

## **Leo Merville - ML OPS LAB5 Automations**

### **1. Introduction**

The goal of this lab was to build a simple but complete CI/CD pipeline around a small Flask application.

Starting from the provided `hello_world.py` file, I had to:

- run the application locally and explore its HTTP endpoints,
- write unit tests and run them with `pytest`,
- configure GitHub Actions to run linting and tests on every push,
- build a binary of the application with `PyInstaller`,
- containerize the binary with Docker and automatically push the image to Docker Hub.

The final deliverables are a GitHub repository with working workflows, a Docker image hosted on Docker Hub, and this short report explaining the process, including problems encountered and how they were fixed.

### **2. Section 1 – Running and Testing the Flask Application**

#### **2.1 Local setup and virtual environment**

I started by creating a local virtual environment in the project folder:

```
python -m venv .venv  
.venv\Scripts\activate  
pip install --upgrade pip  
pip install -r requirements.txt
```

The `.venv/` directory is ignored by Git through the `.gitignore` file, so it is not committed to the repository.

#### **2.2 Running the Flask app**

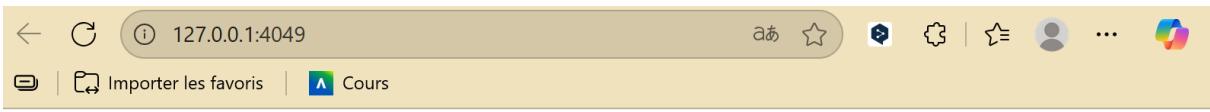
Then I launched the application:

```
python hello_world.py
```

Flask started on port 4049. When I opened `http://127.0.0.1:4049/`, I correctly obtained a **404 Not Found** page, because no route is defined for `/`. When I navigated to `http://127.0.0.1:4049/greeting`, I saw the GitHub Actions image and the message:

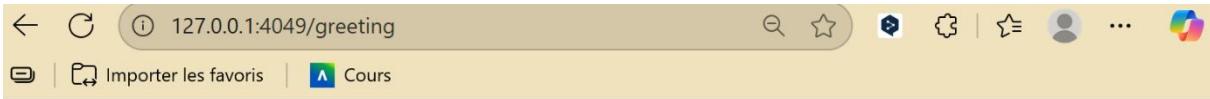
“Welcome to CI/CD 101 using GitHub Actions!”

```
● PS C:\Users\leoje\Desktop\lab5_automation> & C:/Users/leoje/Desktop/lab5_automation/.venv/Scripts/Activate.ps1
○ (.venv) PS C:\Users\leoje\Desktop\lab5_automation> python hello_world.py
>>
* 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://192.168.1.48:4049
Press CTRL+C to quit
127.0.0.1 - - [30/Nov/2025 15:46:03] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [30/Nov/2025 15:46:03] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [30/Nov/2025 15:46:10] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [30/Nov/2025 15:46:34] "GET /greeting HTTP/1.1" 200 -
```



## Not Found

The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.



Welcome to CI/CD 101 using GitHub Actions!

## 2.3 Unit tests with unittest and pytest

To follow the lab instructions, I wrote a `test_hello_world.py` file using the `unittest` framework, and executed it with `pytest`.

The tests cover:

- `greet()` returns the exact expected message string,
- `generate_html(message)` includes the message inside the generated HTML,
- the `/greeting` endpoint responds with HTTP status code 200,
- the `/greeting` response body contains the greeting message.

After activating the virtual environment, I ran:

```
pytest -v
```

```
PS C:\Users\leoj\Desktop\lab5_automation> & C:/Users/leoj/Desktop/lab5_automation/.venv/Scripts/Activate.ps1
(.venv) PS C:\Users\leoj\Desktop\lab5_automation> pytest -v
=====
platform win32 -- Python 3.12.10, pytest-9.0.1, pluggy-1.6.0 -- C:\Users\leoj\Desktop\lab5_automation\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\leoj\Desktop\lab5_automation
collected 4 items

test_hello_world.py::HelloWorldTests::test_generate_html_includes_message PASSED
test_hello_world.py::HelloWorldTests::test_greet_returns_expected_message PASSED
test_hello_world.py::HelloWorldTests::test_greeting_endpoint_contains_message PASSED
test_hello_world.py::HelloWorldTests::test_greeting_endpoint_status_code PASSED

===== 4 passed in 0.20s =====
(.venv) PS C:\Users\leoj\Desktop\lab5_automation>
```

### 3. Section 2 – Continuous Integration with GitHub Actions

In this section, I created two GitHub Actions workflows in the .github/workflows directory: one for linting, one for running the tests.

#### 3.1 Lint workflow (linter.yml)

The lint workflow is triggered on every push or pull request to the main branch. It:

1. Checks out the repository (actions/checkout).
2. Sets up Python 3.12 (actions/setup-python).
3. Creates a virtual environment, upgrades pip, installs the project's dependencies and flake8.
4. Runs flake8 on the project files, excluding the .venv directory.

Originally, the workflow failed because flake8 tried to lint the .venv directory and produced many errors inside third-party libraries such as PyInstaller. I fixed this by adding --exclude=.venv to the flake8 command so that only my source files are checked.

In addition, flake8 reported style issues in hello\_world.py and test\_hello\_world.py (missing blank lines, long lines, no newline at end of file). I updated these files to comply with PEP8 (splitting long lines, adding blank lines before/after functions and before the if \_\_name\_\_ == "\_\_main\_\_" block). After these changes, the lint workflow runs successfully.

The screenshot shows the GitHub Actions interface for the 'Lint' workflow. At the top, there is a 'Filter workflow runs' input field and a three-dot menu icon. Below this, a header row shows '4 workflow runs' on the left and filtering options 'Event', 'Status', 'Branch', and 'Actor' on the right. A single workflow run is listed in the table below:

Fix test_hello_world style for flake8	main	3 minutes ago	21s	...
Lint #4: Commit <a href="#">4d77dc6</a> pushed by <a href="#">lejeulinmerville</a>				

### 3.2 Test workflow (tests.yml)

The test workflow is also triggered on every push or pull request to main. It:

1. Checks out the repository.
2. Sets up Python 3.12.
3. Creates a virtual environment, installs dependencies and pytest.
4. Runs pytest -v.

This ensures that no change can be pushed to main without running the full unit test suite.

The screenshot shows the GitHub Actions interface. On the left, a sidebar lists various management and monitoring options: Actions (selected), New workflow, All workflows, Lint, Tests (selected), Management, Caches, Attestations, Runners, Usage metrics, and Performance metrics. The main area is titled 'Tests' and shows a workflow named 'test.yml'. A search bar at the top right says 'Filter workflow runs'. Below it, a message indicates '1 workflow run'. A table header with columns 'Event', 'Status', 'Branch', and 'Actor' is shown. A single workflow run is listed: 'Section 1 and 2' (status: green checkmark), triggered by 'Tests #1: Commit 8fbfde3 pushed by leojulinmerville' on the 'main' branch, completed 4 minutes ago, and took 25s. There is a three-dot menu icon next to the run details.

## **4. Section 3 – Continuous Delivery to Docker Hub**

The lab also required packaging the application into a Docker image and publishing it to Docker Hub via a deploy workflow.

### **4.1 Dockerfile and PyInstaller**

The provided Dockerfile expects a compiled binary located at dist/hello\_world:

```
COPY requirements.txt .
```

```
COPY dist/hello_world /opt/hello_world/
```

In my first attempt, the GitHub Actions deploy workflow failed with the error:

```
"/dist/hello_world": not found
```

This happened because the binary was never built before docker build was executed.

To fix this, I added a step in the deploy workflow to install dependencies and run PyInstaller:

```
pip install -r requirements.txt
```

```
pyinstaller --onefile hello_world.py
```

This command creates the dist/hello\_world binary, which the Dockerfile can then copy into the image.

## 4.2 Secrets and login to Docker Hub

In the repository settings on GitHub, I configured two Action secrets:

- DOCKER\_LOGIN – my Docker Hub username,
- DOCKER\_PWD – my Docker Hub password / access token.

The deploy workflow uses docker/login-action with these secrets to authenticate against Docker Hub before building and pushing the image.

### Actions secrets and variables

Secrets and variables allow you to manage reusable configuration data. Secrets are **encrypted** and are used for sensitive data. [Learn more about encrypted secrets](#). Variables are shown as plain text and are used for **non-sensitive** data. [Learn more about variables](#).

Anyone with collaborator access to this repository can use these secrets and variables for actions. They are not passed to workflows that are triggered by a pull request from a fork.

The screenshot shows the GitHub repository settings interface. At the top, there are two tabs: 'Secrets' (which is selected) and 'Variables'. Below the tabs, there are two sections: 'Environment secrets' and 'Repository secrets'.

**Environment secrets:** This section displays a message: "This environment has no secrets." It includes a button labeled "Manage environment secrets".

**Repository secrets:** This section lists two secrets: "DOCKER\_LOGIN" and "DOCKER\_PWD". Each secret entry includes a lock icon, the secret name, the last updated time ("1 minute ago"), and edit/delete icons. A green button labeled "New repository secret" is located at the top right of this section.

### 4.3 Deploy workflow (deploy.yml)

The final deploy workflow is triggered on pushes to main and performs the following steps:

1. Checkout the code.
2. Setup Python and build the PyInstaller binary (dist/hello\_world).
3. Log in to Docker Hub using the secrets.
4. Build the Docker image from the Dockerfile, tagging it both as latest and with the current commit SHA.
5. Push both tags to Docker Hub.

Once the workflow succeeded, I verified on Docker Hub that the image leo0679/mllops-lab5-actions had been created and tagged.

The screenshot shows the GitHub Actions interface. On the left, there's a sidebar with 'Actions', 'New workflow', 'All workflows', and several buttons: 'Deploy to Docker Hub' (which is highlighted), 'Lint', 'Tests', 'Management', 'Caches', and 'Attestations'. On the right, under 'Deploy to Docker Hub', there's a 'deploy.yml' link. Below it, it says '2 workflow runs'. A table header includes 'Event', 'Status', 'Branch', and 'Actor'. One row is shown: a green checkmark next to 'Build binary with PyInsta...', followed by 'Deploy to Docker Hub #2: Commit d1f0f11 pushed by leojeulinmerville' (with 'main' in a blue box), '1 minute ago', and '1m 9s'. There's also a three-dot menu icon.

Optionally, the image can be tested locally with:

```
docker pull leo0679/mllops-lab5-actions:latest
```

```
docker run -p 4049:4049 leo0679/mllops-lab5-actions:latest
```

and then by visiting <http://127.0.0.1:4049/greeting> in a browser.

Repositories / [mlops-lab5-actions](#) / General

Using 0 of 1 private repositories.

**leo0679/mlops-lab5-actions** 

Last pushed 1 minute ago • Repository size: 410.2 MB • ⭐0 • ⏪0

Add a description  

Add a category  

**Docker commands**

To push a new tag to this repository:

```
docker push leo0679/mlops-lab5-actions:tagname
```

**General** Tags Image Management BETA Collaborators Webhooks Settings 

**Tags**  DOCKER SCOUT INACTIVE [Activate](#)

This repository contains 0 tag(s).

Tag	OS	Type	Pulled	Pushed
 <a href="#">d1f0f11a62a0c3d7b3...</a>		Image	less than 1 day	3 minutes
 <a href="#">latest</a>		Image	less than 1 day	3 minutes

[See all](#)

## 5. Issues Encountered and Fixes

During the lab, I encountered several issues:

### 1. Git repository confusion

The original ZIP contained a .git folder pointing to the instructor's repository. When I tried to push, Git wanted to push to the wrong remote.

**Fix:** I deleted the existing .git folder, ran git init, and created a fresh GitHub repository (mlops-lab5-actions) with my own remote.

### 2. Linting third-party libraries

The first version of the lint workflow tried to run flake8 on the .venv directory, which produced many errors inside third-party packages.

**Fix:** I added --exclude=.venv to the flake8 command so that only my project files are checked.

### 3. PEP8 style violations in my own files

flake8 reported E302, E305, E501, and W293 in hello\_world.py and test\_hello\_world.py.

**Fix:** I reformatted both files to follow PEP8: added blank lines, split long lines (especially the image URL), removed trailing whitespace, and ensured the files end with a newline.

### 4. Missing PyInstaller binary for Docker build

At first, the deploy workflow failed because dist/hello\_world did not exist when Docker tried to copy it.

**Fix:** I added a step running pyinstaller --onefile hello\_world.py before docker build, which creates the required binary.

These issues are all documented by the corresponding failing and passing workflow runs in GitHub Actions.

## 6. Conclusion

This lab guided me through the complete lifecycle of a small application: from running and testing a simple Flask service to setting up automated CI and CD with GitHub Actions and Docker Hub.

The final setup ensures that:

- All changes pushed to main are automatically linted and tested.
- A binary is built from the Flask app using PyInstaller.
- A Docker image is built from this binary and pushed to Docker Hub without manual intervention.

The repository URL and Docker image URL are:

- GitHub: <https://github.com/leojeulinmerville/mllops-lab5-actions>
- Docker Hub: <https://hub.docker.com/r/leo0679/mllops-lab5-actions>

This provides a reproducible and automated workflow that matches the expectations of the lab for CI/CD using GitHub Actions.