Multi-Container Application: Task Management System

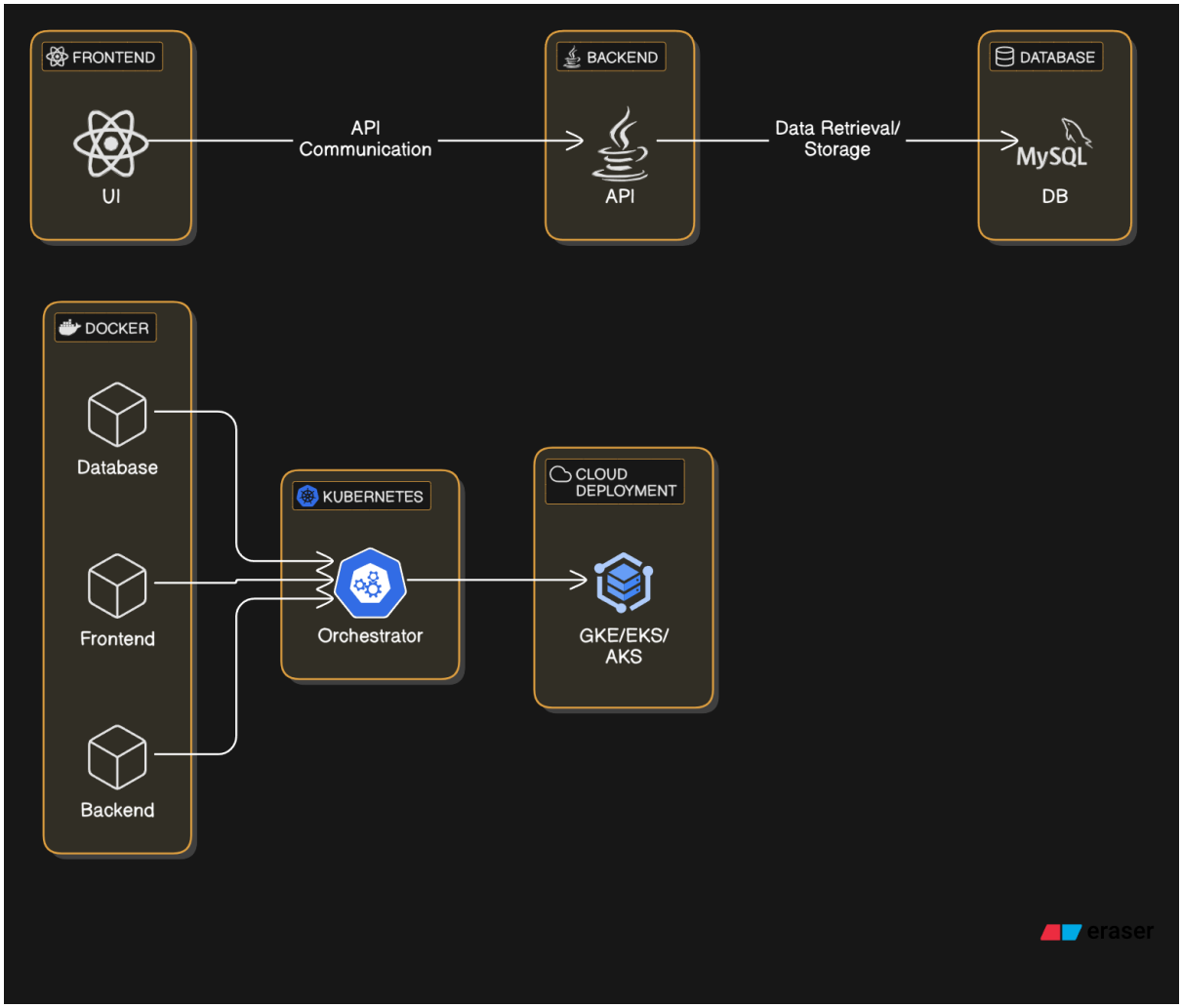
Overview

This project is a **cloud-native, full-stack Multi-Container Application** designed to efficiently manage tasks. It features a ReactJS frontend, a Spring Boot backend, and a MySQL database. The application is fully containerized using Docker and orchestrated using Kubernetes, enabling a scalable and cloud-deployable architecture.

The system allows users to create, view, update, and delete tasks with additional features such as priorities, due dates, and status tracking.

Project Architecture

Architecture Diagram



Architecture Overview

1. **Frontend**:
    - Built with ReactJS.
    - Provides a user-friendly interface for managing tasks.
    - Communicates with the backend via REST APIs.
2. **Backend**:
    - Developed with Java Spring Boot.
    - Manages business logic and processes user requests.
    - Interfaces with the MySQL database for data storage and retrieval.
3. **Database**:
    - Powered by MySQL.
    - Stores user and task-related information.
4. **Containerization**:
    - Dockerized components (frontend, backend, and database) ensure consistency across environments.
5. **Orchestration**:
    - Kubernetes orchestrates the deployment, scaling, and load balancing of the application.

Technologies Used

- **Frontend**: ReactJS, Axios
- **Backend**: Java, Spring Boot, Spring Data JPA
- **Database**: MySQL
- **Containerization**: Docker
- **Orchestration**: Kubernetes
- **Cloud**: Deployable on AWS/GCP/Azure Kubernetes services

Features

1. **Task Management**:
    - Create, view, update, and delete tasks.
    - Track task details such as:
        - Title
        - Description
        - Priority (Low, Medium, High)
        - Status (To Do, In Progress, Done)
        - Due Date
2. **Search and Filter**:
    - Search tasks by title or description.
    - Filter tasks by status, priority, or due date.
3. **Responsive UI**:
    - Built with ReactJS for an intuitive user experience.
4. **Scalable Deployment**:

- The Multi-Container Application is fully orchestrated with Kubernetes for scalability and high availability.

Setup Instructions

Step 1: Prerequisites

Ensure the following tools are installed:

- Node.js (for the frontend)
- JDK 11 or later (for the backend)
- Docker (for containerization)
- Kubectl (for Kubernetes cluster management)
- Minikube or a cloud Kubernetes cluster

Step 2: Running Locally

Backend Setup

1. Navigate to the `backend` directory:

```
bash
CopyEdit
cd backend
```

2. Build and run the Spring Boot application:

```
bash
CopyEdit
./mvnw spring-boot:run
```

Frontend Setup

1. Navigate to the `frontend` directory:

```
bash
CopyEdit
cd frontend
```

2. Install dependencies and start the ReactJS application:

```
bash
CopyEdit
npm install
npm start
```

Database Setup

1. Run the MySQL container:

```
bash
CopyEdit
docker run -d -p 3306:3306 --name mysql-container -e
MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=task_db mysql
```

Step 3: Containerization

Build Docker Images

- Backend:

```
bash
CopyEdit
docker build -t task-backend ./backend
```

- Frontend:

```
bash
CopyEdit
docker build -t task-frontend ./frontend
```

Run with Docker Compose

1. Create a `docker-compose.yml` file:

```
yaml
CopyEdit
version: "3.8"
services:
  backend:
    build: ./backend
    ports:
      - "8080:8080"
    depends_on:
      - database
  frontend:
    build: ./frontend
    ports:
      - "3000:3000"
  database:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: task_db
```

2. Start all services:

```
bash
CopyEdit
docker-compose up
```

Step 4: Deploying the Multi-Container Application with Kubernetes

Steps:

1. Write Deployment and Service YAML files for each component (frontend, backend, database).
2. Deploy all resources to the Kubernetes cluster:

   ```
   bash
   CopyEdit
   kubectl apply -f kubernetes/
   ```

3. Expose the frontend using a Kubernetes Service (LoadBalancer or Ingress).
4. Access the Multi-Container Application:
   - Open the public URL (provided by Kubernetes) in your browser to access the frontend.

Future Enhancements

1. **Authentication**:
   - Implement secure user login and registration using JWT or OAuth.
2. **Notifications**:
   - Add email or in-app reminders for task due dates.
3. **Collaboration**:
   - Enable multi-user support for task sharing and collaboration.
4. **Advanced Filters**:
   - Add options for filtering tasks by multiple criteria.