DevOps Engineer Assignment

Submission Guidelines

Please submit your completed assignment as a ZIP file of your Git repository. The repository should contain all required files including **Dockerfiles, bash scripts, Kubernetes artifacts, and comprehensive documentation**.

Objective

You are provided with the following:

- **KServe Model Service:** A Python service (using KServe) that implements an OCR model using Tesseract. (`model.py`)
- **FastAPI Gateway Service:** A FastAPI application that accepts image uploads, encodes them in base64, and proxies the request to the KServe model service. (`api-gateway.py`)
- **Poetry configuration file:** Poetry dependency management and packaging configuration files for the project (`pyproject.toml`)
- **Instructions:** Poetry installation on Linux environment, setup dependencies, and run the services locally with Postman requests as a starting point. (`commands.sh`)

The objective of this assignment is to design, implement, and deploy a simple two-microservice OCR inference system that integrates a KServe-based model service with a FastAPI gateway on a Kubernetes environment. This will be accomplished using Docker, Helm charts, ArgoCD, bash scripts, and Minikube, with ML model monitoring setup using Prometheus and Grafana.

Tasks

1. Local Setup and Testing

1. Install Poetry (Recommended to use WSL 2 environment or Linux environment - Ubuntu)
2. Set up a Python environment using Poetry dependencies

3. Run the services locally and test using Postman with the commands provided in the `commands.sh` file
4. Verify service functionality as follows:
    a. Open Postman and create a new request
    b. Set method to POST
    c. URL: http://localhost:8001/gateway/ocr
    d. In the Body tab, select "form-data"
    e. Add a key named "image_file" and set type to "File"
    f. Click "Select File" and choose an image for OCR processing
    g. Click "Send"

2. Containerization

1. Create Dockerfiles for both the KServe model server and FastAPI gateway
2. Create bash scripts to automate the build and testing of Docker images locally
3. Push the images to Docker Hub private repository (you can use a free account with one private repository)
4. Document your containerization strategy, including:
    a. Base images selection rationale
    b. Security considerations
    c. Build optimization techniques

3. Infrastructure Setup

1. Create a Minikube cluster with Docker drivers on your local PC
2. Install ArgoCD, Prometheus, and Grafana using public open-source Helm charts
3. Provide appropriate Helm chart values considering the local Minikube setup and resource constraints of the local development environment
4. Create bash scripts to automate this infrastructure setup

4. Kubernetes Deployment

1. Create Helm charts for deploying both services to Minikube and push them to Docker Hub
2. Include all necessary Kubernetes resources:
    a. Deployments
    b. Services
    c. ConfigMaps/Secrets/Roles

    d. Image pull secrets (if needed)
    e. RBAC configurations

5. GitOps with ArgoCD

1. Create ArgoCD Custom Resource Definitions (CRDs) objects (Application, ApplicationSet, AppProject) to deploy both services to Minikube
2. For simplicity, you are not required to focus on the entire GitOps workflow and state repository; just use kubectl to apply ArgoCD resources to the local cluster

6. Monitoring Setup

1. Create Prometheus resources or annotations to scrape the KServe built-in model monitoring metrics exposed via `8080/metrics`
2. Create a simple model monitoring Grafana dashboard using the available built-in metrics of KServe and Prometheus Query Language
3. Examples of metrics to include:
    a. Total number of inference requests received by the model
    b. Resource utilization metrics
    c. Model inference latency histogram metrics
    d. Error rate/request status

7. Documentation

Create comprehensive documentation (README.md or Word document) including:

- Architecture diagram
- Detailed explanation of your implementation
- Step-by-step instructions for the entire process

8. Submission

- Create a ZIP archive of your entire Git repository, maintaining a clean directory structure.
- Ensure all required files (**Dockerfiles**, **scripts**, **Kubernetes artifacts, documentation**) are included.
- Maintain a clean and meaningful Git commit history that demonstrates your development process.

- Follow best practices for code organization, commenting, and version control.
- Include a comprehensive README.md at the root and detailed documentation in the docs/ directory.

Submit your ZIP file through the specified submission channel by the deadline. Ensure the repository is self-contained and includes all necessary instructions for reviewers to evaluate your work.