MLOps Task: Dockerizing and Deploying a Flask API with CI/CD

Objective

You will build a Flask-based calculator API, containerize it using Docker, and set up a CI/CD pipeline using GitHub Actions. Your pipeline will ensure that the application is tested, linted, built, and pushed to Docker Hub only if all checks pass.

Task Steps

1. Create the Flask API

You will create a Flask API that provides basic mathematical operations.

Operation	Endpoint Example	Expected Response
Addition	/add?a=5&b=10	{"result": 15}
Subtraction	/subtract?a=20&b=3	{"result": 17}
Multiplication	/multiply?a=4&b=6	{"result": 24}
Division	/divide?a=30&b=5	{"result": 6}

Additional Requirements:

- Create a Virtual Environment.
- Create a new branch named dev.
- Implement error handling for invalid inputs and division by zero.
- Add a health check endpoint (/healthz) that returns the API uptime.
- Log all incoming requests for monitoring.

2. Write Unit Tests

You must write unit tests to validate the API endpoints.

Testing Requirements:

- Use pytest to write test cases.
- Check both valid and invalid inputs for all endpoints.
- Ensure division by zero is handled correctly.

3. Containerize the Application Using Docker

You will write a Dockerfile to containerize your Flask application securely and efficiently.

Docker Requirements:

- Use a lightweight base image (python:3.9-slim).
- Implement a multi-stage build to optimize the final image size.
- Use a non-root user for security purposes.

- Expose port 5000 for the Flask application.

To build and run locally, use the following commands:

- docker build -t flask-math-api .
- docker run -p 5000:5000 flask-math-api

4. Set Up GitHub Actions for CI/CD

You will create a GitHub Actions workflow to automate testing and linting.

CI/CD Pipeline Requirements:

- Run unit tests before building the Docker image.
- Lint the Python code and Dockerfile.
- Perform a security scan on the Docker image.

Notes

• Follow best practices for Flask, Docker, and GitHub Actions.

• Your API should be secure and scalable to handle multiple requests.

Deliverables

1. A Flask-based calculator API that meets the requirements.

2. A Dockerfile that allows the application to run in a container.

3. A GitHub Actions workflow that automates CI/CD.

4. A Pull Request (PR) submission with all required details.

README:

https://docs.docker.com/get-started/docker-concepts/building-images/multi-stage-builds/