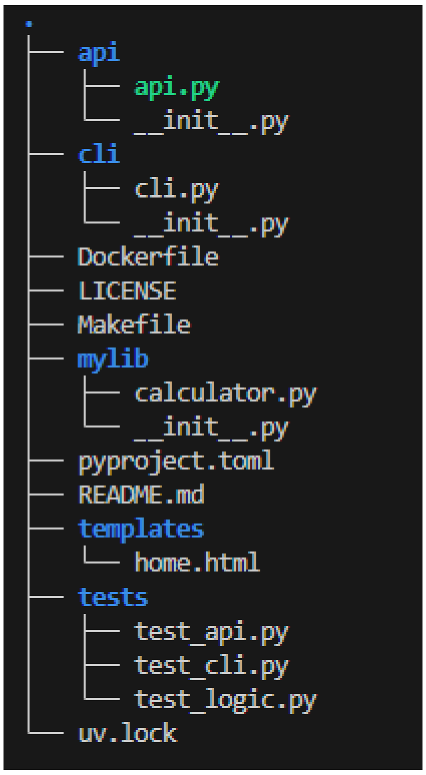
University Master's Degree in
Machine Learning ML

Assignment: Continuous Delivery using GitHub Actions

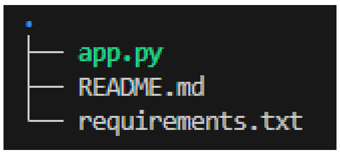
The main objective of this assignment is to develop a Continuous Integration Continuous Delivery (*CICD*) pipeline using GitHub actions. To do it, you must create a new repository for this lab named MLOps-Lab2. In this second part of the series (Lab1-Lab2-La3) you must create all the functionalities devoted to deploying the “random model” into production. Specifically, you will have to create a

- *Dockerfile* to containerize the project so that it can be replicated into the production environment.
- *GUI* for the random prediction of the class of the images using *Gradio*. The *GUI* will be hosted on *HuggingFace spaces*. It is very simple as it only has an input image, and the output is as simple as a text box where you must display the class label predicted for the image.
    - This part of the project is developed in a separated branch named *hf-space* (using the repository associated with the *HuggingFace* spaces as a remote repository).
- *CICD* pipeline with *GitHub Actions*. You will have to create two new jobs for the deployment of the image as well as that of the *GUI* application in *HuggingFace* spaces.
    - Remember to include the status badge in the *README.md* file.

To **help in the development of this lab**, as in the previous lab, we created a demo repository to develop a calculator application devoted to performing some basic arithmetical operations including these new functionalities. The public *GitHub* repository is https://github.com/JoseanSanz/MLOps-Lab2-demo. The scaffold of this calculator application project is illustrated in the following figure.



```
.
├── api
│   ├── api.py
│   └── __init__.py
├── cli
│   ├── cli.py
│   └── __init__.py
├── Dockerfile
├── LICENSE
├── Makefile
├── mylib
│   ├── calculator.py
│   └── __init__.py
├── pyproject.toml
├── README.md
├── templates
│   └── home.html
├── tests
│   ├── test_api.py
│   ├── test_cli.py
│   └── test_logic.py
└── uv.lock
```



```
.
├── app.py
├── README.md
└── requirements.txt
```

Figure 1. Scaffold of the calculator application. Main branch (left) and hf-space branch (right).

As you can see, there are a few changes with respect to the project developed in Lab1. Specifically, the *Dockerfile* devoted to the containerization of the project, the branch devoted to building the *GUI* that will use the *API* created with *FastAPI* and the new jobs added to the *GitHub Actions* pipeline.

Now, the steps performed to produce these new functionalities are explained.

1. **Containerization of the project**

   In first place you must **install Docker** (if you do not have it installed yet).

   sudo apt install docker.io

   Then, you must grant permanent permission to use Docker.

   ```
   sudo groupadd docker      # it creates the group (it it does no exist)
   sudo usermod -aG docker $USER
   newgrp docker
   ```

   Next, you must **create an account in Docker Hub** (if you do not have one).

   Once you have made these two steps and you have **program**med **the *Dockerfile*** you can **manually register the image in Docker Hub** (we will automate this process later using the *CICD* pipeline). To do it you have to

   - Build locally the container image from the *Dockerfile*

     docker build -t <image-name> .

   - Tag the local image so that it can be pushed to the remote registry (*Docker Hub* is used as default) *docker tag <image-name> <dockerhub-username>/<image-name>:latest*

- Push the image to *Docker Hub*

  docker push <dockerhub-username>/<image-name>:latest

Then, you can **check if the image is uploaded to *Ducker Hub*** in the repositories option of My Hub.

To **automate the image registry through GitHub Actions**. First you must **create a personal access token**, named *DOCKERHUB_TOKEN*, in Docker hub. You must select "Read, Write, Delete" and you can select the expiration date. Then, **copy the generated token**.

Then **in GitHub** you must **add the secrets for the GitHub Actions**. You must go to the settings of the repository, then in "Secrets and variables" you must select the actions option and create two secrets

- *DOCKERHUB_USERNAME* and write your username of Docker Hub
- *DOCKERHUB_TOKEN* and paste the token you generated in the previous stage

2. **Generation of a web service to host the API (backend)**

The next step is to **create a web service to host the backend** (API created with FastAPI) we have containerized using Docker. To do it, we **will use Render** as it provides a free solution for hosting APIs. The drawback of Render is that it only uses CPUs and this may imply that our final application, when it uses a deep learning method, can be slow.

The first thing to do is to **create an account on Render** (if you do not have one). Then, in the dashboard you have to **create a new web service** and, in the *Existing Image* option, you have to associate it with the docker image registered in Docker Hub: *docker.io/<dockerhub-username>/<image-name>:latest*. Then press connect and then you will have to deploy the web service (select for **hobby projects options so that it is free**).

Next, we can also **automate the process of deploying again the web service when a new image is automatically registered in Docker Hub in GitHub Actions**. To do it, in Render, you must **create a secret and add it to GitHub actions secrets**. To do it, you must go to the settings of your created web service and scroll down until you see the **Deploy Hook**, which is an *URL*. **The secret is the key** of the URL. That is, you must copy just the part of the *URL* after the = symbol. Then, you must create a GitHub Actions secret named *RENDER_DEPLOY_HOOK_KEY* and you paste the previous secret. **The remainder part of the URL is used in the CICD pipeline** (in the step devoted to deploying again the web service in Render), you only have to copy-paste it.

3. **Generation of the H*uggingFace* space that uses the *API***

The last step is to upload the *Gradio* application to HugginFace spaces. To do it, you first must **create an account** on *HuggingFace* (if you do not have one). Then, you must **create a space** (name, description, Gradio as the SDK, CPU Basic, Public). To automate the process, as in Docker Hub and Render, you must **create new Access Token** with

write permissions, named *HF_TOKEN*, **copy the generated token and save it** (it will be used when pulling the files of the space to the repository of the project). You also must **add this token GitHub Actions secrets** as before. You also must **create a secret** to determine the **username** of *HuggingFace* (name it *HF_USERNAME* and write your username).

Finally, you must **include the files associated with this GUI** (*HuggingFace* space) in your project (as a branch named *hf-space*). To do it, in the terminal of your project (VS Code), you must

- Add the *HuggingFace* repository associated with the space you created as a remote repository

  git remote add huggingface https://huggingface.co/spaces/<huggingface-username>/<hf-space-name>.git

- Create a new branch in the GitHub repository

  git checkout --orphan hf-space

- Pull the files from the *HuggingFace* repository

  git pull huggingface main
  - It asks for the username and access token (you must paste the token)

At this point, you can program the *Gradio* application for the project and add the libraries you need in the *requirements.txt* file (if necessary). **This application uses the API hosted in Render** so that it can use the exposed *endpoint* for the random prediction of the class label. To do it, you **must include the public URL of the API** hosted in Render (you can find it inside the web service created in the second step). Finally, **you can manually push these changes back** both to the *HuggingFace* repository and to the main branch (to make them visible from the main branch and ensure the *CICD* pipeline will work properly). **These steps will be automated in the *CICD* pipeline**.

- Commit the changes

  git add app.py requirements.txt README.md
  git commit -m "commit comment"

- Push it to the HuggingFace repository. **This step is not needed in the automated workflow**, since it will be automatically uploaded from the main branch

  git push huggingface hf-space:main
  - It asks again for the username and access token

- Push these changes also into the main branch

  git push origin hf-space

Finally, the **details of the two new files you have to create/update** are as follows

- The ***Dockerfile*** is composed of **three stages**
  - **Base stage** (*base*): to provide a small Python 3.13 runtime image, set useful Python environment variables and establish the working directory inside the container.
  - **Builder stage** (*builder*): to install the *uv* tool and use it to install all Python dependencies declared in *pyproject.toml* (locked via *uv.lock*) into the system's Python folder (*/usr/local*).
    - This stage is where dependency resolution and package compilation happens. Keeping it separate avoids shipping build-time artifacts in the final image.
  - **Runtime stage** (*runtime*): to start from a clean base image, copy the pre-installed site-packages from the builder (/usr/local) and the application source code (*api*, *mylib* and *templates* folders). This way, the final image is smaller (not all the files in the project are included) and contains only what is necessary to run the app. The container starts with *uvicorn* serving the FastAPI API on port 8000.
- ***CICD* pipeline**, it includes **two new jobs** to
  - **Deploying the API**
    - It checks the repository and sets the *buildx* for both QEMU and Docker
    - It logs to Docker Hub (using the secrets of GitHub Actions)
    - It builds, tags and pushes the image to Docker Hub
    - It triggers the deployment of the web service (in Render) associated with the image created in the previous step
  - **Deploying the *HuggingFace* application**
    - It checks the repository
    - It switches to the *hf-space* branch
    - It adds the *HuggingFace* repository as a remote repository using the secrets of GitHub Actions
    - If pushes the *Gradio* app to the main branch