Projet : Big Data

Rakib SHEIKH
NoSQL 2024-2025
rsheikh1@myges.fr

Lien vers le template du GitHub

Le sujet est à disposition dans le dossier **Docs** ET sur **Teams**.

L'objectif de ce TP sera de déployer une architecture de type CAC 40 afin de vous familiariser avec les termes d'une architecture **Big Data**.

Pour cela, nous allons tout d'abord collecter des données. Mais ces données doivent être stockées quelque part. Comme nous ne savons pas la valeur qu'elles peuvent générer, nous devons alors les stocker dans un Data Lake représenté ici par le service Minio.

Une fois que les données sont stockées sur Minio, nous supposons que nous avons trouvé une valeur potentielle des données à exploiter, nous devons donc préparer les données brutes depuis Minio vers une base de données représentant un Data Warehouse. Cette partie définit donc une intégration de données.

Maintenant, vous vous rendez compte que votre équipe a rédigé un contrat d'intégration de données sous la forme d'un modèle en flocon. Comme vous n'êtes pas sûr que cette donnée sera réutilisée par les équipes externes à la vôtre, vous allez donc vous l'approprier pour votre équipe qui possède sa propre base de **données**. Cette base de **données** est en réalité un sous-ensemble du **Data Warehouse**, qui est physiquement détaché. C'est donc l'**appellation** d'un **Data Mart**. Par soucis de simplicité, nous n'allons pas utiliser de Datamart.

Enfin, les données sont **nettoyées** et prêtes à être **visualisées**, vous allez donc faire **appel** à un outil de visualisation de données de type **Tableau** voire **Power BI**.

⚠️ | Remarque : Java supérieur à 11 et Provided

Sous Intellij, il faut ajouter un paramètre de la VM supplémentaire pour faire sauter les erreurs de type IllegalAccessError

1. Allez dans les trois points vertical
2. Dans la liste déroulante Modify Option : Sélectionnez Add VM Option
3. Ajoutez le code suivant :

```
--add-exports java.base/sun.nio.ch=ALL-UNNAMED
```

Pour l'erreur provided à cause du provided dans le SBT:
· Il faudra sélectionner **Add dependencies with "provided" scope to classpath**

Faites attention à bien choisir une version LTS du backend Java (c'est à dire 8, 11 ou 17, le 21 n'est pas compatible)

Le descriptif des 5 Exercices :

- **Pour l'exercice 1 : La collecte des données et l'intégration de données**
  - L'objectif du TP1 est de récupérer les données des taxi jaunes de la ville de New York, actualisés à chaque mois, directement sur le site d'état de New York. Afin de récupérer les données, nous téléchargeons un fichier parquet du mois en question qui sera stocké vers un Datalake, représenté par le cluster Minio.

  Vous pouvez récupérer ces fichier à l'adresse suivante : https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

  - Afin de simplifier l'acheminement du fichier parquet vers le bucket minio, le TP 1 sera réalisé en deux temps.
    1. Téléchargez un fichier parquet et stockez le dans le dossier data/raw (depuis la racine du projet template).
    2. Uploadez un fichier parquet que vous venez de télécharger dans le dossier `nyc_raw` dans le cluster Minio.
    3. Faites en sorte que cela télécharge directement dans le Minio.
- **Pour le TP 2 : Le nettoyage de données et l'ingestion multi-branche**
  - L'entreprise du CAC40 fictive a finalement pu trouver une finalité amenant à utiliser les fichiers parquets dormants dans le Datalake. Nous allons devoir commencer à récupérer ces fichiers parquets qui sera intégré vers un nouveau service de base donnée postgresql jouant le rôle de Data Warehouse.

1. Nous allons valider l'ensemble des valeurs de notre donnée selon le contrat donnée par l'état de New York. Afin de simplifier la tâche des Machine Learning Engineer ainsi que les Data Scientist, nous stockerons le résultat sous la forme d'un parquet, dans Minio. Cette partie du code sera appelée la branche 1
2. *(Passez directement à l'exercice 3 avant de faire cette question)*

   Vous allez concevoir la suite du programme sous Scala / Spark, prêt à être mis en production afin de récupérer un fichier parquet et permettant de réaliser une ingestion de données vers postgresql Data Warehouse. Nous appellerons cet partie la branche 2. Assurez vous d'avoir en amont, l'ensemble des tables crées définit par l'exercice 3.

```
Parquet Files (Monthly)
  ↓
Single Spark Job with branching:
  ├─→ Parquet/Minio (for ML training - full historical data) (Branche 1)
  └─→ Apply transformations in memory → Postgres Datamart (Branche 2)
```

> **Note**
>
> Si vous n'êtes pas à l'aise avec Postgres, vous êtes libre de changer de SGBD à condition d'appliquer les modifications nécessaire dans le docker compose.

- **Pour l'exercice 3: Data Warehouse et création d'un modèle multi-dimensionnelle.**
  - À ce stade, nos données presque sont prêts à être requêté par notre équipe métier, mais un problème se pose : notre schéma de donnée actuel n'est pas optimisé pour l'analyse de données. En effet, notre table est orientée OLTP, ce qui signifie qu'elle est optimisé pour ajouter une course. Nous allons devoir transformer notre table vers un modèle en étoile, flocon ou constellation, dont vous justifierez votre choix dans le rapport finale.

  Une contrainte : vous êtes dans l'obligation d'utiliser le langage SQL du SGBD de votre choix.
  - Par soucis de simplicité du sujet, vous êtes libre utiliser le SGBD de votre choix sans tenir compte des propriété OLAP.
  - Vous aurez donc un script SQL pour chaque tâche distinct :
    1. `creation.sql` pour la création des tables en flocons avec les contraintes associés.
    2. `insertion.sql` pour insérer les données de référence que le métier vous a mis à disposition.
- **Pour l'exercice 4 : Visualisation de données**

- Lorsque vous avez fait le TP3, vous devriez normalement avoir une idée sur la restitution des données que vous souhaitez faire dans la partie visualisation de données.
  - Si ce n'est pas le cas, vous pouvez ouvrir un Notebook qui sera sauvegardé dans le dossier `notesbooks` pour réaliser votre Analyse Exploratoire de Données (EDA).
  - Pour les plus chaud d'entre vous, vous pouvez concevoir un tableau de bord à l'aide de Streamlit.
- Vous devez connecter votre outil de Data Visualisation à votre base de donnée (jouant le rôle de datamart) afin de produire les visualisations.

· **Pour l'exercice 5 : Implémentation d'un modèle de machine learning**

- Cette partie de l'exercice vous permettra d'avoir un avant goût du cours de MLOps qui aura lieu au deuxième semestre.

1. Votre objectif est de récupérer le fichier parquet qui vous est mis à votre disposition dans le cluster Minio afin de concevoir votre modèle de machine learning traditionnel.

Contrainte : L'entraînement et l'inférence est réalisé sous la forme d'un script python et non notebook !

- Votre target sera le prix payé en fonction des features que vous jugerez utile.
- La précision attendu est de moins de 10 RMSE. Vous êtes libre de choisir une autre métrique en justifiant son utilisation dans le mini-rapport.
- Documentez votre code python à l'aide de `pyment`, nous utiliserons la notation NumpyDoc
- Vérifiez que vous respectez bien la norme PEP 8 avec `flake8`.
- Assurez vous d'implémenter les tests unitaire sur les données entrés à la fois lors de l'entrainement et lors de l'inférence.
- En option : Proposez un front-end sous `streamlit` pour démontrer que votre modèle fonctionne.

Rendu :

· Un lien GitHub vers votre projet
· Par mail ainsi que dans le dossier docs, un mini rapport écrit décrivant l'ensemble des actions menés pour réaliser les 5 exercices ainsi que des captures d'écrans de votre dashboard

Règle de retard :

· Le rendu sera fixé par votre intervenant, à 6h00 du matin.
· Chaque heure de retard entamé sera sanctionné de −5 points. La sanction est appliqué dès 6h00m00s.

Annexes

```
├── README.md          <- The top-level README for developers using this project.
├── docker-compost.yml <- Launch all Data infrastrcture on local. With docker compose up command.
├── data
│   ├── external       <- Data from third party sources.
│   ├── interim        <- Intermediate data that has been transformed.
│   ├── processed      <- The final, canonical data sets for modeling.
│   └── raw            <- The original, immutable data dump.
│
│
├── notebooks          <- Jupyter notebooks. Naming convention is a number (for ordering),
│                         the creator's initials, and a short `-` delimited description, e.g.
│                         `1.0-jqp-initial-data-exploration`.
│
├── references         <- Data dictionaries, manuals, and all other explanatory materials.
│
├── reports            <- Generated analysis as HTML, PDF, LaTeX, etc. C'est
│   └── figures        <- Generated graphics and figures to be used in reporting
```

```
│
├── pyproject.toml   <- The requirements file for reproducing the analysis environment
│
├── spark_data_integration  <- Source code for use in this project. Exercice 1
│   ├── build.sbt    <- Script to deploy an Scala environnement
│   │
│   ├── src/main/scala/fr/cytech/integration  <- Scripts to download or generate data
│   │   └── Main.scala
│   │
│   ├── src/test/scala/fr/cytech/integration       <- Script to run test coverage
│   │   └── MainSpec.py
│
└── spark_data_integration_exercice2 <- Source code for use in this project. Exercice 2
    ├── build.sbt    <- Script to deploy an Scala environnement
    │
    ├── src/main/scala/fr/cytech/integration  <- Scripts to dump
    │   └── Main.scala
    │
    ├── src/test/scala/fr/cytech/integration       <- Script to run test coverage
    │   └── MainSpec.scala
    │
└── spark_data_integration_exercice3 <- Source code for use in this project. Exercice 3
    ├── insertion.sql
    └── creation.sql
└── dataviz_exercice4
│.
└── machine-learning_exercice5
```