

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

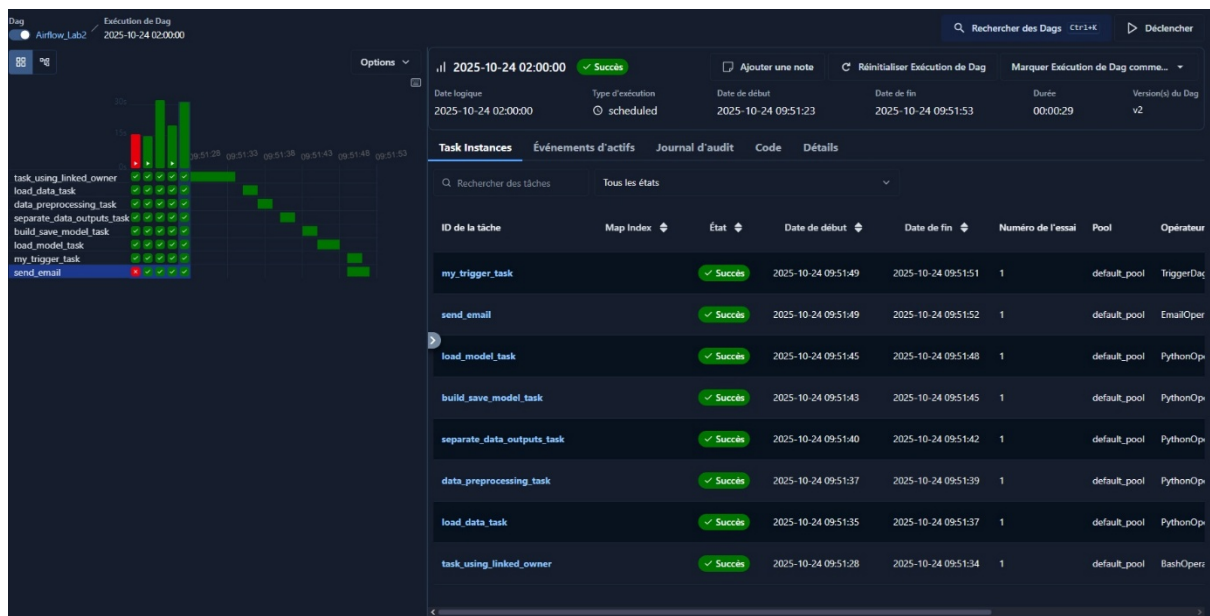
6. send_email

- Envoie une notification email de succès

7. my_trigger_task

- Déclenche automatiquement le DAG Flask API

Résultat (voir screenshot airflow_lab2.jpg) :



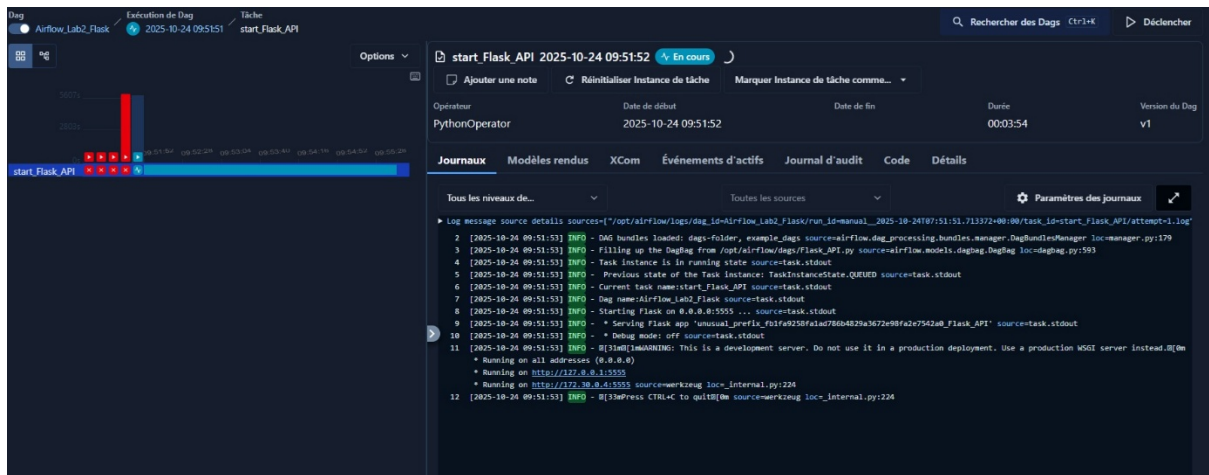
3. WORKFLOW DU DAG 'AIRFLOW_LAB2_FLASK' (API)

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

- ## Résultat :



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) :

Name	UUID	Status	args	kargs	Result	Received	Started	Runtime	Worker
execute_workload	2caadfce-1a34-d4d2-9f70-7ac51bbcb02a	STARTED	[{"token":"","eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJ...}	0		2025-10-24 07:51:52.777	2025-10-24 07:51:52.780		cclery@1bfdbf7ea232c
execute_workload	ba742e98-6704-459b-aa25-bb37db625dda	SUCCESS	[{"token":"","eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJ...}	0	None	2025-10-24 07:51:49.190	2025-10-24 07:51:49.193	2.69	cclery@1bfdbf7ea232c
execute_workload	f608c90-752b-4b0a-a344-8030ff2e1e38	SUCCESS	[{"token":"","eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJ...}	0	None	2025-10-24 07:51:49.185	2025-10-24 07:51:49.190	3.73	cclery@1bfdbf7ea232c
execute_workload	c1cc14dd-1be1-42b7-882d-74fb9a95d	SUCCESS	[{"token":"","eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJ...}	0	None	2025-10-24 07:51:45.632	2025-10-24 07:51:45.634	2.65	cclery@1bfdbf7ea232c
execute_workload	f14d989e-3f00-4ff3-9632-23d08763c6c0	SUCCESS	[{"token":"","eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJ...}	0	None	2025-10-24 07:51:42.920	2025-10-24 07:51:42.923	2.65	cclery@1bfdbf7ea232c
execute_workload	5ff2187e-6be0-44fb-84fd-067114f0361a	SUCCESS	[{"token":"","eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJ...}	0	None	2025-10-24 07:51:40.474	2025-10-24 07:51:40.476	2.21	cclery@1bfdbf7ea232c
execute_workload	1f6dd6ca-dd9c-482d-9dec-ce7b6cd1b77c	SUCCESS	[{"token":"","eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJ...}	0	None	2025-10-24 07:51:37.790	2025-10-24 07:51:37.792	1.98	cclery@1bfdbf7ea232c
execute_workload	10242078-843e-4373-adfa-5fbc0ba7d0be	SUCCESS	[{"token":"","eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJ...}	0	None	2025-10-24 07:51:35.502	2025-10-24 07:51:35.504	1.86	cclery@1bfdbf7ea232c
	dfe5d0841-e7a6-458a-bes01-a850d7397f36	SUCCESS			None			5.97	cclery@1bfdbf7ea232c

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