🔵 QUESTION 1

CI/CD Pipeline for Maven Application using Jenkins + Docker + Docker Swarm

---

🔹 STEP 1: Install Required Software (One Time)

sudo apt update
sudo apt install openjdk-17-jdk maven git -y

👉 Needed for **Java + Maven build**

java -version
mvn -version
git --version

🔹 STEP 2: Install Docker

sudo apt install docker.io -y
sudo systemctl start docker
sudo systemctl enable docker
sudo usermod -aG docker $USER
newgrp docker

Verify:

docker --version
docker ps

🔹 STEP 3: Install Jenkins

sudo apt install fontconfig openjdk-17-jre -y

👉 Jenkins **runs on Java**, so JRE is required

wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc
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
sudo apt update
sudo apt install jenkins -y

sudo systemctl start jenkins

- **STEP 4: Create Maven Application**

mvn archetype:generate \
-DgroupId=com.example \
-DartifactId=my_maven_app \
-DarchetypeArtifactId=maven-archetype-quickstart \
-DinteractiveMode=false

cd my_maven_app
mvn clean package

- **STEP 5: Push Project to GitHub**

git init
git add .
git commit -m "Initial Maven project"
git branch -M main
git remote add origin https://github.com/<USERNAME>/my_maven_app.git
git push -u origin main

- **STEP 6: Create Dockerfile**

nano Dockerfile

FROM eclipse-temurin:21-jdk-alpine
WORKDIR /app
COPY target/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java","-jar","app.jar"]

- **STEP 7: Initialize Docker Swarm**

docker swarm init --advertise-addr <YOUR-IP>

Example:

docker swarm init --advertise-addr 172.20.10.2

- **STEP 8: Create Jenkinsfile**

nano Jenkinsfile

```
pipeline {
    agent any

    environment {
        IMAGE_NAME = "dockerhubusername/my_maven_app"
        DOCKERHUB = credentials('dockerhub')
    }

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

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

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

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

        stage('Deploy to Swarm') {
            steps {
```

```
            sh '''
            docker service rm my_app || true
            docker service create --name my_app -p 8081:8080
$IMAGE_NAME:latest
            '''
        }
      }
   }
}
```

- **STEP 9: Run Jenkins Job**
  - Open Jenkins
  - Create **Pipeline**
  - Select **Pipeline script from SCM**
  - Build Now

Verify:

docker service ls

🔵 QUESTION 2

Maven CI/CD with Jenkins Cron (Scheduler)

- **STEP 1: Open Jenkins Job**
  - Configure → **Build Triggers**
  - Enable **Build periodically**

Cron Example (every 1 min):

* * * * *

👉 Pipeline auto runs every minute

🔹 **STEP 2: Observe Build History**

- Jenkins triggers automatically
- No manual click

🔵 QUESTION 3

Maven CI/CD using Jenkins + Docker + Kubernetes (Minikube)

🔹 **STEP 1: Install Minikube & kubectl**

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
sudo snap install kubectl --classic
```

🔹 **STEP 2: Start Minikube**

```
minikube start
kubectl get nodes
```

🔹 **STEP 3: Create deployment.yaml**

```
nano deployment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-maven-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-maven
  template:
    metadata:
      labels:
        app: my-maven
```

```
    spec:
      containers:
      - name: my-maven
        image: dockerhubusername/my_maven_app:latest
        ports:
        - containerPort: 8080
```

🔹 STEP 4: Create service.yaml

nano service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: my-maven-service
spec:
  type: NodePort
  selector:
    app: my-maven
  ports:
    - port: 8080
      targetPort: 8080
      nodePort: 30007
```

🔹 STEP 5: Update Jenkinsfile (Kubernetes)

Replace Swarm stage with:

```
stage('Deploy to Kubernetes') {
    steps {
        sh 'minikube kubectl -- apply -f deployment.yaml'
        sh 'minikube kubectl -- apply -f service.yaml'
    }
}
```

🔹 STEP 6: Run Jenkins Job

Verify:

```
kubectl get pods
```

kubectl get svc
minikube service my-maven-service

🔵 QUESTION 4

Scheduled Kubernetes Deployment using Jenkins Cron

🔹 STEP 1: Enable Scheduler

Jenkins → Configure → Build periodically

*/1 * * * *

🔹 STEP 2: Observe Auto Deployment

Every minute:

- Build
- Docker push
- Kubernetes redeploy

Verify:

kubectl get pods

✅ FINAL EXAM ONE-LINE SUMMARY

> Jenkins is used to automate build, Docker is used to containerize the Maven application, Docker Swarm/Kubernetes is used for deployment, and Jenkins Cron is used for scheduling the pipeline execution.

If you want next:
👉 **Same FULL command-level steps for React**
👉 **Same FULL command-level steps for Flask**
👉 **Diagram explanation + viva answers**

Just say the word 😄

now same like this give me for Flask

😂🔥 perfect — now we do **FLASK exactly same level**,
NO SHORTCUTS, **NO THEORY SKIP**, **FULL COMMAND-LEVEL STEPS**, **EXAM + LAB READY**.

Save this. This is your **FLASK CI/CD COMPLETE GUIDE (ALL 4 QUESTIONS)**.

🟢 QUESTION 1

CI/CD Pipeline for Flask Application using Jenkins + Docker + Docker Swarm

---

🔹 STEP 1: Install Required Software (One Time)

```
sudo apt update
sudo apt install python3 python3-pip git -y
```

Verify:

```
python3 --version
pip3 --version
git --version
```

🔹 STEP 2: Install Docker

```
sudo apt install docker.io -y
sudo systemctl start docker
sudo systemctl enable docker
sudo usermod -aG docker $USER
newgrp docker
```

Verify:

```
docker --version
docker ps
```

🔹 STEP 3: Install Jenkins

sudo apt install fontconfig openjdk-17-jre -y

👉 Jenkins runs on Java

wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc
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
sudo apt update
sudo apt install jenkins -y
sudo systemctl start jenkins

🔹 STEP 4: Create Flask Application

mkdir flask_app
cd flask_app

Create app file:

nano app.py

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello from Flask CI/CD!"

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

🔹 STEP 5: Create requirements.txt

nano requirements.txt

```
flask
```

Install dependencies:

pip3 install -r requirements.txt

- **STEP 6: Push Flask Project to GitHub**

```
git init
git add .
git commit -m "Initial Flask app"
git branch -M main
git remote add origin https://github.com/<USERNAME>/flask_app.git
git push -u origin main
```

- **STEP 7: Create Dockerfile for Flask**

nano Dockerfile

```
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python","app.py"]
```

- **STEP 8: Initialize Docker Swarm**

docker swarm init --advertise-addr <YOUR-IP>

Example:

docker swarm init --advertise-addr 172.20.10.2

- **STEP 9: Create Jenkinsfile (Flask + Swarm)**

nano Jenkinsfile

```
pipeline {
    agent any

    environment {
        IMAGE_NAME = "dockerhubusername/flask_app"
```

```
        DOCKERHUB = credentials('dockerhub')
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'main', url: 'https://github.com/USERNAME/flask_app.git'
            }
        }

        stage('Install Dependencies') {
            steps {
                sh 'pip3 install -r requirements.txt'
            }
        }

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

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

        stage('Deploy to Swarm') {
            steps {
                sh '''
                docker service rm flask_app || true
                docker service create --name flask_app -p 5001:5000
$IMAGE_NAME:latest
                '''
            }
        }
    }
}
```

🔹 **STEP 10: Run Jenkins Pipeline**

Verify:

```
docker service ls
docker service ps flask_app
```

Open browser:

http://localhost:5001

🟢 QUESTION 2

Scheduled CI/CD Pipeline for Flask using Jenkins Cron

🔹 **STEP 1: Open Jenkins Job**

- Jenkins → Configure
- Enable **Build periodically**

Cron (every 1 minute):

```
* * * * *
```

🔹 **STEP 2: Observe Automatic Builds**

- Jenkins triggers automatically
- Docker image updated
- Swarm service redeployed

🟢 QUESTION 3

CI/CD Pipeline for Flask using Jenkins + Docker + Kubernetes (Minikube)

🔹 STEP 1: Install Minikube & kubectl

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
sudo snap install kubectl --classic

🔹 STEP 2: Start Minikube

minikube start
kubectl get nodes

🔹 STEP 3: Create deployment.yaml

nano deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: flask
  template:
    metadata:
      labels:
        app: flask
    spec:
      containers:
      - name: flask
        image: dockerhubusername/flask_app:latest
        ports:
        - containerPort: 5000
```

🔹 STEP 4: Create service.yaml

nano service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: flask-service
spec:
  type: NodePort
  selector:
    app: flask
  ports:
    - port: 5000
      targetPort: 5000
      nodePort: 30008
```

🔹 STEP 5: Update Jenkinsfile (Kubernetes)

Replace Swarm stage with:

```
stage('Deploy to Kubernetes') {
    steps {
        sh 'minikube kubectl -- apply -f deployment.yaml'
        sh 'minikube kubectl -- apply -f service.yaml'
    }
}
```

🔹 STEP 6: Run Jenkins Pipeline

Verify:

```
kubectl get pods
kubectl get svc
minikube service flask-service
```

🟢 QUESTION 4

Scheduled Kubernetes Deployment for Flask using Jenkins Cron

🔹 STEP 1: Enable Scheduler

Jenkins → Configure → Build periodically

*/1 * * * *

🔹 STEP 2: Observe Auto Deployment

Every minute:

- Flask app rebuilt
- Docker image updated
- Kubernetes redeployed

Verify:

kubectl get pods

✅ FINAL EXAM ONE-LINER (VERY IMPORTANT)

> Jenkins automates the Flask application build and deployment process, Docker containerizes the application, Docker Swarm or Kubernetes handles deployment, and Jenkins Cron schedules the pipeline execution.

🧠 REMEMBER THIS DIFFERENCE (FLASK vs MAVEN)

Maven	Flask
mvn clean package	pip install -r requirements.txt
Java app	Python app
port 8080	port 5000

If you want next:
👉 **React – FULL same level**
👉 **Common viva questions**
👉 **Diagrams explanation**

Just tell me 🔥

yes give me React

🔥😄 NICE — now we complete the **TRILOGY**.
Below is **REACT CI/CD – ALL 4 QUESTIONS**, **FULL COMMAND-LEVEL**, **LAB + EXAM READY**, same depth as **Maven & Flask**.

Save this. This is your **REACT DEVOPS FINAL SET**.

🔴 QUESTION 1

CI/CD Pipeline for React Application using Jenkins + Docker + Docker Swarm

---

🔹 STEP 1: Install Required Software (One Time)

```
sudo apt update
sudo apt install nodejs npm git -y
```

Verify:

```
node -v
npm -v
git --version
```

🔹 STEP 2: Install Docker

```
sudo apt install docker.io -y
sudo systemctl start docker
sudo systemctl enable docker
sudo usermod -aG docker $USER
newgrp docker
```

Verify:

```
docker --version
docker ps
```

🔹 STEP 3: Install Jenkins

```
sudo apt install fontconfig openjdk-17-jre -y
```

👉 Jenkins runs on Java

```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee
/usr/share/keyrings/jenkins-keyring.asc
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
sudo apt update
sudo apt install jenkins -y
sudo systemctl start jenkins
```

🔹 STEP 4: Create React Application

```
npx create-react-app react_app
cd react_app
```

Start locally:

```
npm start
```

Stop with Ctrl + C

🔹 STEP 5: Push React Project to GitHub

```
git init
git add .
git commit -m "Initial React app"
git branch -M main
git remote add origin https://github.com/<USERNAME>/react_app.git
git push -u origin main
```

🔹 STEP 6: Create Dockerfile for React

```
nano Dockerfile
```

```
# Build Stage
FROM node:18-alpine as build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
```

```
RUN npm run build

# Run Stage
FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx","-g","daemon off;"]
```

- **STEP 7: Initialize Docker Swarm**

docker swarm init --advertise-addr <YOUR-IP>

Example:

docker swarm init --advertise-addr 172.20.10.2

- **STEP 8: Create Jenkinsfile (React + Swarm)**

nano Jenkinsfile

```
pipeline {
    agent any

    environment {
        IMAGE_NAME = "dockerhubusername/react_app"
        DOCKERHUB = credentials('dockerhub')
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'main', url: 'https://github.com/USERNAME/react_app.git'
            }
        }

        stage('Install Dependencies') {
            steps {
                sh 'npm install'
            }
        }

        stage('Build React') {
```

```
        steps {
            sh 'npm run build'
        }
    }

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

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

    stage('Deploy to Swarm') {
        steps {
            sh '''
            docker service rm react_app || true
            docker service create --name react_app -p 3000:80 $IMAGE_NAME:latest
            '''
        }
    }
  }
}
```

- **STEP 9: Run Jenkins Pipeline**

Verify:

```
docker service ls
docker service ps react_app
```

Open browser:

http://localhost:3000

🔴 QUESTION 2

Scheduled CI/CD Pipeline for React using Jenkins Cron

🔹 STEP 1: Enable Scheduler

Jenkins → Configure → **Build periodically**

Cron (every 1 min):

* * * * *

🔹 STEP 2: Observe Auto Builds

- Jenkins triggers automatically
- React app rebuilds
- Docker image updated

🔴 QUESTION 3

CI/CD Pipeline for React using Jenkins + Docker + Kubernetes (Minikube)

🔹 STEP 1: Install Minikube & kubectl

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
sudo snap install kubectl --classic
```

🔹 STEP 2: Start Minikube

```
minikube start
kubectl get nodes
```

🔹 STEP 3: Create deployment.yaml

nano deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: react-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: react
  template:
    metadata:
      labels:
        app: react
    spec:
      containers:
      - name: react
        image: dockerhubusername/react_app:latest
        ports:
        - containerPort: 80
```

🔹 STEP 4: Create service.yaml

nano service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: react-service
spec:
  type: NodePort
  selector:
    app: react
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30009
```

🔹 STEP 5: Update Jenkinsfile (Kubernetes)

Replace Swarm stage with:

```
stage('Deploy to Kubernetes') {
    steps {
        sh 'minikube kubectl -- apply -f deployment.yaml'
        sh 'minikube kubectl -- apply -f service.yaml'
    }
}
```

🔹 STEP 6: Run Jenkins Pipeline

Verify:

```
kubectl get pods
kubectl get svc
minikube service react-service
```

🔴 QUESTION 4

Scheduled Kubernetes Deployment for React using Jenkins Cron

🔹 STEP 1: Enable Scheduler

Jenkins → Configure → **Build periodically**

```
*/1 * * * *
```

🔹 STEP 2: Observe Auto Deployment

Every minute:

- React rebuilt
- Docker image pushed
- Kubernetes redeployed

Verify:

kubectl get pods

🧠 DIFFERENCE SUMMARY (WRITE THIS IN EXAM)

App	Build Command	Runtime
Maven	mvn clean package	Java
Flask	pip install	Python
React	npm run build	Nginx
