TP 4 : Pipelines de Données avec Apache Airflow

Cours EFREI 2024-2025

Yvann VINCENT
Machine Learning Engineer

Introduction

Ce TP vous guidera dans la construction d'un pipeline de données en utilisant Apache Airflow. L'objectif est de vous initier à cet outil de gestion de workflows tout en renforçant vos connaissances sur les Data Lakes et les architectures ETL.

Vous travaillerez à nouveau avec le dataset WikiText V2, mais cette fois-ci, vous utiliserez Airflow pour orchestrer un pipeline allant de la couche **Raw** (S3) à **Staging** (MySQL) puis à **Curated** (MongoDB).

—

1 Exercice 1 : Installation et Configuration d'Airflow

1.1 Objectif

Installer et configurer Apache Airflow dans un environnement Docker.

1.2 Étapes

1. **Création du répertoire Airflow :**

```
mkdir airflow-docker
cd airflow-docker
```

2. **Configuration du fichier Docker Compose :** Créez un fichier `docker-compose.yaml` avec le contenu suivant :

```
version: '3.8'
services:
  airflow-scheduler:
    image: apache/airflow:2.5.0
    restart: always
    container_name: airflow-scheduler
    depends_on:
      - airflow-webserver
    volumes:
      - ./dags:/opt/airflow/dags
      - ./logs:/opt/airflow/logs
      - ./plugins:/opt/airflow/plugins
    environment:
      - AIRFLOW__CORE__EXECUTOR=LocalExecutor
    command: scheduler

  airflow-webserver:
    image: apache/airflow:2.5.0
    restart: always
    container_name: airflow-webserver
    ports:
      - "8080:8080"
    depends_on:
      - airflow-init
    volumes:
      - ./dags:/opt/airflow/dags
      - ./logs:/opt/airflow/logs
      - ./plugins:/opt/airflow/plugins
    command: webserver

  airflow-init:
    image: apache/airflow:2.5.0
    container_name: airflow-init
    volumes:
      - ./dags:/opt/airflow/dags
      - ./logs:/opt/airflow/logs
      - ./plugins:/opt/airflow/plugins
    environment:
      - AIRFLOW__CORE__EXECUTOR=LocalExecutor
    command: bash -c "airflow db init && airflow users create \
      --username admin --password admin --firstname Air --lastname Flow --role Admin --emai
```

3. **Démarrage d'Airflow :** Exécutez les commandes suivantes pour initialiser et démarrer les services :

```
docker-compose up airflow-init
docker-compose up -d
```

4. **Accès à l'interface Web d'Airflow :** Rendez-vous sur http://localhost:8080 et connectez-vous avec les identifiants suivants :

   - **Nom d'utilisateur** : `admin`
   - **Mot de passe** : `admin`

—

2 Exercice 2 : Création d'un Pipeline Airflow

2.1 Objectif

Définir un pipeline (DAG) dans Airflow pour orchestrer les étapes ETL.

2.2 Instructions

1. Dans le répertoire `dags`, créez un fichier Python appelé `data_lake_pipeline.py`. 2. Implémentez le pipeline Airflow pour effectuer les tâches suivantes :

- **Extraction** : Téléchargez les données depuis Hugging Face et stockez-les dans le bucket **raw**.
- **Transformation** : Nettoyez les données et insérez-les dans une base MySQL.
- **Chargement** : Récupérez les données nettoyées, tokenisez-les, et insérez-les dans une collection MongoDB.

2.3 Exemple de Code Airflow

Voici un exemple minimal de DAG Airflow pour ce TP :

```python
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime
import boto3
import mysql.connector
import pymongo
from transformers import AutoTokenizer

# Fonctions pour chaque tâche
# Ici, votre but est d'importer les fonctions que vous avez
# codées lors du TP3. Si vous n'avez pas eu le temps de les
# finir, c'est le moment.
# Naturellement les noms que j'ai donnés ici sont des exemples,
# à vous de nommer les fonctions comme vous le souhaitez.

def data_to_raw(**kwargs):
    pass

def raw_to_curated(**kwargs):
    pass

def curated_to_staging(**kwargs):
    pass

# Définition du DAG
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2024, 1, 1),
    'retries': 1,
}

with DAG('data_lake_pipeline', default_args=default_args, schedule_interval=None) as dag:
```

```
    extract_task = PythonOperator(
        task_id='extract_data',
        python_callable=extract_data
    )

    transform_task = PythonOperator(
        task_id='transform_data',
        python_callable=transform_data
    )

    load_task = PythonOperator(
        task_id='load_data',
        python_callable=load_data
    )

    # Dépendances des tâches
    extract_task >> transform_task >> load_task
```

---

3 Exercice 3 : Test et Validation du Pipeline

3.1 Objectif

Exécuter et valider le pipeline dans Airflow.

3.2 Instructions

1. Démarrez le serveur Airflow si ce n'est pas déjà fait :

```
docker-compose up -d
```

2. Dans l'interface Web, activez et exécutez le DAG `data_lake_pipeline`.
3. Vérifiez les journaux pour chaque tâche afin de valider leur exécution.
4. Utilisez les outils CLI pour vérifier les données :

- **Raw (S3)** : Utilisez AWS CLI pour lister les objets dans le bucket `raw`.
- **Staging (MySQL)** : Connectez-vous à la base MySQL pour inspecter les données nettoyées.
- **Curated (MongoDB)** : Utilisez MongoDB Compass ou la CLI pour vérifier les données tokenisées.

—

4 Exercice 4 : Création d'un Pipeline ETL avec Apache Hop

4.1 Objectif

Découvrir une alternative graphique à Apache Airflow pour construire un pipeline ETL : Apache Hop. Vous utiliserez Hop pour orchestrer un pipeline allant de la couche **Raw** (S3) à **Staging** (MySQL) puis à **Curated** (MongoDB).

4.2 Instructions

1. Installation et Configuration

1. Téléchargez Apache Hop depuis le site officiel : https://hop.apache.org/.
2. Décompressez le fichier téléchargé et placez-le dans un répertoire local.
3. Lancez l'interface graphique Hop GUI :

   ```
   ./hop-gui.sh
   ```

   Sous Windows, double-cliquez sur `hop-gui.bat`.
4. Dans l'interface, créez un nouveau projet appelé `HopDataLake`.

2. Création d'un Pipeline ETL

1. Dans votre projet, créez une nouvelle transformation et nommez-la `etl_pipeline.hop`.
2. Ajoutez les étapes suivantes au pipeline :
   - **Extraction (Raw)** :
     - Ajoutez un nœud **S3 Input** pour télécharger les données depuis le bucket `raw`.
     - Ajoutez un nœud **Data Validator** pour vérifier la structure et la qualité des données.
   - **Transformation (Staging)** :
     - Ajoutez un nœud **Database Connection** et configurez une connexion à MySQL.
     - Créez une table `texts` pour stocker les données nettoyées.
     - Ajoutez un nœud **Table Output** pour insérer les données dans MySQL.
   - **Chargement (Curated)** :
     - Ajoutez un nœud **Script Value** pour tokeniser les textes en utilisant un script Python intégré.
     - Ajoutez un nœud **MongoDB Output** pour insérer les données tokenisées dans MongoDB.

3. Configuration des Connexions

- Pour MySQL, configurez les paramètres de connexion avec l'utilisateur et le mot de passe créés précédemment (`root`/`root`).
- Pour MongoDB, ajoutez une connexion à la base `curated` et vérifiez la collection `wikitext`.
- Testez chaque connexion dans Hop pour vérifier leur bon fonctionnement.

4. Exécution et Validation

- Exécutez le pipeline en cliquant sur le bouton d'exécution dans Hop GUI.
- Surveillez l'exécution pour identifier et corriger les éventuelles erreurs.
- Vérifiez les données :
  - **Raw (S3)** : Vérifiez que les données brutes ont été téléchargées correctement.
  - **Staging (MySQL)** : Connectez-vous à MySQL pour valider la table `texts`.
  - **Curated (MongoDB)** : Utilisez MongoDB Compass ou la CLI pour inspecter les documents insérés.

—

4.3 Conclusion

Avec cet exercice, vous avez découvert une alternative graphique à Airflow pour construire un pipeline ETL. Apache Hop permet une approche intuitive et visuelle tout en offrant des fonctionnalités similaires pour orchestrer un pipeline complexe. Comparez cette expérience avec celle d'Airflow pour mieux comprendre les avantages et limites de chaque outil.

—

Conclusion Générale

À la fin de ce TP, vous aurez :

- Exploré Apache Airflow pour orchestrer des pipelines ETL complexes.
- Découvert Talend comme alternative graphique pour la gestion de workflows ETL.
- Comparé les avantages et inconvénients des outils code-first (Airflow) et UI-first (Hop).

Dans le prochain TP, nous nous concentrerons sur l'extraction de données en temps réel à partir d'API REST et leur intégration dans un Data Lake avec un cluster Elasticsearch.

Par ailleurs, une petite minute de silence pour Talend Open-Studio qui a maintenant été retiré de la circulation.