**Project: Deploying a Flask MongoDB Application on Kubernetes using Docker Containers**

**Introduction**

This project report outlines the process of deploying a web application on a local Kubernetes cluster using Docker containers. The deployment involves containerizing a Flask application, setting up a MongoDB service, and deploying both on a Kubernetes cluster using Minikube. The report details each step, from installing necessary tools to testing the final deployment.

**Step 1: Installing Docker**

The first step involved installing Docker, a tool essential for containerizing applications. Docker was installed by downloading Docker Desktop from the official Docker website, following the specific installation instructions for the operating system used (Windows, Mac, or Linux).

**Step 2: Creating a Dockerfile for the Flask Application**

A Dockerfile was created to containerize the Flask application, which is a simple To-Do List app using MongoDB as its database. The Dockerfile script was designed to build a Docker image by performing the following actions:

**1. Base Image:** Using a lightweight Python 3.10 image as the parent image.

**2. Working Directory:** Setting the working directory to `/usr/src/app`.

**3. Copying Files:** Copying the current directory’s contents into the container.

**4. Installing Dependencies:** Installing required Python packages listed in `requirements.txt`.

**5. Exposing Port:** Exposing port 5000 for the Flask app.

**6. Setting Environment Variables:** Defining environment variables necessary for running the Flask app.

**7. Running the Application:** The container runs the Flask application on startup.

**Step 3:** **Defining Services with Docker Compose**

Docker Compose was used to define and manage multi-container Docker applications. A `docker-compose.yml` file was created to define services for the Flask application and MongoDB. The `docker-compose.yml` included:

**1. Web Service:** Defined the Flask app, mapping port 5000 on the host to port 5000 on the container.

**2. MongoDB Service:** Used the official MongoDB image, set environment variables, and created a persistent volume for MongoDB data.

**Step 4:** **Building and Running Containers**

The Docker containers were built and run using the `docker-compose up --build` command. This command built the images (if they didn’t exist) and started the containers. The application was then accessible at `http://localhost:5000`.

**Step 5:** **Pushing Docker Image to Docker Hub**

To deploy the application on Kubernetes, the Docker image of the Flask app was pushed to Docker Hub. This required creating a Docker Hub account, logging in via the command line, tagging the image, and pushing it to Docker Hub.

**Step 6:** **Deploying on Kubernetes using Minikube**

Minikube, a tool by Google that allows running a local Kubernetes cluster, was used for deployment. The process involved:

**1. Installing Minikube:** Minikube was installed to create a local Kubernetes cluster.

**2. Starting Minikube:** Minikube was started to set up the cluster.

**3. Creating Pods:** Two pods were created for the Flask app and MongoDB using Kubernetes deployment files (`mongodb-deployment.yaml` and `flask-deployment.yaml`).

**4. Exposing Services:** Kubernetes services were created to expose the MongoDB and Flask app, allowing external access.

**5. Testing the Application:** The application was tested by accessing the Flask app service through Minikube, ensuring it was running correctly.

**Step 7: Shutting Down the Deployment**

After testing, the services and deployments were shut down to clean up resources. This involved deleting the services and deployments, followed by stopping and deleting the Minikube cluster.

**Conclusion**

This project successfully demonstrated the deployment of a containerized web application on a local Kubernetes cluster using Minikube. By following the outlined steps, we containerized a Flask application, set up a MongoDB service, and managed the entire deployment lifecycle from building the Docker image to testing the application on Kubernetes. This approach offers a robust and scalable method for deploying web applications in a cloud-native environment.

**GitHub Link:** <https://github.com/Chetan4458/Docker-Compose-Flask-Mysql-master>