✅ 📘 End-to-End CI/CD Pipeline Explained (for Spring Boot App)

🔧 1. What is CI/CD?

- **CI (Continuous Integration)**: Automatically test and build code on every change (push).
- **CD (Continuous Delivery/Deployment)**: Automatically deploy the build to a test/production environment.

🧱 CI/CD Pipeline Architecture Overview

```
[Developer]
  ↓ (Push code)
[GitHub Repository] —▶ triggers → [Jenkins]
                        ↓
                  [Maven Build & Test]
                        ↓
                  [SonarQube Analysis]
                        ↓
                  [Docker Image Build]
                        ↓
                  [Push to Docker Hub]
                        ↓
                  [Deploy to Kubernetes]
```

⚙️ Step-by-Step Breakdown

🔹 1. Code pushed to GitHub

- Dev pushes new Spring Boot code to a GitHub repo.
- GitHub triggers a webhook to notify Jenkins.

✅ **Why?**
To detect new changes and automatically trigger the build pipeline.

🔹 **2. Jenkins gets triggered**

- Jenkins receives the GitHub webhook.
- Jenkinsfile (in repo) defines the build steps.

✅ **Why?**
To automate testing, building, scanning, and deployment.

🔹 **3. Maven Build + JUnit Testing**

bash

CopyEdit

mvn clean install

- Runs unit tests
- Builds the .jar file

✅ **Why?**
Ensure code compiles and passes all unit tests.

🔹 **4. SonarQube Analysis**

bash

CopyEdit

mvn sonar:sonar

- Jenkins connects to SonarQube server.
- Analyzes code for bugs, code smells, vulnerabilities.

✅ **Why?**
Improve code quality and maintainability.

🔹 5. Docker Image Build

dockerfile

CopyEdit

```
FROM openjdk:17

COPY target/app.jar app.jar

ENTRYPOINT ["java", "-jar", "app.jar"]
```

bash

CopyEdit

```
docker build -t myapp:1.0 .
```

✅ **Why?**
Package the app into a consistent and portable container.

🔹 6. Push Docker Image to DockerHub

bash

CopyEdit

```
docker tag myapp:1.0 mydockerhub/myapp:1.0

docker push mydockerhub/myapp:1.0
```

✅ **Why?**
So that it can be deployed anywhere via Kubernetes or other platforms.

🔹 7. Deploy to Kubernetes (or Minikube)

yaml

CopyEdit

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: myapp

spec:
```

```
  replicas: 2
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: mydockerhub/myapp:1.0
        ports:
        - containerPort: 8080
```

bash

CopyEdit

```
kubectl apply -f deployment.yaml
```

✅ **Why?**
Auto-scale and manage app instances using K8s.

📦 **Tools Used in the Pipeline**

Tool	Purpose
GitHub	Code repository
Jenkins	CI/CD automation
Maven	Build & test Java app
SonarQube	Code quality analysis
Docker	Containerization

Tool	Purpose
DockerHub	Image registry
Kubernetes	Container orchestration & deployment

🎯 Real-World Usage Example

A Java backend developer pushes code:

- ✅ Jenkins builds it with Maven
- ✅ Tests run and code quality checked with SonarQube
- ✅ Docker image is created and stored on DockerHub
- ✅ Kubernetes takes that image and deploys it on UAT or production

---

🧠 Benefits of CI/CD Pipeline

- Faster development and deployment
- No manual errors
- Consistent environment
- Catch bugs early
- Easy rollback (with tags & deployments)

⚠️ Common Issues / Drawbacks

- Initial setup takes time
- Configuration management must be clean
- Security (DockerHub credentials, secrets) must be managed

🧪 Interview Tip: CI/CD Flow Summary

When asked in an interview:

"Can you explain your CI/CD pipeline?"

Say:

“We use GitHub as the source repo. Jenkins is triggered on each push. It builds our Spring Boot app with Maven, runs unit tests, and performs code quality checks using SonarQube. If everything passes, a Docker image is built and pushed to DockerHub. Kubernetes then pulls the image and deploys it in our UAT/production environment.”

Here's a complete **Jenkinsfile example** for the full CI/CD pipeline we discussed for a **Spring Boot + Docker + Kubernetes** project using Maven and SonarQube:

✅ **Jenkinsfile (Declarative Pipeline)**

groovy

CopyEdit

```
pipeline {
  agent any

  environment {
    SONARQUBE_SERVER = 'SonarQube'
    DOCKER_HUB_REPO = 'yourdockerhubusername/your-app-name'
    IMAGE_TAG = "${BUILD_NUMBER}"
  }

  tools {
    maven 'Maven 3.8.1'
    jdk 'JDK 17'
  }

  stages {
```

```
stage('Checkout Code') {
  steps {
    git url: 'https://github.com/your-username/your-repo.git', branch: 'main'
  }
}

stage('Build with Maven') {
  steps {
    sh 'mvn clean install -DskipTests=false'
  }
}

stage('SonarQube Analysis') {
  steps {
    withSonarQubeEnv("${SONARQUBE_SERVER}") {
      sh 'mvn sonar:sonar'
    }
  }
}

stage('Build Docker Image') {
  steps {
    sh "docker build -t ${DOCKER_HUB_REPO}:${IMAGE_TAG} ."
  }
}

stage('Push Docker Image') {
```

```
        steps {
            withCredentials([usernamePassword(credentialsId: 'dockerhub-creds',
usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')]) {
                sh 'echo "$DOCKER_PASS" | docker login -u "$DOCKER_USER" --password-
stdin'
                sh "docker push ${DOCKER_HUB_REPO}:${IMAGE_TAG}"
            }
        }
    }

    stage('Deploy to Kubernetes') {
        steps {
            sh """
            sed -i 's|IMAGE_PLACEHOLDER|${DOCKER_HUB_REPO}:${IMAGE_TAG}|'
k8s/deployment.yaml
            kubectl apply -f k8s/deployment.yaml
            kubectl apply -f k8s/service.yaml
            """
        }
    }
  }

  post {
    success {
        echo 'Pipeline completed successfully!'
    }
    failure {
        echo 'Pipeline failed. Please check logs.'
    }
```

```
  }
}
```

📂 **Required Files**

✅ **Dockerfile**

dockerfile

CopyEdit

```
FROM openjdk:17
COPY target/*.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]
```

✅ **k8s/deployment.yaml**

yaml

CopyEdit

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: myapp
spec:
 replicas: 2
 selector:
  matchLabels:
   app: myapp
 template:
  metadata:
   labels:
    app: myapp
  spec:
   containers:
```

```
    - name: myapp
      image: IMAGE_PLACEHOLDER
      ports:
        - containerPort: 8080
```

✅ **k8s/service.yaml**

yaml

CopyEdit

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  selector:
    app: myapp
  ports:
    - port: 80
      targetPort: 8080
  type: LoadBalancer
```

🔐 **Required Jenkins Credentials**

- dockerhub-creds: DockerHub Username & Password (store in Jenkins credentials)
- SonarQube server must be configured under Jenkins → Global Tool Configuration

---

✅ **Result**

Once setup:

- Code pushed to GitHub triggers this pipeline
- Jenkins builds, scans, and packages the app

- Docker image is pushed
- App is auto-deployed to Kubernetes