Université Chouaib Doukkali
Ecole Nationale des Sciences Appliquées d'El Jadida
Département Télécommunications, Réseaux et Informatique

Filière : **2ITE**
Niveau : **3ème Année**

Mise en place d'une architecture Lambda pour le traitement et la visualisation des données en utilisant Apache kafka, Spark Streaming, Apache Airflow, Hive, Tableau

Réalisé Par :
- **ZIZ Ilyas**
- **EZZAHI Hamza**

Encadré par :
Prof. KALLOUBI

2023-2024

جامعة شعيب الدكالي
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵛⵄⵢⴻⴱ ⴰⴷⴷⴽⴰⵍⵉ
Université Chouaïb Doukkali

ENSA
المدرسة الوطنية للعلوم التطبيقية بالجديدة
Ecole Nationale des Sciences Appliquées

Table des matières

I. Objectif du projet :

Notre projet vise à mettre en place une architecture Lambda pour le traitement et la visualisation des données en temps réel, en exploitant une source de données en streaming provenant de l'API Finnhub. Cette architecture, basée sur des technologies telles qu'Apache Kafka, Spark ML, Spark Streaming, Apache Airflow, Hive, et Tableau, permettra de gérer efficacement les flux de données financières en temps réel et d'effectuer des analyses approfondies.

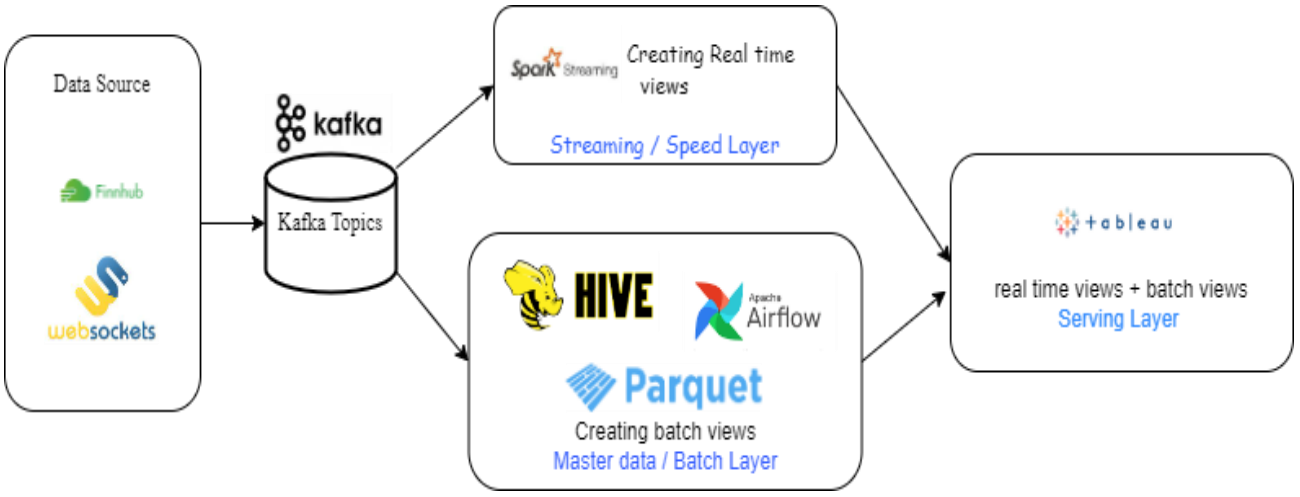


Figure 1: Architecture globale

II. Prérequis pour le projet - Installation et Configuration

1. Environnement de Développement

a. Système d'Exploitation : Vérifiez que vous disposez d'un système d'exploitation compatible avec les outils nécessaires (Exemple : Ici, on utilise Windows).

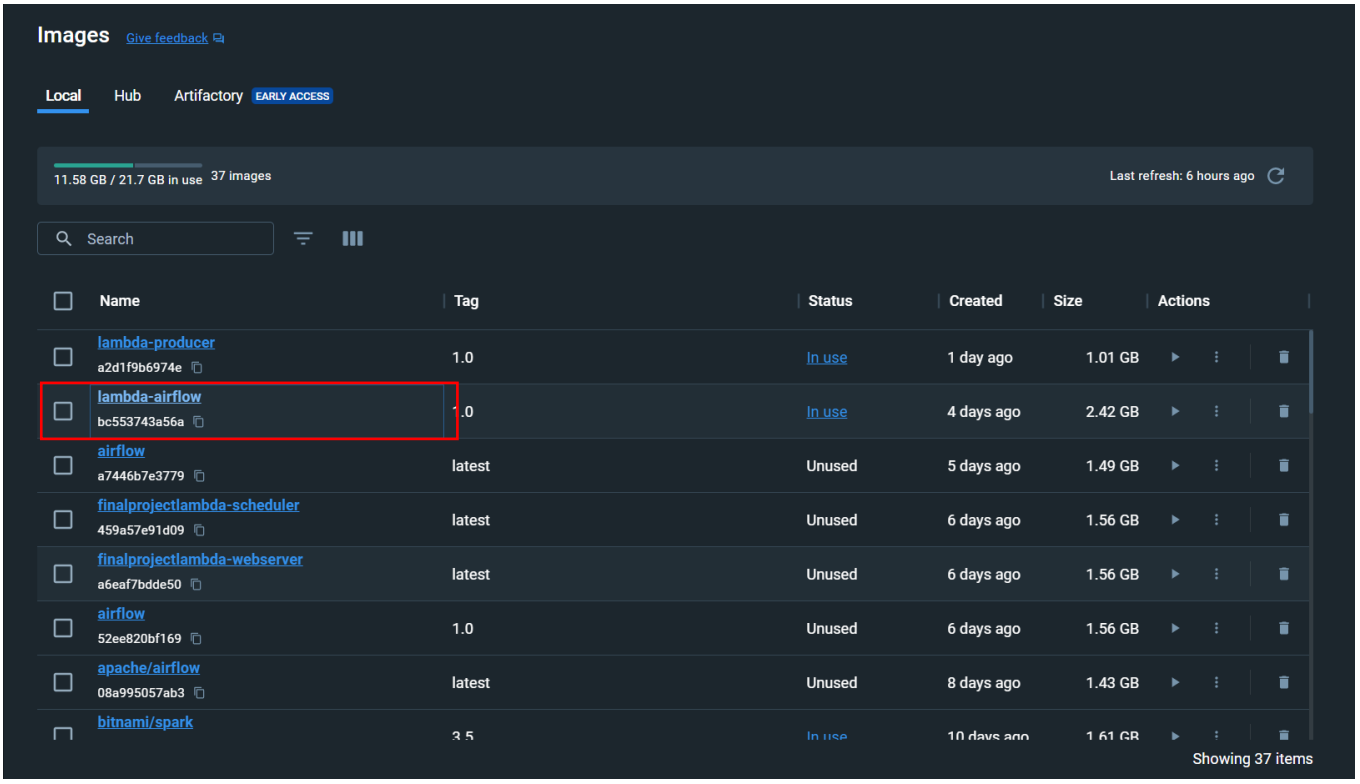
b. Docker:

- Téléchargez et installez Docker en suivant les instructions spécifiques à votre système d'exploitation : lien vers Docker
- Vérifiez l'installation avec la commande : docker –version

2. Configuration de l'Environnement Dockerisé

- Création des conteneurs :
  - Élaborez un fichier docker-compose.yml décrivant les services nécessaires pour cette application. Se référer au fichier docker-compose.yml dans le dossier du projet
- Construire l'mage d'airflow avec les dépendances nécessaires :

- Dans le répertoire de votre docker-compose ajouter le fichier « requierements.txt » qui se trouve dans le repo du projet.
- Puis exécuter la commande suivante : **docker build -t lambda-airflow :1.0 .**
- Vous pouvez la visualiser dans la section images :



III. Collecte des données :

1. Ingestion des données en utilisant Kafka :

- Dans le projet vous aller trouver un dossier producer dans lequel vous allez trouver un docker file et un code python
- Le script Python **producer.py** fourni est un consommateur WebSocket qui s'abonne à divers instruments financiers via l'API Finnhub et envoie les données reçues à deux topic Kafka (speed_topic et batch_topic). De plus, il y a un thread de producteur Kafka (kafka_producer_worker) qui récupère continuellement les messages d'une file d'attente (message_queue) et les envoie aux sujets Kafka spécifiés.
- Pour cette partie de code il faut récupérer le token du site : https://finnhub.io/dashboard

```
ws = websocket.WebSocketApp(
    "wss://ws.finnhub.io?token=clnmfh9r01qqp7jp4asgclnmfh9r01qqp7jp4at0",
    on_message=on_message,
    on_error=on_error,
    on_close=on_close
)
```

- Accéder a ce dossier et exécuter la commande suivante : **docker build -t lambda-producer :1.0 .**
- Vous pouvez visualiser l'image sous le nom de **lambda-producer** dans docker desktop .

2. Execution des conteneurs :

Pour démarrer les conteneurs définis dans votre fichier **docker-compose.yml** et les exécuter en arrière-plan, utilisez la commande suivante : **docker-compose up -d**

Vous pouvez visualiser la liste des conteneurs dans l'interface de docker desktop :

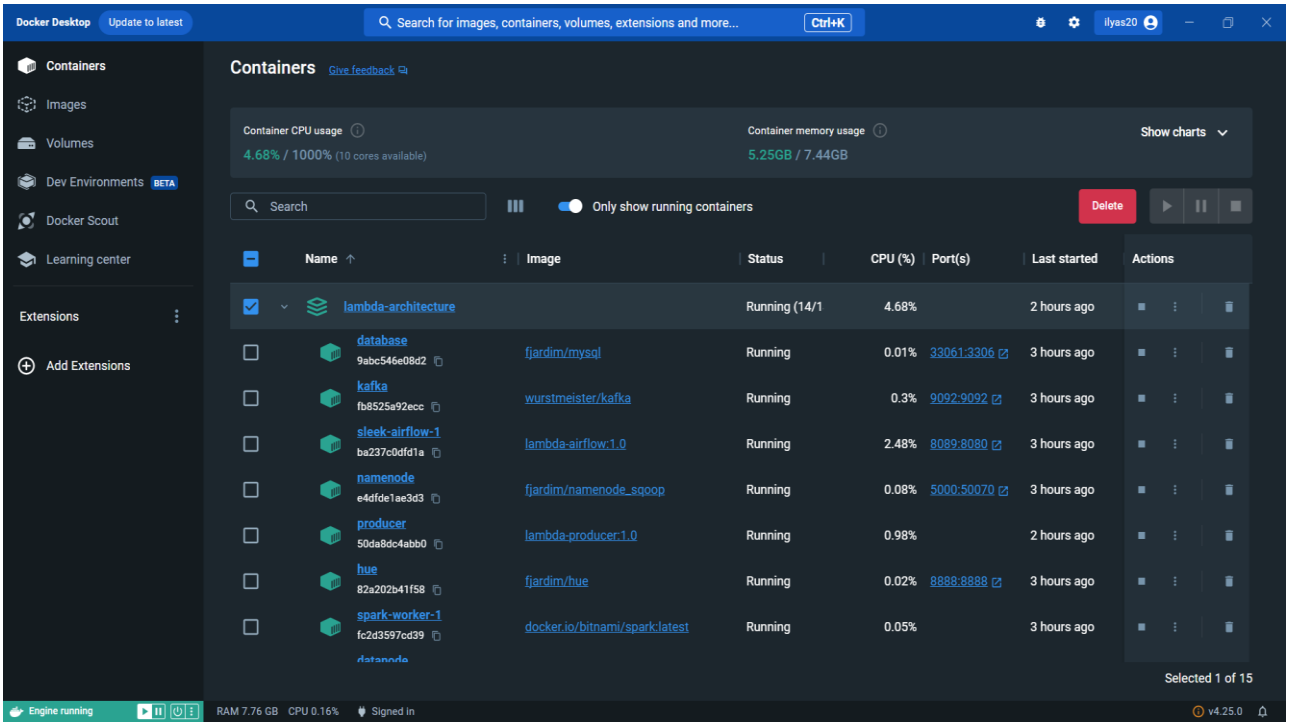


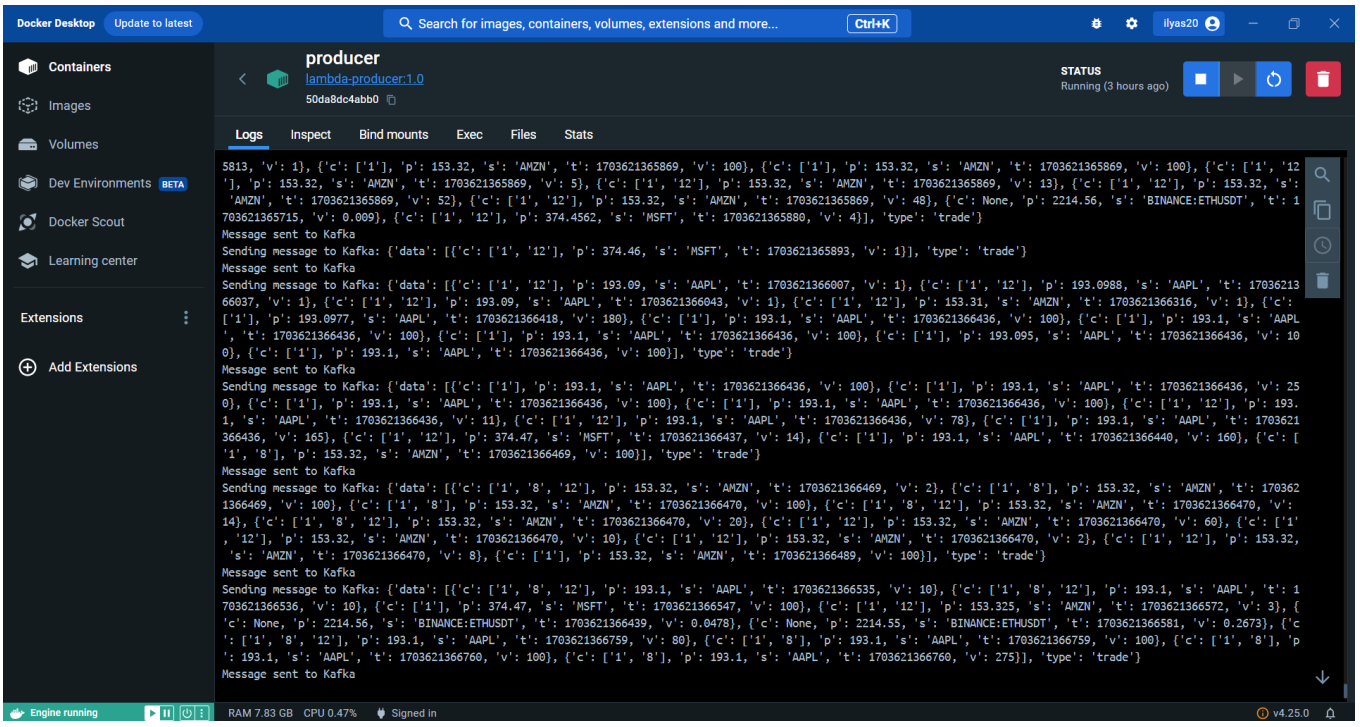
Figure 2: Liste des conteneurs

Pour accéder aux services dans un navigateur web, utilisez les URL suivantes :

- Spark : http://localhost:8080

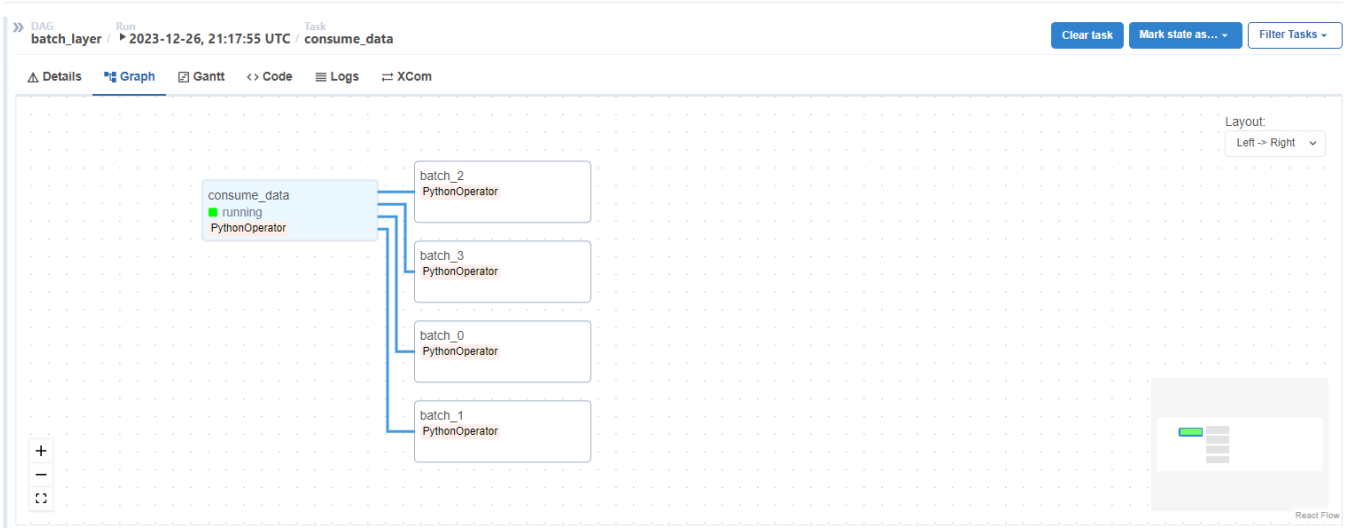
- Airflow : http://localhost:8089
- Namenode : http://localhost: 9870
- Hue : http://localhost:8888

Pour s'assurer que vous consommez l'api finnhub avec succès vous pouvez visualiser les données dans le log du conteneur producer :



IV. Batch Layer :

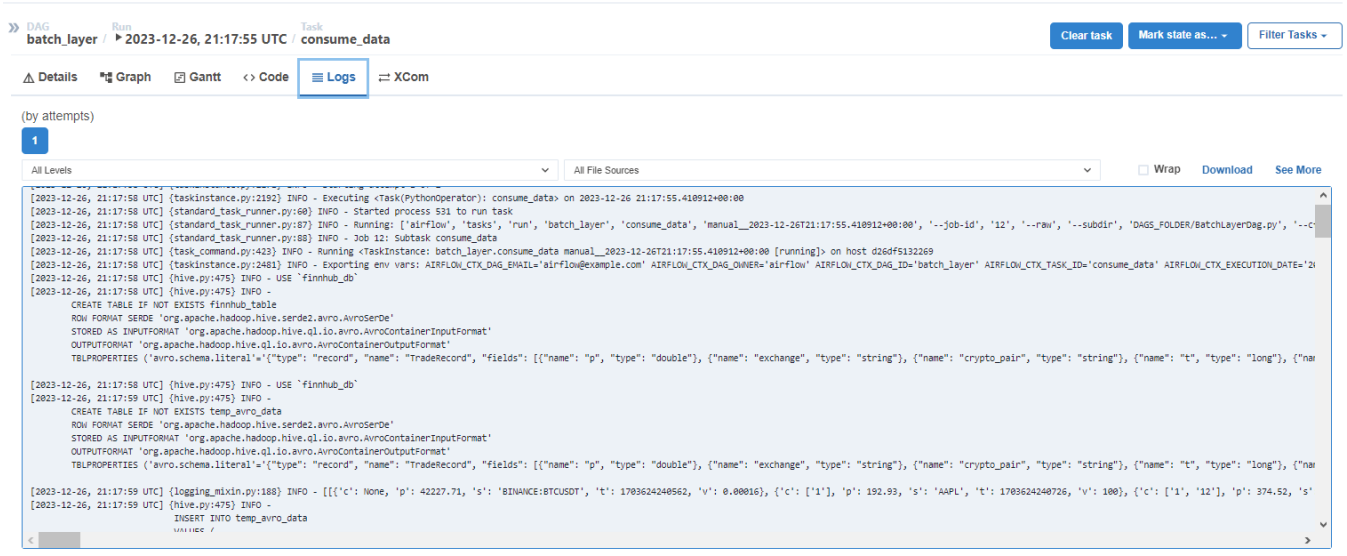
batch layer dans une architecture lambda est utilisé pour effectuer des traitements sur une quantité de données massives. Pour automatiser cette couche, nous avons travaillé avec Airflow, qui exécute le pipeline une fois toutes les 24 heures.



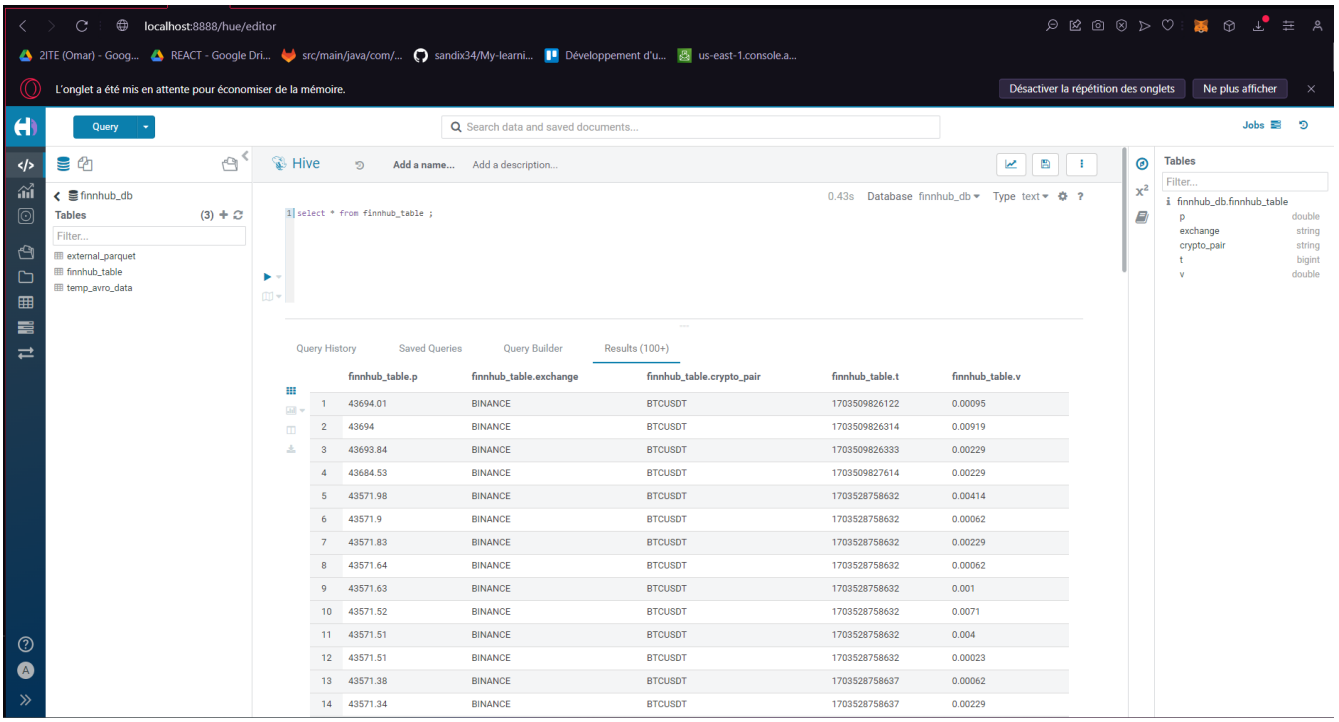
a. Système de stockage de données massives

Pour le système de stockage de données massives dans notre architecture lambda, nous utilisons Hive comme système de gestion de base de données distribué pour le stockage et la récupération des données. De plus, le format Avro est employé pour représenter les données de manière efficace et compacte, facilitant ainsi le traitement au sein du batch layer.

On peut accéder à l'interface d'Airflow pour voir les logs de cette tâche, vérifier qu'aucune erreur n'a été rencontrée et s'assurer que les données ont bien été enregistrées.



Pour visualiser les données, vous pouvez accéder à l'interface web Hue et exécuter des requêtes pour explorer et analyser les informations disponibles.



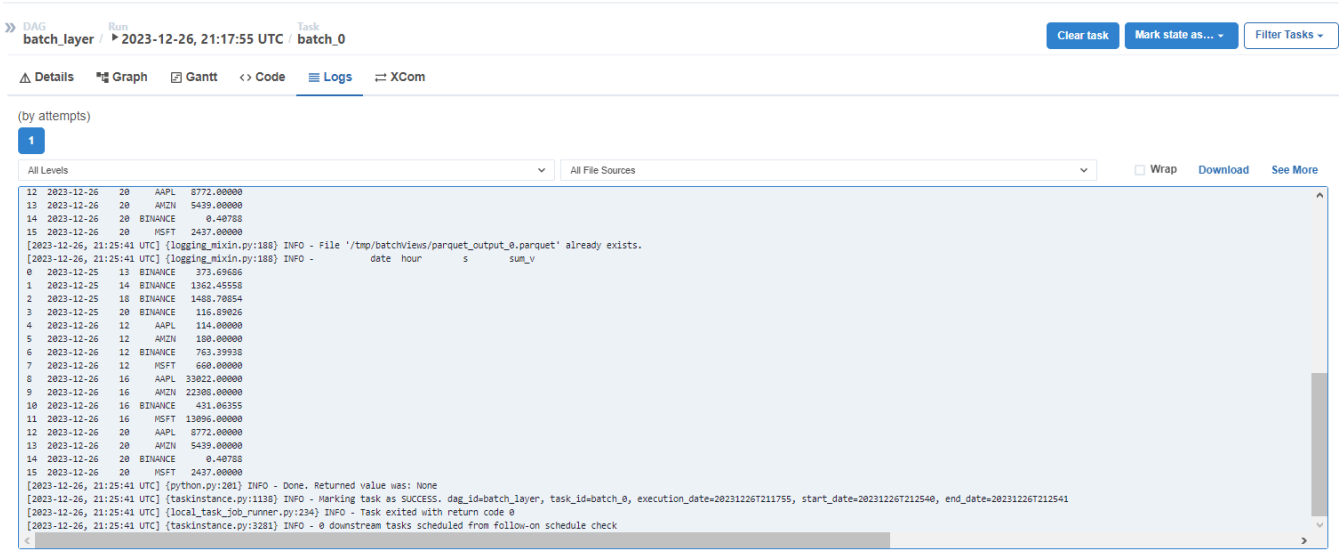
	finnhub_table.p	finnhub_table.exchange	finnhub_table.crypto_pair	finnhub_table.t	finnhub_table.v
1	43694.01	BINANCE	BTCUSDT	1703509826122	0.00095
2	43694	BINANCE	BTCUSDT	1703509826314	0.00919
3	43693.84	BINANCE	BTCUSDT	1703509826333	0.00229
4	43684.53	BINANCE	BTCUSDT	1703509827614	0.00229
5	43571.98	BINANCE	BTCUSDT	1703528758632	0.00414
6	43571.9	BINANCE	BTCUSDT	1703528758632	0.00062
7	43571.83	BINANCE	BTCUSDT	1703528758632	0.00229
8	43571.64	BINANCE	BTCUSDT	1703528758632	0.00062
9	43571.63	BINANCE	BTCUSDT	1703528758632	0.001
10	43571.52	BINANCE	BTCUSDT	1703528758632	0.0071
11	43571.51	BINANCE	BTCUSDT	1703528758632	0.004
12	43571.51	BINANCE	BTCUSDT	1703528758632	0.00023
13	43571.38	BINANCE	BTCUSDT	1703528758637	0.00062
14	43571.34	BINANCE	BTCUSDT	1703528758637	0.00229

b. Jobs de traitement (Batch Jobs)

Les jobs de traitement, également connus sous le nom de Batch Jobs, représentent une composante essentielle de l'architecture lambda. Ces jobs sont responsables de l'analyse en mode batch des données stockées dans le système de stockage massif

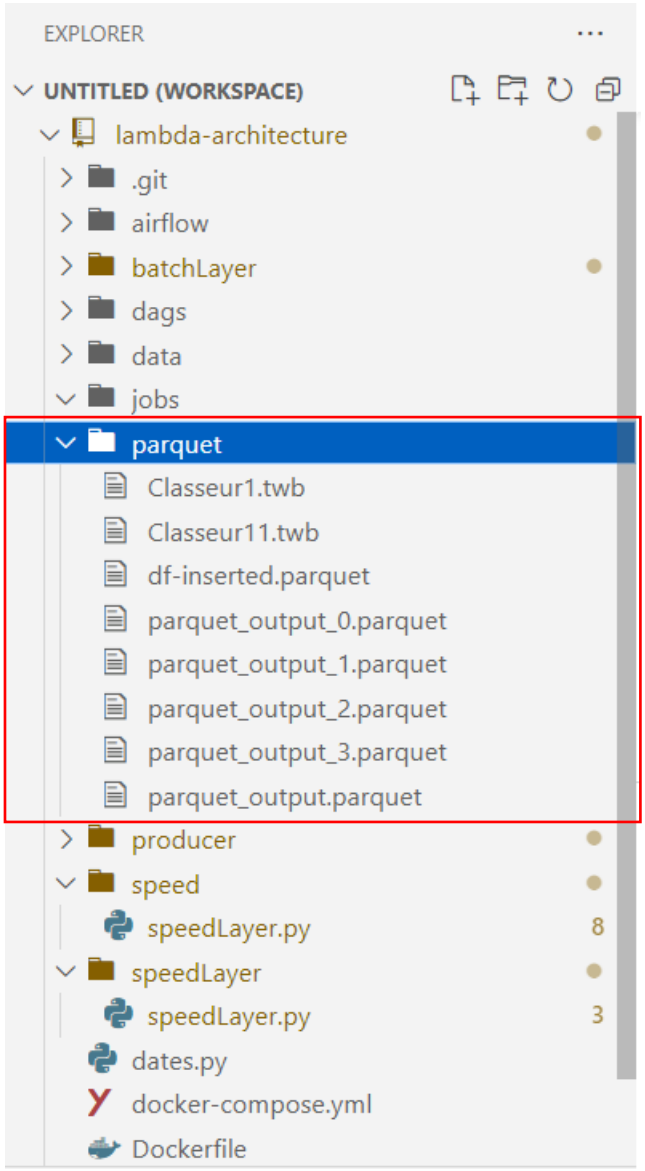
Les tâches accomplies par les Batch Jobs peuvent inclure le filtrage, la transformation, l'agrégation et d'autres opérations nécessaires pour traiter les données en lot de manière efficace.

Pour notre cas, nous effectuons des batch processings afin de calculer la somme des quantités échangées des actions pour les grandes entreprises telles qu'Apple, Amazon, etc., chaque jour. De plus, nous utilisons cette approche pour déterminer le prix minimum et maximum des actions pour chaque entreprise chaque jour. Cette analyse nous permet d'obtenir des informations agrégées sur les volumes d'échange et les variations de prix, fournissant ainsi des indicateurs clés pour la prise de décision et la compréhension des tendances du marché.



Une fois que le batch processing est effectué avec succès, les données résultantes sont insérées dans le format Parquet. Parquet est un format de stockage de colonnes efficace et compressé, particulièrement bien adapté aux analyses ultérieures.

Les fichiers Parquet générés sont localisés dans le répertoire /parquet à la racine du projet.



V. Speed Layer :

Speed Layer, ou couche de vitesse, représente une composante essentielle de l'architecture Lambda, conçue pour le traitement en temps réel des flux de données. Cette couche s'adresse aux données qui nécessitent un traitement immédiat, fournissant ainsi des résultats instantanés et actualisés. Dans le contexte de l'architecture Lambda appliquée à la gestion des données, Speed Layer complète la Batch Layer en permettant une réactivité accrue face aux événements en temps réel. Pour cela nous allons utiliser Spark

- Accéder au dossier speed et ouvrir le script python « speedLayer.py » ce code utilise Apache Spark pour consommer des données en streaming à partir d'un sujet Kafka (speed_topic). Les

données sont ensuite transformées et traitées, puis stockées dans une base de données PostgreSQL.

- S'assurer que les données sont bien crées dans postgress :
  - Ouvrer une invite de commande puis tapez la commande suivante : **docker ps**

```
C:\Users\zizil>docker ps
CONTAINER ID   IMAGE                     COMMAND                  CREATED         STATUS                   PORTS                                                                                  NAMES
8be1d3d45f81   postgres:alpine           "docker-entrypoint.s…"   7 minutes ago   Up 7 minutes (healthy)   0.0.0.0:5432->5432/tcp                                                                 lambda-architecture-postgres
-1
32fa6716323b   fjardim/hive              "entrypoint.sh /bin/…"   53 minutes ago  Up 52 minutes            10002/tcp, 0.0.0.0:10001->10000/tcp                                                    hive-server
aa4cef110677   fjardim/hive              "entrypoint.sh /opt/…"   53 minutes ago  Up 52 minutes            10000/tcp, 0.0.0.0:9083->9083/tcp, 10002/tcp                                           hive_metastore
40a3003513ac   fjardim/hive-metastore    "/docker-entrypoint.…"   53 minutes ago  Up 52 minutes            5432/tcp                                                                               hive-metastore-postgresql
fc2d3597cd39   bitnami/spark:latest      "/opt/bitnami/script…"   53 minutes ago  Up 52 minutes                                                                                                   lambda-architecture-spark-wo
rker-1
b6d68d54f8a7   fjardim/datanode          "/entrypoint.sh /run…"   53 minutes ago  Up 53 minutes (healthy)  0.0.0.0:5075->50075/tcp                                                                datanode
e657eebde781   bitnami/spark:latest      "/opt/bitnami/script…"   53 minutes ago  Up 52 minutes            0.0.0.0:7077->7077/tcp, 0.0.0.0:8080->8080/tcp, 0.0.0.0:10000->10000/tcp               lambda-architecture-spark-1
50da8dc4abb0   lambda-producer:1.0       "python /producer/Pr…"   53 minutes ago  Up 7 minutes                                                                                                    producer
82a202b41f58   fjardim/hue               "./startup.sh"           53 minutes ago  Up 52 minutes            0.0.0.0:8888->8888/tcp                                                                 hue
2d95c6c07252   wurstmeister/zookeeper    "/bin/sh -c '/usr/sb…"   53 minutes ago  Up 53 minutes            22/tcp, 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp                                     ktech_zookeeper
e4dfde1ae3d3   fjardim/namenode_sqoop    "/entrypoint.sh /run…"   53 minutes ago  Up 53 minutes (healthy)  0.0.0.0:5000->50070/tcp                                                                namenode
ba237c0dfd1a   lambda-airflow:1.0        "/usr/bin/dumb-init …"   53 minutes ago  Up 53 minutes            0.0.0.0:8089->8080/tcp                                                                 lambda-architecture-sleek-ai
rflow-1
9abc546e08d2   fjardim/mysql             "docker-entrypoint.s…"   53 minutes ago  Up 53 minutes            33060/tcp, 0.0.0.0:33061->3306/tcp                                                     database
fb8525a92ecc   wurstmeister/kafka        "start-kafka.sh"         53 minutes ago  Up 53 minutes            0.0.0.0:9092->9092/tcp                                                                 kafka
```

  - Récupérer l'id du conteneur postgress-1 et exécuter la commande suivante : **docker exec -it 8be1d3d45f81 psql -U airflow**

```
C:\Users\zizil>docker exec -it 8be1d3d45f81  psql -U airflow
psql (16.1)
Type "help" for help.

airflow=# use airflow
```

  - Acceder a la base données crée initialement avec la commande suivante : **use airflow**
  - Puis afficher le contenu de la table speed table : (select * from speed_table)

```
airflow=# select * from speed_table
;
 id | type  |  p  |        s        |  t  |  v  | percentage_change | absolute_price_change
----+-------+-----+-----------------+-----+-----+-------------------+-----------------------
  1 | trade | 123 | BINANCE:BTCUSDT | 456 | 789 |              0.05 |                  10.5
  2 | trade | 123 | BINANCE:ETHUSDT | 456 | 789 |              0.05 |                  10.5
  3 | trade | 123 | AAPL            | 456 | 789 |              0.05 |                  10.5
  4 | trade | 123 | AMZN            | 456 | 789 |              0.05 |                  10.5
  5 | trade | 123 | MSFT            | 456 | 789 |              0.05 |                  10.5
  6 | trade | 123 | GBP             | 456 | 789 |              0.05 |                  10.5
```

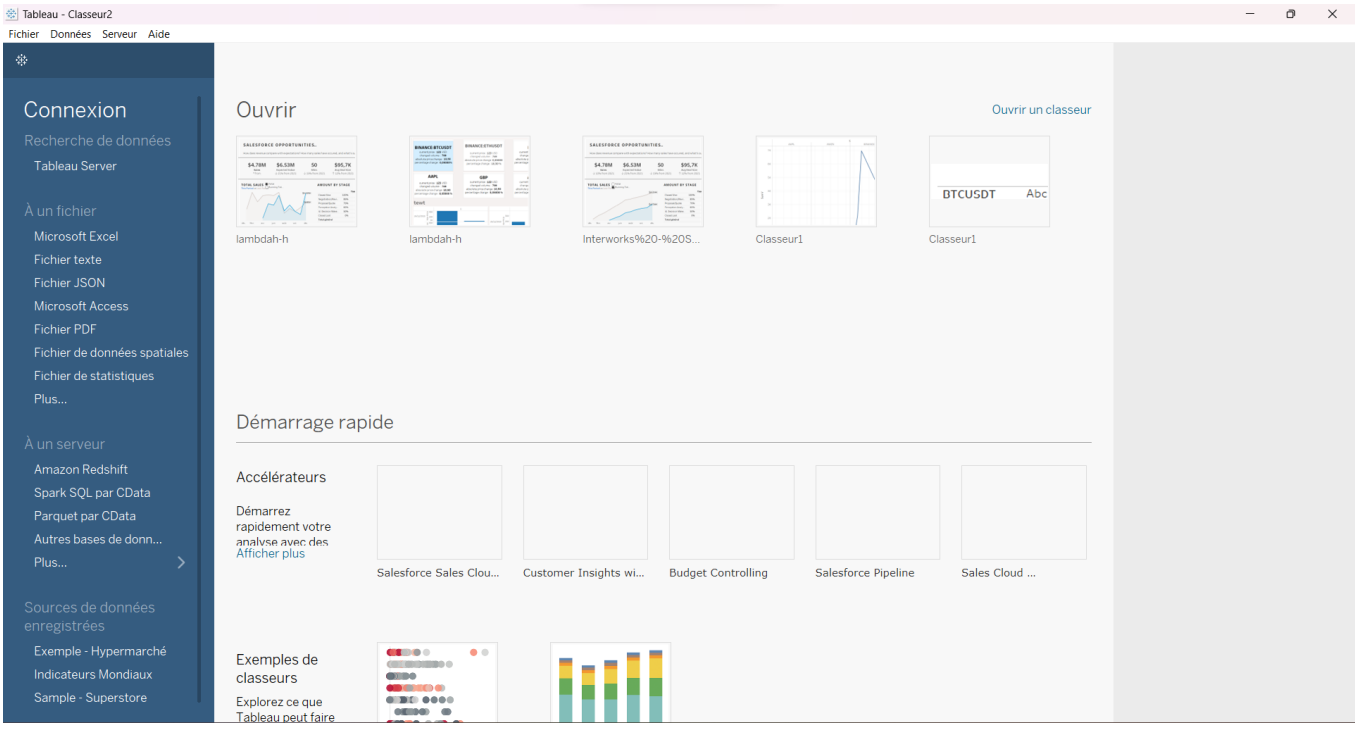
VI. Serving Layer :

Dans cette étape cruciale, les différentes vues générées par la Batch Layer et la Speed Layer sont combinées pour offrir une vision complète et actualisée de l'ensemble des données.

Les données provenant de la Batch Layer, stockées dans des formats optimisés tels que Parquet, sont agrégées avec les résultats de la Speed Layer. Dans le contexte de cette architecture, où les traitements en batch se concentrent sur des volumes importants de données et la Speed Layer réagit aux flux en temps réel, la couche de service assure l'harmonisation de ces perspectives.

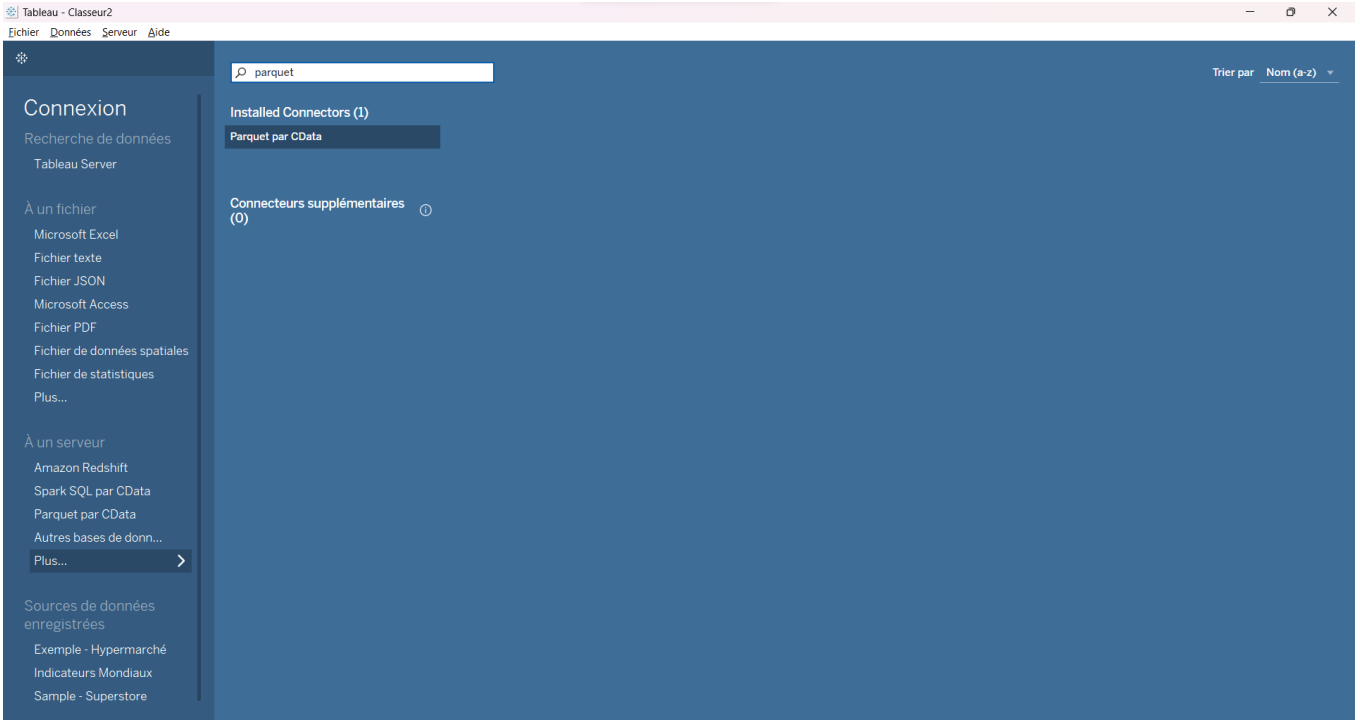
Tableau, en tant qu'outil de visualisation puissant, est utilisé pour créer des tableaux de bord interactifs et informatifs à partir des données agrégées. Les différentes visualisations, telles que les graphiques, les tableaux croisés dynamiques et les indicateurs clés de performance (KPI), permettent aux utilisateurs de tirer des insights significatifs. Les filtres et les options d'interaction offerts par Tableau facilitent l'exploration approfondie des données, aidant ainsi à identifier des tendances, des modèles et des anomalies.

- Installation de tableau :
  Installer tableau depuis le lien suivant :https://www.tableau.com/products/desktop/download
  vous pouvez utiliser la version d'évaluation ou s'inscrire au pack étudiant .

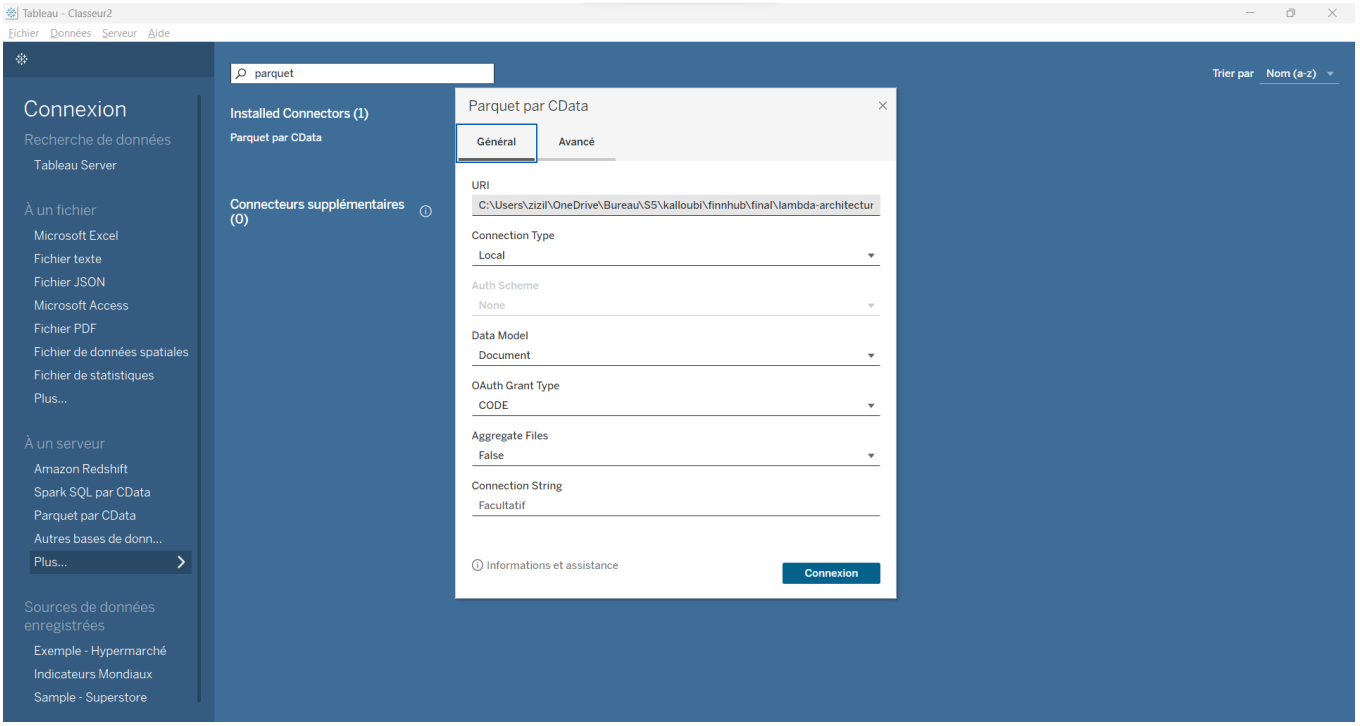


- Connexion de tableau avec nos source de données :

- Installer le driver parquet par cdata depuis le lien suivant : https://www.cdata.com/drivers/parquet/
- Vous pouvez sélectionnez autre source pour se connecter a parquet : puis sélectionner Parquet par cdata :



- Puis spécifier le chemin vers le fichier parquet généré :



- ET maintenant vous pouvez créer vos propre visualisations :