MLOps Project: Image Classification

Deadline: Sunday 2 November at midnight

Group: 5 people

Contexte

The objective of this project is to apply MLOps principles by developing a complete machine learning pipeline for **image classification**.

You will work on a **binary classification** problem: distinguishing images of **dandelions** from those of **grass**.

The images will be stored in a plants_data table with three columns (url_source, url_s3, label), stored in a relational database.

Available resources

For this exercise, we will distinguish between only two environments:

- your local **"dev" environment** for testing/development. Make sure that all necessary services (Airflow, MLflow, etc.) are running locally for the dev phases. Feel free to use **Docker Compose** to initialize and launch your dev environment properly.

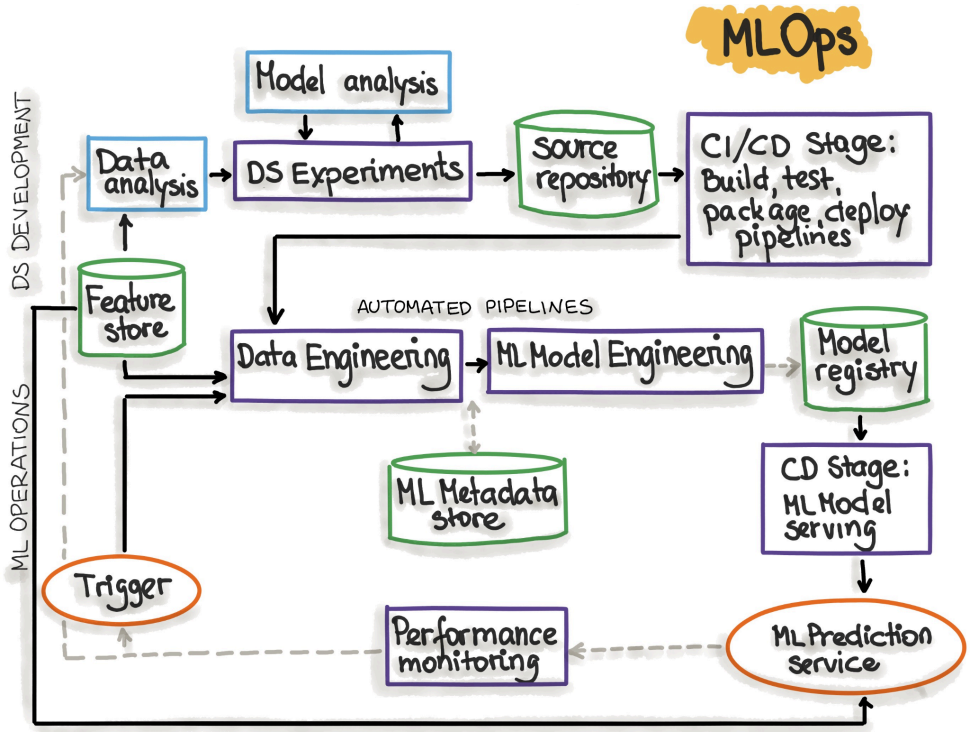
- A **"production" environment**, where all the services you need for your project (Airflow, MLFlow, etc.) must run on **your local Kubernetes cluster** (see: Docker Desktop or Minikube) or on the cloud if you have credits available on AWS, GCP, or Azure.
  Please note that running your project on the cloud is a "nice to have" and is therefore not mandatory. In terms of grading, you will not lose points if you can only stay on your local Kubernetes cluster. Feel free to use existing **Helm Charts** to deploy your services on your Kubernetes cluster.

Objectives

1. **Extract** and **preprocess data**: download images from URLs, clean them, and prepare them for training.

2. Build a **classification model**: develop a deep learning model that can classify images into two categories (dandelion vs. grass). Use the technology best suited to your needs: FastAI, AutoGluon, PyTorch, Tensorflow, etc.
3. Store the model on **AWS S3 (Minio)**: once trained, the model must be saved in an S3 bucket.
4. **Track models and experiments** with **MLFlow**: use MLFlow to record performance and track model versions.
5. Develop an **API**: build an API that allows you to send an image and receive a prediction. Use the technology best suited to your needs: MLFlow, FastAPI, Flask, KServe, Torch Serve, etc.
6. Create a **WebApp to interact** with the model: develop a simple user interface to view prediction results. Use the technology that best suits your needs: Gradio, Streamlit, Voila, etc.
7. **Dockerize your API** and deploy it on a local (or remote) Kubernetes cluster (Docker Desktop or MiniKube). Use **GitHub Actions CI/CD** for this (see github actions self hosted runner).
8. **Version** and **document** the project on GitHub: the entire project must be hosted on GitHub, with a clean file structure and clear documentation.

9. Create a retraining **pipeline** with Apache **Airflow**: set up a pipeline that retrieves new data, updates the model, and redeploys it.
10. Add **monitoring** to view all the metrics you want to monitor (e.g., airflow, API, model performance, etc.). Use the technology best suited to your needs: Elasticsearch + Kibana, Prometheus + Grafana, etc.
11. Use a **feature store**
12. You can add **load tests** (e.g., with Locust).
13. Now we want to do **Continuous Training** (CT). Add one or more Airflow DAGs with **triggers** that you define (new data, weekly training, declining model performance, etc.) to automatically retrain and deploy a new model.



orchestrator / pipelines (for data extraction, etc.)	Up to you: Airflow, Kubeflow Pipelines, Prefect
Deep learning frameworks	Up to you: FastAI, PyTorch, Keras / Tensorflow, AutoGluon
Model registry	AWS S3 (Minio)
Feature store	Up to you. Files by default
ML Metadata Store	MLflow, DVC
Source repository	GitHub
CI/CD	GitHub Actions
Containerization	Docker / Docker compose
Docker registry	Dockerhub (https://hub.docker.com/)
Serving (API)	Up to you: FastAPI, Flask, KServe, Torch Serve, MLflow, etc
Webapp	Up to you: Gradio, Streamlit, Voila, etc.
Deployment	Kubernetes
Monitoring	Up to you : Elasticsearch & Kibana, Prometheus & Grafana, Kubeflow Central Dashboard

Constraints and Best Practices

- **Modularity** and **maintainability**: well-structured and reusable code.
- **CI/CD**: set up tests and deployment pipelines.
- **Logging** and **monitoring**: add relevant logs and a monitoring system.
- **Testing**: add unit tests, integration tests, end-to-end tests, etc.
- **Security** and access management: properly manage access permissions to the database, S3, and API.
- **Documentation**: write a detailed README and document each part of the project.

Data management

The URLs for the images to download are available here:
https://github.com/btphan95/greenr-airflow/tree/master/data

The data is structured according to this pattern:

- label "dandelion" :
  https://raw.githubusercontent.com/btphan95/greenr-airflow/refs/heads/master/data/dandelion/00000000.jpg
  …
  https://raw.githubusercontent.com/btphan95/greenr-airflow/refs/heads/master/data/dandelion/00000199.jpg

- label "grass"
  https://raw.githubusercontent.com/btphan95/greenr-airflow/refs/heads/master/data/grass/00000000.jpg
  …
  https://raw.githubusercontent.com/btphan95/greenr-airflow/refs/heads/master/data/grass/00000199.jpg

Optional: create an SQL script to insert image metadata into the MySQL table, in a table named “plants_data,” with three columns (url_source, url_s3, label). You can then create a data extraction DAG (e.g., with Airflow) that will read this table, retrieve the associated images, and save them in an S3 bucket, which will then be used to train the model. It's up to you to decide which pipelines are best suited to the situation.

Deliverables

At the end of the project, you must send an email to prillard.martin@gmail.com containing:

- A link to a GitHub repository containing:
- all the code (preprocessing, training, operational API, interactive web app for testing the model, optional: supervision with monitoring, etc.)
- your functional pipeline/workflow DAGs (Airflow or other)
- CI/CD with GitHub Actions
- all your tests (unit tests, integration tests, end-to-end tests, etc.)
- project documentation
- a README explaining the technical choices and results obtained, with screenshots of the results on your production environment if possible
- The URLs of your Docker images on Dockerhub
- Ideally, on production resources: a trained model stored on AWS S3 (Minio) versioned model tracking via MLFlow

Notes:

- **Optimize** as much as possible!
- Don't forget to include a list of all the people in the group in the email!
- Optional: if you manage to deploy your API, web app, and/or monitoring on remote servers, you can specify the URLs/authentication to access them.

Evaluation

- Your project will be evaluated according to the following criteria:
- Code quality (compliance with best practices, modularity, documentation)
- Pipeline automation (Airflow and CI/CD)
- Model tracking and management with MLFlow
- Functionality and usability of the API (e.g., Swagger) and WebApp
- Objectives achieved and consistency of the overall architecture created
- Ability to collaborate and version effectively on GitHub

The performance of the model (accuracy, recall, f1-score) for this project **is not the priority**!

Jury Simulation and evaluation process

- Oral defense in RNCP format:
- 10 min demo and presentation by the group
- 15 min Q&A with jury