Devops Programs

1. Maven + Java + Docker Swarm (Manual / On-Demand Pipeline)

Design and implement a CI/CD pipeline for a Java application built using Maven. Use Git/GitHub for version control, Jenkins for pipeline execution, and Docker for containerization. Deploy the containerized application on Docker Swarm and explain each stage of the pipeline during execution.

Aim

To design and implement a **manual (on-demand) CI/CD pipeline** for a **Maven-based Java application** using **Git/GitHub, Jenkins, and Docker**, and deploy the containerized application on **Docker Swarm**.

Tools Used

- Git & GitHub
- Maven
- Jenkins
- Docker
- Docker Swarm
- Java (Spring Boot)

Step 1: Create Maven-based Java Application

1. Go to **https://start.spring.io/**
2. Select:
    - Project: **Maven**
    - Language: **Java**
    - Packaging: **Jar**
    - Java Version: **21**
    - Dependency: **Spring Web**
3. Artifact name: `my_maven_app`
4. Click **Generate** and extract the project.

- **Go To Terminal and do this steps:**
- That my_maven_app zip file will downloaded in downloads folder
- go to download folder **-cd Downloads**
- check that file is exists or not **-ls**
- you will see **my_maven_app.zip** folder in it
- unzip that folder – **unzip my_maven_app.zip**
- and go to that folder – **cd my_maven_app**

Step 2: Add Controller File

Create `HomeController.java` inside `src/main/java/com/example/my_maven_app/`

cd src/main/java/com/example/my_maven_app/

nano HomeController.java

add this code

```
package com.example.my_maven_app;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HomeController {

    @GetMapping("/")
    public String home() {
        return "Hello from Spring Boot Maven App!";
    }
}
```

Step 3: Build the Maven Project

- `sudo apt install maven`
- `mvn clean package -DskipTests`

This generates a **JAR file** inside the `target/` directory.

Do this inside my_maven_app folder not inside src !!

Step 4: Create Dockerfile

Create a `Dockerfile` in the project root:

- **nano Dockerfile (no extension for Dockerfile)**

add this code

```
FROM eclipse-temurin:21-jdk-alpine
WORKDIR /Downloads/my_maven_app       #here you need change your app path
COPY target/*.jar app.jar
EXPOSE 8080
CMD ["java","-jar","app.jar"]
```

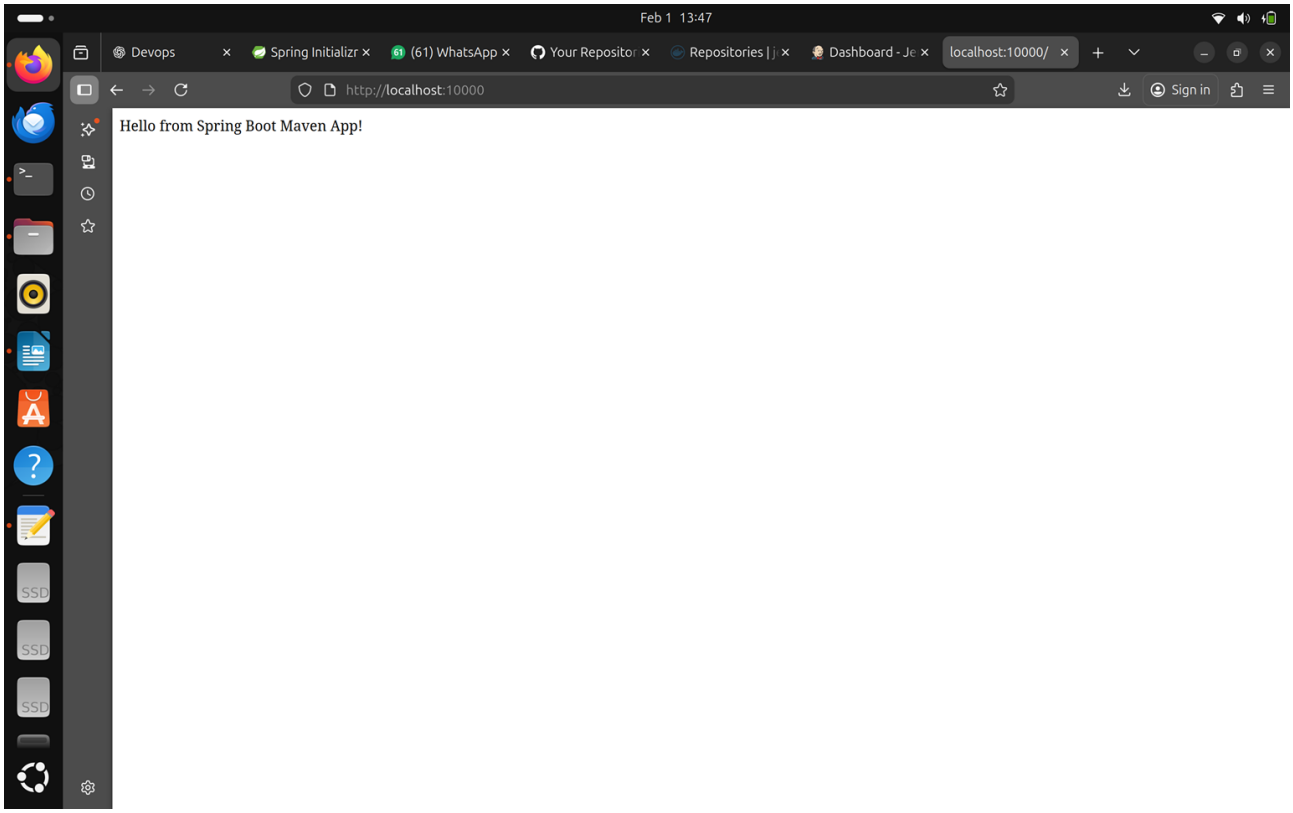
Step 5: Test Docker Image Locally

- `docker build -t my_maven_app .`
- `docker run -p 10000:8080 my_maven_app:latest`

```
jeevan@Jeevan-Shetty:~/Downloads/my_maven_app$ docker run -p 10000:8080 my_maven_app:latest

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v4.0.2)

2026-02-01T08:16:29.695Z  INFO 1 --- [my_maven_app] [           main] c.e.my_maven_app.MyMavenAppApplication   : Starting MyMavenAppAppl
ication v0.0.1-SNAPSHOT using Java 21.0.9 with PID 1 (/Downloads/my_maven_app/app.jar started by root in /Downloads/my_maven_app)
2026-02-01T08:16:29.697Z  INFO 1 --- [my_maven_app] [           main] c.e.my_maven_app.MyMavenAppApplication   : No active profile set,
falling back to 1 default profile: "default"
2026-02-01T08:16:30.153Z  INFO 1 --- [my_maven_app] [           main] o.s.boot.tomcat.TomcatWebServer          : Tomcat initialized with
 port 8080 (http)
2026-02-01T08:16:30.172Z  INFO 1 --- [my_maven_app] [           main] o.apache.catalina.core.StandardService   : Starting service [Tomca
t]
2026-02-01T08:16:30.172Z  INFO 1 --- [my_maven_app] [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine
: [Apache Tomcat/11.0.15]
2026-02-01T08:16:30.192Z  INFO 1 --- [my_maven_app] [           main] b.w.c.s.WebApplicationContextInitializer : Root WebApplicationCont
ext: initialization completed in 455 ms
2026-02-01T08:16:30.409Z  INFO 1 --- [my_maven_app] [           main] o.s.boot.tomcat.TomcatWebServer          : Tomcat started on port
8080 (http) with context path '/'
2026-02-01T08:16:30.419Z  INFO 1 --- [my_maven_app] [           main] c.e.my_maven_app.MyMavenAppApplication   : Started MyMavenAppAppli
cation in 0.961 seconds (process running for 1.223)
2026-02-01T08:16:43.767Z  INFO 1 --- [my_maven_app] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring Dis
patcherServlet 'dispatcherServlet'
2026-02-01T08:16:43.767Z  INFO 1 --- [my_maven_app] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'd
ispatcherServlet'
2026-02-01T08:16:43.768Z  INFO 1 --- [my_maven_app] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initializatio
n in 0 ms
```

Open browser → http://localhost:10000

you will see like this



After this stop the spring using – **ctrl+c**

Step 6: Initialize Git and Push to GitHub

```
- git init
- git branch -m main
- git add .
- git commit -m "Initial Commit"
```

Step 7: Create GitHub Repo

- ➓ Repo name: `my_maven_app`
- ➓ Copy **SSH URL**

```
- git remote add origin git@github.com:USERNAME/my_maven_app.git
- git push origin main
```

Step 8: Docker Hub Setup

- ➓ login Docker Hub account
- ➓ Create repo: `my_maven_app`

Step 9: Add Docker Hub Credentials in Jenkins

Jenkins →
Manage Jenkins → Credentials → Global → Add Credentials

- ➓ Kind: Username & Password
- ➓ ID: `dockerhub`
- ➓ Username: DockerHub username
- ➓ Password: DockerHub password

Step 10: Configure Jenkins SSH Access to GitHub

SSH KEY GEN AND ADDING IT TO GITHUB

- ssh-keygen -t ed25519 -C "<your_mail_id>"

- cat ~/.ssh/id_ed25519.pub (copy this key)

- goto github.com -> settings -> ssh and gpg keys -> add the key

Step 11: Create Jenkinsfile

- nano jenkinsfile

Paste

```
pipeline {
    agent any

    environment {
        IMAGE_NAME = "<dockerhub_username>/my_maven_app"
        DOCKERHUB = credentials('dockerhub')
    }

    stages {

        stage('Checkout') {
            steps {
                git branch: 'main',
                    url: 'git@github.com:USERNAME/my_maven_app.git'
            }
        }

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

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

        stage('Push Docker Image to Docker Hub') {
            steps {
                sh 'docker login -u $DOCKERHUB_USR -p $DOCKERHUB_PSW'
                sh 'docker push $IMAGE_NAME:latest'
            }
        }
    }

    post {
```

```
        success {
            echo "Pipeline executed successfully"
        }
        failure {
            echo "Pipeline execution failed"
        }
        always {
            cleanWs()
        }
    }
}
```

Commit & push:

```
- git add .
- git commit -m "Added Jenkinsfile"
- git push origin main
```

Step 10: Add Jenkins to Docker Group

```
- sudo usermod -aG docker jenkins
- sudo systemctl restart docker
- sudo systemctl restart jenkins
```

Step 11: Create Jenkins Pipeline Job

1. Jenkins → New Item → Pipeline
2. Pipeline from SCM
3. SCM: Git
4. Repo URL: <github_ssh_url>
5. master* - main
6. Script path: jenkinsfile
7. Click **Build Now** (manual trigger)

Step 12: Initialize Docker Swarm

Run ONLY if Swarm is not already active:

- docker swarm init

This step is done **only once per system**, not every time.

If system says:

This node is already part of a swarm

→ Swarm is already ready

Step 13: Deploy Application as Swarm Service

**docker service create **

**--name my_maven_service **

**--publish 8080:8080 **

<dockerhub_username>/my_maven_app:latest

Step 14: Verify Service

- docker service ls

- docker service ps my_maven_service

Step 15: Verify in Browser

http://localhost:8080

DOCKER SWARM – FULL STEP-BY-STEP (MAM'S METHOD)

Setup: Ubuntu Host + VirtualBox + Vagrant
Cluster: 1 Manager + 2 Workers

PHASE 1: HOST SYSTEM SETUP (Ubuntu Laptop)

□ STEP 1: Install VirtualBox

```
sudo apt update
sudo apt install -y virtualbox
```

Verify:

```
vboxmanage --version
```

□ STEP 2: Install Vagrant

Add HashiCorp key:

```
curl -fsSL https://apt.releases.hashicorp.com/gpg | ¥
sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg
```

Add repo:

```
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] ¥
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | ¥
sudo tee /etc/apt/sources.list.d/hashicorp.list
```

Install Vagrant:

```
sudo apt update
sudo apt install -y vagrant
```

Verify:

```
vagrant --version
```

PHASE 2: CREATE DOCKER SWARM VMs

□ STEP 3: Create Project Directory

```
mkdir docker-swarm-vms
cd docker-swarm-vms
```

Initialize Vagrant:

```
vagrant init
```

Edit Vagrantfile:

```
nano Vagrantfile
```

□ STEP 4: Vagrantfile (PASTE FULL CODE)

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/jammy64"

  # Manager VM
  config.vm.define "manager-vm" do |manager|
    manager.vm.hostname = "manager-vm"
    manager.vm.network "private_network", ip: "192.168.56.10"
    manager.vm.provider "virtualbox" do |vb|
      vb.memory = 2048
      vb.cpus = 2
    end
  end
```

```
  # Worker 1
  config.vm.define "worker1-vm" do |worker1|
    worker1.vm.hostname = "worker1-vm"
    worker1.vm.network "private_network", ip: "192.168.56.11"
    worker1.vm.provider "virtualbox" do |vb|
      vb.memory = 1024
      vb.cpus = 1
    end
  end

  # Worker 2
  config.vm.define "worker2-vm" do |worker2|
    worker2.vm.hostname = "worker2-vm"
    worker2.vm.network "private_network", ip: "192.168.56.12"
    worker2.vm.provider "virtualbox" do |vb|
      vb.memory = 1024
      vb.cpus = 1
    end
  end
end
```

Save and exit.

□ STEP 5: Create All VMs

```
vagrant up
```

Verify:

```
vagrant status
```

Expected:

```
manager-vm  running
worker1-vm  running
worker2-vm  running
```

PHASE 3: INSTALL DOCKER ON ALL VMs

□ STEP 6: Login to VMs

Manager:

```
vagrant ssh manager-vm
```

Workers (new terminals):

```
vagrant ssh worker1-vm
vagrant ssh worker2-vm
```

□ STEP 7: Install Docker (RUN ON ALL VMs)

Inside **each VM**:

```
sudo apt update
sudo apt install -y docker.io
sudo systemctl start docker
```

```
sudo systemctl enable docker
sudo usermod -aG docker $USER

exit
```

Re-login to VM after exit.

Verify:

```
docker --version
```

PHASE 4: DOCKER SWARM CLUSTER SETUP

□ STEP 8: Initialize Swarm (MANAGER ONLY)

On `manager-vm`:

```
sudo docker swarm init --advertise-addr 192.168.56.10
```

Copy the **worker join token command**

□ STEP 9: Join Worker Nodes

On `worker1-vm`:

```
sudo docker swarm join --token <TOKEN> 192.168.56.10:2377
```

On `worker2-vm`:

```
sudo docker swarm join --token <TOKEN> 192.168.56.10:2377
```

□ STEP 10: Verify Swarm Cluster (MANAGER)

Back on `manager-vm`:

```
sudo docker node ls
```

Expected:

```
manager-vm   Ready   Leader
worker1-vm   Ready
worker2-vm   Ready
```

Docker Swarm cluster is READY

PHASE 5: DEPLOY APPLICATION ON DOCKER SWARM

(Assuming Jenkins already built & pushed image)

□ STEP 11: Create Docker Swarm Service

```
sudo docker service create ¥
--name my_maven_service ¥
--replicas 3 ¥
-p 8080:8080 ¥
<dockerhub_username>/my_maven_app:latest
```

□ STEP 12: Verify Service

```
docker service ls
docker service ps my_maven_service
```

□ STEP 13: Access Application

From **HOST browser**:

```
http://192.168.56.10:8080
```

You should see:

```
Hello from Spring Boot Maven App!
```

PHASE 6: CLEANUP (EXAM-READY)

□ Remove Service (Manager)

```
docker service rm my_maven_service
```

□ Workers Leave Swarm

On worker VMs:

```
docker swarm leave
```

□ Manager Leave Swarm

```
docker swarm leave --force
```

□ Stop VMs (Host)

```
vagrant halt
```

□ Destroy VMs (ONLY IF ASKED)

```
vagrant destroy -f
```

2. Maven + Java + Docker Swarm (Cron-based Automated Pipeline)

Develop a Java application using Maven and implement a fully automated CI/CD pipeline using Git/GitHub, Jenkins, and Docker. Configure Jenkins cron-based triggers to automatically initiate the pipeline. Containerize the application and deploy it on Docker Swarm. Explain each automated stage of the pipeline.

What you MUST change for Q2 (ONLY THIS)

1. Jenkinsfile

Add **cron trigger**:

```
Go to Triggers → Select Build periodically → in the trigger box add - * * * * *
And Click save
```

That's the **only code change**.

Do NOT click *Build Now!!*

The pipeline will run **automatically**.

3. 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.

Step 1: Install & Start Minikube

```
Install and Setup Minikube
        *Goto official minikube website and find the commands for installation

        - curl -LO https://github.com/kubernetes/minikube/releases/latest/download/minikube-linux-amd64

        - sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64


->Initialize and start kubernetes environment
```

- minikube start

->Install kubectl and verify

- sudo snap install kubectl --classic

- kubectl version

- kubectl cluster info

->Authenticate with dockerhub credentials

- eval $(minikube docker-env)

- docker login

Step 2: Create Kubernetes Deployment

```
kubectl create deployment my-maven-app ¥
--image=<dockerhub_username>/my_maven_app:latest
```

Verify:

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

Step 10: Expose Deployment as Service

```
kubectl expose deployment my-maven-app ¥
--type=NodePort ¥
--port=8080
```

Step 11: Access Application

```
- minikube service my-maven-app
```

Browser opens automatically showing:

```
Hello from Spring Boot Maven App!
```

Step 12: Access Minikube Dashboard

Go to new terminal and do it !!

- minikube dashboard

4. Maven + Java + Kubernetes (Cron-based Automated Pipeline)

Build a Java application using Maven and configure a fully automated CI/CD pipeline with Git/GitHub, Jenkins, and Docker. Use Jenkins cron jobs to trigger the pipeline automatically and deploy the application on Kubernetes. Explain all automated stages of the pipeline.

What you MUST change for Q4(ONLY THIS)

1. Jenkinsfile

Add **cron trigger**:

```
Go to Triggers → Select Build periodically → in the trigger box add - * * * * *
And Click save
```

That's the **only code change**.

Do NOT click *Build Now!!*

The pipeline will run **automatically**.

Q5. Flask + Docker Swarm (Manual / On-Demand)

PHASE 1: CREATE FLASK APPLICATION (HOST SYSTEM)

Step 1: Create Project Directory

```
mkdir my_webapp
cd my_webapp
```

Step 2: Create Flask App

app.py

```
nano app.py
```

```
from flask import Flask
```

```
app = Flask(__name__)

@app.route("/")
def home():
    return "Hello from Jenkins Pipeline Demo"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

Step 3: Create Requirements File

requirements.txt

```
nano requirements.txt

flask
```

Step 4: Create Dockerfile

```
nano Dockerfile

FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python","app.py"]
```

or

```
FROM python:3.10

WORKDIR /app

COPY . .

RUN pip install flask

EXPOSE 5000

CMD ["python", "app.py"]
```

Step 5: Test Locally (Optional but Good)

```
docker build -t my_webapp .
docker run -p 5000:5000 my_webapp
```

Browser:

```
http://localhost:5000
```

Stop container:

```
Ctrl + C
```

PHASE 2: PUSH CODE TO GITHUB

Step 6: Initialize Git

```
git init
git branch -m main
git add .
git commit -m "Initial Flask app"
```

Step 7: Create GitHub Repository

- ❿ Repo name: my_webapp
- ❿ Copy **SSH URL**

```
git remote add origin git@github.com:USERNAME/my_webapp.git
git push origin main
```

PHASE 3: JENKINS (CI – MANUAL PIPELINE)

Step 8: Add Docker Hub Credentials in Jenkins

Jenkins → Manage Jenkins → Credentials → Global

- ❿ Kind: Username & Password
- ❿ ID: dockerhub

Step 9: Jenkinsfile (MANUAL PIPELINE)

jenkinsfile

```
nano jenkinsfile
```

```
pipeline {
    agent any

    environment {
        IMAGE_NAME = "<dockerhub_username>/my_webapp"
        DOCKERHUB = credentials('dockerhub')
    }

    stages {

        stage('Checkout') {
            steps {
                git branch: 'main',
                    url: 'git@github.com:USERNAME/my_webapp.git'
            }
```

```
        }

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

        stage('Push Docker Image') {
            steps {
                sh 'docker login -u $DOCKERHUB_USR -p $DOCKERHUB_PSW'
                sh 'docker push $IMAGE_NAME:latest'
            }
        }
    }

    post {
        success {
            echo "Pipeline Successful"
        }
        failure {
            echo "Pipeline Failed"
        }
        always {
            cleanWs()
        }
    }
}
```

Step 10: Push Jenkinsfile

```
git add jenkinsfile
git commit -m "Added Jenkinsfile for Flask app"
git push origin main
```

Step 11: Create Jenkins Pipeline Job

1. Jenkins → New Item → Pipeline
2. Pipeline from SCM
3. SCM: Git
4. Repo URL: `git@github.com:USERNAME/my_webapp.git`
5. Script path: `jenkinsfile`
6. Save → **Build Now**

Pipeline SUCCESS
Image pushed to Docker Hub

PHASE 4: DOCKER SWARM (MAM'S METHOD)

(Assuming Swarm cluster already created)

Step 12: Deploy Flask App on Docker Swarm

On **manager-vm**:

```
docker service create ¥
--name my_flask_service ¥
--replicas 3 ¥
-p 8080:5000 ¥
<dockerhub_username>/my_webapp:latest
```

Step 13: Verify Service

```
docker service ls
docker service ps my_flask_service
```

Step 14: Verify in Browser

From **host system**:

```
http://192.168.56.10:8080
```

Output:

```
Hello from Jenkins Pipeline Demo
```

CLEANUP (Q5)

Remove Service

```
docker service rm my_flask_service
```

Workers Leave Swarm (optional)

```
docker swarm leave
```

Manager Leave Swarm (optional)

```
docker swarm leave --force
```

Stop VMs

```
vagrant halt
```

Q6. Flask + Docker Swarm (Cron-Based / Automated Pipeline)

What you MUST change for Q6(ONLY THIS)

1. Jenkinsfile

Add **cron trigger**:

Go to Triggers → Select Build periodically → in the trigger box add - * * * * *
And Click save

That's the **only code change**.

Do NOT click *Build Now!!*

The pipeline will run automatically.

Q7. Flask + Kubernetes (Manual / On-Demand Pipeline)

Step 1: Install & Start Minikube

Install and Setup Minikube
*Goto official minikube website and find the commands for installation

- **curl -LO https://github.com/kubernetes/minikube/releases/latest/download/minikube-linux-amd64**

- **sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64**

->Initialize and start kubernetes environment

- **minikube start**

->Install kubectl and verify

- **sudo snap install kubectl --classic**

- **kubectl version**

- **kubectl cluster info**

->Authenticate with dockerhub credentials

- **eval $(minikube docker-env)**

- **docker login**

Step 2: Create Kubernetes Deployment

```
kubectl create deployment my-webapp ¥
--image=<dockerhub_username>/my_webapp:latest
```

Verify:

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

Step 10: Expose Deployment as Service

```
kubectl expose deployment my-webapp ¥
--type=NodePort ¥
--port=5000
```

Step 11: Access Application

```
- minikube service my-maven-app
```

Browser opens automatically showing:

```
Hello from Flask App!
```

Step 12: Access Minikube Dashboard

Go to new terminal and do it !!

- minikube dashboard

CLEANUP (Q7)

```
kubectl delete service my-flask-app
kubectl delete deployment my-flask-app
minikube stop
```

Q8. Flask + Kubernetes (Cron-Based / Automated Pipeline)

What you MUST change for Q8(ONLY THIS)

1. Jenkinsfile

Add **cron trigger**:

```
Go to Triggers → Select Build periodically → in the trigger box add - * * * * *
And Click save
```

That's the **only code change**.

Do NOT click *Build Now!!*

The pipeline will run automatically.

Q9. React + Docker Swarm (Manual / On-Demand Pipeline)

PHASE 1: CREATE REACT APPLICATION

Step 1: Create React App

```
npx create-react-app my_react_app
cd my_react_app
```

Step 2: Verify React App

```
npm start
```

Open:

```
http://localhost:3000
```

Stop:

```
Ctrl + C
```

PHASE 2: DOCKERIZE REACT APPLICATION

Step 3: Create Dockerfile

```
nano Dockerfile
```

```
FROM node:18
```

```
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
RUN npm install -g serve
EXPOSE 3000
CMD ["serve", "-s", "build", "-l", "3000"]
```

Step 4: Test Docker Image Locally

```
docker build -t my_react_app .
docker run -p 3001:3000 my_react_app
```

Open:

```
http://localhost:3001
```

PHASE 3: PUSH CODE TO GITHUB

Step 5: Git Initialization

```
git init
git branch -m main
git add .
git commit -m "Initial React app"
```

Create GitHub repo: `my_react_app`

```
git remote add origin git@github.com:USERNAME/my_react_app.git
git push -u origin main
```

PHASE 4: JENKINS (MANUAL PIPELINE)

Step 6: Create Jenkinsfile

```
nano jenkinsfile
```

```
pipeline {
    agent any

    environment {
        IMAGE_NAME = "<dockerhub_username>/my_react_app"
        DOCKERHUB = credentials('dockerhub')
    }

    stages {

        stage('Checkout') {
            steps {
                git branch: 'main',
                    url: 'git@github.com:USERNAME/my_react_app.git'
            }
        }

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

```
            steps {
                sh 'docker build -t $IMAGE_NAME:latest .'
            }
        }

        stage('Push Docker Image') {
            steps {
                sh 'docker login -u $DOCKERHUB_USR -p $DOCKERHUB_PSW'
                sh 'docker push $IMAGE_NAME:latest'
            }
        }
    }

    post {
        success {
            echo "React Manual Pipeline Successful"
        }
        always {
            cleanWs()
        }
    }
}
```

Step 7: Push Jenkinsfile

```
git add jenkinsfile
git commit -m "Added Jenkinsfile for React"
git push origin main
```

Step 8: Run Jenkins Pipeline (Manual)

1. Open Jenkins Dashboard
2. Open Pipeline Job
3. Click **Build Now**

Build success
Image pushed to Docker Hub

PHASE 5: DOCKER SWARM DEPLOYMENT

(Docker Swarm already created)

Step 9: Deploy React App

On **manager-vm**:

```
docker service create ¥
--name my_react_service ¥
--replicas 3 ¥
-p 8080:3000 ¥
<dockerhub_username>/my_react_app:latest
```

Step 10: Verify Service

```
docker service ls
```

```
docker service ps my_react_service
```

Step 11: Access Application

From host browser:

```
http://192.168.56.10:8080
```

React UI loads successfully.

CLEANUP (Q9)

```
docker service rm my_react_service
```

(Optional if asked)

```
docker swarm leave
docker swarm leave --force
```

Q10. React + Docker Swarm (Cron-Based / Automated Pipeline)

What you MUST change for Q10(ONLY THIS)

1. Jenkinsfile

Add **cron trigger**:

```
Go to Triggers → Select Build periodically → in the trigger box add - * * * * *
And Click save
```

That's the **only code change**.

Do NOT click *Build Now!!*

The pipeline will run automatically.

Q11. React + Kubernetes (Manual / On-Demand Pipeline)

Step 1: Install & Start Minikube

Install and Setup Minikube
*Goto official minikube website and find the commands for installation

- **curl -LO https://github.com/kubernetes/minikube/releases/latest/download/minikube-linux-amd64**

- **sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64**

->Initialize and start kubernetes environment

- **minikube start**

->Install kubectl and verify

- **sudo snap install kubectl ―classic**

- **kubectl version**

- **kubectl cluster info**

->Authenticate with dockerhub credentials

- **eval $(minikube docker-env)**

- **docker login**

Step 2: Create Kubernetes Deployment

```
kubectl create deployment my-react-app ¥
--image=<dockerhub_username>/my_react_app:latest
```

Verify:

- kubectl get deployments
- kubectl get pods

Step 10: Expose Deployment as Service

```
kubectl expose deployment my-react-app ¥
--type=NodePort ¥
--port=5000
```

Step 11: Access Application

```
- minikube service my-react-app
```

Browser opens automatically showing:

```
React Webpage
```

Step 12: Access Minikube Dashboard

Go to new terminal and do it !!

- minikube dashboard

CLEANUP (Q7)

```
kubectl delete service my-react-app
kubectl delete deployment my-react-app
minikube stop
```

Q12. React + Kubernetes (Cron-Based / Automated Pipeline)

What you MUST change for Q8(ONLY THIS)

1. Jenkinsfile

Add **cron trigger**:

```
Go to Triggers → Select Build periodically → in the trigger box add - * * * * *
And Click save
```

That's the **only code change**.

Do NOT click *Build Now!!*

The pipeline will run automatically.