**Amir Mamdouh**

**Task3 Kubernetes full-stack**

## 1. Overview

This document provides a detailed explanation of deploying a full-stack application on Kubernetes, including a frontend, backend, and a MySQL database. The application ensures persistence for static files and database data.

---

## 2. Application Components

- **Frontend:** A React.js application that fetches data from the backend.
- **Backend:** A Spring Boot application exposing APIs.
- **Database:** MySQL database running in a Kubernetes cluster.
- **Persistent Storage:** Ensures that database data and static assets persist across container restarts.
- **Networking:** Services and Ingress for communication between components.

---

## 3. Directory Structure

```
/task6_app_pvc/
├── backend/ (Spring Boot Backend)
├── frontend/ (React Frontend)
├── backendManifest.yml (Backend Deployment)
├── backendService.yml (Backend Service)
├── frontendManifest.yml (Frontend Deployment)
├── frontService.yml (Frontend Service)
├── mysql_pvc.yml (MySQL Persistent Volume Claim)
├── mysqlStatefulset.yml (MySQL Deployment)
├── mysqlService.yml (MySQL Service)
├── front-ingress.yml (Ingress Controller for Routing)
├── mysql-secrets.yml (Database Secrets)
```

---

## 4. Backend (Spring Boot)

### Application Properties (backend/src/main/resources/application.properties)

```
server.port=8080
spring.datasource.url=jdbc:mysql://database-service:3306/amir
spring.datasource.username=root
spring.datasource.password=amir
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
```

### Backend Controller (ItemController.java)

```
@RestController
@RequestMapping("/api/items")
public class ItemController {
```

```
    @Autowired
    private ItemService itemService;

    @GetMapping
    public List<String> getItems() {
        return itemService.getAllItems();
    }
}
```

### CORS Configuration (CorsConfig.java)

```
@Configuration
public class CorsConfig {
    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**").allowedOrigins("*");
            }
        };
    }
}
```

### Backend Deployment (backendManifest.yml)

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

---

## 5. Frontend (React.js)

### App Component (src/App.js)

```
import React from "react";
import ItemList from "./components/ItemList";

function App() {
    return (
        <div>
            <h1>Welcome to My React App</h1>
            <ItemList />
        </div>
    );
```

```
}
export default App;
```

### Fetching Items from Backend (src/components/ItemList.js)

```
import React, { useEffect, useState } from "react";

const ItemList = () => {
    const [items, setItems] = useState([]);

    useEffect(() => {
        fetch("http://localhost:8080/api/items")
            .then(response => response.json())
            .then(data => setItems(data))
            .catch(error => console.error("Error fetching items:", error));
    }, []);

    return (
        <div>
            <h2>Items from Backend</h2>
            <ul>
                {items.map((item, index) => (
                    <li key={index}>{item}</li>
                ))}
            </ul>
        </div>
    );
};
export default ItemList;
```

### Frontend Deployment (frontendManifest.yml)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image: my-frontend-image:latest
        ports:
        - containerPort: 3000
```

---

## 6. Database (MySQL)

### MySQL StatefulSet (mysqlStatefulset.yml)

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
```

```
spec:
  selector:
    matchLabels:
      app: mysql
  serviceName: "mysql"
  replicas: 1
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:latest
        envFrom:
          - secretRef:
              name: mysql-cred
        volumeMounts:
          - name: mysql-storage
            mountPath: /var/lib/mysql
  volumeClaimTemplates:
  - metadata:
      name: mysql-storage
    spec:
      accessModes: ["ReadWriteMany"]
      storageClassName: rook-cephfs
      resources:
        requests:
          storage: 5Gi
```

---

## 7. Services and Ingress

### Backend Service (backendService.yml)

```
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app: backend
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
  type: ClusterIP
```

### Ingress (front-ingress.yml)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend-ingress
spec:
  rules:
  - host: myapp.local
    http:
      paths:
      - path: /
        pathType: Prefix
```

```
        backend:
          service:
            name: frontend-service
            port:
              number: 80
```

---

## 8. Deployment Steps

```
kubectl apply -f mysql-secrets.yml
kubectl apply -f mysqlStatefulset.yml
kubectl apply -f backendManifest.yml
kubectl apply -f backendService.yml
kubectl apply -f frontendManifest.yml
kubectl apply -f frontService.yml
kubectl apply -f front-ingress.yml
```

This ensures your full-stack application is deployed and accessible via Ingress.