Maven + Java + Kubernetes (Manual / On-demand Pipeline)
Create a CI/CD pipeline for a Maven-based Java application using Git/GitHub Jenkins, and Docker. Deploy the containerized application on a Kubernetes cluster and demonstrate the execution of each pipeline stage

Below is a **complete, end-to-end CI/CD flow** for a **Maven + Java application** using **GitHub → Jenkins → Docker → Kubernetes**, built as a **Manual / On-Demand Jenkins Pipeline** and with **clear demonstration of each stage**.

This is written in a **DevOps-lab / interview-ready format** ✅

Maven + Java + Kubernetes

Manual / On-Demand CI/CD Pipeline

1. Architecture Overview

mathematica Copy code

```
Developer
   |
   |  git push
   v
GitHub Repository
   |
   |  Manual trigger
   v
Jenkins Pipeline
   ├── Checkout Code
   ├── Maven Build & Test
   ├── Docker Image Build
```

```
    ├── Push Image to Docker Hub
    ├── Deploy to Kubernetes
    v
Kubernetes Cluster (Minikube / EKS / AKS)
```

2. Prerequisites

Jenkins Server

- Jenkins installed
- Plugins:
  - Git
  - Pipeline
  - Docker Pipeline
  - Kubernetes CLI (kubectl)
- Jenkins user added to Docker group

Tools Installed on Jenkins

bash Copy code

```
java -version
mvn -version
docker --version
kubectl version --client
```

Accounts

- GitHub repository
- Docker Hub account
- Kubernetes cluster (Minikube is fine)

3. Sample Maven Java Application

Project Structure

css Copy code

```
maven-java-app/
├── Dockerfile
```

```
├── Jenkinsfile
├── pom.xml
├── deployment.yaml
└── src
    └── main
        └── java
            └── com/example/App.java
```

App.java

java Copy code

```java
package com.example;

public class App {
    public static void main(String[] args) {
        System.out.println("Hello from Maven Java App running in Kubernetes
        while(true) {}
    }
}
```

pom.xml

xml Copy code

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>maven-java-app</artifactId>
  <version>1.0</version>

  <properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
  </properties>

</project>
```

4. Dockerfile (Containerization Stage)

dockerfile Copy code

```
FROM openjdk:11-jre-slim
WORKDIR /app
COPY target/maven-java-app-1.0.jar app.jar
CMD ["java", "-jar", "app.jar"]
```

5. Kubernetes Deployment Manifest

`deployment.yaml`

yaml Copy code

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: maven-java-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: maven-java
  template:
    metadata:
      labels:
        app: maven-java
    spec:
      containers:
      - name: maven-java
        image: viveknshet112/maven-java-app:latest
        ports:
        - containerPort: 8080
```

6. Jenkins Pipeline (Manual / On-Demand)

Jenkinsfile

groovy Copy code

```
pipeline {
    agent any

    environment {
        IMAGE_NAME = "viveknshet112/maven-java-app"
        DOCKER_TAG = "latest"
    }

    stages {

        stage('Checkout Code') {
            steps {
                git branch: 'main',
                    url: 'https://github.com/<your-username>/maven-java-ap
            }
        }

        stage('Maven Build') {
            steps {
                sh 'mvn clean package'
            }
        }

        stage('Docker Build') {
            steps {
                sh "docker build -t ${IMAGE_NAME}:${DOCKER_TAG} ."
            }
        }

        stage('Docker Push') {
            steps {
                withCredentials([usernamePassword(
                    credentialsId: 'dockerhub-creds',
                    usernameVariable: 'DOCKER_USER',
                    passwordVariable: 'DOCKER_PASS'
                )]) {
                    sh '''
                    echo $DOCKER_PASS | docker login -u $DOCKER_USER --pas
```

```
                    docker push ${IMAGE_NAME}:${DOCKER_TAG}
                    '''
                }
            }
        }

        stage('Deploy to Kubernetes') {
            steps {
                sh '''
                kubectl apply -f deployment.yaml
                kubectl rollout status deployment/maven-java-app
                '''
            }
        }
    }

    post {
        success {
            echo "✅ CI/CD Pipeline executed successfully"
        }
        failure {
            echo "❌ Pipeline failed"
        }
    }
}
```

7. Manual / On-Demand Execution

1. Open **Jenkins Dashboard**
2. Select **Pipeline Job**
3. Click **Build Now**
4. Pipeline executes **only when triggered manually**

✔ This satisfies **Manual / On-Demand CI/CD**

8. Demonstration of Each Pipeline Stage

1️⃣ Checkout Code

text Copy code

```
Cloning repository from GitHub
```

2️⃣ Maven Build

text Copy code

```
[INFO] Building jar: target/maven-java-app-1.0.jar
BUILD SUCCESS
```

3️⃣ Docker Build

text Copy code

```
Successfully built image viveknshet112/maven-java-app:latest
```

4️⃣ Docker Push

text Copy code

```
Pushed image to Docker Hub
```

5️⃣ Kubernetes Deployment

bash Copy code

```
kubectl get pods
kubectl get deployments
```

Output:

text Copy code

```
maven-java-app-xxxxx   Running
```

9. Verification

bash Copy code

```
kubectl describe deployment maven-java-app
kubectl logs <pod-name>
```

Logs:

text Copy code

```
Hello from Maven Java App running in Kubernetes!
```

10. Key Interview Points

- Manual pipeline using **Build Now**
- Maven used for build & dependency management
- Docker used for containerization
- Kubernetes for orchestration
- Jenkins as CI/CD orchestrator
- Fully automated but **manually triggered**

11. Enhancements (Optional)

- Add `mvn test`
- Use image versioning (`BUILD_NUMBER`)
- Add Kubernetes Service (NodePort)
- Use Helm
- Add approval stage

If you want, I can also provide:

- 🎯 **Pipeline diagram**
- 🎯 **Jenkins screenshots explanation**
- 🎯 **Helm-based deployment**
- 🎯 **GitHub webhook (auto trigger)**

Just tell me 👍