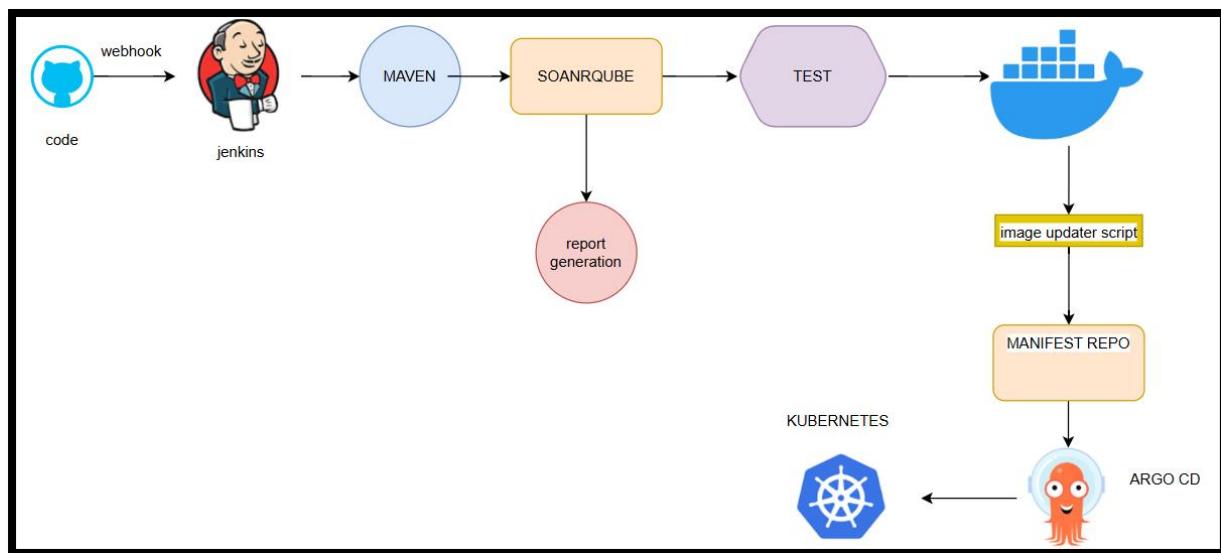


CI/CD PROJECT



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

LET'S START

Create an EC2 instance.

The screenshot shows the 'Instance type' section of the AWS CloudFormation template editor. A dropdown menu is open, showing the 't2.large' option selected. The 't2' family is highlighted. The 'Info' and 'Get advice' buttons are visible at the top of the dropdown. To the right, there is a toggle switch for 'All generations' and a link to 'Compare instance types'. Below the dropdown, a note states 'Additional costs apply for AMIs with pre-installed software'.

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

Edit inbound rules Info

Inbound rules control the incoming traffic that's allowed to reach the instance.

The screenshot shows the 'Edit inbound rules' section of the AWS CloudFront console. It displays two rules:

- Security group rule ID:** sgr-03327b6db02e500df
Type: SSH
Protocol: TCP
Port range: 22
Source: Custom
Description - optional: (empty)
Action: Delete
- Security group rule ID:** (empty)
Type: Custom TCP
Protocol: TCP
Port range: 8080
Source: Anywhere...
Description - optional: (empty)
Action: Delete

An 'Add rule' button is located at the bottom left.

Checking if Jenkins is running in ec2 instance

```
ubuntu@ip-172-31-43-160:~$ ps -ef |grep jenkins
jenkins   4694      1  4 17:43 ?    00:00:15 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080
ubuntu     5874  1438  0 17:49 pts/0  00:00:00 grep --color=auto jenkins
```

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

And perform necessary configurations like

1. Install suggested plugins

The screenshot shows the Jenkins dashboard. At the top, it says "Welcome to Jenkins!" and provides instructions: "This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project." Below this, there are sections for "Start building your software project" and "Set up a distributed build".

- Start building your software project:**
 - Create a job
- Set up a distributed build:**
 - Set up an agent
 - Configure a cloud
 - Learn more about distributed builds

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

New Item

Enter an item name

cicd-pipeline

Select an item type



Freestyle project

Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.



Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.



Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.



Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.



Multibranch Pipeline

Creates a set of Pipeline projects according to detected branches in one SCM repository.



Organization Folder

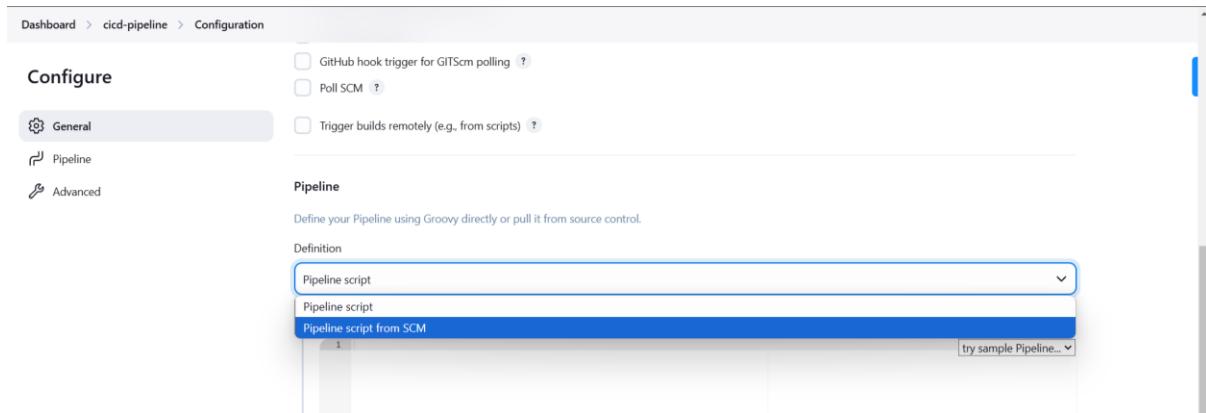
Creates a set of multibranch project subfolders by scanning for repositories.

OK

Here I am using the pipeline option

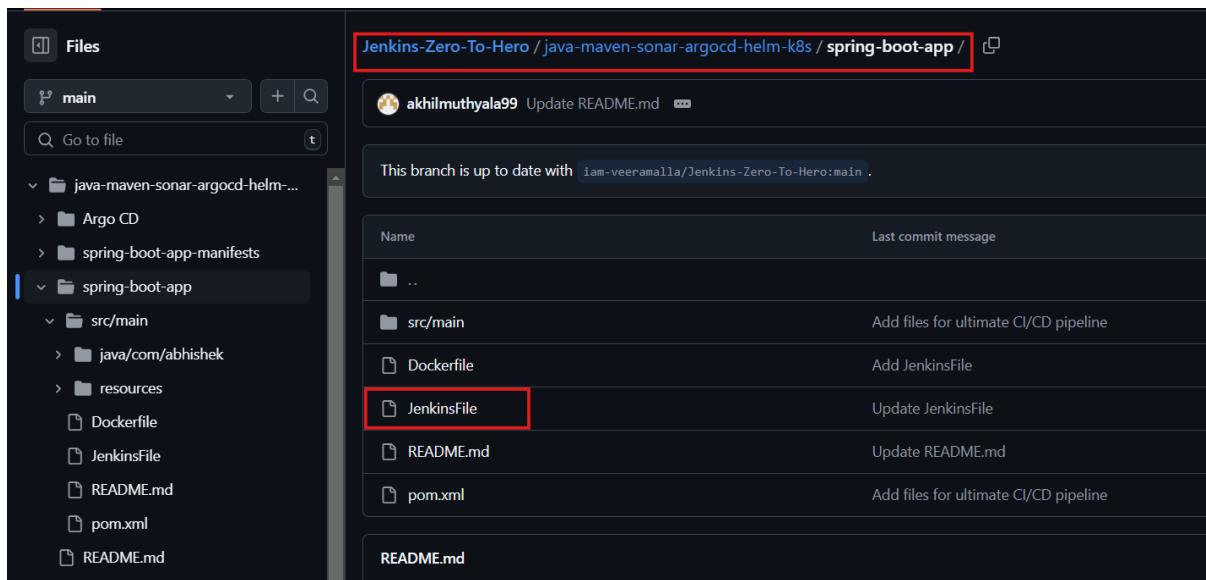
Key Benefits of Jenkins Pipeline:

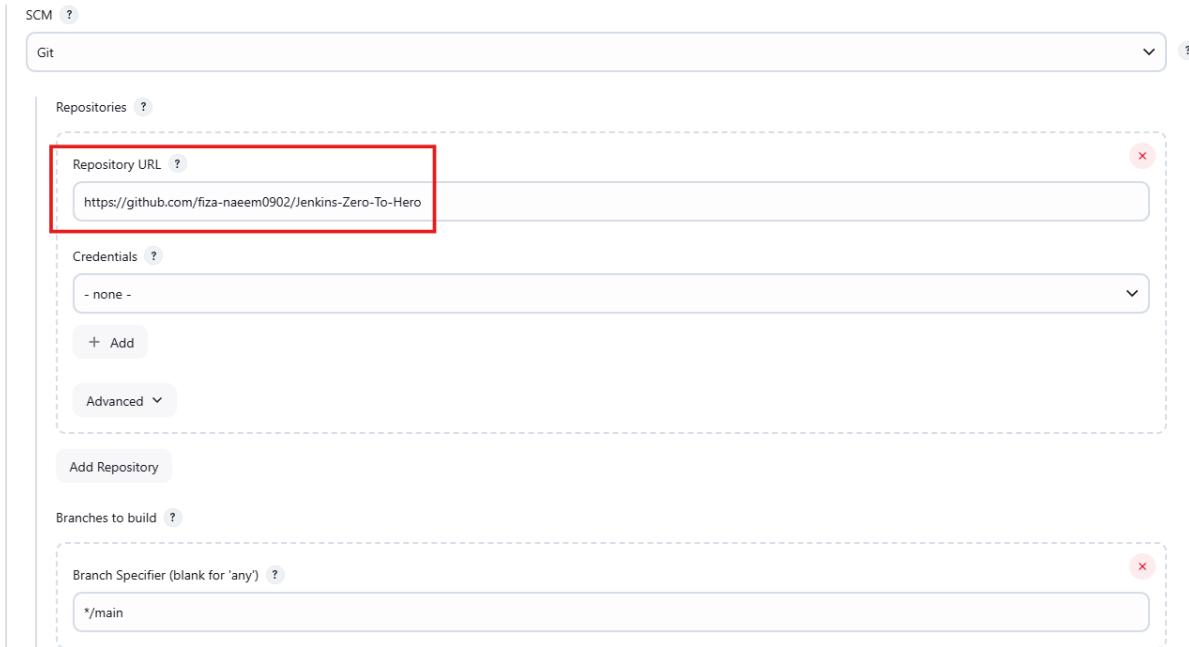
- Pipeline as Code** – Uses a Jenkinsfile (written in Groovy) for version-controlled workflows.
- Automation & Efficiency** – Automates the entire CI/CD process.
- Error Handling & Resilience** – Can retry failed steps, handle exceptions, and improve stability.
- Parallel Execution** – Runs tasks in 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 Jenkins

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

The screenshot shows the Jenkins Manage Jenkins interface. The 'Manage Jenkins' tab is active. On the left, there are several links: 'New Item', 'Build History', 'Manage Jenkins' (which is highlighted with a red box), 'My Views', 'Build Queue' (with a note 'No builds in the queue.'), and 'Build Executor Status'. On the right, there are sections for 'System Configuration' (with 'System', 'Tools', 'Clouds', and 'Appearance'), 'Plugins' (which is also highlighted with a red box; it says 'Add, remove, disable or enable plugins that can extend the functionality of Jenkins.'), and 'Nodes'.

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

The screenshot shows the Jenkins Plugins page. The 'Available plugins' tab is selected. A search bar at the top contains 'sonarqube'. A list of available plugins is shown, with 'SonarQube Scanner 2.18' checked for installation. The plugin details show it was released '1 day 6 hr ago' and integrates with 'SonarQube', an open source platform for Continuous Inspection of code quality. An 'Install' button is visible.

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/
```

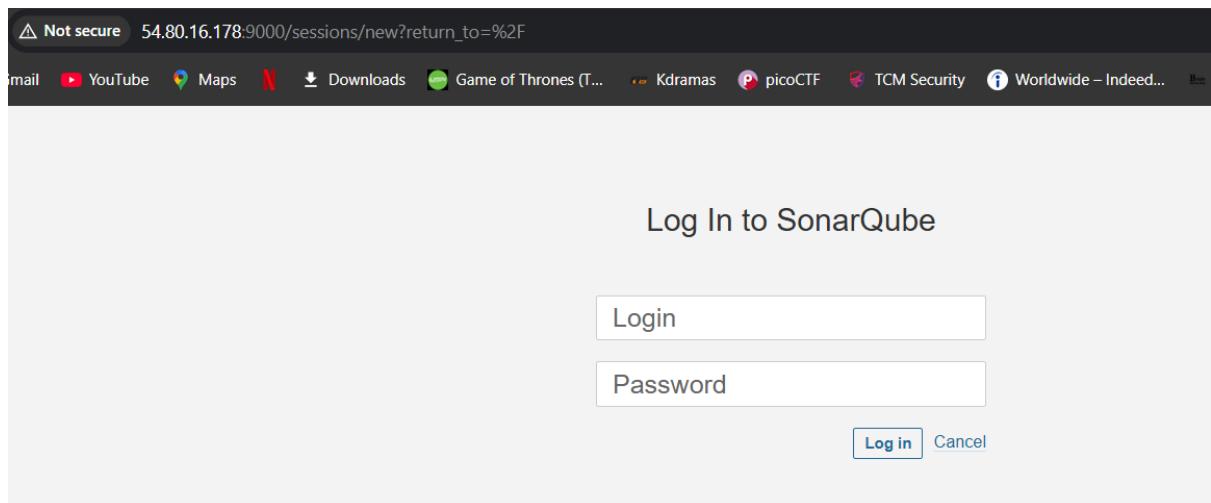
```
./sonar.sh start
```

The screenshot shows the AWS CloudFront Inbound rules table. It lists three rules:

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
-	sgr-03327b6db02e500df	IPv4	SSH	TCP	22	0.0.0.0/0	-
-	sgr-06d58f3c7c1fd429	IPv4	Custom TCP	TCP	9000	0.0.0.0/0	-
-	sgr-05388fd29ddb2726c	IPv4	Custom TCP	TCP	8080	0.0.0.0/0	-

A red box highlights the second row where the port range is set to 9000.

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

The screenshot shows the "Tokens" section under the "Security" tab in the SonarQube user interface. On the left, there's a green profile icon with a white letter "A" and the text "Administrator". To the right, there are tabs for "Profile", "Security" (which is underlined), "Notifications", and "Projects".

The main area is titled "Tokens". It contains a sub-section with a "Generate Tokens" button (which is highlighted with a red border). Below it is an input field labeled "Enter Token Name" and a "Generate" button. A yellow notification box says: "New token \"for jenkins\" has been created. Make sure you copy it now, you won't be able to see it again!". Below this is a "Copy" button with a blue background and white text.

At the bottom, there's a table with columns: "Name", "Last use", and "Created". It lists a single token: "for jenkins" (which is also highlighted with a red border), "Never", and "January 30, 2025". To the right of the last column is a "Revoke" button.

Name	Last use	Created
for jenkins	Never	January 30, 2025

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

New credentials

Kind

Secret text

Scope ?

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

Secret

.....

ID ?

sonarqube

Description ?

for linking sonarqube with jenkins

Create

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 resources 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

A terminal window showing the output of the 'minikube start' command. The process involves pulling a base image, downloading Kubernetes v1.32.0, starting a Docker container, generating certificates, booting up the control plane, and configuring RBAC rules. It also configures the bridge CNI and verifies components like storage-provisioner and default-storageclass. Finally, it indicates that kubectl is configured to use the 'minikube' cluster and 'default' namespace by default. A red box highlights the command 'minikube start --driver=docker'. Another red box highlights the command 'minikube status'. The status output shows the minikube node is a Control Plane, host is Running, kubelet is Running, apiserver is Running, and kubeconfig is Configured.

```
kali@DESKTOP-RK2B14M:~$ minikube start --driver=docker
minikube v1.35.0 on Debian 12.8 (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 v0.0.46 ...
  Downloading Kubernetes v1.32.0 preload ...
    > preloaded-images-k8s-v18-v1...: 333.57 MiB / 333.57 MiB 100.00% 1.20 Mi
    > gcr.io/k8s-minikube/kicbase...: 500.31 MiB / 500.31 MiB 100.00% 1.31 Mi
  Creating docker container (CPUs=2, Memory=2200MB) ...
  Preparing Kubernetes v1.32.0 on Docker 27.4.1 ...
    - 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
kali@DESKTOP-RK2B14M:~$ minikube status
minikube
  type: Control Plane
  host: Running
  kubelet: Running
  apiserver: Running
  kubeconfig: Configured
```

Welcome to OperatorHub.io

OperatorHub.io is a new home for the Kubernetes community to share Operators. Find an existing Operator or list your own today.

3 ITEMS

 Argo CD provided by Argo CD Community Argo CD is a declarative, GitOps continuous delivery	 Argo CD Operator (Helm) provided by Disposable Zone Declarative Continuous Delivery following Gitops.	 Cluster as a service operator provided by Stolostron Easily install fully configured clusters with guard-rails.
---	--	--



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()
            }
        }
    }
}
```

New credentials

Kind

Username with password

Scope ?

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

Username ?

fizza05

Treat username as secret ?

Password ?

.....

ID ?

docker-cred

Description ?

docker hub connection setup

Create

Now generate access token for github and connect github with Jenkins

New credentials

Kind

Secret text

Scope ?

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

Secret

.....

ID ?

GITHUB_TOKEN

Description ?

GITHUB TOKEN

Create

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

The Jenkins Pipeline Overview for Build #10 shows a successful pipeline run. The pipeline stages are: Start, Checkout SCM, Checkout, Build and Test, Static Code Anal., Build and Push..., and Update Deploy... All stages are marked with green checkmarks, indicating success. The 'Details' panel shows the run was manually triggered by admin, started 1 minute 7 seconds ago, queued for 3 ms, and took 41 seconds.

And volla, after about 10 attempts here its done

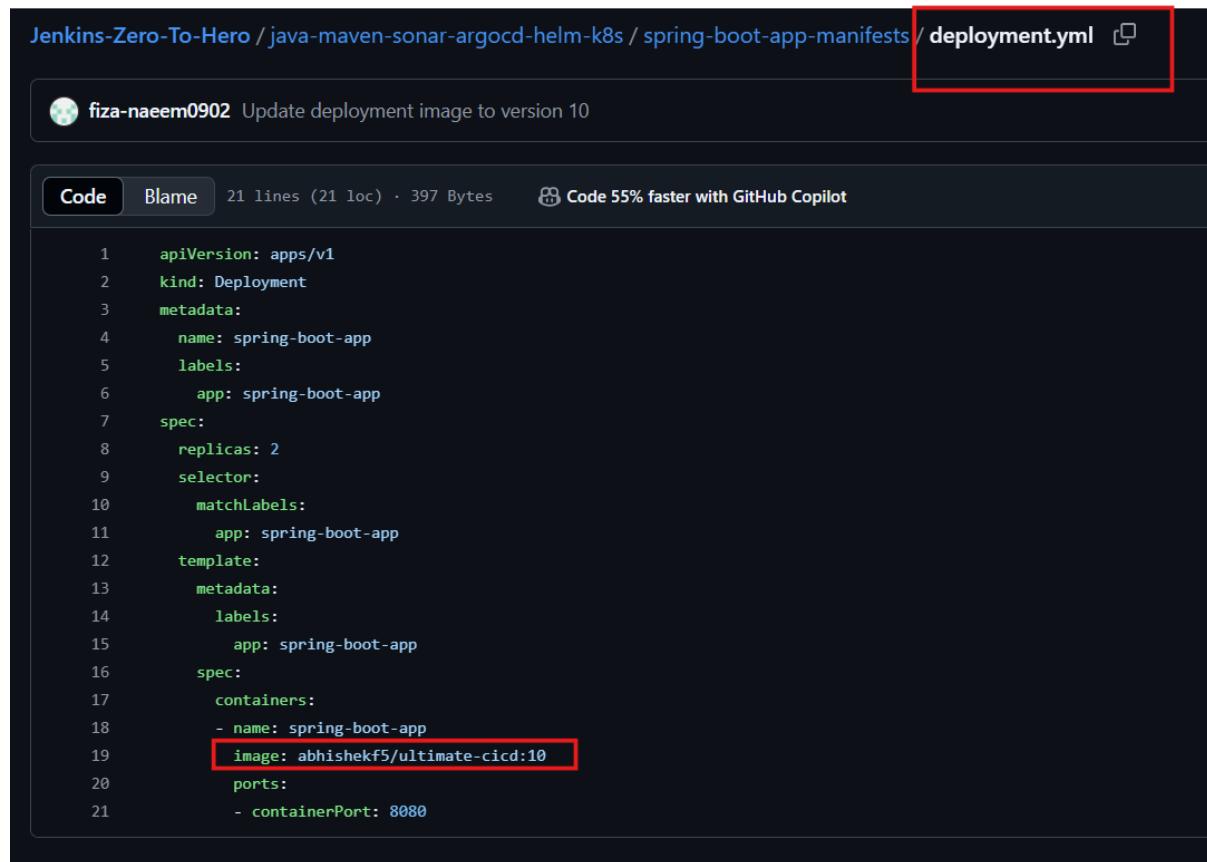
Sonarqube results

The Sonarqube dashboard displays analysis results for the 'spring-boot-demo' project. The project status is 'Passed'. Key metrics shown: Bugs (0), Vulnerabilities (0), Hotspots Reviewed (-), Code Smells (0), Coverage (0.0%), Duplications (0.0%), and Lines (79 XML, Java).

And thus the docker image is created at ec2, dockerhub

```
root@ip-172-31-43-160:/home/ubuntu# docker images
REPOSITORY          TAG      IMAGE ID   CREATED    SIZE
fizza05/ultimate-cicd    10      01654d800be5  8 minutes ago  170MB
fizza05/ultimate-cicd    9       a834063d97cb  12 minutes ago  170MB
abhishekf5/ultimate-cicd  8       4d66d19c5070  17 hours ago  170MB
abhishekf5/maven-abhishek-docker-agent  v1      3fb9145e2467  22 months ago  913MB
fizza05/maven-abhishek-docker-agent    v1      3fb9145e2467  22 months ago  913MB
root@ip-172-31-43-160:/home/ubuntu#
```

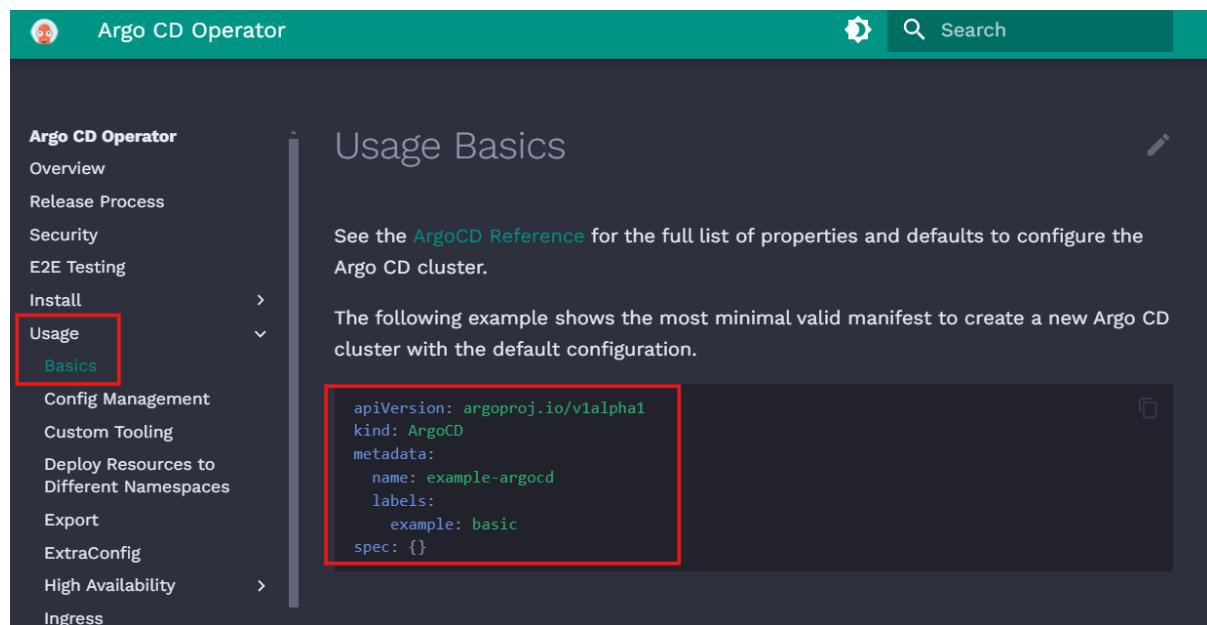
While the latest one will be present in my manifest repo



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spring-boot-app
  labels:
    app: spring-boot-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: spring-boot-app
  template:
    metadata:
      labels:
        app: spring-boot-app
    spec:
      containers:
        - name: spring-boot-app
          image: abhishekf5/ultimate-ci-cd:10
      ports:
        - containerPort: 8080
```

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 >



Argo CD Operator

Argo CD Operator

Overview

Release Process

Security

E2E Testing

Install

Usage

Basics

Config Management

Custom Tooling

Deploy Resources to Different Namespaces

Export

ExtraConfig

High Availability

Ingress

Usage Basics

See the [ArgoCD Reference](#) for the full list of properties and defaults to configure the Argo CD cluster.

The following example shows the most minimal valid manifest to create a new Argo CD cluster with the default configuration.

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec: {}
```

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 automatically. 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

```
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 8080
    - name: https
      port: 443
      protocol: TCP
      targetPort: 8080
  selector:
    app.kubernetes.io/name: example-argocd-server
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

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