Projet CryptoBot

Partie 2 : Stockage de la donnée

Projet Informatique - Data Engineering

Participants au projet :

Thomas Saliou

Florent Rigal

Philippe Kirstetter-Fender

Nancy Frémont

Formation : DataScientest

Année : 2025

Étape 2 : Stockage de la donnée

Sommaire :

Introduction :

Après la création de scripts nous permettant de récupérer les données historiques et streaming des cryptomonnaies définis. La suite logique est donc de stocker ces données dans des espaces de stockage dédiés afin de pouvoir les traiter dans notre future et prochaine étape de machine learning. Il s'agit d'une étape importante. En effet, tout notre processus dépend de la qualité et de la quantité des données que l'on va utiliser.

L'objectif de cette étape va être de trouver la solution la plus adaptée à la problématique de cette étape et obtenir un ETL fonctionnel permettant de récupérer, traiter et stocker nos données.

Dans notre cas nous avons 2 types d'entrées de données récoltés :

- les données historiques, qui sont figées dans le temps. Ont peut obtenir grâce aux script toutes les valeurs d'une crypto au format OHLCV entre 2 dates précisés par l'utilisateur sur un intervalle de temps donné. Ces données auront pour objectif d'entraîner le modèle de machine learning et serviront à la visualisation.

- les données streaming, qui sont soit en flux continu (streaming) soit en bloc (API). Ces données vont nous servir à valider le modèle de machine learning et à appliquer nos requêtes analytiques d'aide à la décision sur l'achat ou la vente de crypto en temps réel.

Il faut une ou deux structures permettant de recueillir ces 2 types de données.

1 Comparatif des solutions et diagramme UML

1.1 Comparatif des solutions

Avantages et Inconvénients des SGBD proposées :

SYSTEME	TYPE	AVANTAGES	INCONVENIENTS
SQL (MySQL, PostgreSQL)	Relationnelle	- simple, structuré - bon pour les jointures - connu et bien documenté	- moins efficace pour les données massives en temps réel - moins flexible pour schémas évolutifs
ElasticSearch	NoSQL (search et analyse en temps réel)	- indexation rapide - agrégation sur séries temporelles - très bon outil de visualisation (Kibana)	- moins adapté pour relations complexes - pas optimal pour le stockage à long terme
Snowflake	Data Warehouse Cloud	- bon pour les gros volumes analytiques - langage SQL - optimisé pour la BI et la Data Science	- coût élevé - période d'essai trop courte - moins bon pour ingestion en temps réel
MongoDB	NoSQL (document)	- flexible - bon pour le streaming - stockage Json - scalabilité	- requetage plus complexe - moins performant pour des requêtes analytiques
BigQuery	Data Warehouse Cloud	- très scalable - bon pour les requêtes sur de gros volumes - facile à intégrer avec GCP AI tools	- latence avec données en batch - non créé pour l'ingestion en temps réel

En comparant les critères des données avec les avantages/inconvénients des systèmes de gestion de bases de données, il ressort en ressort plusieurs éléments.

Tout d'abord, nous pensons qu'il est préférable de séparer les données en deux SGBD différents. En effet, nos données bien que semblables dans la forme, ne s'importent pas de la même façon, les données historiques sont figées dans le temps selon un intervalle de dates tandis que les données en streaming vont importer la donnée en continue en temps réel jusqu'à l'arrêt du script. De plus, ces données vont traiter deux aspects différents, l'entraînement du ML avec avec les données historiques et la validation de celui-ci avec les données en streaming.

Concernant les SGBD à proprement parler, il semble logique de s'orienter vers un schéma relationnel pour les données historiques et un schéma No-SQL pour les données en temps réel. Les solutions proposées pour les schémas relationnels sont postgreSQL ou Snowflake. Le principal inconvénient pour Snowflake est son coût élevé et sa courte période d'essai qui ne couvre pas la durée totale de la formation, de ce fait postgreSQL semble convenir parfaitement. D'autant plus que nous n'avons pas beaucoup de relations à créer et le format des données importées est et sera toujours le même. Pour ce qui est du No-SQL, Elasticsearch aurait pu être une bonne alternative mais il est surtout conçu pour le traitement de données textuel et d'indexation, MogoDB semble être une bonne alternative pour traiter les données en streaming.

En fonctionnant de la sorte, nous nous rapprochons d'une architecture lambda. L'architecture Lambda permet de traiter simultanément les données historiques (batch) et les données en temps réel (streaming). Dans notre cas, les données historiques stockées dans PostgreSQL servent à l'entraînement du modèle de Machine Learning, tandis que les données en continu stockées dans MongoDB permettent sa validation en temps réel.

Cette approche combine la précision du traitement batch et la réactivité du streaming, offrant une solution complète et évolutive.

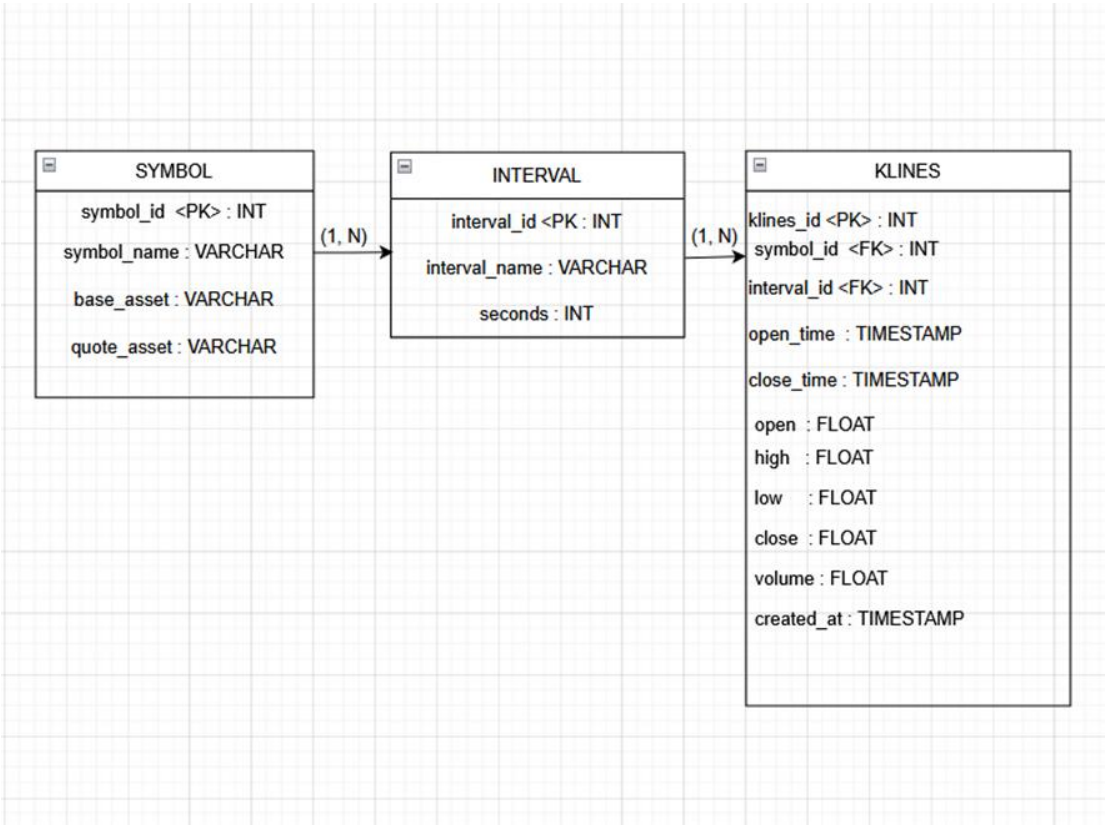
1.2 Diagramme UML

Avec les données que nous avons nous pouvons faire le constat suivant :

- Un symbole a plusieurs Klines (Chandeliers). Chaque Kline appartient à un seul symbole.
- Un symbole peut avoir plusieurs intervalles. Un intervalle peut être lié à plusieurs symboles.
- Chaque kline (une Kline est définie par un triplet : un symbole, un intervalle, un timestamp d'ouverture) et n'appartient qu'à un seul intervalle. Un intervalle peut être lié à plusieurs klines.

Pour passer en diagramme UML, nous nous sommes basées sur les cardinalités maximales uniquement, donc cela correspond à une relation (N, 1) ou (1,N). Car selon la 1ère règle de Merise : Si la relation est (1,N) ou (N,1), la clé primaire côté N descend dans l'entité côté 1 et devient clé étrangère dans la table Klines.

En tenant compte des règles établis ci-dessus on obtient le schéma ci-dessus contenant les éléments suivants :



Légende :

Symbol_id, interval_id et klines_id sont des primary keys. Il s'agit d'index des tables.

Symbol_name : BTCUSDT, ETHUSDT, etc.

Base_asset : BTC

Quote_asset : USDT

Interval_name : "1m", "5m", "15m", "1h", "4h", "1d"

Seconds : durée en seconde

Open_time : heure ouverture du klines

Close_time : heure de fermeture du klines

Timestamps : date de création

2 Import des données

2.1 Choix des intervalles

Type de trading	Intervalle Binance recommandé	Pourquoi / Utilisation
Scalping	1m, 3m, 5m	Très court terme, prise de positions sur quelques minutes, nécessite réactivité.
Day trading	15m, 30m, 1h	Permet de suivre les tendances intrajournalières sans le bruit des 1m/5m.
Swing trading	4h, 6h, 12h	Capture les mouvements sur plusieurs jours, moins sensible aux fluctuations mineures.
Position trading / Long terme	1D, 3D, 1W	Analyse macro, suit les tendances sur plusieurs semaines ou mois.

Légendes : m=minutes, h=heures, D=day, W=Week

D'après le tableau ci-dessus, plusieurs types de trading existent : scalping, day trading, swing trading et analyse long terme.
Pour le projet crypto, nous voulons entraîner un modèle de Machine Learning, puis calculer des indicateurs techniques (MSE, MAO, RMSE). Ainsi, nous n'avons pas besoin de scalping. Pour le machine Learning, il nous faut beaucoup de données (précision fine) donc un intervalle entre 1h et 4h. Pour les indicateurs, nous avons besoin de moins de données donc entre 15 minutes et 1h. Pour la visualisation, on peut passer à un intervalle 1d ou 1w.

On peut donc garder la liste d'intervalles suivants : [15m, 1h, 4h, 1d, 1w].

2.2 Import des données sur PosgreSQL

Pour l'import des données sur PostgreSQL, nous avons collecté les données sur les intervalles choisis précédemment et pour quatre symboles (BNBUSDT, BTCUSDT, ETHUSDT et SOLUSDT).

Nous avons réalisé un pipeline ETL (Extract Transform Load), avec enregistrement dans des fichiers csv (un fichier par symbol_interval) après traitement des données.

Pour la visualisation des données importées dans les tables (selon diagramme UML défini), nous avons utilisé PgAdmin4.

Voici un aperçu des données d'un fichier csv avant import :

open_time	open	high	low	close	volume	close_time
2023-01-30	317.1	320.8	302.0	307.2	413716.212	2023-01-30 23:59:59.999
2023-01-31	307.2	314.4	305.3	312.0	278061.958	2023-01-31 23:59:59.999
2023-02-01	312.1	319.4	305.6	317.0	390978.523	2023-02-01 23:59:59.999
2023-02-02	317.1	334.4	316.3	323.5	587655.881	2023-02-02 23:59:59.999
2023-02-03	323.5	335.5	318.3	332.2	579577.162	2023-02-03 23:59:59.999
2023-02-04	332.3	334.1	327.0	330.4	212109.626	2023-02-04 23:59:59.999
2023-02-05	330.4	337.8	324.0	327.8	396514.711	2023-02-05 23:59:59.999
2023-02-06	327.8	331.5	320.7	324.3	290965.0	2023-02-06 23:59:59.999
2023-02-07	324.4	333.7	323.2	333.0	265904.833	2023-02-07 23:59:59.999

2.3 Import des données sur MongoDB

Pour mongodb nous avons choisi de recueillir les données en streaming. Nous avons choisi d’organiser les données avec une collection par symbole.

Voici un exemple de données stockées :

{

  _id: ObjectId('68f0f0fbb58dfa2bbac90d8d'),

  open_time: ISODate('2025-09-25T18:00:00.000Z'),

  open: 108833.63,

  high: 109801.78,

  low: 108631.51,

  close: 109760.2,

  volume: 1427.42291,

  close_time: ISODate('2025-09-25T18:59:59.999Z'),

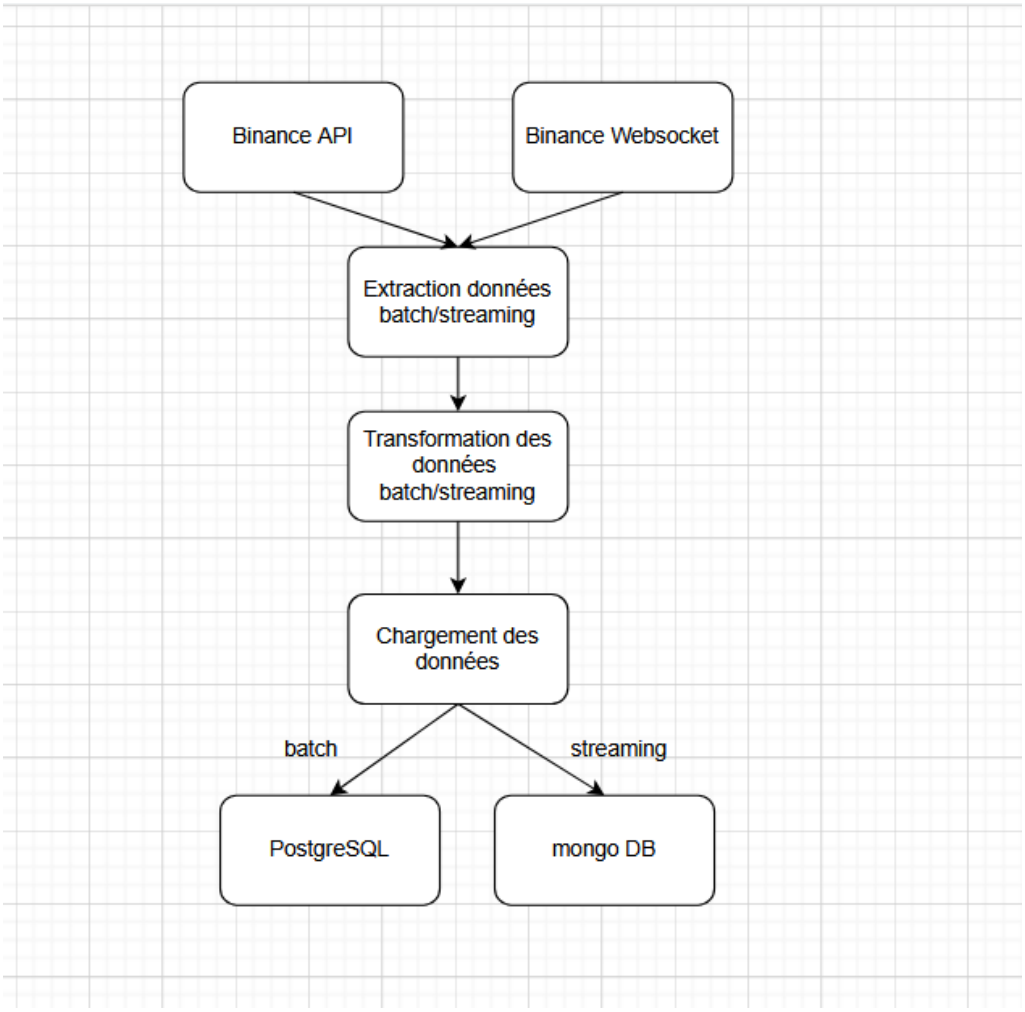
  symbol: 'BTCUSDT'

}

On ne stocke que les klines qui sont fermées.

3 Résultats

Voici de manière synthétique une représentation de notre pipeline ETL :



Pour terminer, quelques visualisations ci-dessous.

Donnée historique -> Visualisation et requêtage avec vue postgreSQL

	klines_id [PK] integer	symbol_id integer	interval_id integer	open_time timestamp without time zone	close_time timestamp without time zone	open double precision	high double precision	low double precision	close double precision	volume double precision	created_at timestamp without time zone
1	1	3	4	2023-01-30 00:00:00	2023-01-30 23:59:59.999	317.1	320.8	302	307.2	413716.212	2025-10-25 12:59:49.164639
2	2	3	4	2023-01-31 00:00:00	2023-01-31 23:59:59.999	307.2	314.4	305.3	312	278061.958	2025-10-25 12:59:49.164639
3	3	3	4	2023-02-01 00:00:00	2023-02-01 23:59:59.999	312.1	319.4	305.6	317	390978.523	2025-10-25 12:59:49.164639
4	4	3	4	2023-02-02 00:00:00	2023-02-02 23:59:59.999	317.1	334.4	316.3	323.5	587655.881	2025-10-25 12:59:49.164639
5	5	3	4	2023-02-03 00:00:00	2023-02-03 23:59:59.999	323.5	335.5	318.3	332.2	579577.162	2025-10-25 12:59:49.164639
6	6	3	4	2023-02-04 00:00:00	2023-02-04 23:59:59.999	332.3	334.1	327	330.4	212109.626	2025-10-25 12:59:49.164639
7	7	3	4	2023-02-05 00:00:00	2023-02-05 23:59:59.999	330.4	337.8	324	327.8	396514.711	2025-10-25 12:59:49.164639
8	8	3	4	2023-02-06 00:00:00	2023-02-06 23:59:59.999	327.8	331.5	320.7	324.3	290965	2025-10-25 12:59:49.164639
9	9	3	4	2023-02-07 00:00:00	2023-02-07 23:59:59.999	324.4	333.7	323.2	333	265904.833	2025-10-25 12:59:49.164639
10	10	3	4	2023-02-08 00:00:00	2023-02-08 23:59:59.999	333.1	335.8	323.6	328.2	286865.867	2025-10-25 12:59:49.164639

Donnée en streaming -> Visualisation avec streamlit