Full-Stack Web Application Kubernetes Deployment

Author: Bhanu Reddy
Roll No: 2022bcd0026

This repository contains all the necessary files to containerize and deploy a full-stack web application (React frontend, Node.js backend, MySQL database) using Minikube.

Table of Contents

Project Overview

This project demonstrates how to:

1. Create a simple full-stack application with React, Node.js, and MySQL
2. Containerize each component with Docker
3. Configure and deploy the application on Kubernetes using Minikube
4. Set up proper communication between components using Kubernetes Services
5. Manage configuration and secrets using Kubernetes ConfigMaps and Secrets
6. Create persistent storage for database data

Prerequisites

Ensure you have the following installed on your system:

- Docker
- Minikube
- kubectl
- Node.js and npm
- Git

Project Structure

```
kubernetes-fullstack-app/
├── backend/                    # Node.js backend
│   ├── src/
│   │   ├── controllers/
│   │   ├── models/
│   │   ├── routes/
│   │   └── app.js
│   ├── package.json
│   ├── Dockerfile
│   └── .env
├── frontend/                   # React frontend
│   ├── public/
│   ├── src/
│   ├── package.json
│   ├── Dockerfile
│   └── .env
├── k8s/                        # Kubernetes configuration files
│   ├── mysql/
│   │   ├── mysql-configmap.yaml
│   │   ├── mysql-secret.yaml
│   │   ├── mysql-pv.yaml
│   │   ├── mysql-pvc.yaml
│   │   ├── mysql-deployment.yaml
│   │   └── mysql-service.yaml
│   ├── backend/
│   │   ├── backend-configmap.yaml
│   │   ├── backend-deployment.yaml
│   │   └── backend-service.yaml
│   └── frontend/
│       ├── frontend-configmap.yaml
│       ├── frontend-deployment.yaml
│       └── frontend-service.yaml
├── docker-compose.yaml         # Docker Compose configuration
└── README.md                   # This file
```

alt text

Step 1: Setting Up the Application

Backend Setup

1. Create a Node.js Express application:

```
mkdir -p backend/src
cd backend
npm init -y
npm install express mysql2 cors dotenv
touch src/app.js
```

2. Create a simple CRUD API in `src/app.js`:

```
const express = require('express');
const mysql = require('mysql2/promise');
const cors = require('cors');
require('dotenv').config();

const app = express();
app.use(cors());
app.use(express.json());

// MySQL connection pool
const pool = mysql.createPool({
  host: process.env.DB_HOST || 'mysql',
  user: process.env.DB_USER || 'root',
  password: process.env.DB_PASSWORD || 'password',
  database: process.env.DB_NAME || 'tasksdb',
  waitForConnections: true,
  connectionLimit: 10,
  queueLimit: 0
});

// Health check endpoint
app.get('/api/health', (req, res) => {
  res.status(200).json({ status: 'healthy' });
});

// Get all tasks
app.get('/api/tasks', async (req, res) => {
  try {
    const [rows] = await pool.query('SELECT * FROM tasks');
    res.json(rows);
  } catch (error) {
    console.error('Error fetching tasks:', error);
    res.status(500).json({ error: 'Failed to fetch tasks' });
  }
});

// Get a specific task
```

```
app.get('/api/tasks/:id', async (req, res) => {
  try {
    const [rows] = await pool.query('SELECT * FROM tasks WHERE id = ?',
[req.params.id]);
    if (rows.length === 0) {
      return res.status(404).json({ error: 'Task not found' });
    }
    res.json(rows[0]);
  } catch (error) {
    console.error('Error fetching task:', error);
    res.status(500).json({ error: 'Failed to fetch task' });
  }
});

// Create a new task
app.post('/api/tasks', async (req, res) => {
  const { title, description } = req.body;

  if (!title) {
    return res.status(400).json({ error: 'Title is required' });
  }

  try {
    const [result] = await pool.query(
      'INSERT INTO tasks (title, description) VALUES (?, ?)',
      [title, description || '']
    );

    res.status(201).json({
      id: result.insertId,
      title,
      description: description || ''
    });
  } catch (error) {
    console.error('Error creating task:', error);
    res.status(500).json({ error: 'Failed to create task' });
  }
});

// Update a task
app.put('/api/tasks/:id', async (req, res) => {
  const { title, description } = req.body;

  try {
    const [result] = await pool.query(
      'UPDATE tasks SET title = ?, description = ? WHERE id = ?',
      [title, description || '', req.params.id]
    );

    if (result.affectedRows === 0) {
      return res.status(404).json({ error: 'Task not found' });
    }
```

```
    res.json({ id: req.params.id, title, description });
  } catch (error) {
    console.error('Error updating task:', error);
    res.status(500).json({ error: 'Failed to update task' });
  }
});

// Delete a task
app.delete('/api/tasks/:id', async (req, res) => {
  try {
    const [result] = await pool.query('DELETE FROM tasks WHERE id = ?',
[req.params.id]);

    if (result.affectedRows === 0) {
      return res.status(404).json({ error: 'Task not found' });
    }

    res.status(204).end();
  } catch (error) {
    console.error('Error deleting task:', error);
    res.status(500).json({ error: 'Failed to delete task' });
  }
});

// Initialize database function
async function initDb() {
  try {
    // Create connection to MySQL server (without database selection)
    const tempPool = mysql.createPool({
      host: process.env.DB_HOST || 'mysql',
      user: process.env.DB_USER || 'root',
      password: process.env.DB_PASSWORD || 'password',
      waitForConnections: true,
      connectionLimit: 10,
      queueLimit: 0
    });

    // Create database if it doesn't exist
    await tempPool.query(`CREATE DATABASE IF NOT EXISTS
${process.env.DB_NAME || 'tasksdb'}`);

    // Connect to the database
    await pool.query(`
      CREATE TABLE IF NOT EXISTS tasks (
        id INT AUTO_INCREMENT PRIMARY KEY,
        title VARCHAR(255) NOT NULL,
        description TEXT,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
      )
    `);

    console.log('Database and tables initialized successfully');
  } catch (error) {
```

```
    console.error('Error initializing database:', error);
    // Retry after delay
    setTimeout(initDb, 5000);
  }
}

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
  // Initialize database
  initDb();
});
```

3. Create a `package.json` file:

```
{
  "name": "backend-2022bcd0026-bhanu-reddy",
  "version": "1.0.0",
  "description": "Backend API for task management",
  "main": "src/app.js",
  "scripts": {
    "start": "node src/app.js",
    "dev": "nodemon src/app.js"
  },
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "mysql2": "^3.2.0"
  },
  "devDependencies": {
    "nodemon": "^2.0.22"
  }
}
```

PROF

Frontend Setup

1. Create a React application:

```
npx create-react-app frontend
cd frontend
npm install axios
```

2. Replace the content of `src/App.js` with:

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import './App.css';

const API_URL = process.env.REACT_APP_API_URL ||
'http://localhost:5000/api';

function App() {
  const [tasks, setTasks] = useState([]);
  const [newTask, setNewTask] = useState({ title: '', description: ''
});
  const [editingTask, setEditingTask] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  // Fetch tasks on component mount
  useEffect(() => {
    fetchTasks();
  }, []);

  const fetchTasks = async () => {
    try {
      setLoading(true);
      const response = await axios.get(`${API_URL}/tasks`);
      setTasks(response.data);
      setError(null);
    } catch (err) {
      console.error('Error fetching tasks:', err);
      setError('Failed to fetch tasks. Please try again later.');
    } finally {
      setLoading(false);
    }
  };

  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setNewTask({ ...newTask, [name]: value });
  };

  const handleEditChange = (e) => {
    const { name, value } = e.target;
    setEditingTask({ ...editingTask, [name]: value });
  };

  const createTask = async (e) => {
    e.preventDefault();
    if (!newTask.title.trim()) return;

    try {
      await axios.post(`${API_URL}/tasks`, newTask);
      setNewTask({ title: '', description: '' });
      fetchTasks();
```

```
    } catch (err) {
      console.error('Error creating task:', err);
      setError('Failed to create task. Please try again.');
    }
  };

  const startEditing = (task) => {
    setEditingTask({ ...task });
  };

  const cancelEditing = () => {
    setEditingTask(null);
  };

  const updateTask = async (e) => {
    e.preventDefault();
    if (!editingTask.title.trim()) return;

    try {
      await axios.put(`${API_URL}/tasks/${editingTask.id}`, {
        title: editingTask.title,
        description: editingTask.description
      });
      setEditingTask(null);
      fetchTasks();
    } catch (err) {
      console.error('Error updating task:', err);
      setError('Failed to update task. Please try again.');
    }
  };

  const deleteTask = async (id) => {
    if (!window.confirm('Are you sure you want to delete this task?'))
return;

    try {
      await axios.delete(`${API_URL}/tasks/${id}`);
      fetchTasks();
    } catch (err) {
      console.error('Error deleting task:', err);
      setError('Failed to delete task. Please try again.');
    }
  };

  return (
    <div className="App">
      <header className="App-header">
        <h1>Task Manager</h1>
        <h2>by Bhanu Reddy (2022bcd0026)</h2>
      </header>

      <div className="container">
        {error && <div className="error-message">{error}</div>}
```

PROF

```
        <div className="form-container">
          <h2>Add New Task</h2>
          <form onSubmit={createTask}>
            <div className="form-group">
              <label htmlFor="title">Title:</label>
              <input
                type="text"
                id="title"
                name="title"
                value={newTask.title}
                onChange={handleInputChange}
                required
              />
            </div>
            <div className="form-group">
              <label htmlFor="description">Description:</label>
              <textarea
                id="description"
                name="description"
                value={newTask.description}
                onChange={handleInputChange}
              />
            </div>
            <button type="submit" className="btn">Add Task</button>
          </form>
        </div>

        <div className="tasks-container">
          <h2>Tasks</h2>
          {loading ? (
            <p>Loading tasks...</p>
          ) : tasks.length === 0 ? (
            <p>No tasks found. Add a new task to get started.</p>
          ) : (
            <ul className="tasks-list">
              {tasks.map((task) => (
                <li key={task.id} className="task-item">
                  {editingTask && editingTask.id === task.id ? (
                    <form onSubmit={updateTask} className="edit-form">
                      <div className="form-group">
                        <label htmlFor={`edit-title-${task.id}`}>Title:
</label>

                        <input
                          type="text"
                          id={`edit-title-${task.id}`}
                          name="title"
                          value={editingTask.title}
                          onChange={handleEditChange}
                          required
                        />
                      </div>
                      <div className="form-group">
```

PROF

```
                        <label htmlFor={`edit-
description-${task.id}`}>Description:</label>
                        <textarea
                          id={`edit-description-${task.id}`}
                          name="description"
                          value={editingTask.description}
                          onChange={handleEditChange}
                        />
                      </div>
                      <div className="button-group">
                        <button type="submit" className="btn
save">Save</button>
                        <button type="button" className="btn cancel"
onClick={cancelEditing}>Cancel</button>
                      </div>
                    </form>
                  ) : (
                    <>
                      <div className="task-content">
                        <h3>{task.title}</h3>
                        <p>{task.description}</p>
                        <small>Created: {new
Date(task.created_at).toLocaleString()}</small>
                      </div>
                      <div className="task-actions">
                        <button onClick={() => startEditing(task)}
className="btn edit">Edit</button>
                        <button onClick={() => deleteTask(task.id)}
className="btn delete">Delete</button>
                      </div>
                    </>
                  )}
                </li>
              ))}
            </ul>
          )}
        </div>
      </div>
    </div>
  );
}

export default App;
```

3. Add some basic styling in `src/App.css`:

```
.App {
  text-align: center;
  font-family: Arial, sans-serif;
}
```

```
.App-header {
  background-color: #282c34;
  padding: 20px;
  color: white;
}

.container {
  max-width: 1200px;
  margin: 0 auto;
  padding: 20px;
}

.form-container {
  background-color: #f5f5f5;
  padding: 20px;
  border-radius: 8px;
  margin-bottom: 20px;
}

.form-group {
  margin-bottom: 15px;
  text-align: left;
}

.form-group label {
  display: block;
  margin-bottom: 5px;
  font-weight: bold;
}

.form-group input,
.form-group textarea {
  width: 100%;
  padding: 8px;
  border: 1px solid #ddd;
  border-radius: 4px;
}

.btn {
  background-color: #4CAF50;
  color: white;
  border: none;
  padding: 10px 15px;
  cursor: pointer;
  border-radius: 4px;
  font-size: 16px;
}

.btn:hover {
  background-color: #45a049;
}

.tasks-list {
```

```
  list-style: none;
  padding: 0;
}

.task-item {
  background-color: #fff;
  border: 1px solid #ddd;
  border-radius: 8px;
  margin-bottom: 15px;
  padding: 15px;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.task-content {
  text-align: left;
  flex-grow: 1;
}

.task-content h3 {
  margin-top: 0;
}

.task-content p {
  color: #666;
}

.task-actions {
  display: flex;
  gap: 10px;
}

.edit {
  background-color: #2196F3;
}

.edit:hover {
  background-color: #0b7dda;
}

.delete {
  background-color: #f44336;
}

.delete:hover {
  background-color: #d32f2f;
}

.save {
  background-color: #4CAF50;
}
```

```
.cancel {
  background-color: #9e9e9e;
}

.edit-form {
  width: 100%;
}

.button-group {
  display: flex;
  gap: 10px;
  justify-content: flex-end;
}

.error-message {
  background-color: #ffebee;
  color: #c62828;
  padding: 10px;
  border-radius: 4px;
  margin-bottom: 20px;
}
```

Database Setup

We'll be using MySQL as our database. The database initialization will be handled by our backend application, but we need to prepare the Kubernetes files for it.

Step 2: Containerization

Backend Dockerfile

Create a `Dockerfile` in the backend directory:

PROF

```
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./

RUN npm install --production

COPY . .

EXPOSE 5000

CMD ["node", "src/app.js"]
```

Frontend Dockerfile

Create a `Dockerfile` in the frontend directory:

```
# Stage 1: Build the React application
FROM node:18-alpine AS build

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

# Pass the correct backend API URL as a build argument
ARG REACT_APP_API_URL=http://192.168.58.2:30001/api
ENV REACT_APP_API_URL=${REACT_APP_API_URL}

RUN npm run build

# Stage 2: Serve the React application using Nginx
FROM nginx:alpine

COPY --from=build /app/build /usr/share/nginx/html

# Expose port 80
EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

Building Images

```
# Build backend image
cd backend
docker build -t bhanureddy1973/backend-2022bcd0026:latest .
```

Push the image to a container registry (if necessary):

```
docker push bhanureddy1973/backend-2022bcd0026:latest
```

alt text

```
# Build frontend image
cd ../frontend
docker build -t bhanureddy1973/frontend-2022bcd0026-bhanu-reddy:latest .
```

Push the image to a container registry (if necessary):

```
docker push bhanureddy1973/frontend-2022bcd0026-bhanu-reddy:latest
```

alt text

Step 3: Kubernetes Configuration

Create Kubernetes Configuration Files

Let's create the necessary Kubernetes configuration files.

MySQL ConfigMap, Secret, and PV/PVC

`k8s/mysql/mysql-configmap.yaml`:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: db-config-2022bcd0026
data:
  DB_HOST: mysql
  DB_NAME: tasksdb
```

`k8s/mysql/mysql-secret.yaml`:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret-2022bcd0026
type: Opaque
data:
  mysql-root-password: cGFzc3dvcmQ=  # 'password' encoded in base64
  mysql-user: cm9vdA==  # 'root' encoded in base64
  mysql-password: cGFzc3dvcmQ=  # 'password' encoded in base64
```

`k8s/mysql/mysql-pv.yaml`:

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv-2022bcd0026
  labels:
```

```
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

k8s/mysql/mysql-pvc.yaml:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pvc-2022bcd0026
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

k8s/mysql/mysql-deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-2022bcd0026
spec:
  selector:
    matchLabels:
      app: mysql
      owner: 2022bcd0026
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql
        owner: 2022bcd0026
    spec:
      containers:
      - image: mysql:8.0
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
```

```
            valueFrom:
              secretKeyRef:
                name: mysql-secret-2022bcd0026
                key: mysql-root-password
          - name: MYSQL_DATABASE
            valueFrom:
              configMapKeyRef:
                name: db-config-2022bcd0026
                key: DB_NAME
          ports:
          - containerPort: 3306
            name: mysql
          volumeMounts:
          - name: mysql-persistent-storage
            mountPath: /var/lib/mysql
            - name: init-script
            mountPath: /docker-entrypoint-initdb.d/
          resources:
            requests:
              memory: "256Mi"
              cpu: "250m"
        volumes:
        - name: mysql-persistent-storage
          persistentVolumeClaim:
            claimName: mysql-pvc-2022bcd0026
         - name: init-script
          configMap:
            name: mysql-init-script
```

k8s/mysql/mysql-service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: mysql-2022bcd0026
spec:
  selector:
    app: mysql
    owner: 2022bcd0026
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
  clusterIP: None
```

PROF

Backend ConfigMap and Deployment

k8s/backend/backend-configmap.yaml:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: backend-config
data:
  PORT: "5000"
  DB_HOST: "mysql-2022bcd0026.default.svc.cluster.local"
  DB_USER: "root"
  DB_PASSWORD: "password"
  DB_NAME: "tasks"
  NODE_ENV: "production"
```

`k8s/backend/backend-deployment.yaml`:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deploy-2022bcd0026
spec:
  replicas: 1
  selector:
    matchLabels:
      app: backend
      owner: 2022bcd0026
  template:
    metadata:
      labels:
        app: backend
        owner: 2022bcd0026
    spec:
      containers:
      - name: backend
        image: bhanureddy1973/2022bcd0026-bhanu-reddy:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 5000
        env:
        - name: PORT
          valueFrom:
            configMapKeyRef:
              name: backend-config
              key: PORT
        - name: DB_HOST
          valueFrom:
            configMapKeyRef:
              name: backend-config
              key: DB_HOST
        - name: DB_NAME
```

```
          valueFrom:
            configMapKeyRef:
              name: backend-config
              key: DB_NAME
        - name: DB_USER
          valueFrom:
            configMapKeyRef:
              name: backend-config
              key: DB_USER
        - name: DB_PASSWORD
          valueFrom:
            configMapKeyRef:
              name: backend-config
              key: DB_PASSWORD
              ```

`k8s/backend/backend-service.yaml`:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend-service-2022bcd0026
spec:
  selector:
    app: backend
    owner: 2022bcd0026
  ports:
  - port: 5000
    targetPort: 5000
    nodePort:  3001
  type: NodePort
```
```

PROF

**Frontend Deployment and Service**

`k8s/frontend/frontend-deployment.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deploy-2022bcd0026
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
      owner: 2022bcd0026
  template:
    metadata:
      labels:
```
```

```
        app: frontend
        owner: 2022bcd0026
    spec:
      containers:
      - name: frontend
        image: bhanureddy1973/frontend-2022bcd0026-bhanu-reddy:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 80
        env:
        - name: REACT_APP_API_URL
          valueFrom:
            configMapKeyRef:
              name: frontend-config
              key: REACT_APP_API_URL
              ```

`k8s/frontend/frontend-service.yaml`:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-service-2022bcd0026
  labels:
    app: frontend
spec:
  selector:
    app: frontend

  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  type: NodePort
```

PROF

`k8s/frontend/frontend-config.yaml`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: frontend-config
data:
  REACT_APP_API_URL: "http://$(minikube ip):30001/api"
  APP_ENV: "production"
```

Step 4: Deploying to Minikube

Now we'll deploy our application to Minikube:

1. Start Minikube:

```
minikube start
```

alt text

2. Set your Docker environment to use Minikube's Docker daemon:

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

3. Build the Docker images:

```
cd backend
docker build -t 2022bcd0026-bhanu-reddy .```
![alt text](image-8.png)
![alt text](image-9.png)


```bash
cd ../frontend
docker build -t frontend-2022bcd0026-bhanu-reddy .
```

alt text

```
docker-compose.yml
version: '3.8'

services:
  # MySQL Database
  mysql:
    image: mysql:8.0
    container_name: mysql-2022bcd0026
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: tasksdb
    ports:
      - "3306:3306"
    volumes:
      - mysql-data:/var/lib/mysql
    networks:
      - app-network
    healthcheck:
```

PROF
```

```yaml
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 10s
      timeout: 5s
      retries: 5

  # Node.js Backend
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    image: 2022bcd0026-bhanu-reddy:latest
    container_name: backend-2022bcd0026
    restart: unless-stopped
    environment:
      PORT: 5000
      DB_HOST: mysql
      DB_USER: root
      DB_PASSWORD: password
      DB_NAME: tasksdb
    ports:
      - "5000:5000"
    depends_on:
      - mysql
    networks:
      - app-network

  # React Frontend
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
      args:
        - REACT_APP_API_URL=http://localhost:5000/api
    image: frontend-2022bcd0026-bhanu-reddy:latest
    container_name: frontend-2022bcd0026
    restart: unless-stopped
    ports:
      - "80:80"
    depends_on:
      - backend
    networks:
      - app-network

networks:
  app-network:
    driver: bridge

volumes:
  mysql-data:
    driver: local
```

PROF

alt text
alt text

4. Deploy MySQL:

```
#move to the k8s/mysql directory
kubectl apply -f mysql-pv.yaml
kubectl apply -f mysql-pvc.yaml
kubectl apply -f mysql-configmap.yaml
kubectl apply -f mysql-secret.yaml
kubectl apply -f mysql-deployment.yaml
kubectl apply -f mysql-service.yaml
```

alt text
alt text
alt text
alt text

5. Deploy the backend:

```
kubectl apply -f k8s/backend/backend-configmap.yaml
kubectl apply -f k8s/backend/backend-deployment.yaml
kubectl apply -f k8s/backend/backend-service.yaml
```

alt text

6. Deploy the frontend:

```
kubectl apply -f k8s/frontend/frontend-deployment.yaml
kubectl apply -f k8s/frontend/frontend-service.yaml
kubectl apply -f k8s/frontend/frontend-configmap.yaml
```

alt text

7. Check that everything is running:

Verify Services:

```
#FRONTEND SERVICE:
kubectl get svc -l app=frontend
#BACKEND SERVIICE:
kubectl get svc -l app=backend
#MYSQL SERVICE:
kubectl get svc -l app=backend
```

```
#TO GET ALL PODS AND SERVICES
kubectl get pods
kubectl get services
```

alt text
alt text

```
#Delete all existing pods :-
Delete all existing pods
```

MySQL Tasks Database Setup

Access MySQL

```
kubectl exec -it $(kubectl get pod -l app=mysql -o
jsonpath='{.items[0].metadata.name}') -- /bin/bash

# Inside the pod
mysql -u root -p
```

###Enter the password (password) when prompted. mentioned in the secret.yaml file

Steps to Set Up the Tasks Database

1. Use the `tasksdb` Database

Switch to the `tasksdb` database:

```
USE tasksdb;
```

2. Create a Table for Tasks

Create a table named `tasks` with columns for task ID, name, status, and creation date:

```
CREATE TABLE tasks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    task_name VARCHAR(255) NOT NULL,
    status VARCHAR(50) DEFAULT 'Pending',
```

```
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

3. Insert Sample Tasks

Insert a few tasks into the `tasks` table:

```
INSERT INTO tasks (task_name, status) VALUES
('Complete Kubernetes setup', 'Pending'),
('Test MySQL connectivity', 'Done'),
('Deploy backend service', 'In Progress'),
('Configure frontend application', 'Pending');
```

4. Verify the Inserted Tasks

Retrieve all tasks from the `tasks` table:

```
SELECT * FROM tasks;
```

You should see output similar to this:

```
+----+----------------------------+-------------+---------------------+
| id | task_name                  | status      | created_at          |
+----+----------------------------+-------------+---------------------+
|  1 | Complete Kubernetes setup  | Pending     | 2025-04-05 11:00:00 |
|  2 | Test MySQL connectivity    | Done        | 2025-04-05 11:05:00 |
|  3 | Deploy backend service     | In Progress | 2025-04-05 11:10:00 |
|  4 | Configure frontend application | Pending | 2025-04-05 11:15:00 |
+----+----------------------------+-------------+---------------------+
```

PROF

5. Exit MySQL

Once done, exit the MySQL shell:

```
EXIT;
```

alt text

Troubleshooting

- Ensure that MySQL is running and accessible.
- Verify that the `tasksdb` database exists or create it if necessary.

- Check for any syntax errors in SQL commands.

Step 5: Testing the Application

Let's test our deployed application:

```
# Get Minikube IP
MINIKUBE_IP=$(minikube ip)
echo "Minikube IP: $MINIKUBE_IP"

# Get service NodePorts
BACKEND_PORT=$(kubectl get svc backend-service-YOUR_REGISTER_NUMBER -o
jsonpath='{.spec.ports[0].nodePort}')
FRONTEND_PORT=$(kubectl get svc frontend-service-YOUR_REGISTER_NUMBER -o
jsonpath='{.spec.ports[0].nodePort}')

echo "Backend available at: http://$MINIKUBE_IP:$BACKEND_PORT"
echo "Frontend available at: http://$MINIKUBE_IP:$FRONTEND_PORT"

# Test backend API using curl
curl http://$MINIKUBE_IP:$BACKEND_PORT/api/health
curl http://$MINIKUBE_IP:$BACKEND_PORT/api/tasks

# For testing POST requests
curl -X POST -H "Content-Type: application/json" \
  -d '{"title":"Task from curl","description":"Created using curl
command"}' \
  http://$MINIKUBE_IP:$BACKEND_PORT/api/tasks
```

1. Get the URL of the frontend service:

PROF

```
minikube service frontend-service-2022bcd0026 --url
```

alt text

2. Test the backend API using curl:

```
# Port-forward the backend service to localhost
kubectl port-forward svc/backend-service-2022bcd0026 5000:5000
```

alt text

```
# In another terminal, test the API
curl http://localhost:5000/api/health
```

```
curl http://localhost:5000/api/tasks
curl -X POST -H "Content-Type: application/json" -d '{"title":"Test
Task","description":"This is a test task"}'
http://localhost:5000/api/tasks
```

alt text

3. Access the frontend in your browser using the URL provided by Minikube.
   ###GET THE MINIKUBE IP:-

```
minikube ip
```

###Access the Frontend: Use the Minikube IP and the frontend service's NodePort to access the application:

```
http://<minikube-ip>:<frontend-nodeport>
```

###Verify API Requests: Open the browser's developer tools (Network tab) and confirm that the API requests are being sent to the correct backend URL (http://:30001/api/tasks).

Troubleshooting

If you encounter issues with your deployment, here are some common troubleshooting steps:

1. Check pod status:

```
# Get all pods
kubectl get pods

# Get detailed information about a pod
kubectl describe pod <pod-name>

# Check pod logs
kubectl logs <pod-name>

# For previous container logs (if container crashed)
kubectl logs <pod-name> --previous

# Check specific container logs in a multi-container pod
kubectl logs <pod-name> -c <container-name>
```

2. Check service status:

```
```

```
# List all services
kubectl get services

# Get details about a specific service
kubectl describe service <service-name>

# Test a service's endpoint directly
kubectl run -i --tty --rm debug --image=busybox -- wget -O- <service-
name>:<port>
```

alt text

3. Check ConfigMaps and Secrets:

```
# List all ConfigMaps
kubectl get configmaps

# Get details about a specific ConfigMap
kubectl describe configmap <configmap-name>

# List all secrets
kubectl get secrets

# Get details about a specific secret
kubectl describe secret <secret-name>

# Decode a secret value
kubectl get secret <secret-name> -o jsonpath="{.data.<key>}" | base64 --
decode
```

alt text

PROF

4. Check Network Policies

```
# If you're using Network Policies, check them
kubectl get networkpolicies
kubectl describe networkpolicy <policy-name>
```

5. Common issues:
    - Images not found: Make sure you've built the images in the Minikube Docker environment

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

docker images | grep YOUR_REGISTER_NUMBER

```
   - Database connection issues: Check the logs of your backend pod for
connection errors
   ```bash
   kubectl logs $(kubectl get pod -l
app=backend,owner=YOUR_REGISTER_NUMBER -o
jsonpath='{.items[0].metadata.name}')
```

- Services not accessible: Use `kubectl port-forward` to debug connectivity

```
kubectl port-forward svc/backend-service-YOUR_REGISTER_NUMBER 5000:5000
# In another terminal
curl http://localhost:5000/api/health
```

```
- Check for imagePullBackOff errors:
```bash
kubectl get pods
```

If you see ImagePullBackOff, switch to minikube's docker daemon

eval $(minikube docker-env)
docker images # Check if your image exists

```
  - Restart a deployment after making changes:
  ```bash
  kubectl rollout restart deployment/backend-deploy-YOUR_REGISTER_NUMBER
```

PROF

#OUTPUT

##CRUD OPERATIONS

###INTIALLY FETCHED DATA , DATA ADDED IN TERMINAL - GET ()
alt text
alt text

alt text

###NEW TASK IS ADDED
alt text
```

###TASK 1 IS DELETED
alt text

###TASK 5 (ASSIGNMENT 3) IS UPDATED
alt text

###AFTER CHANGES
####FRONTEND
alt text
####BACKEND
alt text
alt text

####MYSQL
alt text

Cleaning Up

To clean up all the resources:

```
# Delete frontend resources
kubectl delete -f k8s/frontend/

# Delete backend resources
kubectl delete -f k8s/backend/

# Delete MySQL resources
kubectl delete -f k8s/mysql/

# Delete PersistentVolumeClaims and PersistentVolumes if needed
kubectl delete pvc mysql-pvc-YOUR_REGISTER_NUMBER
kubectl delete pv mysql-pv-YOUR_REGISTER_NUMBER

# Stop Minikube
minikube stop

# Optional: Delete the Minikube cluster
minikube delete
```

Additional Commands

```
# Get all resources in the namespace
kubectl get all

# View Pod details in YAML format
kubectl get pod <pod-name> -o yaml

# Open a shell in a running pod
```

```
kubectl exec -it <pod-name> -- /bin/bash
# or for containers that don't have bash
kubectl exec -it <pod-name> -- /bin/sh

# View real-time logs of a pod
kubectl logs -f <pod-name>

# Check Minikube status
minikube status

# Access Minikube dashboard
minikube dashboard

# Check resource usage
kubectl top nodes
kubectl top pods

# Generate YAML manifests for debugging
kubectl create deployment test --image=nginx --dry-run=client -o yaml

# Check all events in the cluster
kubectl get events --sort-by='.metadata.creationTimestamp'
```