GLM Model Deployment with FastAPI, Docker, Kubernetes, and GitLab

Overview

This project demonstrates the deployment of a Generalized Linear Model (GLM) using the FastAPI framework, Docker containers, Kubernetes orchestration, and GitLab for Continuous Integration/Continuous Deployment (CI/CD). The GLM model is served as a web API, providing predictions based on input data.

Project Structure

- `app/`: Contains the FastAPI application code.
- `app/model/`: Holds the pre-trained GLM model (pickle file).
- `docker/`: Contains Dockerfile for building the Docker image.
- `kubernetes/`: Includes Kubernetes deployment and service YAML files.
- `tests/`: Holds unit tests for the FastAPI application.
- `gitlab-ci.yml`: GitLab CI/CD pipeline configuration file.
- `README.md`: This documentation file.

Getting Started

1. Clone this repository: `git clone https://github.com/ZCai25/glm-fastapi-app.git`
2. Navigate to the project directory: `cd glm-fastapi-app`
3. Follow the instructions in each directory to deploy the GLM model locally or in a Kubernetes cluster.

End-to-End Process

1. **FastAPI Application:**

    - The FastAPI application (`app/main.py`) defines API endpoints for model prediction.
    - Input data is received via HTTP requests and passed to the pre-trained GLM model.

2. **Model Loading:**

    - The pre-trained GLM model is stored in the `model/` directory.
    - The model is loaded during the FastAPI application startup.

3. **Docker Containerization:**

    - The Dockerfile (`docker/Dockerfile`) specifies the environment and dependencies for running the FastAPI application.
    - Docker image is built using the `docker build` command, then you can build by running the `run_api.sh`
        - From the project directory, build the container with tag `docker build -t glm-fast-api:1.0 .`
        - Make sure the script has execute permissions by running `chmod +x run_api.sh`.

    - Run the scipt to run the container by typing `./run_api.sh 1313:80`, you will see the api server started, you can access the server document at `http://localhost:1313/docs` image

4. **Orchestration:**

    - Orchestration using Docker compose
        - From the project directory, run `docker-compose up` and you will see the api server up and running like the previous image.
        - Run `docker-compose down` to stop the service
    - Orestration Using Kubernetes
        - Kubernetes deployment YAML (`kubernetes/deployment.yml`) defines how the FastAPI application should run as pods.
        - Kubernetes service YAML (`kubernetes/service.yml`) exposes the application within the cluster.
        - To start the orchestration process locally, start minikube by running `minikube start`
        - Check list deployment runing `kubectl get deployments`
        - Check pod by runing `kubectl get pod`
        - Check services by running `kubectl get services`
        - Here is a example of the output of the above commands, you can see we create 3 replica in the pod and we deploy them as load balancer to handle large amount of requests
        - image

5. **GitLab CI/CD Pipeline:**

    - `.gitlab-ci.yml` contains the CI/CD pipeline configuration.

    - The pipeline includes stages for linting, testing, building the Docker image, and deploying to Kubernetes.

    - To use this CI/CD configuration, make sure you have GitLab CI/CD configured for your repository, and the necessary variables (e.g., Docker registry credentials) are set in your GitLab project settings.

    - When you push changes to the master branch, GitLab CI/CD will automatically trigger the pipeline, and it will execute the defined stages. The Docker image will be built, pushed to the registry, and then the application will be deployed to Kubernetes.

6. **CI/CD Workflow:**

    - Code changes trigger the GitLab CI/CD pipeline.
    - Automated testing ensures code quality.
    - Docker image is built and pushed to the container registry.
    - Kubernetes deployment is updated with the new image.

Notes

- Update configuration files (`docker/Dockerfile`, `kubernetes/deployment.yml`) based on your model and requirements.
- Adjust GitLab CI/CD settings and environment variables in the GitLab project.

Feel free to explore the project directories for detailed instructions and customization options.