**Deployment**

**A) Spring Boot Application Containerization**

(1) Create a new DockerFile: In order to containerize the application into a Docker container.

Make sure the application is packaged (in this case with maven) with this command : ./mvnw package

For example :

# Prioritize a base slim image containing Java runtime

FROM openjdk:11-jdk-slim as build

# Set the working directory location of the container

WORKDIR /app

# Add the application's jar to the container

COPY target/csvmanager-\*.jar app.jar

# Run the jar file inside the container

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

(2) Build Docker Image: A must in order to run the containerize application.

Make sure you’re in the app-level top project directory.

Name : csvmanager, Tag: latest.

For example : docker build -t csvmanager:latest .

(3) Locally Test Image: A must in order to push a valid image later to the registry.

docker run -p 8080:8080 csvmanager:latest

**B) Registry**

(1) Tagging the Docker Image: In order to push the docker image to a registry, you need to tag it with the Registry’s URL. You can use Docker Hub, AWS ECR etc.

For example : docker tag csvmanager:latest <registry-url>/csvmanager:latest

(2) Push the Docker Image: In order to push the docker image to a registry, you need to tag it with the Registry’s URL. You can use Docker Hub, AWS ECR etc.

**C) Kubernetes Configration File**

(1) Define a Deployment configuration file: Define a deployment.yaml file responsible for guiding Kubernetes how to run the application, specifying the container/s, label, resources etc.

apiVersion: apps/v1

kind: Deployment

metadata:

name: csvmanager-deployment

spec:

replicas: 3

selector:

matchLabels:

app: csvmanager

template:

metadata:

labels:

app: csvmanager

spec:

containers:

- name: csvmanager

image: <registry-url>/csvmanager:latest

ports:

- containerPort: 8080

resources:

requests:

memory: "512Mi"

cpu: "500m"

limits:

memory: "1Gi"

cpu: "1000m"

(2) Define a Service configuration file: Create a ‘service.yaml’ to define how the application’s service will be exposed in the Cluster.

apiVersion: v1

kind: Service

metadata:

name: csvmanager-service

spec:

selector:

app: csvmanager

ports:

- protocol: TCP

port: 80

targetPort: 8080

type: LoadBalancer

**D) Deployment to Kubernetes**

(1) Connect to the Kubernetes Cluster: Ensure your personal kubeconfig is correctly setup.

Use the command ‘kubectl’ to test the interaction in your cluster.

(2) Apply configuration files to the cluster

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

(3) Verify deployment:

Check that your pods are running: kubectl get pods

Verify your service is up and running: kubectl get service csvmanager-service

Then you could use the external IP assigned to the deployed LoadBalancer.

**E) CI/CD**

(1) **CI/CD Pipelines**: Use tools like Jenkins, or GitHub Actions to automate the deployment process. You can create jobs that could be responsible for running tests, building the Docker image, pushing to the registry, and updating the Kubernetes cluster.

(2) **Helm Charts**: It’s a good practice to define Helm Charts in order to manage your Kubernetes application. They help you define, install, and upgrade even the most complex Kubernetes applications.

(3) **Monitoring and Logging**: Implement monitoring and logging with tools such as Prometheus, Grafana.