Projet MLOps : classification d'Images

Modalité du projet

Deadline : vendredi 16 Mai 2025 à minuit
Groupe : 2 à 4 personnes

Contexte

L'objectif de ce projet est de mettre en application les principes du MLOps en développant un pipeline complet de machine learning pour la classification d'images.
Vous travaillerez sur un problème de **classification binaire** : distinguer les images de pissenlits (**dandelion**) de celles d'herbe (**grass**).

Les images seront stockées dans une table `plants_data` avec trois colonnes (url_source, url_s3, label), stockée dans une base de données relationnelle.

Ressources disponibles

Nous distinguerons pour l'exercice seulement **deux environments** :

- votre environnement de "**dev**" en local pour vos tests / développements. Faites en sorte que tous les services nécessaires (Airflow, MLflow, etc.) tournent en local pour les phases de dev. N'hésitez pas à passer par un Docker Compose pour initialiser et lancer proprement votre environnement de dev.

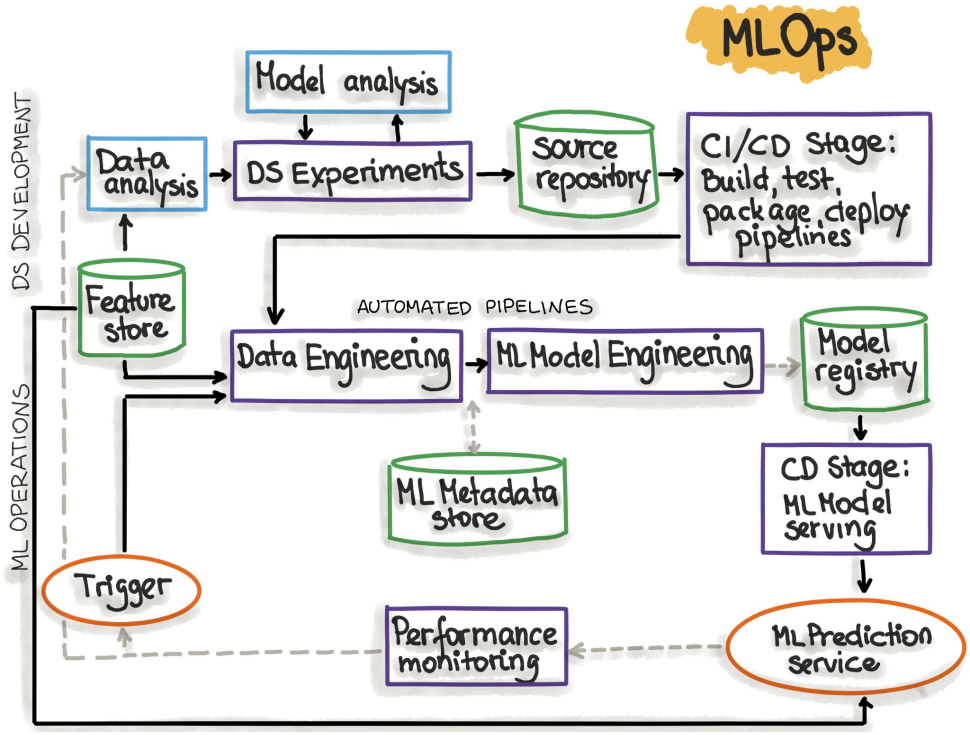
- un environnement de "**production**", oú l'entièreté de vos services nécessaires pour votre projet (Airflow, MLFlow, etc.) doit tourner sur votre cluster kubernetes en local (cf : Docker Desktop ou Minikube) ou sur le Cloud si vous avez des crédits disponibles sur AWS, GCP ou Azure. A noter que faire tourner votre projet sur le Cloud est un "nice to have" et n'est donc pas obligatoire. En ce qui concerne la notation, vous ne perdrez pas de points si vous ne pouvez que rester sur votre cluster Kubernetes en local.

Pensez à distinguer les exécutions sur les environnements (dev ou production) via la **CI/CD**. N'hésitez pas à utiliser des **Helm Chart** existants pour déployer vos services sur votre cluster Kubernetes.

Objectifs

1. **Extraire et prétraiter les données** : télécharger les images depuis les URLs, les nettoyer et les préparer pour l'entraînement.
2. **Construire un modèle de classification** : développer un modèle de deep learning permettant de classer les images en deux catégories (dandelion vs grass).
   Utilisez la techno la plus adaptée à vos besoins : FastAI, PyTorch, Tensorflow, etc.
3. **Stocker le modèle sur AWS S3** (Minio) : une fois entraîné, le modèle devra être sauvegardé dans un bucket **S3**.
4. **Créer un pipeline de réentraînement avec Apache Airflow** : mettre en place un pipeline qui récupère les nouvelles données, met à jour le modèle et le redéploie.
5. **Suivre les modèles et les expériences avec MLFlow** : utiliser MLFlow pour enregistrer les performances et suivre les versions du modèle.
6. **Développer une API** : construire une API permettant d'envoyer une image et de recevoir une prédiction. Utilisez la techno la plus adaptée à vos besoins : FastAPI, Flask, KServe, Torch Serve, etc.
7. **Créer une WebApp pour interagir avec le modèle** : développer une interface utilisateur simple pour visualiser les résultats des prédictions.
   Utilisez la techno la plus adaptée à vos besoins : Gradio, Streamlit, Voila, etc.
8. **Dockeriser** votre API et **déployer** là sur un cluster Kubernetes en local (Docker Desktop ou MiniKube). Utilisez la **CI/CD** de **GitHub Actions** pour celà.
9. **Versionner et documenter le projet sur GitHub** : tout le projet devra être hébergé sur **GitHub**, avec une structure de fichiers propre et une documentation claire.

10. Ajouter du **monitoring** pour visualiser l'ensemble des métriques que vous souhaitez monitorer (ex: airflow, API, performances du modèle, etc.). Utilisez la techno la plus adaptée à vos besoins : Elasticsearch + Kibana, Prometheus + Grafana, etc.
11. Vous pouvez ajouter des **tests de montée en charge** (ex: avec Locust)
12. On souhaite maintenant faire de l'**Entraînement Continue (CT)**. Ajouter un ou plusieurs DAG Airflow avec des triggers que vous définirez (nouvelle données, entrainement hebdomadaire, performances du modèle en baisse, etc.) pour réentraîner et déployer automatiquement un nouveau modèle.



orchestrateur / pipelines (pour extraction de données, etc.)	Airflow, Kubeflow Pipelines, Prefect
Deep learning frameworks	FastAI, PyTorch, Keras / Tensorflow, AutoGluon
Model registry	AWS S3 (Minio)
Feature store	Simulé par une BDD MySQL
ML Metadata Store	MLflow, DVC
Source repository	GitHub
CI/CD	GitHub Actions
Conteneurisation	Docker / Docker compose
Docker registry	Dockerhub (https://hub.docker.com/)
Serving (API)	Au choix : FastAPI, Flask, KServe, Torch Serve, MLflow, etc
Webapp d'inférence	Au choix : Gradio, Streamlit, Voila, etc.
Déploiement	Kubernetes
Monitoring	Au choix : Elasticsearch & Kibana, Prometheus & Grafana, Kubeflow Central Dashboard

Contraintes et Bonnes Pratiques

- **Modularité et maintenabilité** : code bien structuré et réutilisable.
- **CI/CD** : mettre en place des tests et des pipelines de déploiement.
- **Logging et monitoring** : ajouter des logs pertinents et un système de monitoring.
- **Testing** : ajouter des tests unitaires, test d'intégration, test end-to-end, etc.
- **Sécurité et gestion des accès** : gérer correctement les permissions des accès à la base de données, S3 et l'API.
- **Documentation** : rédiger un README détaillé et documenter chaque partie du projet.

La gestion des données

Les URLs des images à télécharger sont disponibles ici :
https://github.com/btphan95/greenr-airflow/tree/master/dat

faire un script SQL pour insérer les métadonnées des images dans la table MySQL, dans une **table** nommée "plants_data", avec trois colonnes (url_source, url_s3, label).
Aide : les données sont structurées selon ce pattern :

- Pour les données avec le 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

- Pour les données avec le 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

Vous pouvez ensuite faire un DAG d'extraction de données (ex: avec Airflow) qui va lire cette table, récupérer les images associées et les sauvegarder dans un bucket S3, utilisé ensuite pour entraîner le modèle. A vous de voir les pipelines les plus adaptés à la situation.

Livrables

À la fin du projet, vous devrez fournir via un mail à prillard.martin@gmail.com :

1. Un lien d'un **repository GitHub** contenant :
    - tout le code (preprocessing, entraînement, API opérationnelle, Webapp intéractive permettant de tester le modèle, optionnel : supervision avec le monitoring, etc.)
    - vos DAG de pipelines / workflows fonctionnels (Airflow ou autre)
    - la CI/CD avec Github Actions
    - l'ensemble de vos tests (tests unitaires, test d'intégration, test end-to-end, etc.)
    - la documentation du projet
    - un **README** expliquant les choix techniques et les résultats obtenus, avec si possible des screenshots des résultats sur votre environnement de production
2. Les URLs de vos images docker sur Dockerhub
3. Idéalement, sur les ressources de production :
    - un **modèle entraîné** stocké sur AWS S3 (Minio)
    - un **suivi des modèles versionnés via MLFlow**

Notes :

- Optimisez un maximum de choses !
- N'oubliez pas de préciser dans le mail la **liste de toutes les personnes** du groupe !
- Optionnel : si vous arrivez à déployer votre API, Webapp et/ou supervision sur des serveurs distants, vous pouvez préciser les URLs / authentification pour y accéder.

Évaluation

Votre projet sera évalué selon les critères suivants :

- **Qualité du code** (respect des bonnes pratiques, modularité, documentation)
- **Automatisation du pipeline** (Airflow et CI/CD)
- **Suivi et gestion des modèles avec MLFlow**
- **Fonctionnalité et ergonomie de l'API (ex: Swagger) et de la WebApp**
- **Objectifs réalisés et cohérence de l'architecture globale créée**
- **Capacité à collaborer et versionner efficacement sur GitHub**

La performance du modèle (précision, recall, f1-score) pour ce projet **n'est pas la priorité !**

Bon courage et bon développement ! 🚀