.io/k8s-minikube/kicbase...:  463.69 MiB / 463.69 MiB  100.00% 4.57 Mi
🔥  Creating docker container (CPUs=2, Memory=4096MB) ...
🐳  Preparing Kubernetes v1.33.1 on Docker 28.1.1 ...
    ▪ 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: default-storageclass, storage-provisioner
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
jyoshnabonagiri@Macbook ~ %
```

→instal kubernetes controller >ArgoCD >goto operationshub.io



```
jyoshnabonagiri@Macbook ~ % curl -sL https://github.com/operator-framework/operator-lifecycle-manager/releases/download/v0.32.0/install.sh | bash -s v0.32.0
customresourcedefinition.apiextensions.k8s.io/catalogsources.operators.coreos.com created
customresourcedefinition.apiextensions.k8s.io/clusterserviceversions.operators.coreos.com created
customresourcedefinition.apiextensions.k8s.io/installplans.operators.coreos.com created
customresourcedefinition.apiextensions.k8s.io/olmconfigs.operators.coreos.com created
customresourcedefinition.apiextensions.k8s.io/operatorconditions.operators.coreos.com created
customresourcedefinition.apiextensions.k8s.io/operatorgroups.operators.coreos.com created
customresourcedefinition.apiextensions.k8s.io/operators.operators.coreos.com created
customresourcedefinition.apiextensions.k8s.io/subscriptions.operators.coreos.com created
customresourcedefinition.apiextensions.k8s.io/catalogsources.operators.coreos.com condition met
customresourcedefinition.apiextensions.k8s.io/clusterserviceversions.operators.coreos.com condition met
customresourcedefinition.apiextensions.k8s.io/installplans.operators.coreos.com condition met
customresourcedefinition.apiextensions.k8s.io/olmconfigs.operators.coreos.com condition met
customresourcedefinition.apiextensions.k8s.io/operatorconditions.operators.coreos.com condition met
customresourcedefinition.apiextensions.k8s.io/operatorgroups.operators.coreos.com condition met
customresourcedefinition.apiextensions.k8s.io/operators.operators.coreos.com condition met
customresourcedefinition.apiextensions.k8s.io/subscriptions.operators.coreos.com condition met
namespace/olm created
```

```
subscription.operators.coreos.com/my-argocd-operator created
jyoshnabonagiri@Macbook ~ % kubectl get csv -n operators
No resources found in operators namespace.
jyoshnabonagiri@Macbook ~ % kubectl get csv -n operators
NAME                     DISPLAY   VERSION   REPLACES                  PHASE
argocd-operator.v0.14.1   Argo CD   0.14.1    argocd-operator.v0.14.0   Succeeded
jyoshnabonagiri@Macbook ~ %
```

This is the Argo CD Operator's controller pod — it runs inside the operators namespace and is responsible for:

Watching for ArgoCD custom resources (CRs)
Deploying and managing Argo CD instances

```
No resources found in operators namespace.
jyoshnabonagiri@Macbook ~ % kubectl get csv -n operators
NAME                     DISPLAY   VERSION   REPLACES                  PHASE
argocd-operator.v0.14.1   Argo CD   0.14.1    argocd-operator.v0.14.0   Succeeded
jyoshnabonagiri@Macbook ~ % kubectl get pods -n operators
NAME                                          READY   STATUS    RESTARTS   AGE
argocd-operator-controller-manager-6f9747bff7-pgt7z   1/1     Running   0          5m22s
jyoshnabonagiri@Macbook ~ %
```

>>Create and update Jenkins File accordingly

```
pipeline {

  agent {

    docker {

      image 'jyoshnasakala/maven-docker-agent:latest'

      args '--user root -v /var/run/docker.sock:/var/run/docker.sock'

    }
```

```
  }
  stages {
    stage('Checkout') {
      steps {
        sh 'echo passed'
        //git branch: 'main', url: 'https://github.com/iam-veeramalla/Jenkins-Zero-To-Hero.git'
      }
    }
    stage('Build and Test') {
      steps {
        sh 'ls -ltr'
        sh 'cd java-maven-sonar-argocd-helm-k8s/spring-boot-app && mvn clean package'
      }
    }
    stage('Static Code Analysis') {
      environment {
        SONAR_URL = "http://3.148.232.139:9000"
      }
      steps {
        withCredentials([string(credentialsId: 'sonarqube', variable: 'SONAR_AUTH_TOKEN')]) {
          sh 'cd java-maven-sonar-argocd-helm-k8s/spring-boot-app && mvn sonar:sonar
-Dsonar.login=$SONAR_AUTH_TOKEN -Dsonar.host.url=${SONAR_URL}'
        }
      }
    }
```

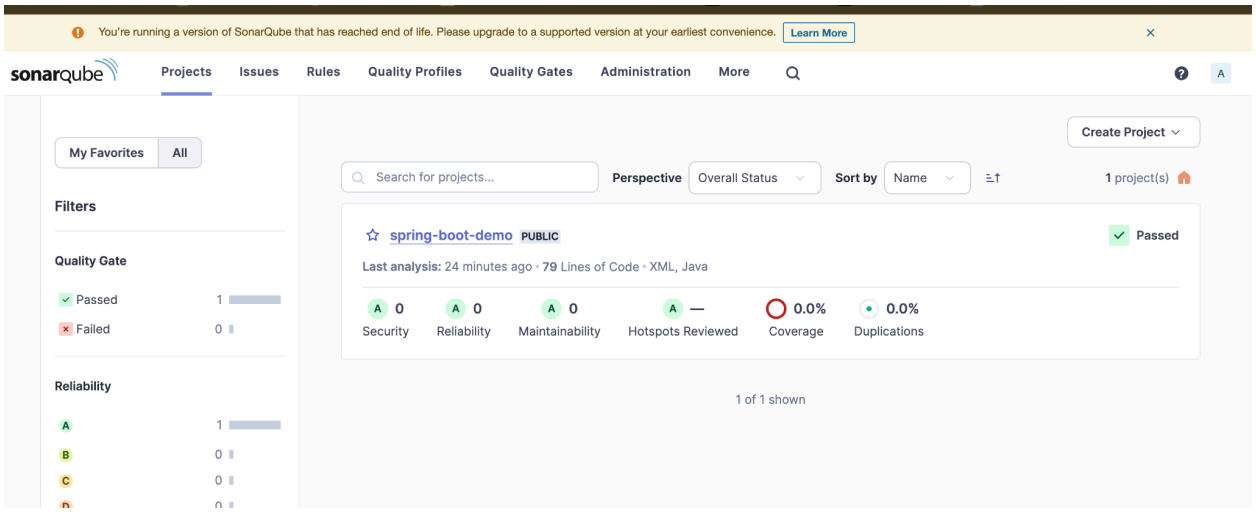
```
    stage('Build and Push Docker Image') {
      environment {
        DOCKER_IMAGE = "jyoshnasakala/ultimate-cicd:${BUILD_NUMBER}"
        REGISTRY_CREDENTIALS = credentials('docker-cred')
      }
      steps {
        script {
          sh 'cd java-maven-sonar-argocd-helm-k8s/spring-boot-app && docker build -t ${DOCKER_IMAGE} .'
          def dockerImage = docker.image("${DOCKER_IMAGE}")
          docker.withRegistry('https://index.docker.io/v1/', "docker-cred") {
            dockerImage.push()
          }
        }
      }
    }
    stage('Update Deployment File') {
      environment {
        GIT_REPO_NAME = "cicd-pipeline"
        GIT_USER_NAME = "jyoshnabonagirirp7"
      }
      steps {
        withCredentials([string(credentialsId: 'github', variable: 'GITHUB_TOKEN')]) {
          sh '''
            rm -rf $GIT_REPO_NAME
```

```
            git clone
https://${GITHUB_TOKEN}@github.com/${GIT_USER_NAME}/${GIT_REPO_NAME}.git

            cd $GIT_REPO_NAME



            git config user.email "jyoshnarp7@gmail.com"

            git config user.name "Jyoshna Bonagiri"



            sed -i "s/replaceImageTag/${BUILD_NUMBER}/g"
java-maven-sonar-argocd-helm-k8s/spring-boot-app-manifests/deployment.yml



            git add java-maven-sonar-argocd-helm-k8s/spring-boot-app-manifests/deployment.yml

            git commit -m "Update deployment image to version ${BUILD_NUMBER}"

            git push origin main

          '''

        }

      }

    }

  }

}
```

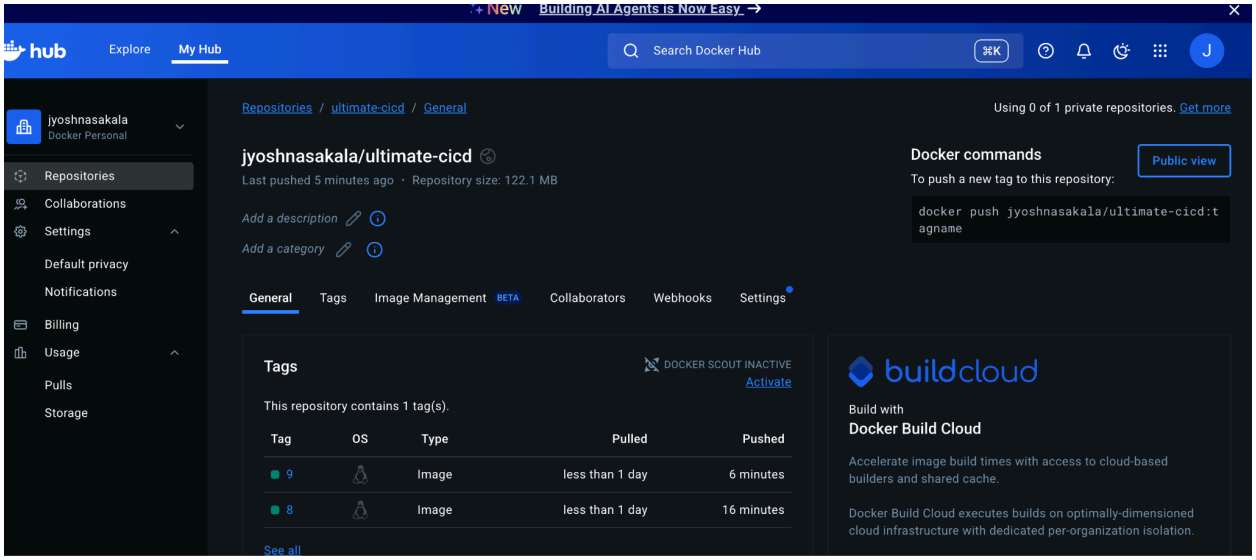
My build was success in jenkins

```
[main 491e1c2] Update deployment image to version 9
 1 file changed, 1 insertion(+), 1 deletion(-)
+ git push origin main
To https://github.com/jyoshnabonagirirp7/cicd-pipeline.git
   8b029b2..491e1c2  main -> main
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
$ docker stop --time=1 4765196de3716220faa4d37590531a2bc8fd93f4bf5cf61919715230d73db73b
$ docker rm -f --volumes 4765196de3716220faa4d37590531a2bc8fd93f4bf5cf61919715230d73db73b
[Pipeline] // withDockerContainer
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

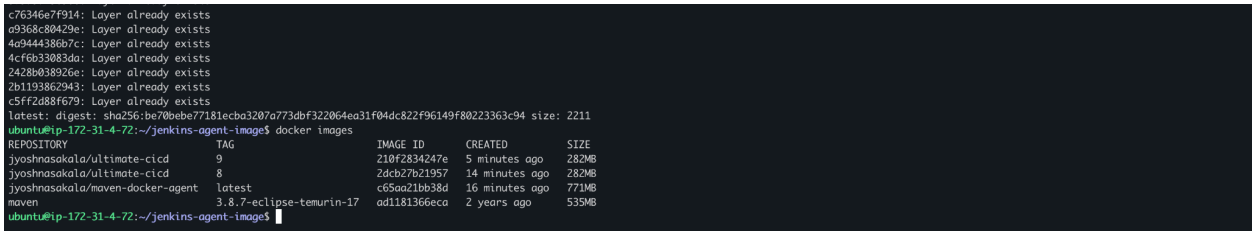
And as part of pipeline my code is no complaints



Docker image was pushed to my docker

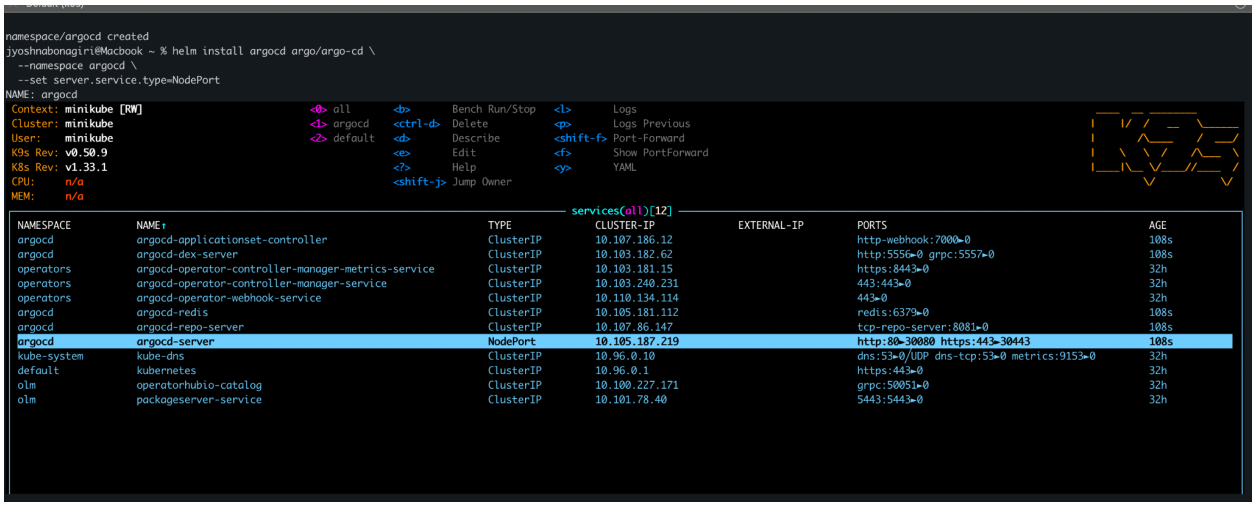


Checking from my instance as well

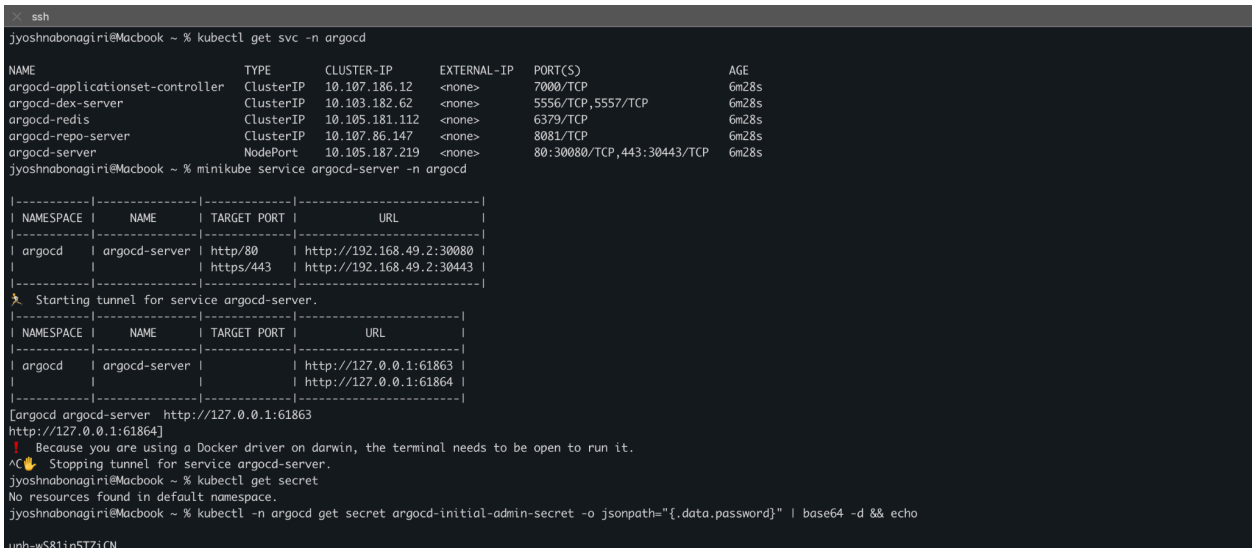


The CI part is done now it's time for the CD…

Since i already install minikube
Let me create helm

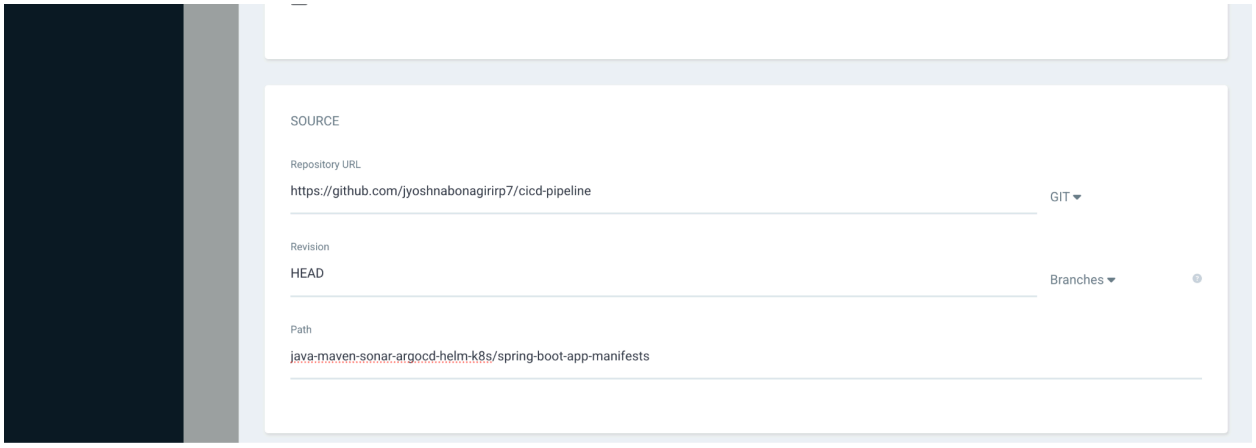


Give us an temporary url for argocd

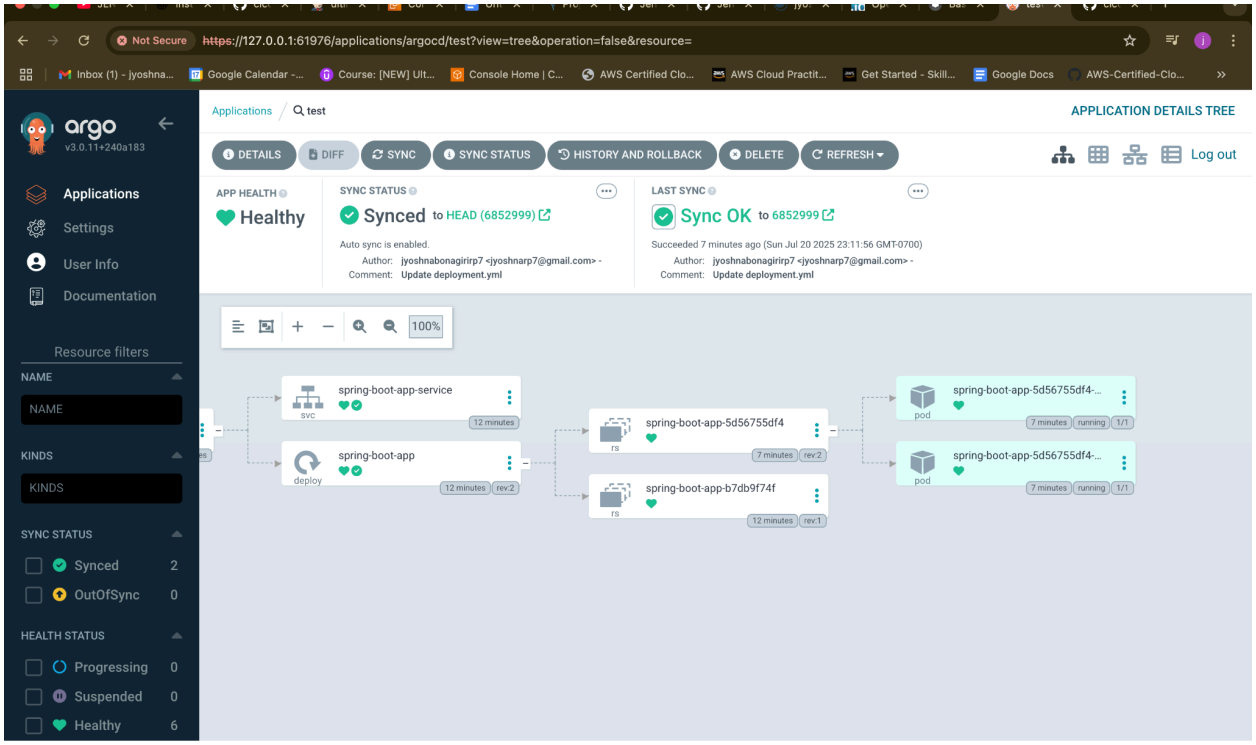


```
ssh
jyoshnabonagiri@Macbook ~ % kubectl get svc -n argocd

NAME                               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
argocd-applicationset-controller   ClusterIP   10.107.186.12    <none>        7000/TCP                     6m28s
argocd-dex-server                  ClusterIP   10.103.182.62    <none>        5556/TCP,5557/TCP            6m28s
argocd-redis                       ClusterIP   10.105.181.112   <none>        6379/TCP                     6m28s
argocd-repo-server                 ClusterIP   10.107.86.147    <none>        8081/TCP                     6m28s
argocd-server                      NodePort    10.105.187.219   <none>        80:30080/TCP,443:30443/TCP   6m28s
jyoshnabonagiri@Macbook ~ % minikube service argocd-server -n argocd

|-----------|---------------|-------------|---------------------------|
| NAMESPACE |     NAME      | TARGET PORT |            URL            |
|-----------|---------------|-------------|---------------------------|
| argocd    | argocd-server | http/80     | http://192.168.49.2:30080 |
|           |               | https/443   | http://192.168.49.2:30443 |
|-----------|---------------|-------------|---------------------------|
🏃  Starting tunnel for service argocd-server.
|-----------|---------------|-------------|------------------------|
| NAMESPACE |     NAME      | TARGET PORT |          URL           |
|-----------|---------------|-------------|------------------------|
| argocd    | argocd-server |             | http://127.0.0.1:61863 |
|           |               |             | http://127.0.0.1:61864 |
|-----------|---------------|-------------|------------------------|
[argocd argocd-server  http://127.0.0.1:61863
http://127.0.0.1:61864]
❗  Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
^C✋  Stopping tunnel for service argocd-server.
jyoshnabonagiri@Macbook ~ % kubectl get secret
No resources found in default namespace.
jyoshnabonagiri@Macbook ~ % kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d && echo

unh-wS81inSTZiCN
```

Now can access through browser
And do some configs and update the source with git rep
namespace>default



Pods got deployed successfully..



Check your pods in default namespace:kubectl get pods -n default #argocd in this case

Access Your Application kubectl get svc -n default Get Minikube IP:minikube ip

Access your app in browser:

http://<minikube-ip>:<node-port>

Thankyou