Project Summary – LLM Query Text Similarity Microservice

1. Overview

This project is an end-to-end Flask-based microservice designed to support LLM-related workflows by comparing two text prompts and identifying whether they match or are semantically similar.

2. Purpose

The purpose of this project is to validate similarity between two input prompts before passing them to a language model. It includes methods like cosine and Jaccard similarity for text comparison.

3. Project Structure

The application is organized into the following components:

- `run.py` – Entry point for the Flask app
- `app.py` – App factory and logger initialization
- `analyze.py` – API route to handle prompt similarity requests
- `similarity.py` – Implements cosine and Jaccard similarity logic
- `text_sanitization.py` – Tokenization, stopword filtering, text normalization
- `llm_handler.py` – Stub for LLM interaction or post-similarity querying
- `tests/` – Contains unit and integration test cases
- `Dockerfile` – For containerization and deployment
- `locustfile.py` – Load testing using Locust
- `logs/` – Log directory (rotating handler)
- `.github/workflows/` – Contains GitHub Actions workflow
- `README.md` – Setup, usage, and deployment documentation

4. CI/CD with GitHub Actions

A GitHub Actions workflow has been integrated to automate testing and Docker image creation.

The workflow performs the following steps:
- Triggers on push events

- Installs dependencies
- Runs all test cases using `pytest`
- Builds a Docker image of the service
- Optionally pushes the image to a container registry (if configured)

This ensures continuous integration and validation of code quality and deployment readiness.

5. Key Features

- Input sanitization using NLTK and custom filtering
- Similarity scoring using Cosine (TF-IDF) and Jaccard
- Modular, extensible service design
- GitHub Actions for CI/CD with automated tests and Docker build
- Pro
- Load testing with Locust

6. Testing Strategy

- Unit tests cover similarity and sanitization logic
- Integration tests validate full request-response flow
- Locust simulates concurrent users for performance testing

7. Deployment Process

- Run locally with Flask development server (`python run.py`)
- Build Docker image with `docker build -t text-similarity-service .`

8. Scaling Considerations

- Use nginx workers for concurrent request handling
- Deploy multiple container replicas behind a load balancer
- Add Redis caching for common or repeated prompt queries
- Integrate ELK/CloudWatch for logging and observability
- Use environment variables for config management and secrets

9. Conclusion

This project is a foundational component for building intelligent LLM query pipelines. It validates input prompts efficiently, avoids redundancy, and is built with scalability and automation in mind. With CI/CD pipelines, container support, and modular logic, it is ready for production deployment.