Lab 5 : Automation with Github Actions

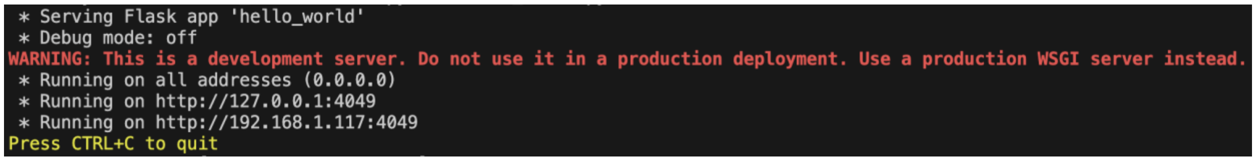
This lab will walk through implementing a simple CI/CD pipeline into a codebase using GitHub Actions. By the end of it, you will learn how to:

- Integrate GitHub Actions with your GitHub project.
- Implement a CI/CD pipeline in the codebase using a GitHub Actions workflow file
- Automate the build and deploy of a Docker image to Docker Hub.

Section 1 – Testing the Code

This code base contains a Python Flask application.The app is a simple web server that renders ~~HTML when a request is~~ made. The Flask application lives in the `hello_world.py` file.

1. Run the python file
2. On your browser, navigate to one of the URLs that appear on your terminal



```
 * Serving Flask app 'hello_world'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:4049
 * Running on http://192.168.1.117:4049
Press CTRL+C to quit
```

3. It seems you have a 404 error. Now using the same URL, append "/greeting" to it. You should see a new page.
4. All code must be tested to ensure that stable, quality code is being released. Python comes with a testing framework named unittest. Create unit tests file ***test_hello_world.py*** and add your tests in it. These tests should be able to cover most of the ***hello_world*** app.
5. Run pytest to validate your unit tests.

Section 2 - CI/CD Pipeline using GitHub Actions

It's time to implement a CI/CD pipeline into the codebase using GitHub Actions. The integration is very simple because GitHub Actions is built right into GitHub.

The CI/CD configuration for GitHub Actions is defined in a YAML file that must be placed in a directory named `.github/workflows/` at the root of your repository.

1. Similar to what's been done in class. Implement a linter workflow linter.yml that checks your code for style issues on every push or pull request to the main branch.

2. Implement a test workflow that sets up a Python virtual environment, installs your dependencies, and runs your unittests.

Section 3 – Adding a Deploy Workflow

1. Implement a deploy workflow builds your application binary, creates a Docker image, logs in to Docker Hub (using secrets), and pushes the image. It triggers on pushes to the `main` branch.

Additional Notes

- **Repository Secrets:**
  Make sure you set up the following secrets in your GitHub repository settings (under **Settings > Secrets and variables > Actions**):
  - DOCKER_LOGIN (your Docker Hub username)
  - DOCKER_PWD (your Docker Hub password)
- **Trigger Conditions:**
  The linter and test workflows run on both pushes and pull requests to the main branch. The deploy workflow runs on pushes to main—you can adjust these triggers as needed.