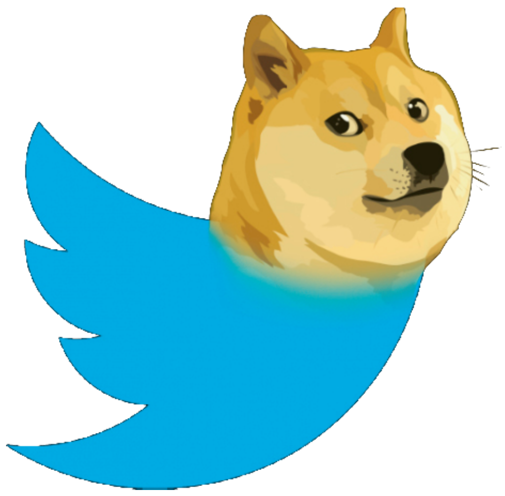
Assessment-4 Charlie: Deploying Python Flask-Based Twitter Alternative 'Twoge' Using Kubernetes and AWS EKS

Objective

The objective of this assessment is to gain hands-on experience with Kubernetes while deploying twoge application on kubernetes. It will help you to understand its architecture, and use it to manage containerized applications effectively both on minikube and AWS EKS.

You will be deploying a Python Flask-based Twitter alternative called 'Twoge' initially on your local Kubernetes cluster (For us it is Minikube) and later on AWS Elastic Kubernetes Service (EKS).



Source code:

https://github.com/chandradeoarya/twoge/tree/k8s (use k8s branch)

Assessment Steps

1. **Create a Docker image of the Twoge application**: The first step involves Dockerizing the Twoge application. Write a Dockerfile for your Python Flask application and build the Docker image.

   Please be mindful that the last Dockerfile may not work exactly the same. So, you may want to Dockerize the application again and test using compose (optionally).

2. **Deploy Twoge on Minikube**: Write deployment and service YAML configuration files to deploy the Twoge application and expose it within the Minikube cluster.

   Note: If you are having issues with database deployment and services then for testing you can use RDS as environment or hard coded value.

3. **Configure Database**: Use ConfigMap and Secrets to pass database configuration to the application. Your database will also be deployed on the Kubernetes cluster.
4. **Namespace**: Create a namespace for the application. All deployments should be within this namespace.
5. **Probes**: Implement **any one** of the liveness, readiness, and startup probes for your deployments to manage the application's lifecycle effectively. If you implement two, one will be considered a bonus.
6. **Deploy on AWS EKS(optional or not, will confirm by Monday)**: Migrate your deployment to AWS EKS. Ensure you change the service type to LoadBalancer when deploying on EKS for public access.
   Please be very very mindful of using your namespace only.
7. **Persistent Volume Claim(optional)**: Create a Persistent Volume Claim (PVC) using storage class.
8. **EKS CICD (optional):** Perform CICD using github actions using Dockerhub and EKS.
   Please be very very mindful of using your namespace only in CICD as well.

Deliverables

- Dockerfile for the Twoge application.
- YAML configuration files for the deployment, service, ConfigMap, Secrets, Namespace, Probes. Optionally storage class and PVC as well.
- A diagram outlining the architecture of your deployment.
- Screenshots or screen recordings demonstrating successful deployments on both Minikube and AWS EKS.

Key Files to write

1. **Dockerfile**: This is essential for creating a Docker image of your Twoge application. Please don't put any sensitive information like live database details etc. on this. You can use your Dockerfile written already.
2. **Namespace.yaml**: This YAML file will define the namespace where your Twoge application will reside.
3. **Twoge-Deployment.yaml**: This YAML file will define the Deployment for the Twoge application. The Deployment manages the desired state for your Pods and ReplicaSets.
    1. This will be having a quota definition as well.
    2. This will also have probes definition as well. It will define the liveness, readiness, and startup probes for the Twoge application. You are required to create just one of the probes of your choice.
4. **Twoge-Service.yaml**: This file will define the Service for the Twoge application. The Service is an abstract way to expose an application running on a set of Pods as a network service.
    1. This will be NodePort for local testing but LoadBalancer for EKS.
5. **ConfigMap.yaml**: This file will define ConfigMap, which will hold your configuration details like database connection strings, ports, etc.
6. **Secrets.yaml**: This file will contain Secrets, such as sensitive database credentials.
7. **Database-Deployment.yaml** and **Database-Service.yaml**: These files will define the Deployment and Service for your database.
8. **Storage class and PVC(optional):** For persistent storage
9. **Github Actions CICD yml (optional)**

Optional

1. You can also write phpmyadmin related yml files (deployment and services) optionally.

Note:

1. Use the same cluster shared among the classroom. **(optional or not, will confirm by Monday)**
2. You can do storage class and pvc implementation at the end. **(optional or not, will confirm by Monday)**
3. You can do CICD at the end **(optional)**

DODIIGE