Rapport de Projet : Pipeline de Données Météo avec Airflow et MySQL

Introduction

Le but de ce projet est de créer un pipeline **end-to-end** qui collecte des données météorologiques depuis l'API OpenWeather, les nettoie, puis les stocke dans une base de données **MySQL** pour les analyser. Ce pipeline est automatisé à l'aide d'Apache **Airflow** et implémente les concepts clés de l'ingénierie des données : collecte, transformation, chargement (ETL) et stockage.

Architecture du Pipeline

1. Collecte des données

Les données météorologiques sont récupérées via l'API OpenWeather. Nous avons utilisé une liste de villes pour interroger l'API et obtenir des données météo sous forme de fichiers **JSON**.

- API utilisée : **OpenWeather API**
- Liste de villes : **Dakar, Thies, Diourbel, etc.**
- Stockage : Les données brutes sont sauvegardées sous forme de fichiers JSON dans un répertoire local nommé `data/`.

2. Transformation des données

Les fichiers JSON collectés contiennent des informations météorologiques complexes. Nous avons utilisé une fonction Python pour **nettoyer** et **transformer** ces données. Cela inclut :

- Extraction des informations clés comme : température, humidité, pression, condition météo, vitesse du vent, etc.
- Gestion des valeurs manquantes et des structures imbriquées.
- Conversion des données transformées en **DataFrame Pandas**.

3. Chargement dans MySQL

Les données nettoyées sont ensuite chargées dans une base de données **MySQL**. Une table appelée `engine` a été créée pour stocker les données météo, avec les colonnes suivantes :

- `city`, `temperature`, `humidity`, `pressure`, `weather_condition`, `wind_speed`, `wind_deg`, `clouds`, `latitude`, `longitude`, `rain_1h`.

L'insertion dans MySQL se fait via une connexion Python utilisant le module `mysql-connector-python`.

4. Automatisation avec Apache Airflow

Le pipeline est automatisé à l'aide de **Apache Airflow**. Airflow permet de planifier et d'orchestrer les tâches, y compris :

- Collecter les données à des intervalles réguliers.
- Nettoyer et transformer les données.
- Charger les données nettoyées dans MySQL.

Étapes de développement

Étape 1 : Collecte des données météo

Nous avons créé une fonction Python `fetch_weather_data()` qui interroge l'API OpenWeather pour chaque ville dans la liste et sauvegarde les réponses dans des fichiers **JSON**.

Extrait de code :

```
def fetch_weather_data(city):
    url = f"http://api.openweathermap.org/data/2.5/weather?
q={city}&appid={API_KEY}&units=metric"
    response = requests.get(url)
    if response.status_code == 200:
```

```
        weather_data = response.json()
        with open(f"data/{city}.json", "w") as json_file:
            json.dump(weather_data, json_file, indent=4)
        print(f"Weather data for {city} saved.")
    else:
        print(f"Error fetching data for {city}")
```

Étape 2 : Nettoyage et transformation des données

Une fonction `clean_weather_data()` a été mise en place pour traiter les fichiers JSON bruts et en extraire les informations pertinentes, en les formatant pour être compatibles avec le modèle de base de données MySQL.

Extrait de code :

```
def clean_weather_data(file_path):
    with open(file_path, "r") as file:
        data = json.load(file)
    cleaned_data = {
        "city": data.get("name"),
        "temperature": data.get("main", {}).get("temp"),
        "weather_condition": data.get("weather", [{}])[0].g
et("description"),
        # Autres colonnes...
    }
    df_cleaned = pd.DataFrame([cleaned_data])
    return df_cleaned
```

Étape 3 : Chargement des données dans MySQL

La fonction `load_data_to_mysql()` prend en entrée les fichiers nettoyés et les insère dans la base de données MySQL.

Extrait de code :

```
def load_data_to_mysql(file_path):
    conn = mysql.connector.connect(
```

```
        host="127.0.0.1",
        user="boot",
        password="",
        database="test"
    )
    cur = conn.cursor()
    df_cleaned = clean_weather_data(file_path)
    for index, row in df_cleaned.iterrows():
        cur.execute(
            "INSERT INTO engine (city, temperature, weather
_condition) VALUES (%s, %s, %s)",
            (row["city"], row["temperature"], row["weather_
condition"])
        )
    conn.commit()
    cur.close()
    conn.close()
```

Étape 4 : Automatisation avec Airflow

Une fois le pipeline ETL défini, nous l'avons intégré à Airflow pour automatiser les tâches. Un **DAG (Directed Acyclic Graph)** dans Airflow permet de définir le flux des tâches de collecte, transformation, et chargement.

Configuration Airflow :

- **Base de données Airflow** : Initialisée avec `airflow db init`.
- **DAG** : Programmé pour exécuter les tâches de collecte, transformation, et chargement sur une base quotidienne.

Défis rencontrés et solutions

1. Problèmes d'installation d'Airflow

Lors de l'installation d'Apache Airflow, une erreur liée au module `google-re2` a été rencontrée. L'installation des outils de compilation natifs (`build-essential`, `libre2-dev`) a résolu le problème.

2. Gestion des erreurs d'insertion dans MySQL

Certaines erreurs se sont produites lors de l'insertion des données dans MySQL (valeurs manquantes, erreurs de types). Nous avons ajouté des vérifications de type et des valeurs par défaut pour gérer ces cas.

3. Orchestration et planification avec Airflow

Le démarrage du scheduler et du webserver d'Airflow a posé quelques difficultés (commande non reconnue), qui ont été résolues en ajoutant le bon chemin vers le binaire Airflow dans la variable **PATH**.

Résultats et bénéfices

- **Pipeline automatisé** : Grâce à Airflow, nous avons automatisé la collecte, transformation et chargement des données météorologiques pour un ensemble de villes.
- **Base de données structurée** : Les données nettoyées et bien structurées sont maintenant disponibles dans MySQL pour une analyse ultérieure.
- **Scalabilité** : Ce pipeline peut facilement être étendu pour ajouter plus de villes ou collecter des données supplémentaires comme la précipitation ou la qualité de l'air.

Améliorations futures

- **Ajout de tests automatisés** pour valider la transformation des données.
- **Visualisation des données** : Utiliser des outils comme **Tableau** ou **Matplotlib** pour visualiser les tendances météorologiques.
- **Intégration avec des services cloud** comme AWS S3 ou Google Cloud pour stocker les données brutes et les résultats.

Conclusion

Ce projet montre comment construire un pipeline de données end-to-end, depuis la collecte de données météo jusqu'au stockage et à l'automatisation à l'aide d'Airflow. Il est également extensible à d'autres cas d'utilisation comme l'intégration avec des systèmes de monitoring en temps réel ou la prédiction de tendances météorologiques à l'aide de modèles de machine learning.

Adama Traoré

le 15/09/2024