

Jenkins-Zero-To-Hero

Are you looking forward to learn Jenkins right from Zero(installation) to Hero (Build end to end pipelines)? then you are at the right place.

Installation on EC2 Instance

YouTube Video -

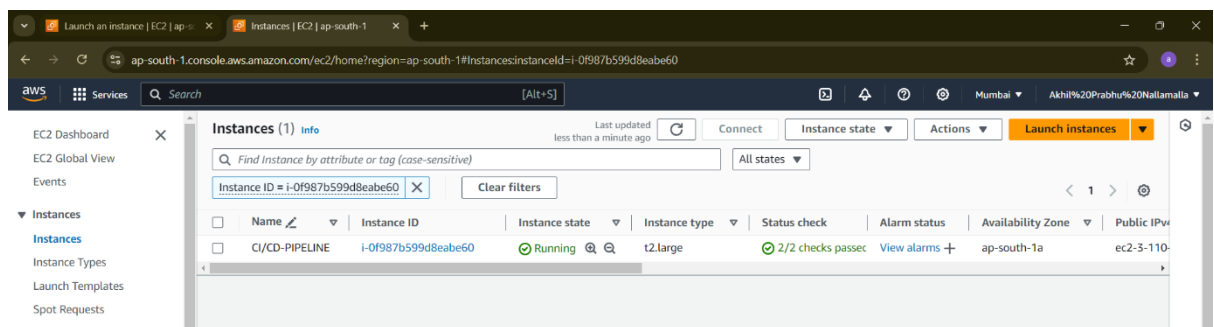
> <https://www.youtube.com/watch?v=zZfhAXfBvVA&list=RDCMUCnnQ3ybuyFdzv2Ky5jnAA&index=1>



Install Jenkins, configure Docker as agent, set up cicd, deploy applications to k8s and much more.

AWS EC2 Instance

- Go to AWS Console
- Instances(running)
- Launch instances



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

- ```
curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo tee \
```

```
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
```
- ```
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
```



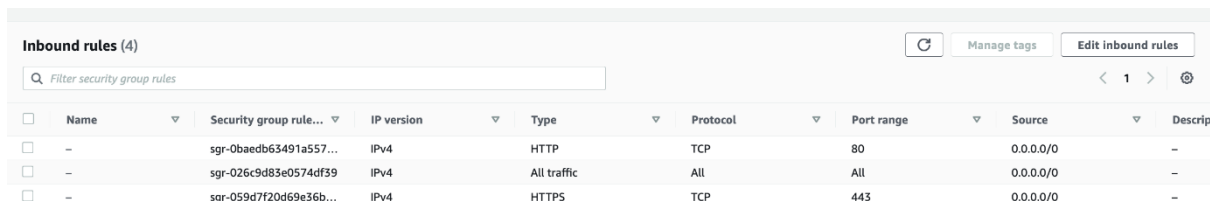
```
https://pkg.jenkins.io/debian binary/ | sudo tee \
```



```
/etc/apt/sources.list.d/jenkins.list > /dev/null
```
- ```
sudo apt-get update
```
- ```
sudo apt-get install jenkins
```

****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 rules as show below.

- EC2 > Instances > Click on
- In the bottom tabs -> Click on Security
- Security groups
- Add inbound traffic rules as shown in the image (you can just allow TCP 8080 as well, in my case, I allowed All traffic).



Inbound rules (4)										
<input type="text" value="Filter security group rules"/>										
<input type="checkbox"/>	Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Descrip		
<input type="checkbox"/>	-	sgr-0baedb63491a557...	IPv4	HTTP	TCP	80	0.0.0.0/0	-		
<input type="checkbox"/>	-	sgr-026c9d83e0574df39	IPv4	All traffic	All	All	0.0.0.0/0	-		
<input type="checkbox"/>	-	sgr-059d7f20d69e36b...	IPv4	HTTPS	TCP	443	0.0.0.0/0	-		

Login to Jenkins using the below URL:

<http://:8080> [You can get the ec2-instance-public-ip-address from your AWS EC2 console page]

Note: If you are not interested in allowing All Traffic to your EC2 instance 1. Delete the inbound traffic rule for your instance 2. Edit the inbound traffic rule to only allow custom TCP port 8080

After you login to Jenkins, - Run the command to copy the Jenkins Admin Password -

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

 - Enter the Administrator password

```
ubuntu@ip-172-31-47-125:~$ java -version
openjdk version "17.0.12" 2024-07-16
OpenJDK Runtime Environment (build 17.0.12+7-Ubuntu-1ubuntu224.04)
OpenJDK 64-Bit Server VM (build 17.0.12+7-Ubuntu-1ubuntu224.04, mixed mode, sharing)
ubuntu@ip-172-31-47-125:~$ jenkins --version
2.477
ubuntu@ip-172-31-47-125:~$ ps -ef | grep jenkins
jenkins 3958 1 48 06:41 ? 00:00:20 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080
ubuntu 4158 1145 0 06:42 pts/0 00:00:00 grep --color=auto jenkins
```

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

`/var/lib/jenkins/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password

Click on Install suggested plugins

Getting Started



Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

Wait for the Jenkins to Install suggested plugins

Getting Started

Getting Started

✓ Folders	Formatter			Git
✓ Timestamper	✓ Workspace Cleanup	✓ Ant	✓ Gradle	** GitHub
✓ Pipeline	✓ GitHub Branch Source	✓ Pipeline: GitHub Groovy Libraries	✓ Pipeline: Stage View	GitHub Branch Source
✓ Git	✓ SSH Build Agents	✓ Matrix Authorization Strategy	✓ PAM Authentication	Pipeline: GitHub Groovy Libraries
✓ LDAP	✓ Email Extension	✓ Mailer		** Pipeline: Graph Analysis
				** Pipeline: REST API
				Pipeline: Stage View
				Git
				SSH Build Agents
				Matrix Authorization Strategy
				PAM Authentication
				LDAP
				Email Extension
				Mailer

Create First Admin User or Skip the step [If you want to use this Jenkins instance for future use-cases as well, better to create admin user]

Getting Started

Create First Admin User

Username

Password

Confirm password

Full name

E-mail address

Jenkins Installation is Successful. You can now starting using the Jenkins

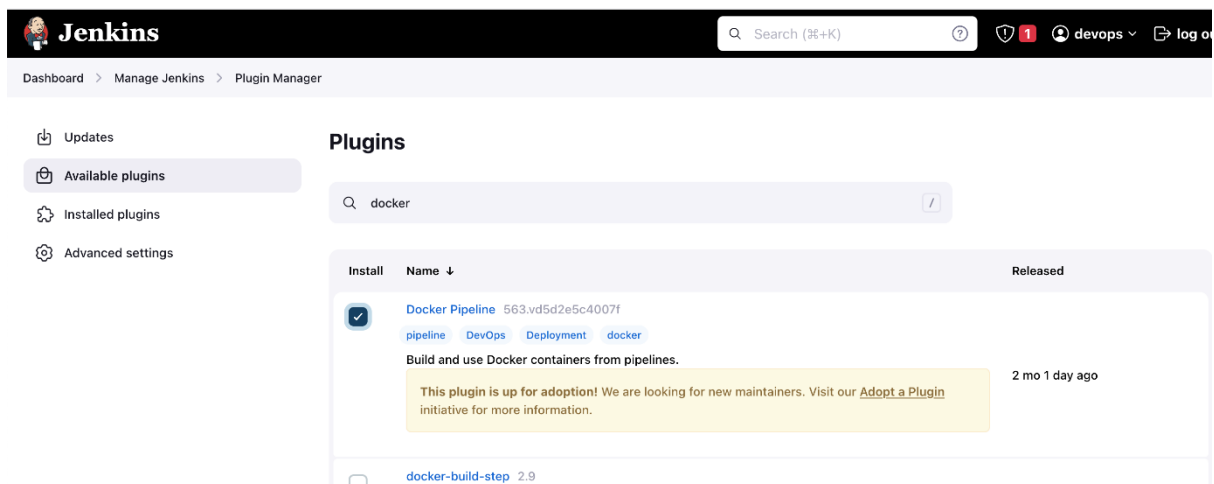
Jenkins is ready!

Your Jenkins setup is complete.

Start using Jenkins

Install the Docker Pipeline plugin in Jenkins:

- Log in to Jenkins.
- Go to Manage Jenkins > Manage Plugins.
- In the Available tab, search for "Docker Pipeline".
- Select the plugin and click the Install button.
- Restart Jenkins after the plugin is installed.



The screenshot shows the Jenkins web interface. At the top, there's a navigation bar with the Jenkins logo, a search bar, and user information. Below the navigation bar, the breadcrumb trail reads "Dashboard > Manage Jenkins > Plugin Manager". On the left sidebar, there are links for "Updates", "Available plugins" (which is highlighted), "Installed plugins", and "Advanced settings". The main content area is titled "Plugins" and contains a search bar with "docker" entered. Below the search bar, there's a table of plugins. The first plugin listed is "Docker Pipeline" with version "563.vd5d2e5c4007f". It has tags for "pipeline", "DevOps", "Deployment", and "docker". The description says "Build and use Docker containers from pipelines." There's a yellow warning box stating "This plugin is up for adoption! We are looking for new maintainers. Visit our [Adopt a Plugin](#) initiative for more information." The release date is "2 mo 1 day ago". Below this, the "docker-build-step" plugin version "2.9" is partially visible.

Wait for the Jenkins to be restarted.

Docker Slave Configuration

Run the below command to Install Docker

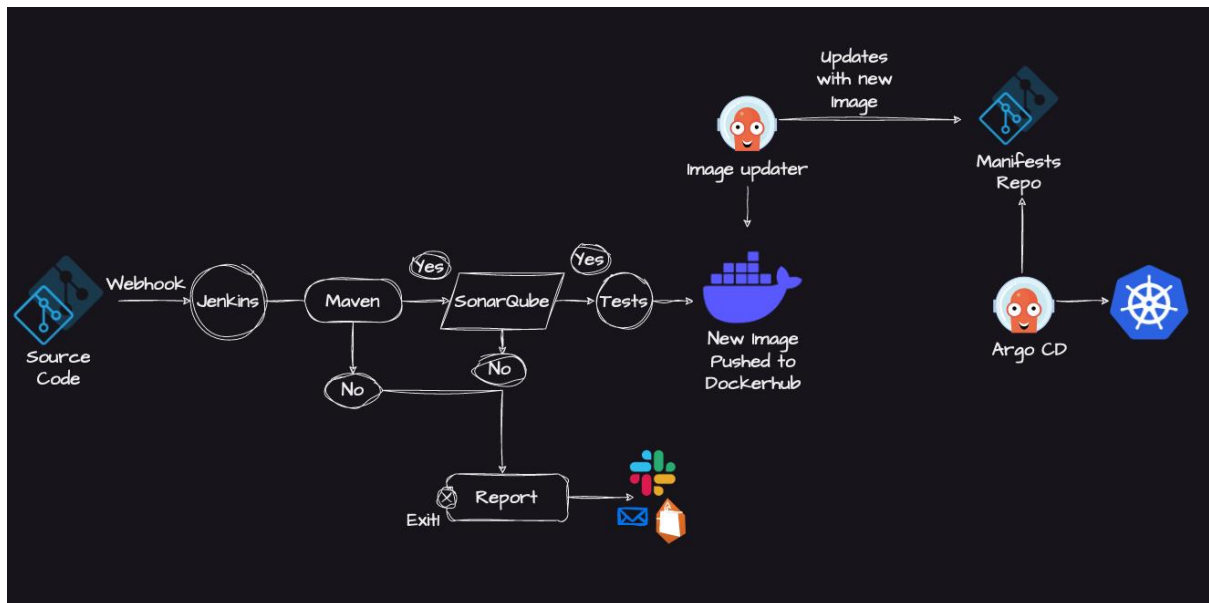
- `sudo apt update`

- `sudo apt install docker.io`

Grant Jenkins user and Ubuntu user permission to docker daemon.

- `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://<ec2-instance-public-ip>:8080/restart>
- The docker agent configuration is now successful.

Jenkins Pipeline for Java based application using Maven, SonarQube, Argo CD, Helm and Kubernetes



Here are the step-by-step details to set up an end-to-end Jenkins pipeline for a Java application using SonarQube, Argo CD, Helm, and Kubernetes:

Prerequisites:

- Java application code hosted on a Git repository
- Jenkins server
- Kubernetes cluster
- Argo CD

Steps:

1. Install the necessary Jenkins plugins:

1.1 Git plugin

1.2 Maven Integration plugin

1.3 Pipeline plugin

1.4 Kubernetes Continuous Deploy plugin

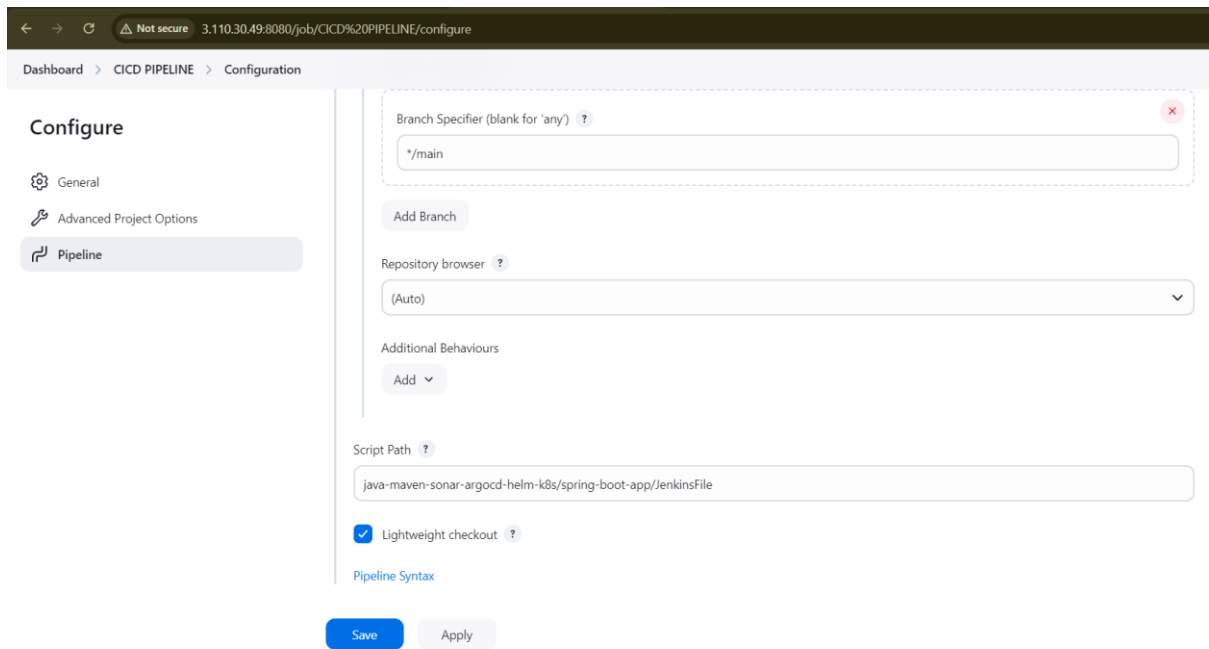
2. Create a new Jenkins pipeline:

2.1 In Jenkins, create a new pipeline job and configure it with the Git repository URL for the Java application.

2.2 Add a Jenkinsfile to the Git repository to define the pipeline stages.

The image shows two screenshots of the Jenkins web interface. The top screenshot is the 'New Item' page. It has a header with the Jenkins logo, a search bar, and a user profile 'akhil'. The breadcrumb is 'Dashboard > All > New Item'. The main section is titled 'New Item'. It asks to 'Enter an item name' with a text box containing 'CICD PIPELINE'. Below that, it asks to 'Select an item type' and lists four options: 'Freestyle project' (Classic, general-purpose job type), 'Pipeline' (Orchestrates long-running activities), 'Multi-configuration project' (Suitable for projects that need a large number of different configurations), and 'Folder' (Creates a container that stores nested items). The 'Pipeline' option is highlighted. At the bottom is an 'OK' button.

The bottom screenshot is the 'Configure' page for the 'CICD PIPELINE' job. The breadcrumb is 'Dashboard > CICD PIPELINE > Configuration'. On the left, there's a 'Configure' sidebar with 'General', 'Advanced Project Options', and 'Pipeline' (selected). The main area is titled 'Pipeline'. It has a 'Definition' section with a dropdown menu set to 'Pipeline script from SCM'. Below that is an 'SCM' dropdown set to 'Git'. Under 'Repositories', there's a 'Repository URL' field with the value 'https://github.com/iam-veeramalla/jenkins-Zero-To-Hero/'. There's also a 'Credentials' dropdown set to '- none -' and an '+ Add' button. At the bottom are 'Save' and 'Apply' buttons.



Dashboard > CICD PIPELINE > Configuration

Configure

- General
- Advanced Project Options
- Pipeline**

Branch Specifier (blank for 'any') ?

*/main

Add Branch

Repository browser ?

(Auto)

Additional Behaviours

Add

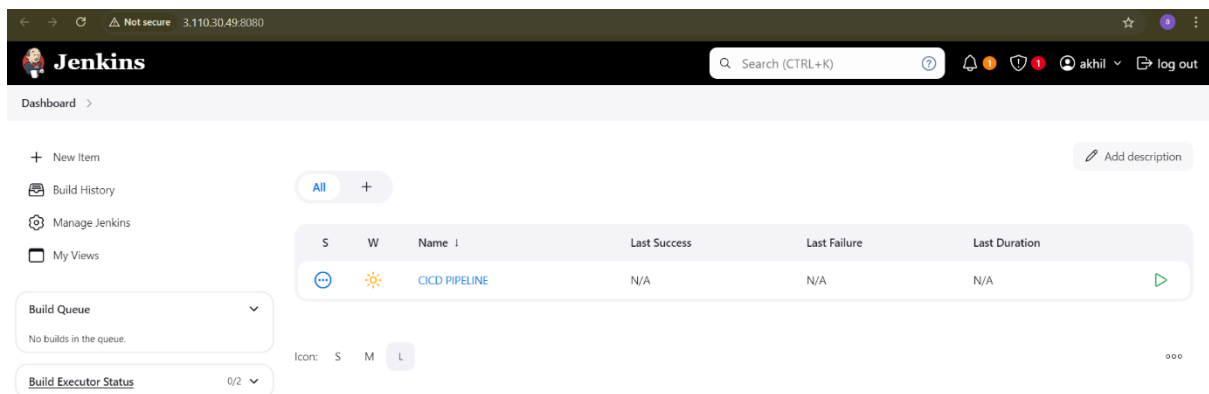
Script Path ?

java-maven-sonar-argocd-helm-k8s/spring-boot-app/JenkinsFile

☒ Lightweight checkout ?

[Pipeline Syntax](#)

Save Apply



Dashboard >

+ New Item

Build History

Manage Jenkins

My Views

Build Queue

No builds in the queue.

Build Executor Status

0/2

Search (CTRL+K)

akhil

log out

Add description

S	W	Name	Last Success	Last Failure	Last Duration
🔄	☀️	CICD PIPELINE	N/A	N/A	N/A

Icons: S M L

3. Define the pipeline stages:

Stage 1: Checkout the source code from Git.

Stage 2: Build the Java application using Maven.

Stage 3: Run SonarQube analysis to check the code quality.

Stage 4: Package the application into a JAR file.

Stage 5: Deploy the application to a test environment using Helm.

Stage 6: Run user acceptance tests on the deployed application.

Stage 7: Promote the application to a production environment using Argo CD.

INTERVIEW QUESTIONS:

1. Can JenkinsFile can be in any location?

a. Yes, It can be in any location.

2. JenkinsFile can have any different names?

a. Obviously can have a different name.

Yes Jenkins file can be in any location and can have any name.

3. How to create a Jenkins file and what is its purpose?

a. In Jenkins we can write jenkinsfile from jobs of pipeline or we can import it using GitHub.

The purpose of it is to execute the steps of Continuous Integration.

4. Difference between add user and user add?

a. In summary, adduser is more interactive and easier for general use, while useradd is more flexible but requires more manual steps.

This is a **low-level command** for adding users.

This is a **higher-level script** that is more user-friendly and often provides defaults for user creation.

Maven:

5. Difference between mvn clean package and mvn clean install?

a. If you want to push your enterprise archive, java archive, eb archive to the artifactory / nexus in such cases we can use mvn clean install.

In my case i dont need to push to any artifactory i need to push the image to docker registry so i used mvn clean package.

Basically we have pom.xml in our projects directory, it is responsible for getting the dependencies runtime and building the application.

Here using maven our application will complete the process of build, by this artifacts will be generated into jar/war files.

We can see the target folder where the archive of web s present. In DockerFile we may use jar file by copying it.

Static Code Analysis:

Here we need to configure the URL of our SonarQube configured with public ip and port. And also mention credentials and token. Here we use another maven target.

Docker:

Here we build and push the image by using the docker registry credentials.

Update Deployment File:

Here we can use argocd updater/shell script, we are using a shell script file.

We are using Docker as agent for containers, Jenkins uses master-slave architecture instead of docker we can have used ec2 instances as worker nodes. But these instances should run for a long time. We may get more cost. And also the configuration will need to be done on all the instances.

The efficient way is to use docker as agent where we include all of our stages in pipeline that will containerize and run.

Once all the stages are passed the container will be deleted by the pipeline. So that the resources can be used by another stages.

- * Install Docker Pipeline Plugin.

- Docker Pipeline

- * Install SonarQube plugin and also install SonarQube severit on ec2 instance

- SonarQube Scanner

Configure a Sonar Server:

```
apt install unzip
```

```
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
```

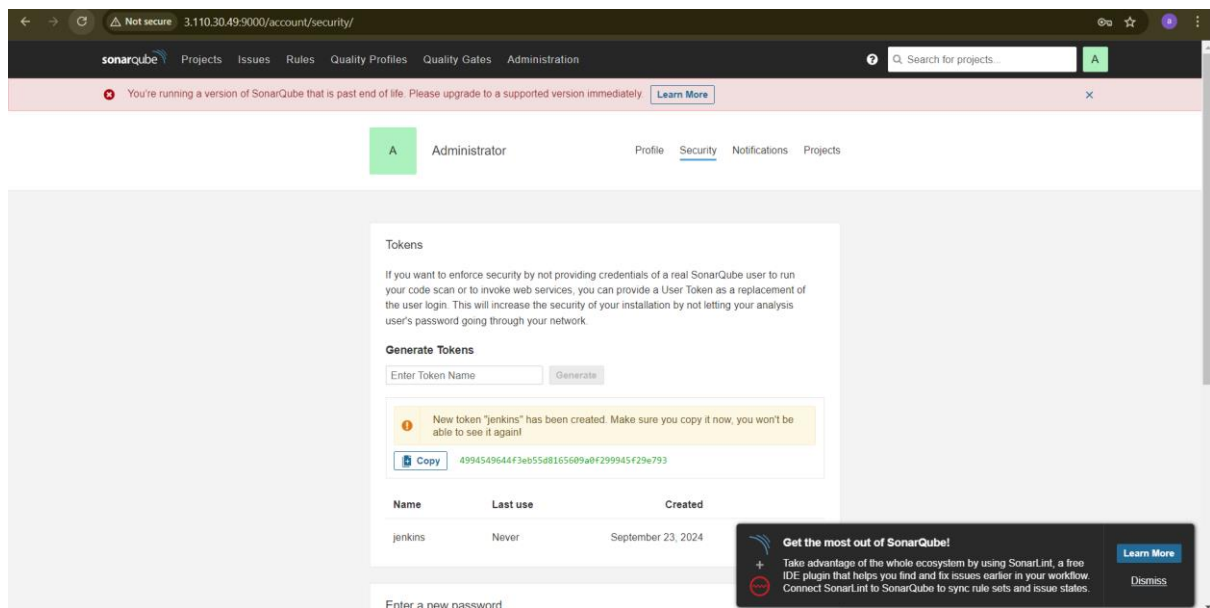
```
root@ip-172-31-47-125:/home/ubuntu# adduser sonarqube
info: Adding user `sonarqube' ...
info: Selecting UID/GID from range 1000 to 59999 ...
info: Adding new group `sonarqube' (1001) ...
info: Adding new user `sonarqube' (1001) with group `sonarqube (1001)' ...
info: Creating home directory `/home/sonarqube' ...
info: Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for sonarqube
Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] y
info: Adding new user `sonarqube' to supplemental / extra groups `users' ...
info: Adding user `sonarqube' to group `users' ...
root@ip-172-31-47-125:/home/ubuntu#
```

```

sonarqube@ip-172-31-47-125:~$ ls
sonarqube-9.4.0.54424 sonarqube-9.4.0.54424.zip
sonarqube@ip-172-31-47-125:~$ chmod -R 755 /home/sonarqube/sonarqube-9.4.0.54424
sonarqube@ip-172-31-47-125:~$ chown -R sonarqube:sonarqube /home/sonarqube/sonarqube-9.4.0.54424
sonarqube@ip-172-31-47-125:~$ cd sonarqube-9.4.0.54424/bin/linux-x86-64/
sonarqube@ip-172-31-47-125:~/sonarqube-9.4.0.54424/bin/linux-x86-64$ ls
lib sonar.sh wrapper
sonarqube@ip-172-31-47-125:~/sonarqube-9.4.0.54424/bin/linux-x86-64$ ./sonar.sh
Usage: ./sonar.sh { console | start | stop | force-stop | restart | status | dump }
sonarqube@ip-172-31-47-125:~/sonarqube-9.4.0.54424/bin/linux-x86-64$ ^C
sonarqube@ip-172-31-47-125:~/sonarqube-9.4.0.54424/bin/linux-x86-64$ ./sonar.sh start
Starting SonarQube...
Started SonarQube.
sonarqube@ip-172-31-47-125:~/sonarqube-9.4.0.54424/bin/linux-x86-64$ |

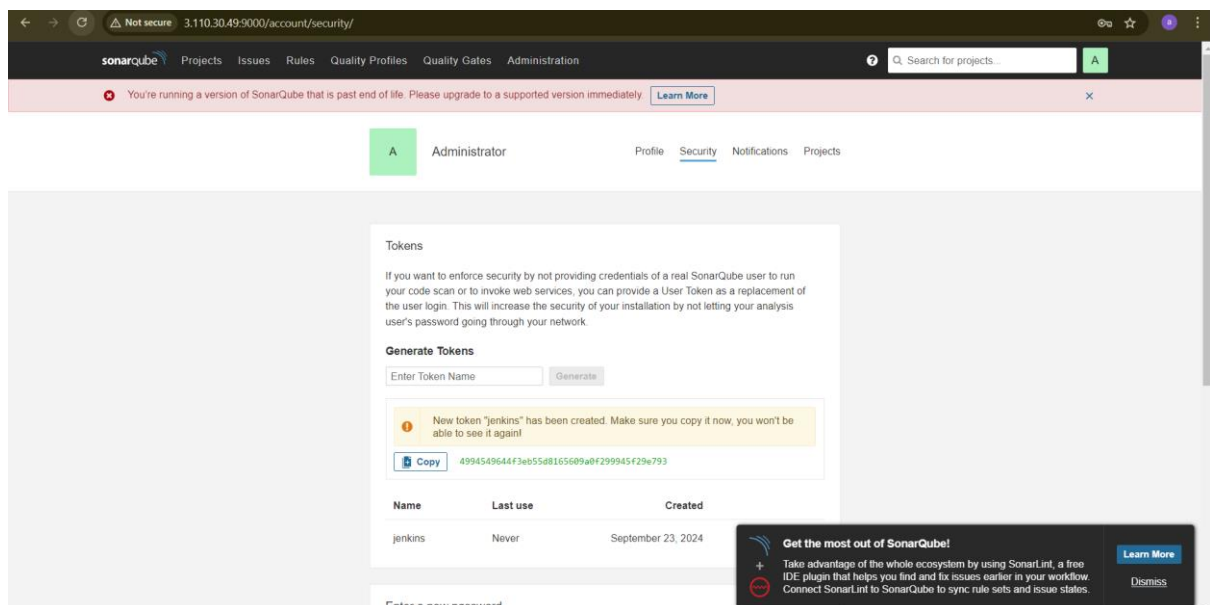
```

We can launch SonarQube using 3.110.30.49:9000. In login admin as username and password. And change the password to 12345.



Maven need not to install on instance because it is already integrated with docker container.

To make connection between Jenkins and SonarQube, SonarQube >> My account >> security >> generate token – Jenkins



Now go to manage Jenkins >> Credentials >> System >> Global >> Secret text >> paste and save it.

```
sudo su -
```

```
exit
```

Install Docker On ec2 instance.

```
sudo apt update
```

```
sudo apt install docker.io
```

Grant Jenkins user and Ubuntu user permission to docker deamon.

```
sudo su -
```

```
usermod -aG docker jenkins
```

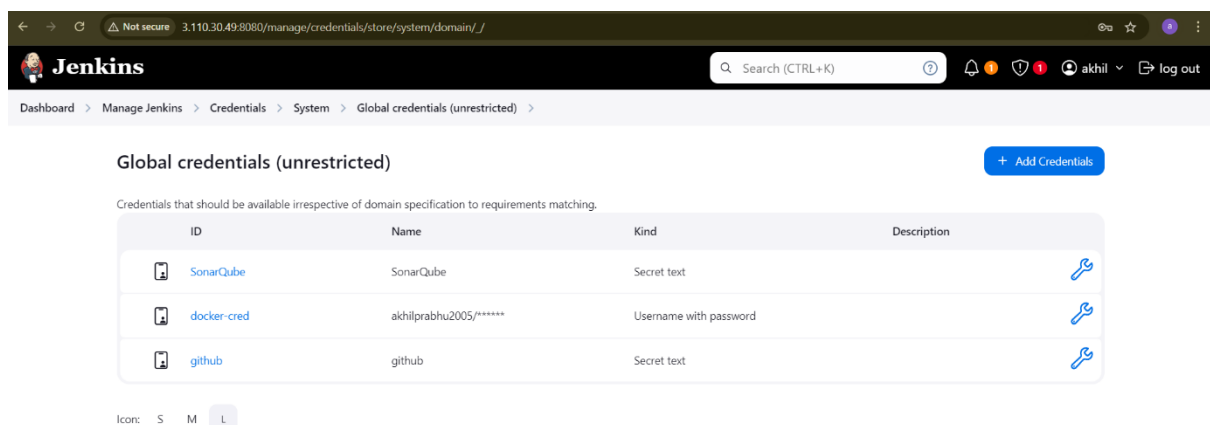
```
usermod -aG docker ubuntu
```

```
systemctl restart docker
```

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

<http://<ec2-instance-public-ip>:8080/restart>

The docker agent configuration is now successful.



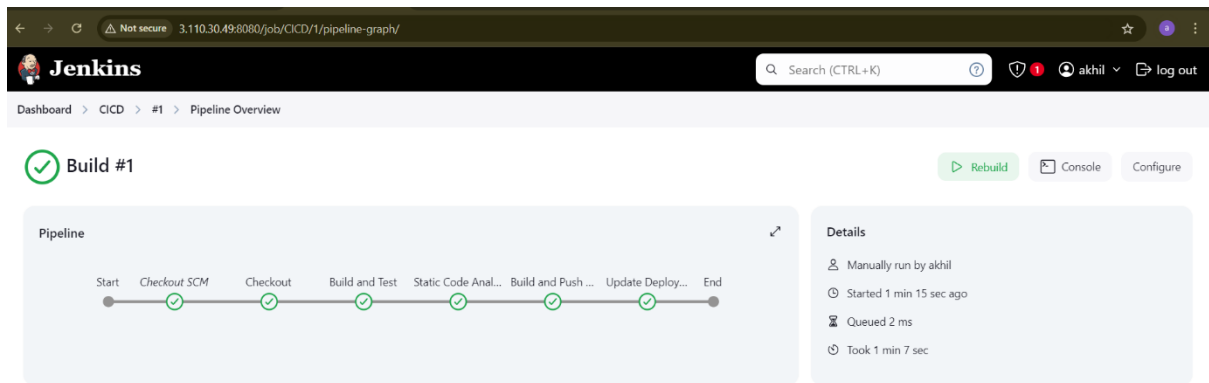
4. Configure Jenkins pipeline stages:

Stage 1: Use the Git plugin to check out the source code from the Git repository.

Stage 2: Use the Maven Integration plugin to build the Java application.

Stage 3: Use the SonarQube plugin to analyze the code quality of the Java application.

Stage 4: Use the Maven Integration plugin to package the application into a JAR file



Stage 5: Use the Kubernetes Continuous Deploy plugin to deploy the application to a test environment using Minikube and ArgoCD.

Stage 7: Use Argo CD to promote the application to a production environment.

```
commit: 210b148df93a80eb872ecbeb7e35281b3c582c61
akhil@LAPTOP-ARIT216A:~$ minikube start --driver=docker --memory=2200mb --cpus=2
🐳 minikube v1.34.0 on Ubuntu 22.04 (amd64)
🔧 Using the docker driver based on user configuration

⚠️ The requested memory allocation of 2200MiB does not leave room for system overhead (total system memory: 2934MiB). You may face stability issues.
💡 Suggestion: Start minikube with less memory allocated: 'minikube start --memory=2200mb'

🚀 Using Docker driver with root privileges
🔥 Starting "minikube" primary control-plane node in "minikube" cluster
📡 Pulling base image v0.0.45 ...
📦 Downloading Kubernetes v1.31.0 preload ...
> preloaded-images-k8s-v18-v1...: 326.69 MiB / 326.69 MiB 100.00% 4.49 Mi
> gcr.io/k8s-minikube/kicbase...: 487.90 MiB / 487.90 MiB 100.00% 2.85 Mi
🔥 Creating docker container (CPUs=2, Memory=2200MB) ...
📦 Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
  ▪ 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
akhil@LAPTOP-ARIT216A:~$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

akhil@LAPTOP-ARIT216A:~$ kubectl get nodes
NAME       STATUS   ROLES    AGE   VERSION
minikube   Ready    control-plane   24s   v1.31.0
akhil@LAPTOP-ARIT216A:~$
```

minikube start --memory=2200mb --driver=docker

5. Set up Argo CD:

Install Argo CD on the Kubernetes cluster.

Set up a Git repository for Argo CD to track the changes in the Kubernetes manifests.

Argo CD
0.12.0 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/v0.28.0/install.sh | bash -s v0.28.0
```

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.

```
akhil@LAPTOP-ARIT216A:~$ kubectl get pods -n operators -w
^C
akhil@LAPTOP-ARIT216A:~$ kubectl get csv -n operators
NAME          DISPLAY          VERSION  REPLACES          PHASE
argocd-operator.v0.12.0  Argo CD  0.12.0   argocd-operator.v0.11.0  Installing
akhil@LAPTOP-ARIT216A:~$ kubectl get csv -n operators
NAME          DISPLAY          VERSION  REPLACES          PHASE
argocd-operator.v0.12.0  Argo CD  0.12.0   argocd-operator.v0.11.0  Succeeded
akhil@LAPTOP-ARIT216A:~$ ls
Deployment      admin.sh          cronlog           inventory.ini      openshift-client-linux.tar.gz  pullfail.sh  terraform
Docker-Zero-to-Hero  ansible-prac.pem  dead.letter       kubectl           professional-scripts  pullwarn.sh
README.md       cpu_usage.log     file.sh           minikube-linux-amd64  pullerror.sh          snap
```

vim argocd-basic.yml

apiVersion: argoproj.io/v1alpha1

kind: ArgoCD

metadata:

name: example-argocd

labels:

example: basic

spec: {}

```
akhil@LAPTOP-ARIT216A:~$ vim argocd-basic.yml
akhil@LAPTOP-ARIT216A:~$ kubectl apply -f argocd-basic.yml
Warning: ArgoCD v1alpha1 version is deprecated and will be converted to v1beta1 automatically. Moving forward, please use v1beta1 as the ArgoCD API version.
argocd.argoproj.io/example-argocd created
akhil@LAPTOP-ARIT216A:~$ kubectl get pods
NAME                                READY   STATUS             RESTARTS   AGE
example-argocd-application-controller-0  0/1     ContainerCreating   0           28s
example-argocd-repo-server-869d5757c7-s9kt8  0/1     Init:0/1            0           29s
example-argocd-server-76bb84cddc-7tt9j  0/1     ContainerCreating   0           29s
akhil@LAPTOP-ARIT216A:~$ kubectl get svc
NAME                                TYPE               CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
example-argocd-metrics              ClusterIP          10.98.14.87      <none>            8082/TCP          112s
example-argocd-redis                 ClusterIP          10.110.106.151   <none>            6379/TCP          112s
example-argocd-repo-server           ClusterIP          10.109.12.57     <none>            8081/TCP, 8084/TCP 112s
example-argocd-server                ClusterIP          10.104.242.29    <none>            80/TCP, 443/TCP   112s
example-argocd-server-metrics        ClusterIP          10.105.158.37    <none>            8083/TCP          112s
kubernetes                           ClusterIP          10.96.0.1        <none>            443/TCP           9h
akhil@LAPTOP-ARIT216A:~$ kubectl edit svc example-argocd-server
service/example-argocd-server edited
akhil@LAPTOP-ARIT216A:~$ kubectl get svc
NAME                                TYPE               CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
example-argocd-metrics              ClusterIP          10.98.14.87      <none>            8082/TCP          3m23s
example-argocd-redis                 ClusterIP          10.110.106.151   <none>            6379/TCP          3m23s
example-argocd-repo-server           ClusterIP          10.109.12.57     <none>            8081/TCP, 8084/TCP 3m23s
example-argocd-server                NodePort           10.104.242.29    <none>            80:32098/TCP, 443:31599/TCP 3m23s
example-argocd-server-metrics        ClusterIP          10.105.158.37    <none>            8083/TCP          3m23s
kubernetes                           ClusterIP          10.96.0.1        <none>            443/TCP           9h
```

kubectl get pods

kubectl get svc

kubectl edit svc example-argocd-server

kubectl get svc

minikube service argocd-server

minikube service example-argocd-server

minikube service list

kubectl port-forward svc/example-argocd-server 8080:80

kubectl port-forward svc/example-argocd-server 8443:443

kubectl get secret

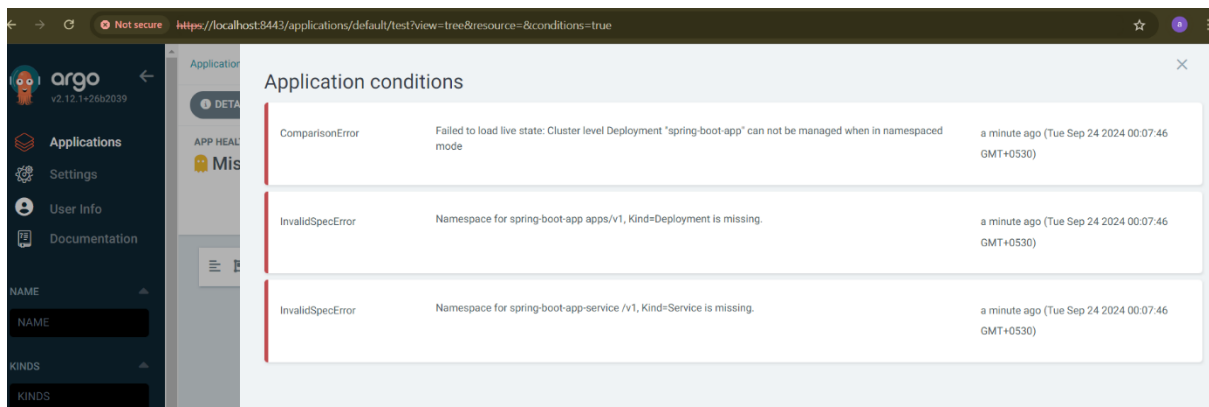
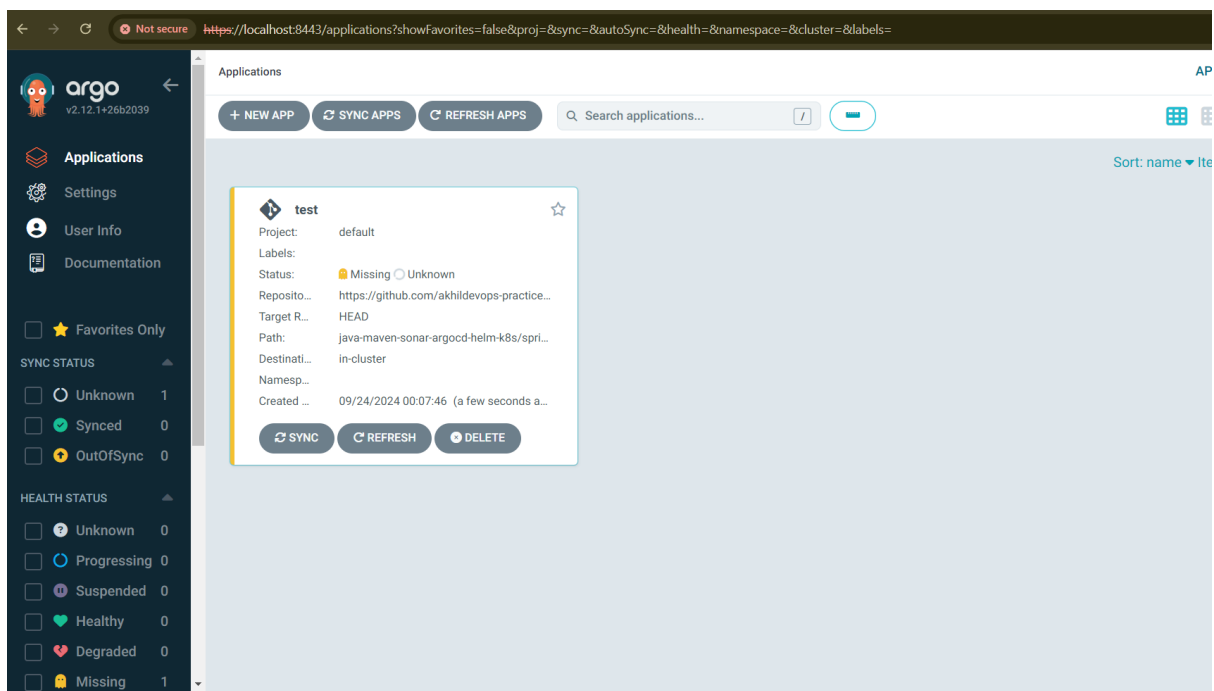
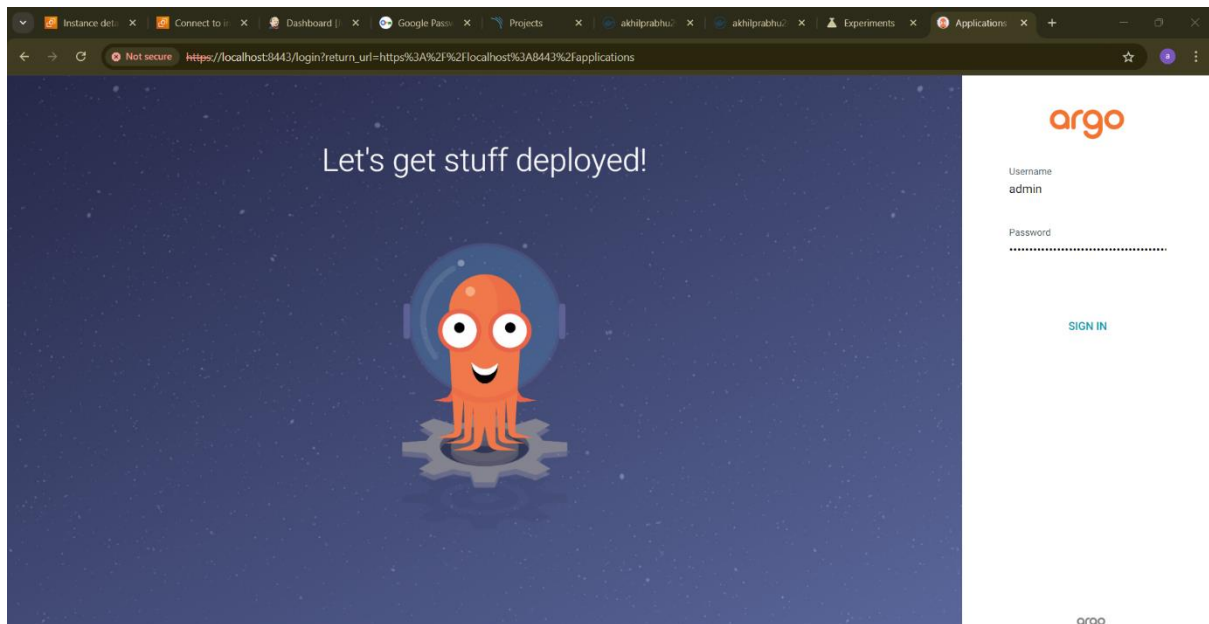
kubectl edit secret example-argocd-cluster

```
root@LAPTOP-ARIT216A: /hoi  X + v
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  admin.password: OFhaNEZyTGxIcFJvOTN5T25UbWFDanZnSTBlUURQZkc=
kind: Secret
metadata:
  creationTimestamp: "2024-09-23T18:05:59Z"
  labels:
    app.kubernetes.io/managed-by: example-argocd
    app.kubernetes.io/name: example-argocd-cluster
    app.kubernetes.io/part-of: argocd
  name: example-argocd-cluster
  namespace: default
  ownerReferences:
    - apiVersion: argoproj.io/v1beta1
      blockOwnerDeletion: true
      controller: true
      kind: ArgoCD
      name: example-argocd
      uid: a78a1e7d-02a4-4f91-9d80-99966e627931
  resourceVersion: "29891"
  uid: 1a540214-d0a3-40a8-b505-8a0671220246
type: Opaque
```

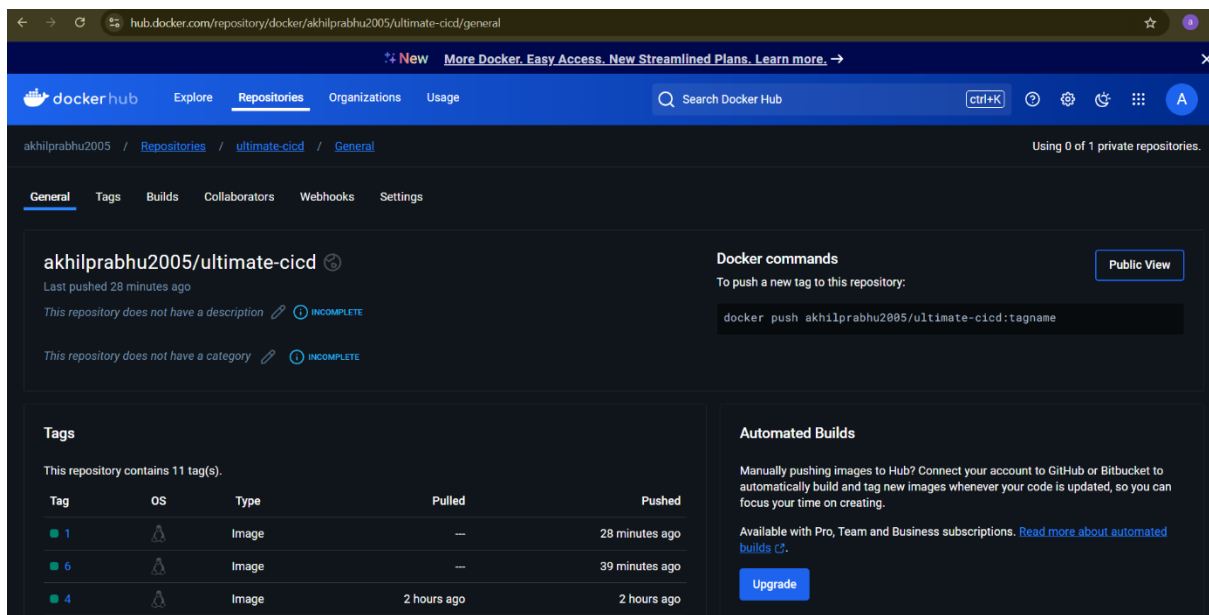
echo OFhaNEZyTGxIcFJvOTN5T25UbWFDanZnSTBlUURQZkc= | base64 -d

```
akhil@LAPTOP-ARIT216A:~$ minikube service example-argocd-server
-----
| NAMESPACE | NAME | TARGET PORT | URL |
-----
| default | example-argocd-server | http/80 | http://192.168.49.2:32098 |
| | | https/443 | http://192.168.49.2:31599 |
-----
[default example-argocd-server http/80
https/443 http://192.168.49.2:32098
http://192.168.49.2:31599]
akhil@LAPTOP-ARIT216A:~$ minikube service list
-----
| NAMESPACE | NAME | TARGET PORT | URL |
-----
| default | example-argocd-metrics | No node port | |
| default | example-argocd-redis | No node port | |
| default | example-argocd-repo-server | No node port | |
| default | example-argocd-server | http/80 | http://192.168.49.2:32098 |
| | | https/443 | http://192.168.49.2:31599 |
| default | example-argocd-server-metrics | No node port | |
| default | kubernetes | No node port | |
| kube-system | kube-dns | No node port | |
| olm | operatorhubio-catalog | No node port | |
| olm | packageserver-service | No node port | |
| operators | argocd-operator-controller-manager-metrics-service | No node port | |
| operators | argocd-operator-controller-manager-service | No node port | |
| operators | argocd-operator-webhook-service | No node port | |
-----
akhil@LAPTOP-ARIT216A:~$ kubectl get pods
NAME READY STATUS RESTARTS AGE
example-argocd-application-controller-0 1/1 Running 0 5m41s
example-argocd-redis-6545fd6d6c-6pklc 1/1 Running 0 5m42s
example-argocd-repo-server-869d5757c7-s9kt8 1/1 Running 0 5m42s
example-argocd-server-76bb84cddc-7tt9j 1/1 Running 0 5m42s
```

kubectl port-forward svc/example-argocd-server 8443:443



Mention the namespace as default.



```
ubuntu@ip-172-31-47-125: ~
Swap usage: 0%

* Ubuntu Pro delivers the most comprehensive open source security and
compliance features.

https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

143 updates can be applied immediately.
41 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Mon Sep 23 15:54:17 2024 from 45.112.202.181
ubuntu@ip-172-31-47-125:~$ git config --global user.name "akhilprabhu20"
ubuntu@ip-172-31-47-125:~$ git config --global user.email "akhilprabhu20@gmail.com"
ubuntu@ip-172-31-47-125:~$ docker images
REPOSITORY                                TAG                IMAGE ID           CREATED           SIZE
akhilprabhu2005/ultimate-cicd            1                 3676d980495f      28 minutes ago   170MB
akhilprabhu2005/ultimate-cicd            6                 78e5fbbf4119      39 minutes ago   170MB
akhilprabhu2005/ultimate-cicd            4                 d9ea5569e9ae      2 hours ago      170MB
akhilprabhu2005/ultimate-cicd            3                 eea3e415674b      2 hours ago      170MB
akhilprabhu2005/ultimate-cicd            2                 06729813370f      3 hours ago      170MB
akhilprabhu2005/ultimate-cicd            <none>            80a07b35a9bc      3 hours ago      170MB
akhilprabhu2005/ultimate-cicd            12                2e0de2377060      3 hours ago      170MB
akhilprabhu2005/ultimate-cicd            11                e67445e7ba2c      3 hours ago      170MB
akhilprabhu2005/ultimate-cicd            10                dd2b8601b593      3 hours ago      170MB
akhilprabhu2005/ultimate-cicd            9                 106a81322c3c      3 hours ago      170MB
akhilprabhu2005/ultimate-cicd            8                 3f6ab4edf042      4 hours ago      170MB
akhilprabhu2005/ultimate-cicd            7                 0e0e3f047f9d      4 hours ago      170MB
abhishekf5/ultimate-cicd                 6                 b75ffd354682      4 hours ago      170MB
abhishekf5/ultimate-cicd                 5                 ba6609488271      4 hours ago      170MB
abhishekf5/ultimate-cicd                 4                 c8d790181d1d      4 hours ago      170MB
abhishekf5/ultimate-cicd                 3                 595c67ddcb81      5 hours ago      170MB
abhishekf5/ultimate-cicd                 1                 fb643cfffca20      5 hours ago      170MB
abhishekf5/maven-abhishek-docker-agent    v1                 3fb9145e2467      17 months ago    913MB
ubuntu@ip-172-31-47-125:~$
```

kubectl get deploy

kubectl get pods

6. Configure Jenkins pipeline to integrate with Argo CD:

6.1 Add the Argo CD API token to Jenkins credentials.

6.2 Update the Jenkins pipeline to include the Argo CD deployment stage.

7. Run the Jenkins pipeline:

7.1 Trigger the Jenkins pipeline to start the CI/CD process for the Java application.

7.2 Monitor the pipeline stages and fix any issues that arise.

This end-to-end Jenkins pipeline will automate the entire CI/CD process for a Java application, from code checkout to production deployment, using popular tools like SonarQube, Argo CD, Helm, and Kubernetes.

- Errors:

1. Got an Issue of at the stage of SonarQube url?

a. I have changed the url based on my ip address, committed and pushed. And i also added the SonarQube security issue by adding the credentials of SonarQube at credentials by adding token and all.

2. Got an issue of docker image at build and push docker image?

a. Solved it by changing the docker registry of my account to it. Because we are using our docker credentials.

3. Got an issue at last deployment stage?

a. Solved it by configuring my GitHub username and email in ec2 instance.

Forked the repository and updated my credentials in Jenkins File and deleted the present token and created fine grained token with all the permissions.

4. After configuring argocd we need to host it in browser but it is not worked?

a. First made a port forwarding to http and https, to local host.

Add an Exception for the Self-Signed Certificate

If you must use HTTPS, you can add an exception for the self-signed certificate. In Chrome:

Go to `chrome://flags/#allow-insecure-localhost`.

Set Allow invalid certificates for resources loaded from localhost to Enabled.