PROJET AIRFLOW LAB2

Pipeline ML avec orchestration Airflow

1. COMMENT FONCTIONNE AIRFLOW ?

Airflow orchestre des workflows via des DAGs (Directed Acyclic Graphs).

Dans ce projet, on utilise l'architecture suivante :

- Scheduler : surveille et déclenche les DAGs selon leur planning
- CeleryExecutor : distribue les tâches aux workers
- Workers : exécutent les tâches Python/Bash en parallèle
- PostgreSQL : stocke les métadonnées et l'état des exécutions
- Webserver : interface UI pour monitorer et déclencher les DAGs

Le principe est simple : le scheduler lit les DAGs, planifie les tâches, et les envoie aux workers qui les exécutent de manière distribuée.

2. WORKFLOW DU DAG 'AIRFLOW_LAB2' (PIPELINE ML)

Ce DAG orchestré en séquence entraîne un modèle de régression logistique pour prédire les clics publicitaires.

Étapes du pipeline :

1. load_data_task

   → Charge le dataset advertising.csv (1000 lignes)

   → Sauvegarde en pickle

2. data_preprocessing_task
   → Split train/test (70/30)
   → Feature scaling (MinMaxScaler + StandardScaler)
   → Features : temps sur site, âge, revenu, usage internet, genre

3. separate_data_outputs_task
   → Prépare les données pour l'entraînement

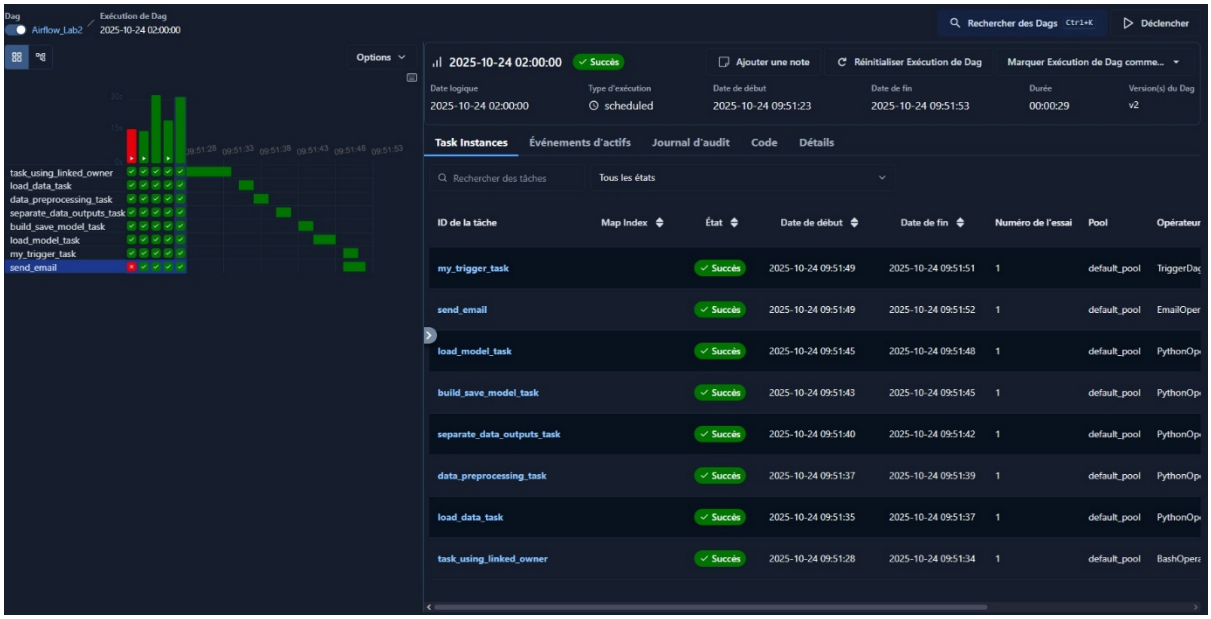
4. build_save_model_task
   → Entraîne un modèle LogisticRegression
   → Sauvegarde dans /model/model.sav

5. load_model_task
   → Charge le modèle sauvegardé
   → Évalue la performance sur le test set

6. send_email
   → Envoie une notification email de succès

7. my_trigger_task
   → Déclenche automatiquement le DAG Flask API

Résultat (voir screenshot airflow_lab2.jpg) :



3. WORKFLOW DU DAG 'AIRFLOW_LAB2_FLASK' (API)

Ce DAG démarre une API Flask qui interroge l'API REST d'Airflow pour afficher le statut du dernier run du pipeline ML.

Fonctionnement :

- Tâche unique : start_Flask_API
  → Lance un serveur Flask sur le port 5555
  → La tâche bloque volontairement pour maintenir l'API active

- Routes exposées :
  - / : redirection selon statut
  - /success : affiche page de succès si le dernier run est OK
  - /failure : affiche page d'échec sinon
  - /health : health check

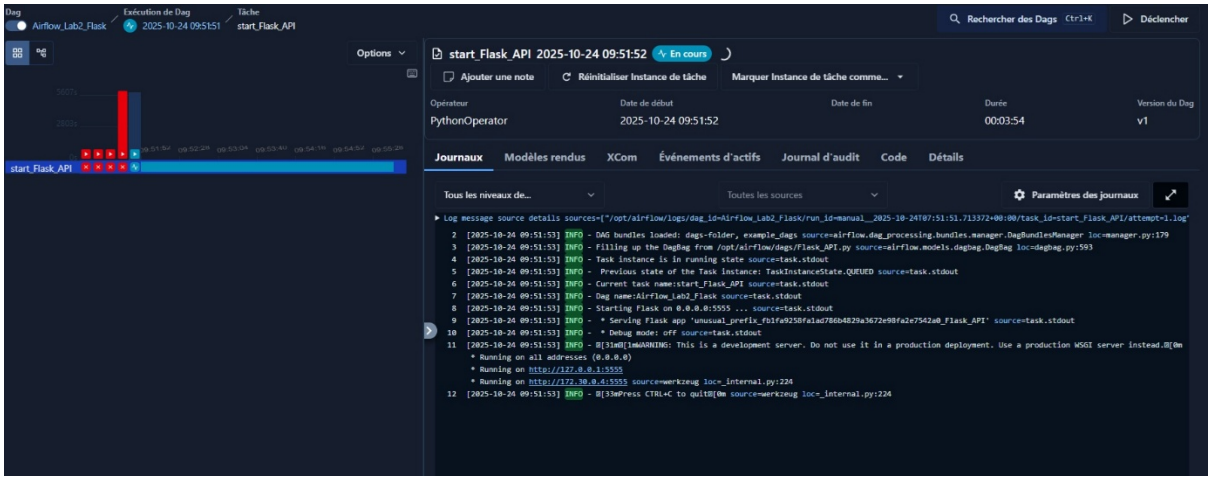
• Mécanisme :

- Flask interroge l'API REST Airflow (http://airflow-apiserver:8080/api/v2)

- Récupère le dernier dag_run de 'Airflow_Lab2'

- Affiche le statut avec templates HTML

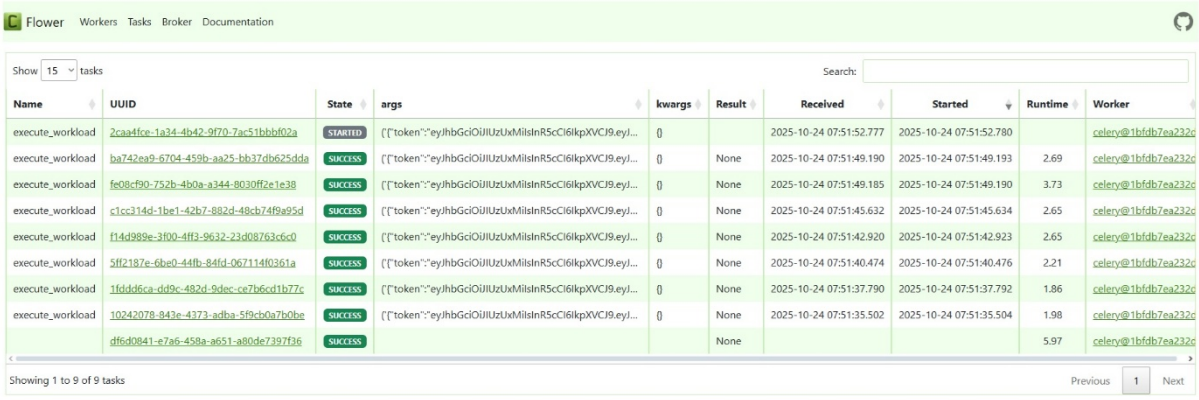
Résultat :



4. MONITORING DES WORKERS (FLOWER)

Flower est l'interface de monitoring de Celery. Elle affiche l'état des workers et les tâches distribuées en temps réel.

Observations (voir screenshot flower.jpg) :



Cela confirme que CeleryExecutor fonctionne correctement et distribue

les tâches Python du pipeline ML aux workers disponibles.

5. SYNTHÈSE

Ce projet démontre l'utilisation d'Airflow pour orchestrer un pipeline

ML complet :

✓ Chargement de données

✓ Prétraitement et feature engineering

✓ Entraînement d'un modèle ML

✓ Évaluation et sauvegarde

✓ Notification par email

✓ Déploiement automatique d'une API Flask

L'architecture distribuée avec CeleryExecutor permet l'exécution parallèle et scalable des tâches. Le système est conteneurisé avec Docker Compose pour un déploiement reproductible.

Stack technique :

- Apache Airflow 3.1.0
- CeleryExecutor + Redis
- PostgreSQL 16
- Flask + scikit-learn
- Docker Compose