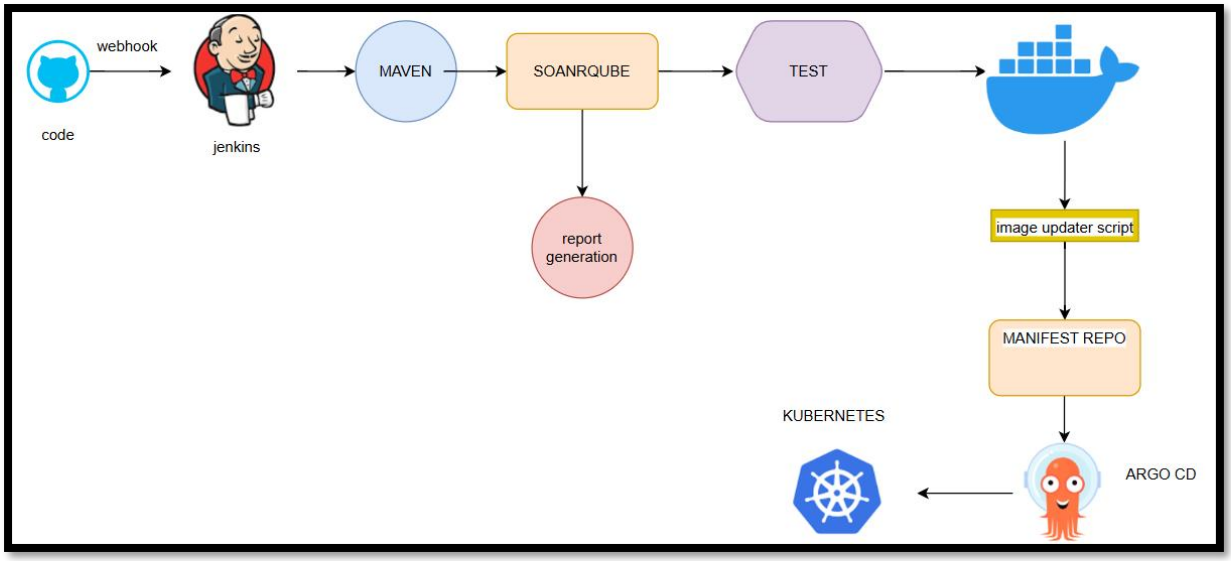
CI/CD PROJECT



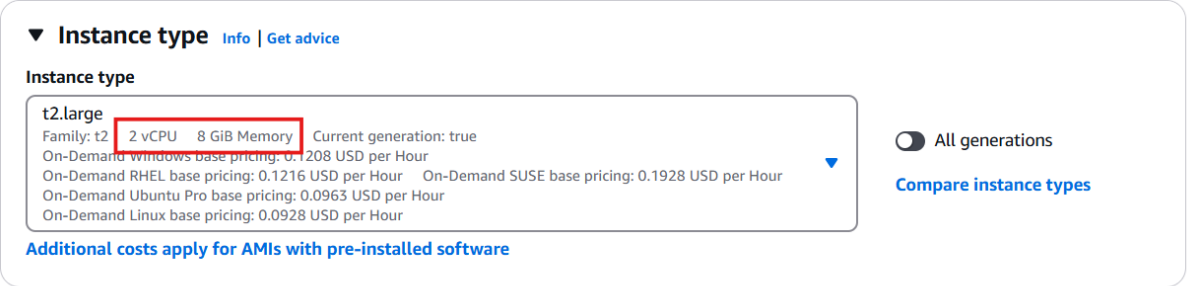
This project represents a **CI/CD (Continuous Integration and Continuous Deployment) pipeline** that automates the software development and deployment process using popular DevOps tools.

Pipeline Workflow:

1. **Source Code Management (GitHub)**
    - Developers push code changes to a GitHub repository.
    - A **webhook** triggers the CI/CD process.
2. **Continuous Integration (Jenkins & Maven)**
    - **Jenkins** is responsible for running the pipeline.
    - **Maven** is used to build the project.
3. **Code Quality and Testing (SonarQube & Test Stage)**
    - **SonarQube** analyzes the code for quality and security issues.
    - If the code passes quality checks, automated **tests** are executed.
4. **Containerization and Image Deployment (DockerHub)**
    - If tests pass, a new **Docker image** is built and pushed to **DockerHub**.
5. **Continuous Deployment (ArgoCD & Kubernetes)**
    - An **image updater script** updates the deployment configuration.
    - The updated image is committed to the **Manifests Repository**.
    - **ArgoCD** detects changes in the repository and deploys the updated application to **Kubernetes**.

LET'S START

Create an EC2 instance.



Ssh into the machine

Now install Jenkins

Install Jenkins.

Pre-Requisites:

- Java (JDK)

Run the below commands to install Java and Jenkins

Install Java

sudo apt update

sudo apt install openjdk-17-jre

Verify Java is Installed

java -version

Now, you can proceed with installing Jenkins, Since it's a Linux system, refer to Jenkins documentation for installation commands

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]" \

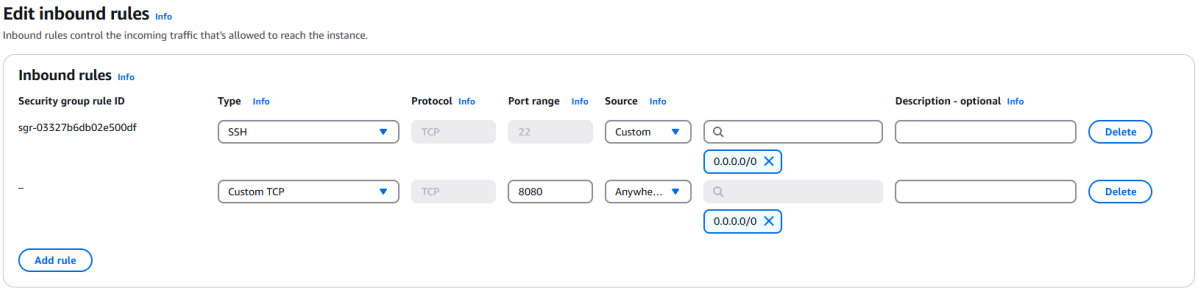
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

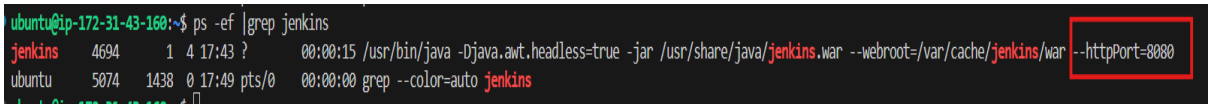
sudo apt-get update

sudo apt-get install Jenkins

**Note: ** By default, Jenkins will not be accessible to the external world due to the inbound traffic restriction by AWS. Open port 8080 in the inbound traffic



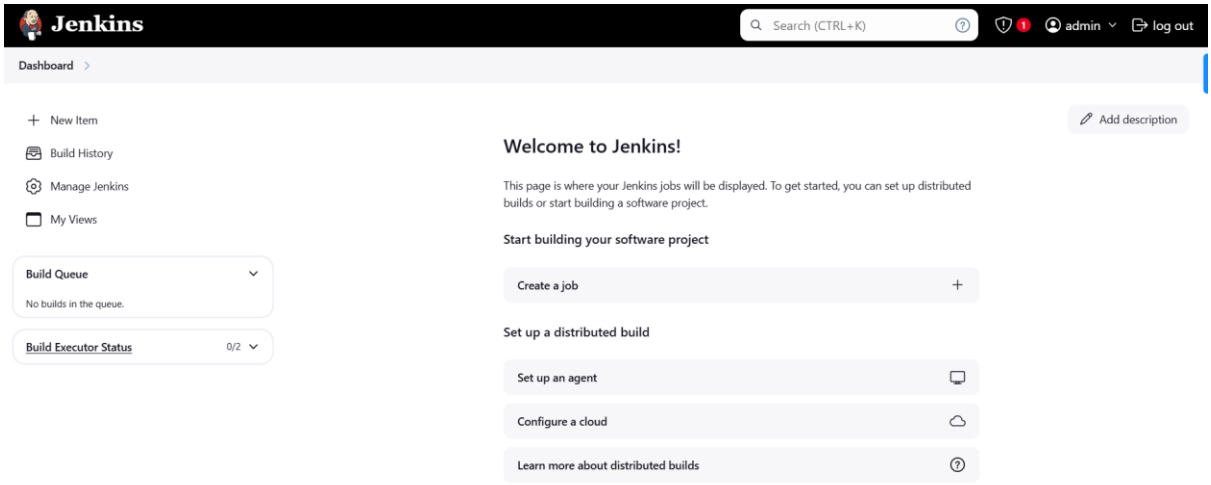
Checking if Jenkins is running in ec2 instance



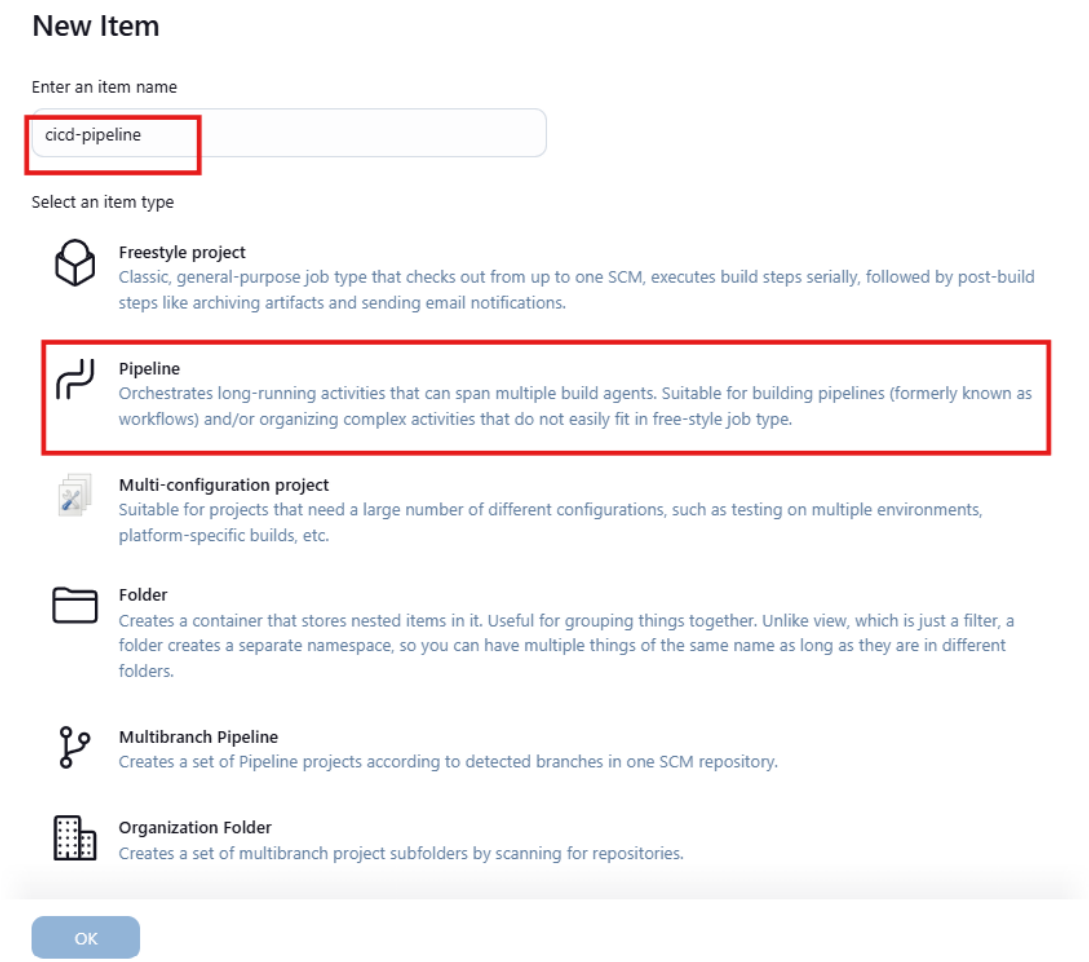
Accessing Jenkins through browser at http://54.80.16.178:8080/

And perform necessary configurations like

1. Install suggested plugins



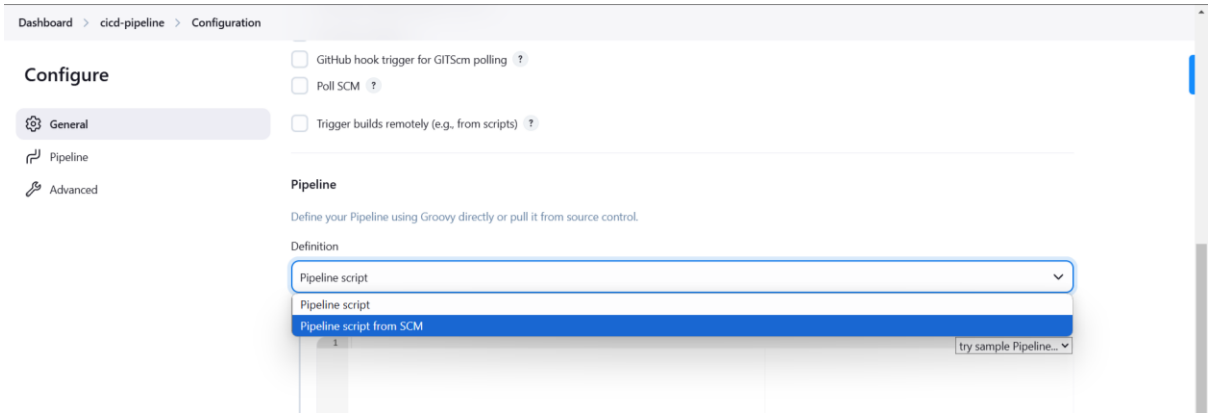
Once Jenkins is setup. Go to new items, here we'll create a Jenkins pipeline



Here I am using the pipeline option

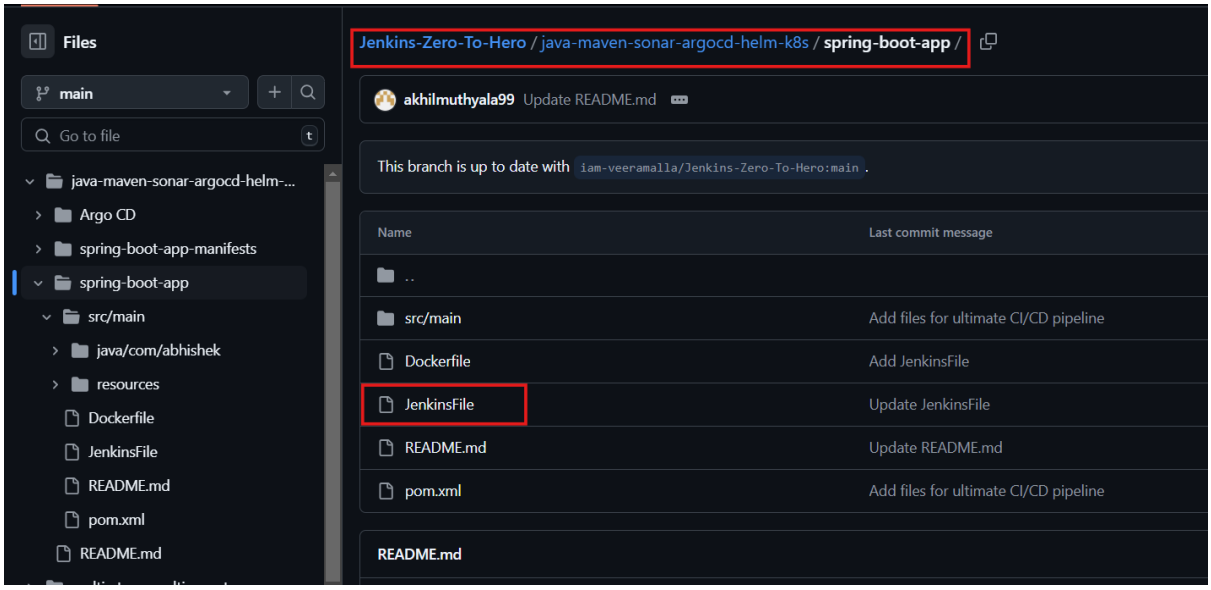
Key Benefits of Jenkins Pipeline:

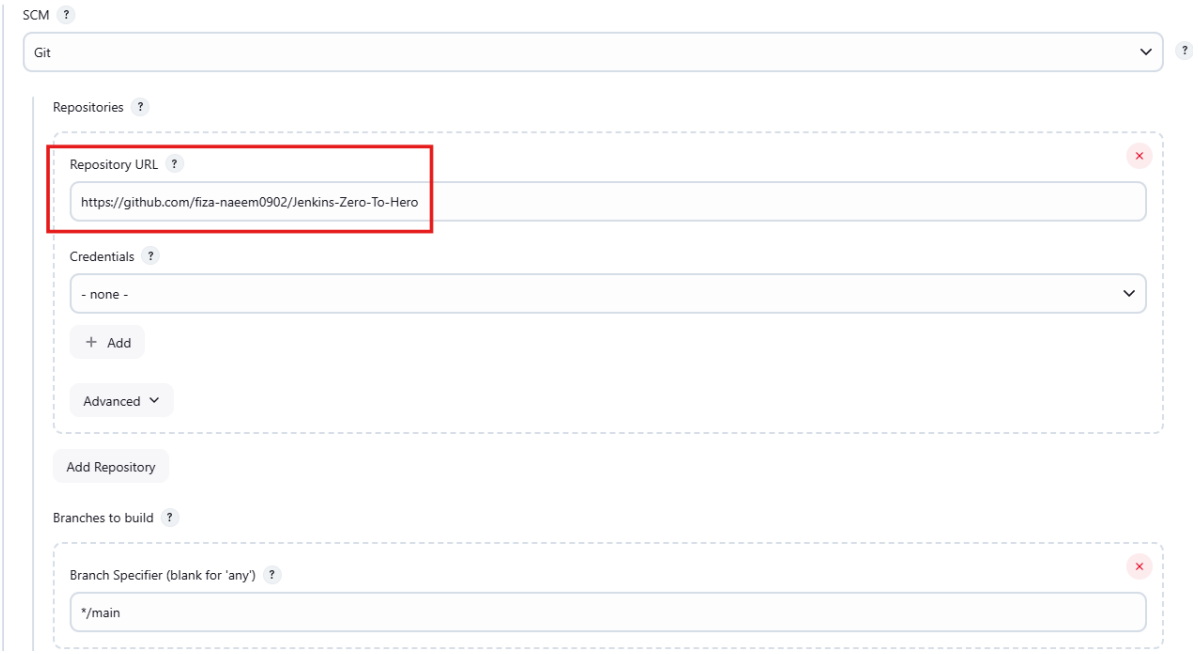
✅ **Pipeline as Code** – Uses a Jenkinsfile (written in Groovy) for version-controlled workflows.
✅ **Automation & Efficiency** – Automates the entire CI/CD process.
✅ **Error Handling & Resilience** – Can retry failed steps, handle exceptions, and improve stability.
✅ **Parallel Execution** – Runs tasks in parallel, speeding up builds.
✅ **Modular & Reusable** – Supports functions, shared libraries, and parameterized builds.
✅ **Better Visibility** – Provides real-time graphical views of the pipeline stages.



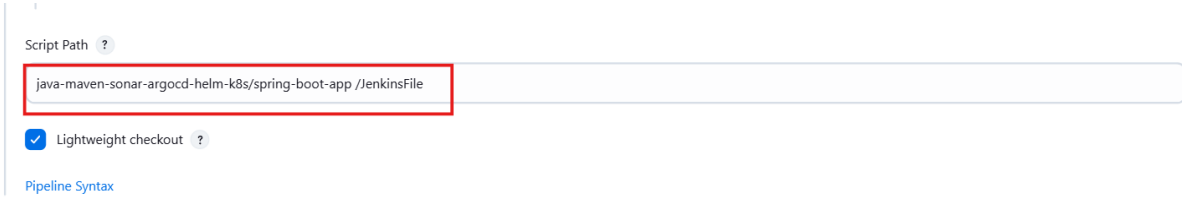
There is this “Pipeline script from SCM” option. Using it we can provide our jenkinsfile path present on Github in our case, to jekins

Referring to this Jenkins file, we’ll be using it. This file contains all the stages to be executed for CI.





Jenkins clone the repo and search for the path where the Jenkinsfile is present, and it can have any name. we are going to provide its path here



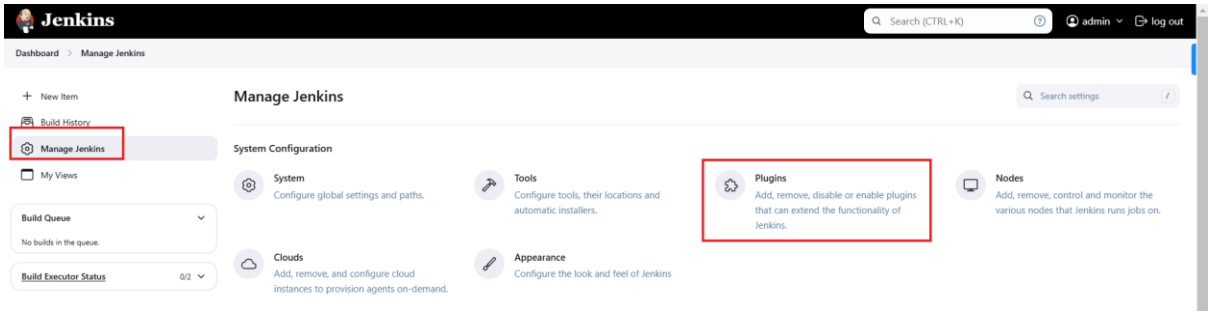
And now simply APPLY

Good practice is to always use DOCKER AS AN AGENT for Jenkins pipeline

1. It lessens the configurations
2. Containers are light weight, along with it once the pipeline is executed they get created and executes all the stages inside the container and gets deleted once the job is done. Therefore parallel tasks execution can take place and multiple containers can run at once in a single instance.

In comparison to it EC2 instances will require more configurations and will be costly as well

For it a plugin has to installed -> DOCKER PIPELINE



Now we'll configure sonar server on ec2 also in Jenkins we'll install SONAR PLUGIN

Now back to the ec2 instance, installing sonarqube server. By default it runs on port 9000. So we'll also add that in inbound rules.

apt install unzip

Sudo su - //switching to root

adduser sonarqube

sudo su - sonarqube

wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.4.0.54424.zip

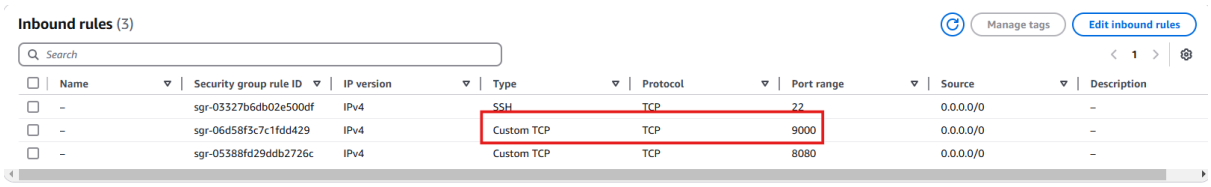
unzip *

chmod -R 755 /home/sonarqube/sonarqube-9.4.0.54424

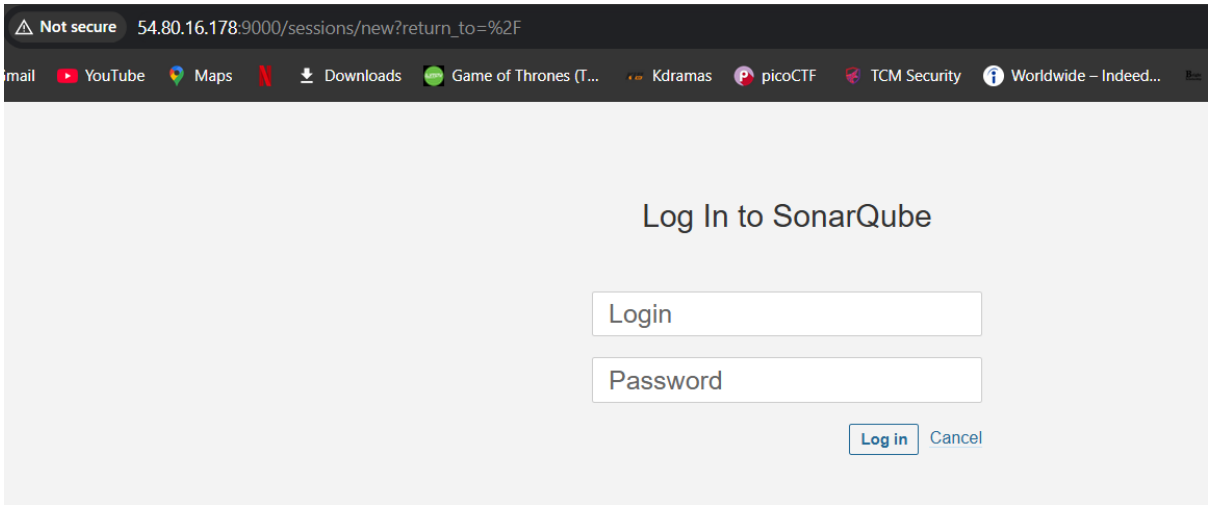
chown -R sonarqube:sonarqube /home/sonarqube/sonarqube-9.4.0.54424

cd sonarqube-9.4.0.54424/bin/linux-x86-64/

./sonar.sh start



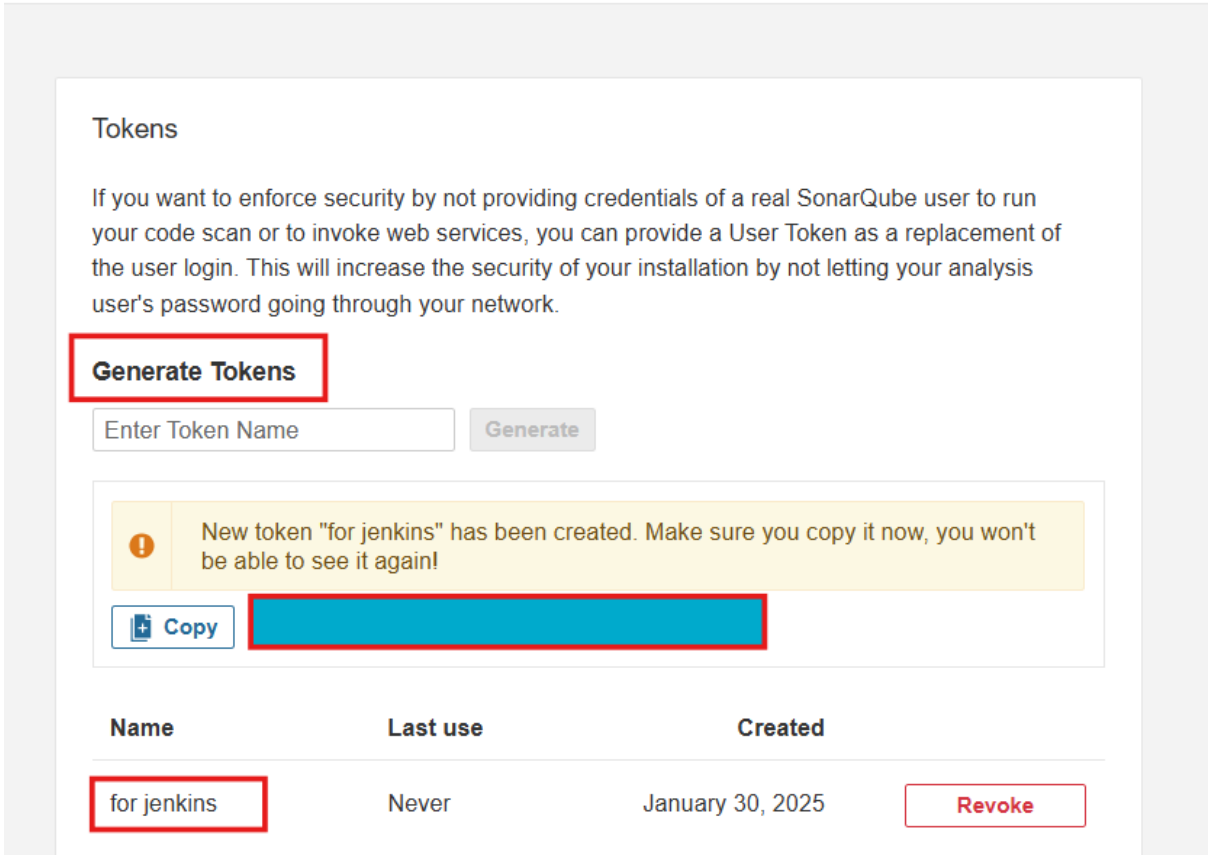
And here its accessible



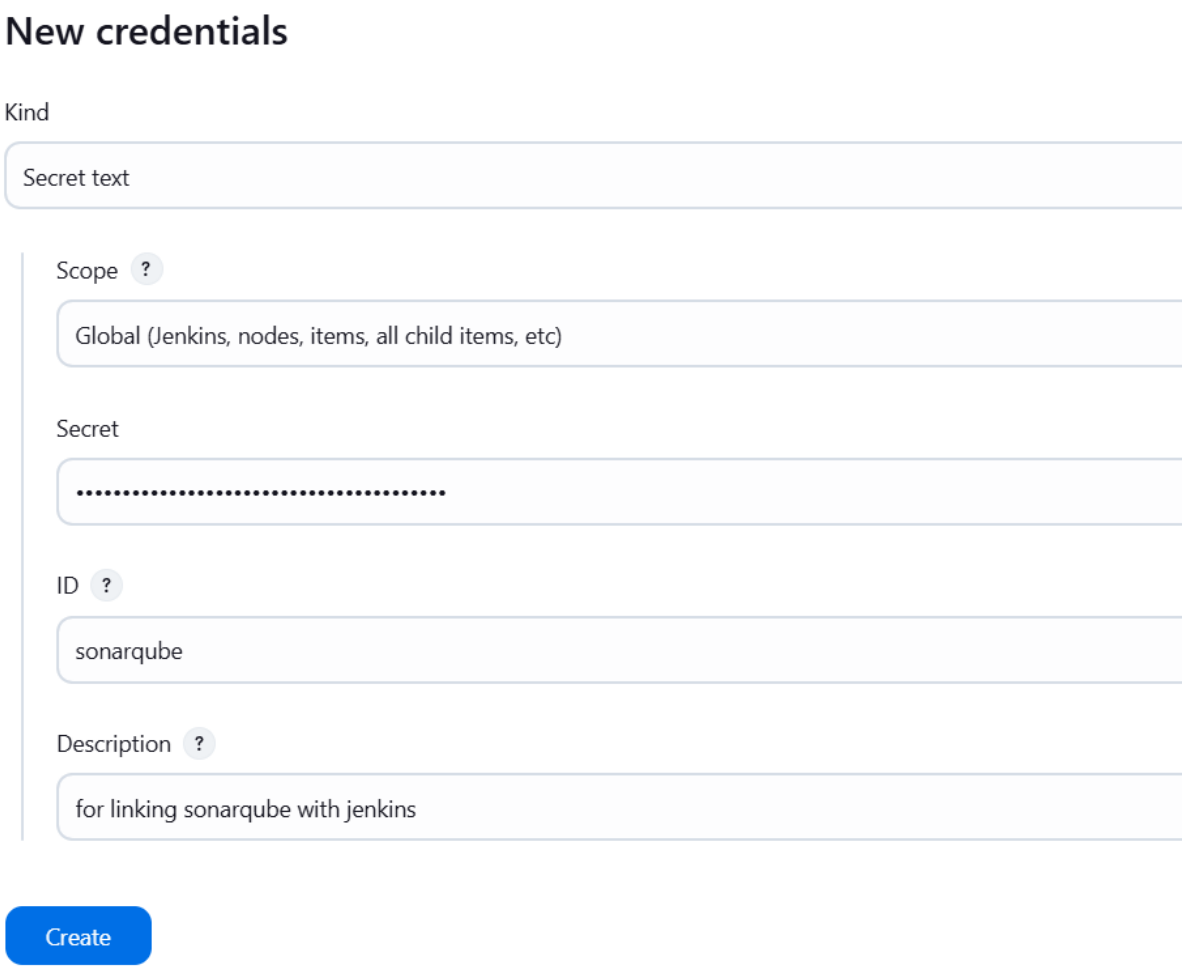
Our docker container has already maven installed, so we're not downloading that

Now time to establish link between sonarqube and Jenkins

Go to sonarqube > my account > security



Copy this token. Go to Jenkins > manage Jenkins > credentials > system> global credentials> add credentials



Now sonarqube configuration Is all done

Next step is to install docker on ec2. For it logout of sonarqube user and shift to root

```
sonarqube@ip-172-31-43-160:~/sonarqube-9.4.0.54424/bin/linux-x86-64$ logout
root@ip-172-31-43-160:~#
```

Docker Slave Configuration

Run the below command to Install Docker

sudo apt update

sudo apt install docker.io

Grant Jenkins user and Ubuntu user permission to docker deamon.

sudo su -

usermod -aG docker jenkins

usermod -aG docker ubuntu

systemctl restart docker

Once you are done with the above steps, it is better to restart Jenkins.

http://54.80.16.178:8080/restart

The docker agent configuration is now successful.

For CD, here we are using k8s and its controller argo cd. We'll be setting them on our host machine, reason is: ec2 reources aren't enough to deploy them on our instance

Best practice is to always download k8s controllers using operators

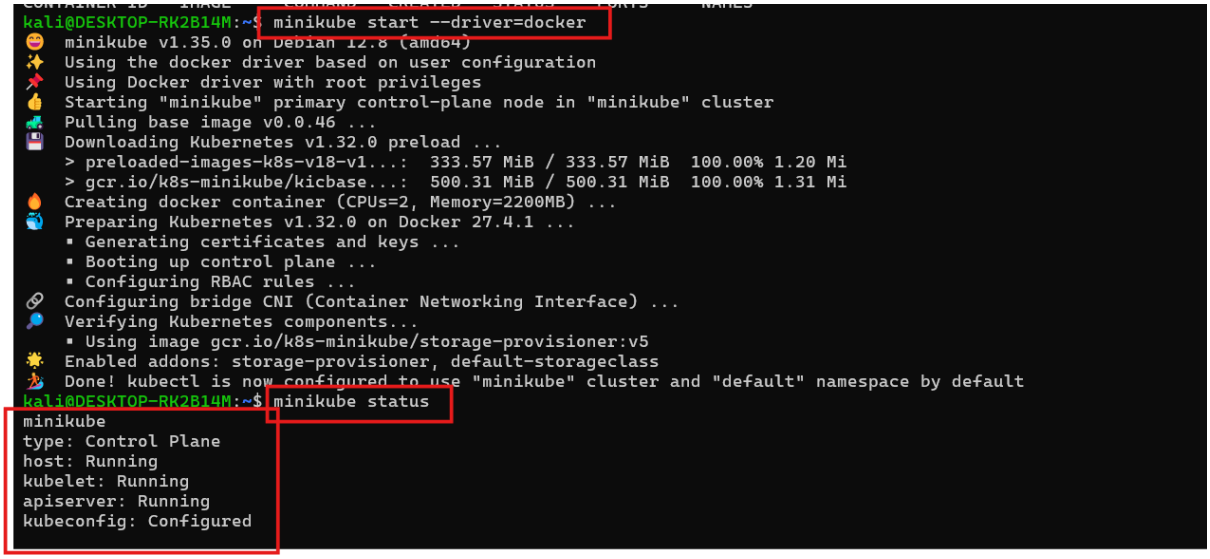
Analogy:

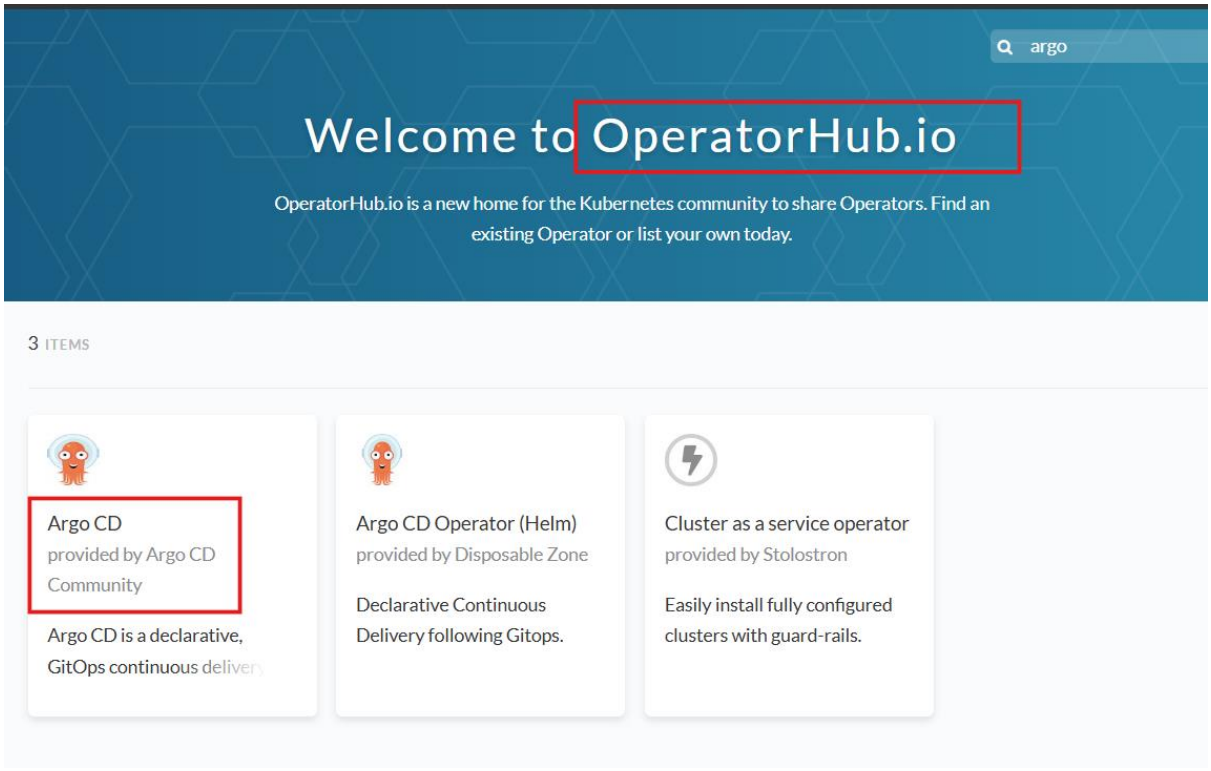
Think of the **controller** as the specific tool (Argo CD) performing a task and the **operator** as the manager overseeing the tool's health, upgrades, and configurations.

The Argo CD Operator might automate:

- Installing and configuring Argo CD.
- Upgrading it to newer versions.
- Managing configurations for high availability or scaling.

Before installing argo cd, make sure minikube is running





Argo CD

0.13.0 provided by Argo CD Community

Install on Kubernetes

1. Install Operator Lifecycle Manager (OLM), a tool to help manage the Operators running on your cluster.

```
$ curl -sL https://github.com/operator-framework/operator-lifecycle-manager/releases/download/v0.31.0/install.sh | bash -s v0.31.0
```

2. Install the operator by running the following command: What happens when I execute this command?

```
$ kubectl create -f https://operatorhub.io/install/argocd-operator.yaml
```

This Operator will be installed in the "**operators**" namespace and will be usable from all namespaces in the cluster.

3. After install, watch your operator come up using next command.

```
$ kubectl get csv -n operators
```

To use it, checkout the custom resource definitions (CRDs) introduced by this operator to start using it.

I am running these commands in cmd using wsl

Lets talk abot the Jenkins file contents

```
pipeline {
  agent {
    docker {
      image 'abhishekf5/maven-abhishek-docker-agent:v1'
      args '--user root -v /var/run/docker.sock:/var/run/docker.sock' // mount Docker socket to access the host's Docker daemon
    }
  }
  stages {
    stage('Checkout') {
      steps {
        sh 'echo passed'
        //git branch: 'main', url: 'https://github.com/iam-veeramalla/Jenkins-Zero-To-Hero.git'
      }
    }
```

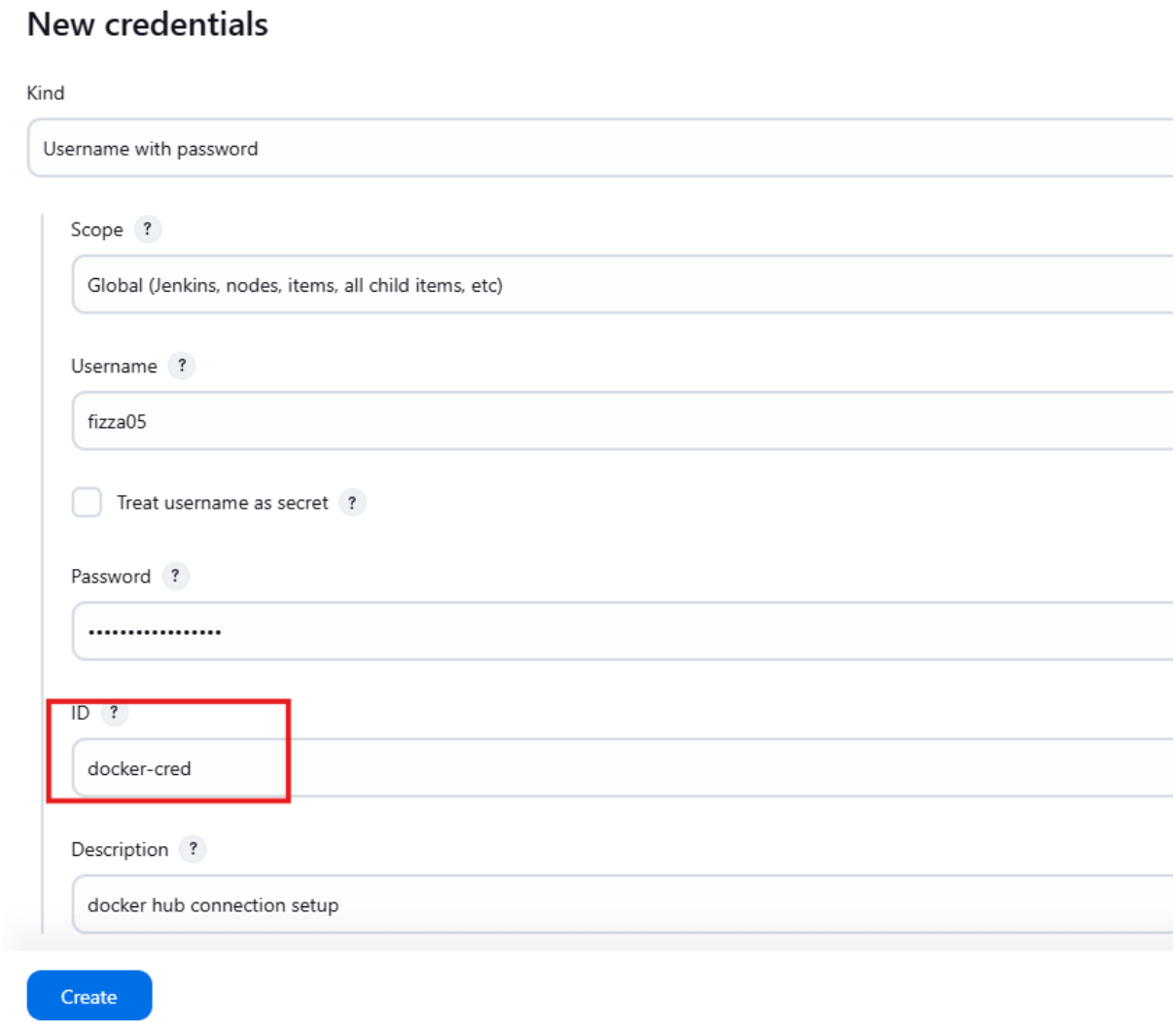
Here I am using docker as an agent, and in that image I have maven installation done

Secondly the stage 1, here in it as I have used SCM for Jenkins pipeline therefore I can simply say its passed

Now setting up Jenkins to have connection with dockerhub and our repository(through)

As here for docker hub credentials are named as “docker-cred” therefore naming id the same

```
stage('Build and Push Docker Image') {
  environment {
    DOCKER_IMAGE = "abhishekf5/ultimate-cicd:${BUILD_NUMBER}"
    // DOCKERFILE_LOCATION = "java-maven-sonar-argocd-helm-k8s/spring-boot-app/Dockerfile"
    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()
```



Now generate access token for github and connect github with Jenkins

Now restart Jenkins

Lastly pull the docker image, run powershell as administrator

Minikube start

docker pull abhishekf5/maven-abhishek-docker-agent:v1

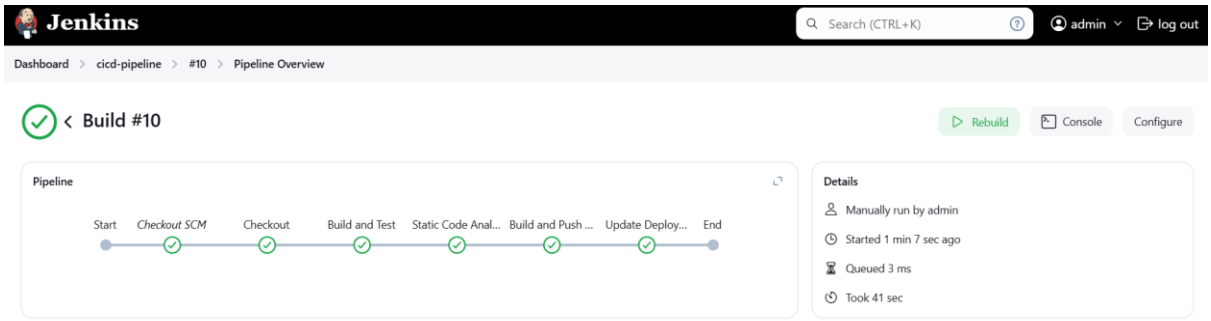
docker tag abhishekf5/maven-abhishek-docker-agent:v1 fizza05/maven-abhishek-docker-agent:v1

docker login

push fizza05/maven-abhishek-docker-agent:v1

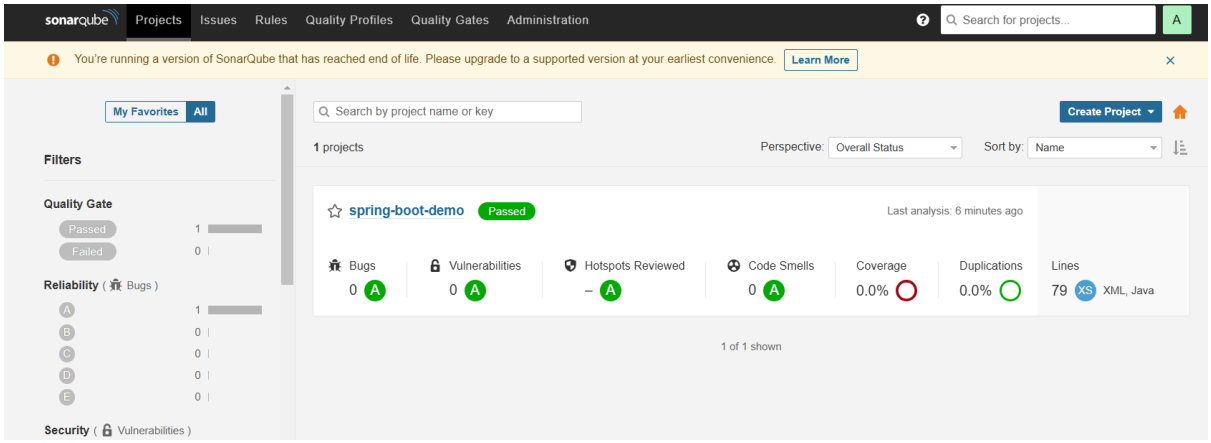
And build the ci

AT THIS POINT KEEP YOUR CALM

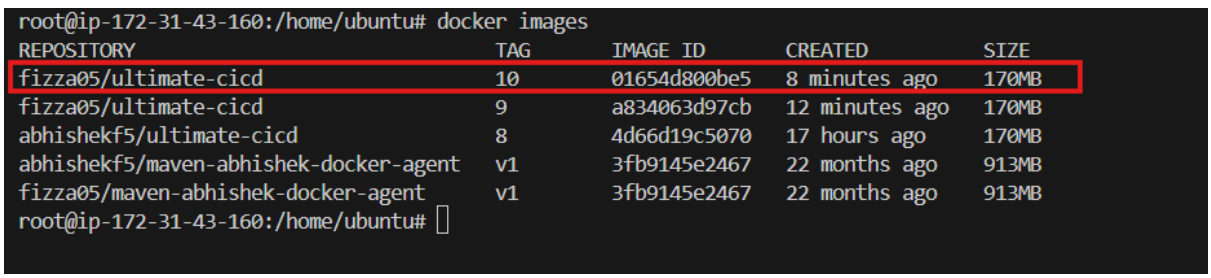


And volla, after about 10 attempts here its done

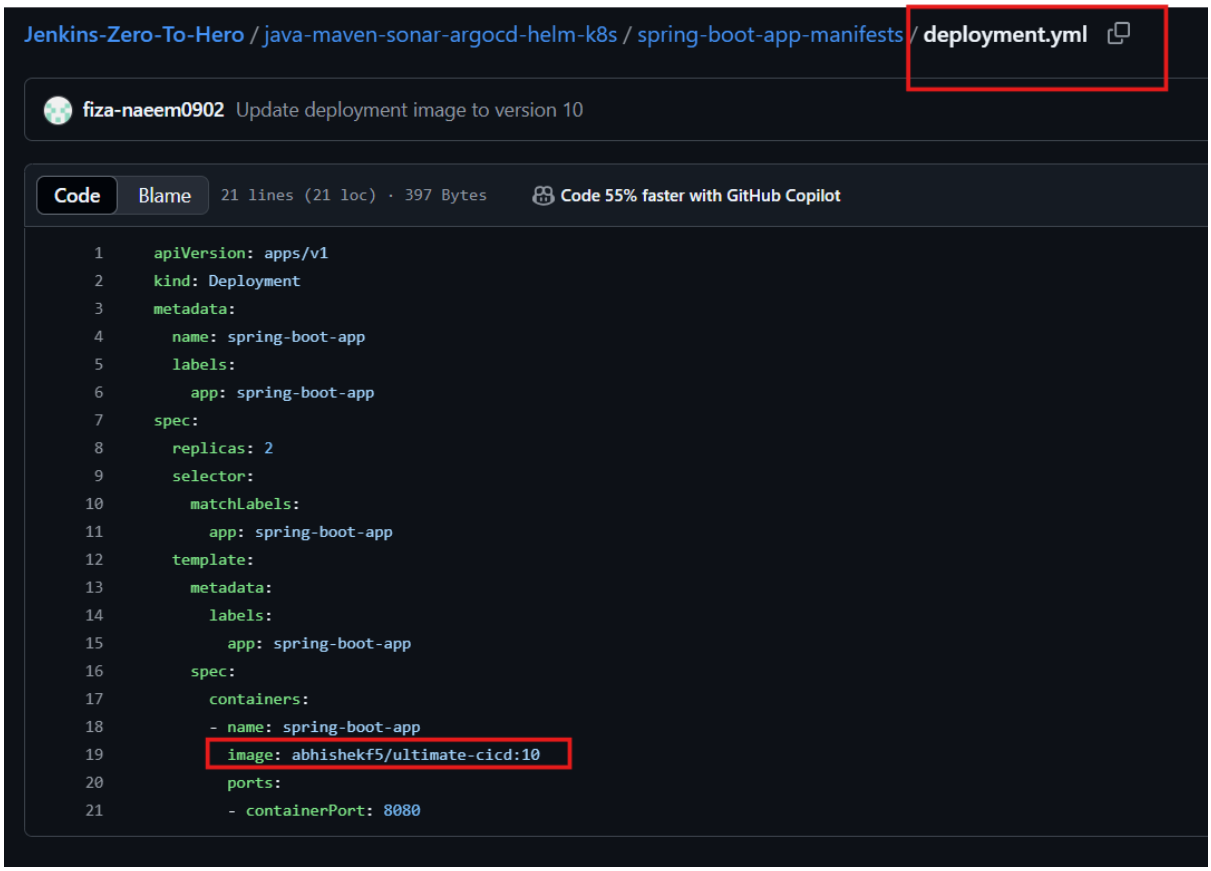
Sonarqube results



And thus the docker image is created at ec2, dockerhub

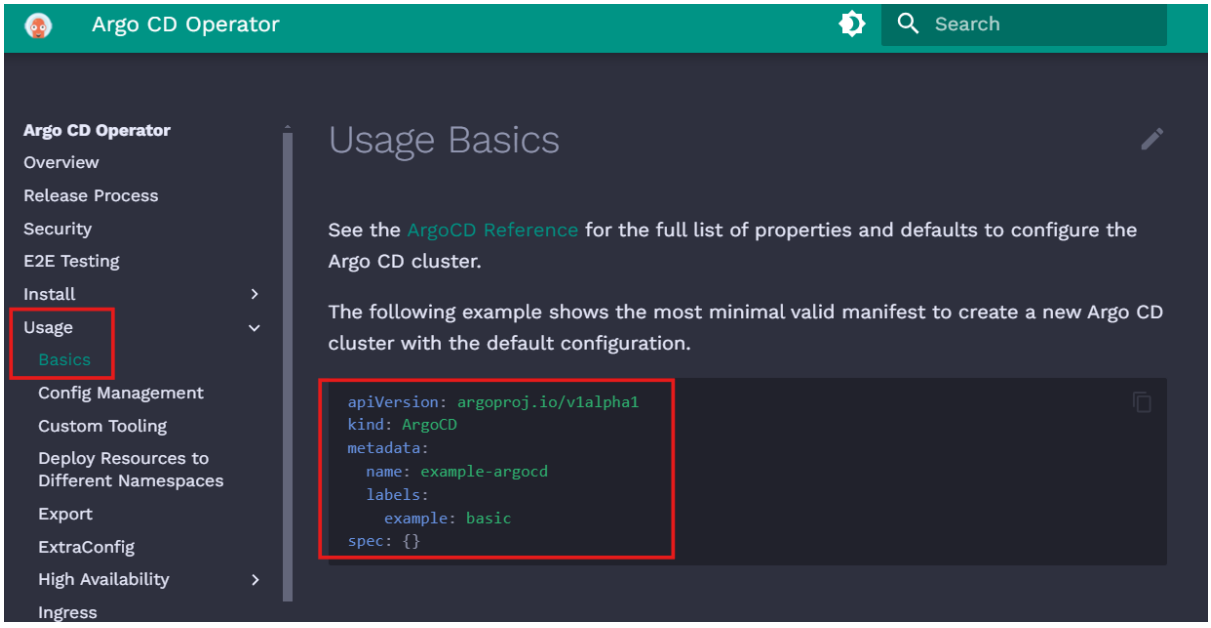


While the latest one will be present in my manifest repo



Now starting off with CD part. As argo cd operator's installation is done. Now creating the argo cd controller

Go to operatorhub.io> search argo cd> select operator documentation >



Copy this code and put in the file

And apply this file

```
kali@DESKTOP-RK2B14M:~$ nano argocd-basic.yml
kali@DESKTOP-RK2B14M:~$ kubectl apply -f argocd-basic.yml
Warning: ArgoCD v1alpha1 version is deprecated and will be converted to v1beta1 automatical
ly. Moving forward, please use v1beta1 as the ArgoCD API version.
argocd.argoproj.io/example-argocd created
```

```
kali@DESKTOP-RK2B14M:~$ kubectl get pods
NAME                                          READY   STATUS              RESTARTS   AGE
example-argocd-application-controller-0       0/1     ContainerCreating   0          41s
example-argocd-redis-5678d59479-h8599         0/1     ContainerCreating   0          42s
example-argocd-repo-server-6ddfc9947f-5bbtf   0/1     Init:0/1            0          42s
example-argocd-server-5b9d85449b-qdqbj        0/1     ContainerCreating   0          42s
kali@DESKTOP-RK2B14M:~$
```

So argocd workloads are getting created

Now we'll pull the latest image from the git manifest repo

```
kali@DESKTOP-RK2B14M:~$ kubectl get pods -w
NAME                                          READY   STATUS              RESTARTS   AGE
example-argocd-application-controller-0       0/1     ContainerCreating   0          2m59s
example-argocd-redis-5678d59479-h8599         0/1     ContainerCreating   0          3m
example-argocd-repo-server-6ddfc9947f-5bbtf   0/1     Init:0/1            0          3m
example-argocd-server-5b9d85449b-qdqbj        0/1     ContainerCreating   0          3m
```

So containers are getting created

For running in browser as well:

```
kali@DESKTOP-RK2B14M:~$ kubectl get svc
NAME                            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)             AGE
example-argocd-metrics          ClusterIP   10.107.140.79   <none>        8082/TCP            4m27s
example-argocd-redis            ClusterIP   10.104.184.44   <none>        6379/TCP            4m27s
example-argocd-repo-server      ClusterIP   10.96.154.43    <none>        8081/TCP,8084/TCP   4m27s
example-argocd-server           ClusterIP   10.97.177.118   <none>        80/TCP,443/TCP      4m27s
example-argocd-server-metrics   ClusterIP   10.102.90.142   <none>        8083/TCP            4m27s
kubernetes                      ClusterIP   10.96.0.1       <none>        443/TCP             19h
kali@DESKTOP-RK2B14M:~$
```

This server is responsible for argo cd UI

```
kali@DESKTOP-RK2B14M:~$ export EDITOR=nano
kubectl edit svc example-argocd-server
service/example-argocd-server edited
kali@DESKTOP-RK2B14M:~$
```

Since I want to run it on browser so Change the type from clusterip to nodeport

To execute it on browser use: minikube service argocd-server

With this minikube will generate a url, by which it can be accessed

To get the password of ARGOCD, use command: kubectl get secret, while username is: admin

from the example-argocd-cluster, copy the secret and decrypt it using base64

now provide the necessary info, provide it with the deployment.yaml file

AND YOU ARE DONE