TP 3 : Pipelines de Données avec Intégration SQL et NoSQL

Cours EFREI 2024-2025

Yvann VINCENT
Machine Learning Engineer

Introduction

Ce TP vous guidera à travers la création d'un pipeline de données pour un Data Lake en intégrant des bases de données SQL et NoSQL dans les zones **Staging** et **Curated**.

Vous utiliserez le dataset WikiText V2 (cliquez ici) contenant des textes issus de Wikipédia. J'ai écoute votre feedback au sujet du dataset PFAM qui était trop gros et avait un temps de traitement trop long sur vos ordinateurs. Par ailleurs, Wikitext n'est pas constitué de sous-dossiers avec chacun des fichiers csv à recombiner.

Le workflow est similaire au TP2, mais avec deux nouveautés :

- Utilisation de **SQLite** pour structurer les données nettoyées dans **Staging**.
- Intégration de **MongoDB** pour organiser les données tokenisées dans **Curated**.
- J'ai aussi essayé de rendre le TP plus simple de façon à ce que tout le monde puisse l'avoir fini à la fin de la séance.

—

1 Exercice 1 : Configuration de l'Environnement

1.1 Objectif

Suite à la dernière fois, vous devriez avoir Docker et LocalStack déjà installés. Vous devrez installer MongoDB si ce n'est pas déjà fait, et vous assurer que SQLite est fonctionnel pour ce TP.

1.2 Étapes

1. Cette fois-ci, vous allez créer votre propre repository GitHub et mettre en place les bonnes pratiques que nous avons vues lors des deux premières séances. Vous êtes libres de structurer votre solution comme vous le voulez. Le but est de vous familiariser avec le workflow complet du travail avec un Data Lake. Naturellement, je partagerai quand même la correction.

2. Installez les dépendances nécessaires à l'environnement Python :

   ```
   pip install pandas boto3 localstack-client pymongo transformers
   ```

   Note : SQLite est intégré nativement dans Python. Si vous utilisez un environnement Conda, vous pouvez vous assurer de sa disponibilité en exécutant :

   ```
   conda install sqlite
   ```

3. Configurez et démarrez LocalStack et MongoDB : Assurez-vous d'avoir un fichier `docker-compose.yml` configuré pour inclure les services LocalStack et MongoDB :

   ```yaml
   version: '3.8'

   services:
     localstack:
       image: localstack/localstack
       container_name: localstack
       ports:
         - "4566:4566"
         - "4572:4572"
       environment:
         - SERVICES=s3
         - DOCKER_HOST=unix:///var/run/docker.sock

     mongodb:
       image: mongo:latest
       container_name: mongodb
       ports:
         - "27017:27017"
   ```

   Placez ce fichier dans votre répertoire de travail, puis exécutez :

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

   Vous pouvez vérifier que les conteneurs sont en cours d'exécution avec :

   ```
   docker ps
   ```

   Si vous préférez démarrer MongoDB séparément, utilisez :

```
docker run -d --name mongodb -p 27017:27017 mongo:latest
```

4. Testez SQLite : Vérifiez que SQLite fonctionne correctement en exécutant une commande Python pour créer et interroger une base de données SQLite minimale :

```
import sqlite3

# Connexion à une base de données (ou création si elle n'existe pas)
conn = sqlite3.connect('test.db')
cursor = conn.cursor()

# Création d'une table de test
cursor.execute('CREATE TABLE IF NOT EXISTS test_table (id INTEGER PRIMARY KEY, value TEXT)
cursor.execute('INSERT INTO test_table (value) VALUES ("Hello, SQLite!")')
conn.commit()

# Interrogation de la base
cursor.execute('SELECT * FROM test_table')
print(cursor.fetchall())

conn.close()
```

5. Configurez les buckets S3 si nécessaire : Vérifiez la disponibilité des buckets S3 du dernier TP et recréez-les si nécessaire :

```
aws --endpoint-url=http://localhost:4566 s3 mb s3://raw
aws --endpoint-url=http://localhost:4566 s3 mb s3://staging
aws --endpoint-url=http://localhost:4566 s3 mb s3://curated
```

6. Si besoin, n'hésitez pas à vous référer au TP2 pour l'installation et les étapes précédentes.

—

2 Exercice 2 : Téléchargement et Intégration des Données dans Raw

2.1 Objectif

Télécharger les données WikiText V2, les diviser en sous-dossiers (`train`, `test`, `dev`), et téléverser les fichiers dans le bucket **raw**.

2.2 Instructions

1. Téléchargez les données WikiText V2 depuis Hugging Face. Vous êtes libres de les télécharger manuellement depuis la source de votre choix, utiliser la librairie transformers, où bien le faire en CLI :

```
python -m datasets.download --dataset Salesforce/wikitext
```

2. Organisez les données dans un dossier `data/raw` avec les sous-dossiers `train`, `test`, et `dev`.

3. Créez un script d'unpacking pour extraire les données et les téléverser dans le bucket raw :

- Combiner les fichiers `train`, `test`, et `dev`.
- Téléverser le fichier combiné dans le bucket **raw**.
- N'oubliez pas d'ajouter un main avec un argparse dans votre script

4. Exécutez le script. Si tout fonctionne, vous pouvez passer à la suite.

—

3 Exercice 3 : Nettoyage et Structuration avec une Base Distante

3.1 Objectif

Voir comment un DBMS relationnel distant peut s'intégrer dans une architecture de Data Lake, en remplaçant le file storage ou le blob storage pour la zone de staging **staging**.

3.2 Instructions

1. Configuration de la Base Distante

- Si une instance MySQL ou PostgreSQL n'est pas disponible, démarrez-en une avec Docker :

```
docker run --name mysql -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=staging -p 3306:3306 -d mysql:
```

- Notez les identifiants :

- **Utilisateur** : `root`
- **Mot de passe** : `root`
- **Base de données** : `staging`

- Installez le client Python nécessaire pour interagir avec MySQL :

```
pip install mysql-connector-python
```

2. Créez un script pour préparer les données au staging

Ajoutez les étapes suivantes dans le fichier :

- Télécharger les données depuis le bucket **raw**.
- Nettoyer les données (suppression des doublons et des lignes vides).
- Se connecter à la base MySQL distante.
- Créer une table `texts` (si elle n'existe pas) avec les colonnes nécessaires.
- Insérer les données nettoyées dans cette table.
- Vérifier que les données sont bien insérées.

3. Requêtes SQL pour la Validation

Ajoutez un script ou une étape dans votre pipeline pour exécuter une requête SQL et vérifier que les données sont correctes. Par exemple :

```
SELECT COUNT(*) FROM texts WHERE text IS NOT NULL;
```

4. Interrogez la Base pour la Prochaine Étape

- À l'étape **Curated**, le script peut interroger la base pour récupérer les données directement, au lieu de passer par un fichier intermédiaire.

5. Exécutez le Script

Exécutez le script avec les arguments suivants :

```
python src/preprocess_to_staging.py --bucket_raw raw --db_host localhost --db_user root --db_passwor
```

6. Vérification Manuelle

Utilisez un client MySQL (par exemple, `mysql-cli` ou `DBeaver`) pour vous connecter à la base et inspecter les données. Assurez-vous que la table `texts` contient les données nettoyées.

3.3 Exemple de Code SQL

Pour interagir avec la base de données, voici un exemple de script Python minimaliste qui peut être intégré :

```python
import mysql.connector

# Connexion à la base MySQL
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="root",
    database="staging"
)

cursor = conn.cursor()

# Vérification des données
cursor.execute("SELECT COUNT(*) FROM texts WHERE text IS NOT NULL;")
count = cursor.fetchone()
print(f"Nombre de lignes valides : {count[0]}")

cursor.close()
conn.close()
```

Cette approche met davantage l'accent sur une utilisation réaliste des bases SQL dans un pipeline, en tirant parti des avantages qu'elles offrent pour le stockage et les requêtes complexes.

—

4 Exercice 4 : Tokenisation et Stockage NoSQL (Curated)

4.1 Objectif

Télécharger les données nettoyées depuis MySQL, tokeniser les textes avec un modèle de NLP, et organiser les données dans une base MongoDB locale pour leur utilisation future dans la zone **Curated**.

4.2 Instructions

1. Préparation

Assurez-vous que les services MySQL et MongoDB sont en cours d'exécution. Si ce n'est pas le cas, démarrez-les à l'aide de Docker :

```
docker start mysql
docker start mongodb
```

Installez les dépendances nécessaires pour interagir avec MySQL et MongoDB :

```
pip install pymysql pymongo transformers
```

2. Script pour passer les données de staging à curated

Ajoutez les étapes suivantes :

- **Connexion à MySQL** : Récupérez les données depuis la table `texts` créée à l'étape précédente.
- **Tokenisation des textes** : Utilisez le tokenizer de votre choix pour tokenizer les données. Si vous avez du mal à choisir, gpt2 ou distilbert-base-uncased peuvent être de bonnes options.
- **Connexion à MongoDB** : Connectez-vous à la base MongoDB locale.
- **Insertion dans la collection MongoDB** : Créez une collection appelée `wikitext` dans la base `curated` et insérez les données tokenisées tout en conservant les métadonnées.

Voici un exemple de structure de document JSON inséré dans MongoDB :

```
{
    "id": "1",
    "text": "This is a sample sentence.",
    "tokens": [2023, 2003, 1037, 7099, 6251, 1012],
    "metadata": {
        "source": "mysql",
        "processed_at": "2024-01-01T10:00:00Z"
    }
}
```

3. Vérification des Données dans MongoDB

Après avoir exécuté le script, utilisez un client MongoDB pour vérifier que les données ont été correctement insérées. Vous pouvez utiliser l'interface MongoDB Compass ou la CLI :

```
docker exec -it mongodb mongo
> use curated
> db.wikitext.find().limit(5).pretty()
```

4.3 Exemple de Script Python

Voici un exemple minimal de ce que pourrait contenir votre script :

```python
import pymysql
import pymongo
from transformers import AutoTokenizer
from datetime import datetime

# Connexion à MySQL
mysql_conn = pymysql.connect(
    host="localhost",
    user="root",
    password="password",
    database="staging"
)
cursor = mysql_conn.cursor(pymysql.cursors.DictCursor)
cursor.execute("SELECT * FROM texts")
rows = cursor.fetchall()

# Initialisation MongoDB
mongo_client = pymongo.MongoClient("mongodb://localhost:27017/")
mongo_db = mongo_client["curated"]
mongo_collection = mongo_db["wikitext"]

# Initialisation du tokenizer
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")

# Traitement et insertion dans MongoDB
for row in rows:
    tokens = tokenizer(row["text"], truncation=True, padding=True, max_length=128)["input_ids"]
    document = {
        "id": row["id"],
        "text": row["text"],
        "tokens": tokens,
        "metadata": {
            "source": "mysql",
            "processed_at": datetime.utcnow().isoformat()
        }
    }
    mongo_collection.insert_one(document)

print(f"Data successfully inserted into MongoDB collection '{mongo_collection.name}'.")

# Fermeture des connexions
cursor.close()
mysql_conn.close()
```

5. Nettoyage (Facultatif)

Si vous souhaitez vider la collection après des tests :

```
docker exec -it mongodb mongo
> use curated
> db.wikitext.drop()
```

Cela supprimera toutes les données dans la collection `wikitext`.

5 Exercice 5 (Facultatif) : Création et Exécution du Pipeline DVC

J'ai volontairement été moins directif dans mes instructions dans ce TP pour amener cet exercice. En vous inspirant du TP précédent, à vous de revoir (si nécessaire) l'implémentation de votre code de façon à pouvoir en faire un pipeline reproductible avec DVC.

Dans les prochains TP, nous travaillerons sur l'intégration de données à partir d'une API, avec une approche un peu différente puisque je vous ferai utiliser Apache Airflow pour faire la pipeline. Je vous montrerai également comme faire ce travail avec des outils plus graphiques comme Talend.