Uma Ahirwar

ahirwaruma51@gmail.com

91-9294586442

Uma8269/Devops-assignment-techdome: Multi-Container application development with docker compose and Kubernetes (github.com)

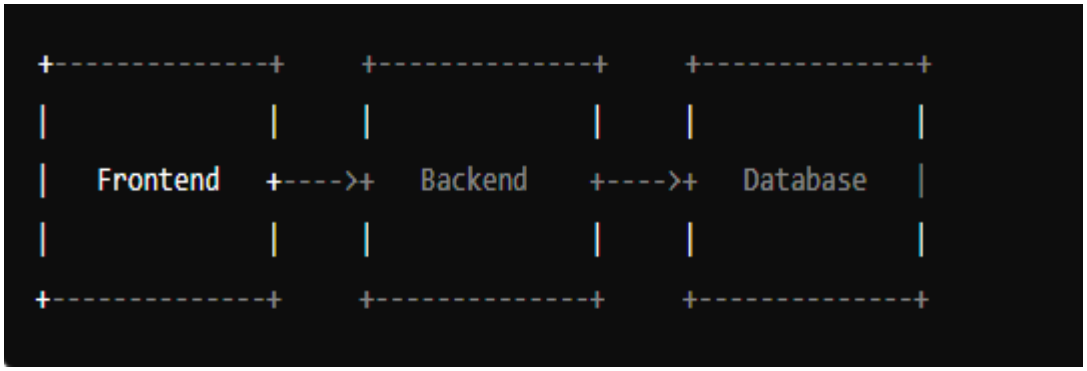
DevOps Assignment

Multi-Container Application Deployment with Docker Compose and Kubernetes

1. Application Architecture

The application architecture consists of a multi-container setup that includes three primary components:

1. **Frontend Service**
2. **Backend Service**
3. **Database Service**



These services are containerized and defined using Docker Compose for local development. For deployment in a production-like environment, Kubernetes manifests are used to manage these containers in a Minikube cluster.

2. Deployment Strategy

Using Docker Compose

Docker Compose is used to define and run the multi-container Docker application locally. The `docker-compose.yml` file specifies the configuration for each service.

Using Kubernetes

For a more scalable and robust deployment, Kubernetes is used. The application is deployed on a local Minikube cluster. Kubernetes deployment manifests define the configuration for each component and its corresponding service.

3. Instructions for Building, Deploying, and Managing the Application

Step-1. Set Up Your Development Environment

Make sure you have the necessary tools installed:

- Docker
- Docker Compose
- Minikube
- kubectl (Kubernetes command-line tool)

Step-2. Clone the Repositories

Clone the provided backend and frontend repositories:

```
git clone https://github.com/Anand-1432/Techdome-backend
git clone https://github.com/Anand-1432/Techdome-frontend
```

Step- 3. Create a Docker Compose File

Create a `docker-compose.yml` file to define the multi-container application. Here is an example configuration:

`Docker-compose.yml`

```yaml
version: '3.8'

services:
  frontend:
    build: ./Techdome-frontend
    ports:
      - "3000:3000"
    depends_on:
      - backend
    networks:
      - techdome-net

  backend:
    build: ./Techdome-backend
    ports:
      - "5000:5000"
    networks:
      - techdome-net

  db:
    image: postgres:13
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: passwd
      POSTGRES_DB: techdome
    ports:
      - "5432:5432"
    networks:
      - techdome-net

networks:
  techdome-net:
```

Step- 4. Deploy Locally with Docker Compose

Run the following command to start your application locally:

```
docker-compose up
```

Step- 5. Set Up Minikube

```
minikube start
```

Step- 6. Create Kubernetes Manifests

Create Kubernetes manifests for each component (frontend, backend, db).

Frontend Deployment (frontend-deployment.yml)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image: uma8269/frontend:latest
        ports:
        - containerPort: 3000
---
apiVersion: v1
kind: Service
metadata:
  name: frontend
spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
  type: NodePort
```

Backend Deployment (backend-deployment.yml)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: backend
        image: uma8269/backend:latest
        ports:
        - containerPort: 5000
---
apiVersion: v1
kind: Service
metadata:
  name: backend
spec:
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 5000
      targetPort: 5000
  type: ClusterIP
```

Database Deployment (db-deployment.yml)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: db
spec:
  replicas: 1
  selector:
    matchLabels:
      app: db
  template:
    metadata:
      labels:
        app: db
    spec:
      containers:
      - name: db
        image: postgres:13
        env:
        - name: POSTGRES_USER
          value: "admin"
        - name: POSTGRES_PASSWORD
          value: "passwd"
        - name: POSTGRES_DB
          value: "techdome"
        ports:
        - containerPort: 5432
---
apiVersion: v1
kind: Service
metadata:
  name: db
spec:
  selector:
    app: db
  ports:
    - protocol: TCP
      port: 5432
      targetPort: 5432
  type: ClusterIP
```

Step- 7. Deploy to Kubernete

```
kubectl apply -f frontend-deployment.yml
kubectl apply -f backend-deployment.yml
kubectl apply -f db-deployment.yml
```

Step- 8. Verify the Deployment

```
minikube service frontend-service
minikube service backend-service
minikube service db-service
```

Docker

Docker is an open-source platform that automates the deployment, scaling, and management of applications inside lightweight, portable containers. Containers package an application with all of its dependencies, libraries, and configuration files, allowing it to run consistently across different computing environments. Docker includes tools and utilities to facilitate container creation, management, and orchestration.

Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. Using a YAML file, you can specify the services, networks, and volumes that your application requires, enabling you to manage multiple containers as a single application. Docker Compose simplifies the setup and management of complex applications, making it easier to define and run multi-container Docker applications.

Minikube

Minikube is a tool that allows you to run a single-node Kubernetes cluster locally on your computer. It provides a way to create and manage a local Kubernetes environment, ideal for development and testing. Minikube supports most of the Kubernetes features, enabling developers to experiment with and develop against a local Kubernetes cluster without needing a full-scale production cluster.

Kubernetes

Kubernetes, often referred to as K8s, is an open-source platform designed to automate the deployment, scaling, and operation of application containers. It provides a robust framework for managing containerized applications across a cluster of machines, handling tasks such as load balancing, scaling, resource allocation, and failover. Kubernetes can manage complex, distributed applications, ensuring high availability and efficiency.