



PROJET OPA – DE NOV 2022 **DATASCIENTEST**

Equipe projet : Benjamin AYELA / Bertrand WULFF / Loïc CHEVALIER

Mentor Datascientest : Fall Lewis YOMBA

Chef de cohorte Datascientest : Dimitri CONDORIS

SUJET :

CryptoBot avec Binance

Cursus concerné : Data Engineer

Difficulté : 8/10

Description détaillée :

De nos jours, le monde des crypto commence à prendre une place importante et grossi. Il s'agit tout simplement de marchés financiers assez volatiles et instables se basant sur la technologie de la Blockchain. Le but principal de ce projet est de créer un bot de trading basé sur un modèle de Machine Learning et qui investira sur des marchés crypto.

Vous trouverez ci-dessous les étapes du projet tel qu'elles sont décrites dans le sujet du projet.

Etape	Description	Objectif	Modules / Masterclass / Templates	Conditions de validation du projet
1	Récolte des données	<p>Récolter deux types de données en passant par l'API Binance en passant par une architecture de streaming.</p> <ol style="list-style-type: none"> Grâce à cette API, on peut aller récupérer des informations sur les cours des différents marchés (BTC-USDT, BTC-ETH, ...). Le but sera de créer une fonction de récupération de données générique afin de pouvoir avoir les données de n'importe quel marché. Il faudra aussi créer un script de pré-processing pour réorganiser les données sortant du streaming afin qu'elles soient propres. Récupérer les données historiques, pré-processé pour pouvoir entraîner notre futur modèle 		<p>Fichier explicatif du traitement (doc / pdf)</p> <p>Un fichier d'exemple de récupération. json</p>
2	Architecture de la donnée	<p>Il s'agit de choisir la solution de stockage la plus adaptée.</p> <ul style="list-style-type: none"> 2 tables SQL, une pour les données historiques et une autre pour les données streaming Une DB Mongo comportant 2 collections: une pour les données stream et une autre pour les données 	<p>142 - SQL</p> <p>(Architecture des données)</p> <p>Elasticsearch</p> <p>MongoDB</p>	<p>Une base de données relationnelle</p> <p>Un fichier de requête SQL pour montrer que c'est bien fonctionnel</p> <p>Même rendu mais exemples de requêtes Elastic/Mongo</p>

3	Consommation de la donnée	Utiliser un algo de Machine Learning appliqué à la finance qui permettra de retourner une décision d'achat ou non. (Quelle stratégie pour la vente ?) Aller plus loin: prédiction de gains Si pas le temps de faire ce travail, possible de faire un dashboard de suivi de la crypto	DE120 Dash	Notebook de ML
4	Mise en production	Faire une API pour tester le modèle de ML et pourquoi pas requêter les données historique Dockeriser tout le projet pour qu'il soit reproductible sur n'importe quel machine	Docker FastAPI	Fichier Yaml du docker-compose API FastAPI
5	Automatisation des flux	<u>Étape</u> Facultative Il faudra requêter l'API quotidiennement via Airflow. Nous ne pouvons pas utiliser Airflow pour les données en streaming, mais vous pouvez tester Nifi.	Airflow	Fichier python du DAG
6	Soutenance	Démonstration de leur appli et explication du raisonnement effectué lors de leur projet.	X	Soutenance Rapport

Table des matières

Sprint 1 : Recolte des données.....	6
1. Créez un compte sur la plateforme Binance pour obtenir une clé API.....	6
2. Créer notre environnement de travail.	6
3. Sécurisé les clefs API (car le Github est public)	7
4. Librairies / packages	7
5. Utilisez une bibliothèque Python, telle que "python-binance" pour interagir avec l'API Binance et récupérer les données de marché (hist).	8
6. Établissez une architecture de streaming pour recevoir les données de manière continue.....	9
7. Enregistrez les données dans un format approprié, tel que JSON.....	11


Sprint 1 : Recolte des données

Afin de recueillir les données des marchés cryptomonnaie, l'énoncé nous impose de passer par la plateforme d'échange BINANCE (<https://www.binance.com/fr>). Pour commencer, nous avons dû créer un compte afin d'obtenir des clés (une publique et une privée) pour utiliser l'API.

1. Créez un compte sur la plateforme Binance pour obtenir une clé API

L'API est aujourd'hui paramétrée uniquement en lecture pour éviter de mauvaises manipulations. En effet, pour obtenir la clé, il fallait faire un premier dépôt en € ou acheter une cryptomonnaie. Nous avons acheté pour 15€ de BTC pour commencer.

HMAC projet OPA Modifier les restrictions Supprimer



API Key

eccWMDlc1dNmfgVHz2As8XjibvMrU1Tvm0n8aA5J7oKeF0z3NkwNTWZtHcMHEZv9 Copy

Secret Key

Restrictions API :

☒ Permettre la lecture

☐ Activer le trading Spot et sur marge

☐ Activer les retraits

☐ Activer les Prêts sur marge, Remboursements et Transferts

☐ Permet les transferts universels

☐ Activer les options européennes

☐ Activer la liste blanche des symboles Modifier

Restrictions d'accès IP :

☒ Sans restriction (moins sûr) Cette clé API permet l'accès à partir de n'importe quelle adresse IP. Ceci n'est pas recommandé.
Afin de protéger la sécurité de vos fonds, si l'IP n'est pas restreinte et qu'aucune autorisation autre que Lecture n'est activée, cette clé API sera supprimée.

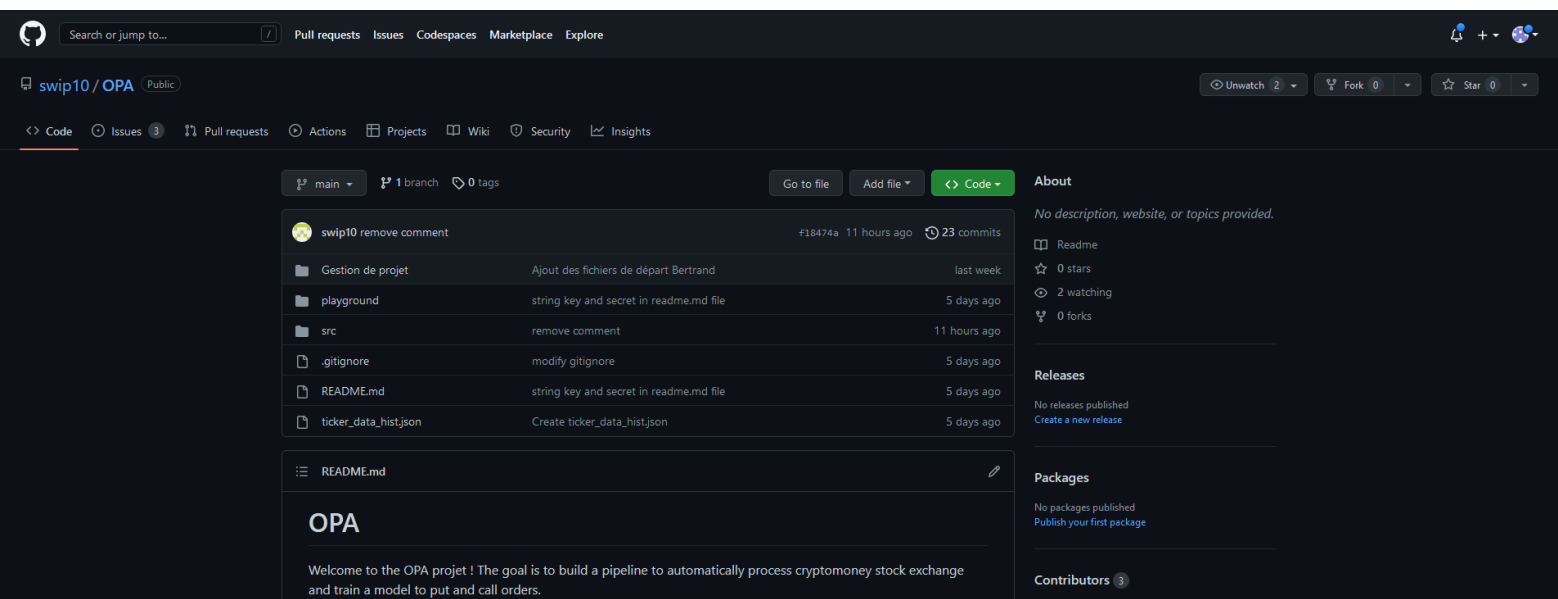
☐ Restreindre l'accès aux adresses IP de confiance uniquement (recommandé)

2. Créer notre environnement de travail.

Nous avons créé un Github pour pouvoir nous organiser dans notre travail en équipe.

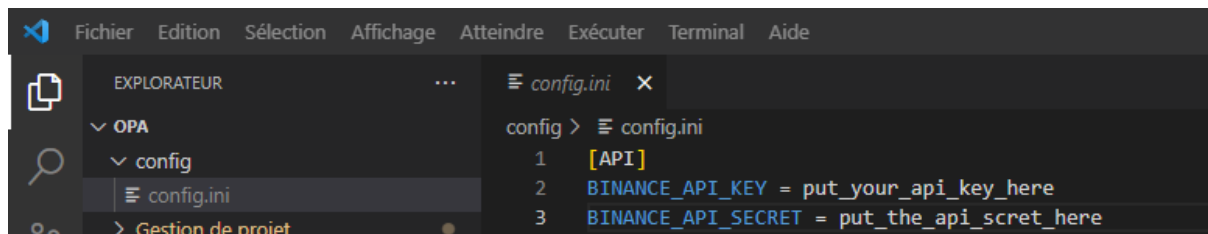
Il est organisé de la façon suivante :

- Dossier **Gestion de projet** pour tout ce qui est relatif aux rendus du projet (dont ce document)
- Dossier **playground** pour tester du code en autonomie
- Dossier **src** (source) pour le code validé ensemble.



3. Sécurisé les clefs API (car le Github est public)

Les clefs fournis par Binance ne doivent pas être visibles sur Github car il est public. Pour cela, un fichier `.gitignore` permet d'ignorer le fichier `config/config.ini` dans lequel nous stockons la clef localement sur nos machines.



4. Librairies / packages

Afin de simplifier notre organisation, nous avons créé un fichier **requirements.txt** dans lequel nous mettons les différents paquets Python dont l'installation est requise dans un environnement virtuel pour que l'application s'exécute correctement.

```
requirements.txt X
src > requirements.txt
1 python-binance
2 pandas
3 openpyxl
4 websocket
5 plotly
6 tqdm
```

5. Utilisez une bibliothèque Python, telle que "python-binance" pour interagir avec l'API Binance et récupérer les données de marché (hist).

Ci-dessous est décrit le code nous permettant d'extraire les données historique de 10 'ticker' (pour commencer). Fichier : ***data_api_hist.py***

Le code permet de se connecter à l'API avec les clefs, de récupérer tout les 'tickers', puis pour les 10 premiers de récupérer les données, de les convertir en Dataframe, puis de le stocker dans un dictionnaire et enfin d'exporter en format JSON.

```
from binance.client import Client
import json
from tqdm import tqdm
import pandas as pd
import config

# Initialise le client Binance
client = Client(config.BINANCE_API_KEY, config.BINANCE_API_SECRET)
client.ping()

# Récupère tout les 'tickers' de l'API
tickers = client.get_all_tickers()

# Initialise un dictionnaire pour stocker les données de chaque ticker
ticker_data = {}
loading_tickers = tqdm(tickers[0:10])
```



```

# Récupère les données de l'API sur chacun des tickers
for ticker in loading_tickers:
    loading_tickers.set_description(f"loading symbol {ticker['symbol']}")
    klines = client.get_historical_klines(ticker['symbol'],
Client.KLINE_INTERVAL_8HOUR, "1000 days ago")

    # Transforme les données en DataFrame
    data = pd.DataFrame(klines, columns=["timestamp", "open", "high", "low",
"close", "volume", "close_time", "quote_asset_volume", "number_of_trades",
"taker_buy_base_asset_volume", "taker_buy_quote_asset_volume", "ignore"])
    data.drop(columns="ignore", inplace=True)

    # Change le type de la colonne timestamp en datetime
    data["timestamp"] = pd.to_datetime(data["timestamp"],
unit="ms").dt.strftime('%Y-%m-%d %H:%M:%S')

    # Stocke la DataFrame correspondante dans le dictionnaire
    ticker_data[ticker['symbol']] = data.to_dict(orient='records')

# Exporte les données au format JSON
with open('ticker_data_hist.json', 'w') as f:
    json.dump(ticker_data, f)

```

En faisant ça, nous recoltons toutes les valeurs des 10 'tickers' depuis 1000 jours (donc + de 2,5 ans) avec un intervalle de 8 heures.

6. Établissez une architecture de streaming pour recevoir les données de manière continue.

Comme demandé dans l'énoncé, nous avons cherché à collecter les données du marché de manière continue en utilisant une architecture de streaming.

```

import asyncio
import websockets
import json

```

```
import time

async def subscribe(websocket):
    # Abonnement au ticker de BTCUSDT

    await websocket.send('{"method": "SUBSCRIBE", "params":
["btcusdt@ticker"], "id": 1}')
```



```
async def receive(websocket, data_list):
    async for message in websocket:
        data = json.loads(message)
        print(data)
        data_list.append(data)
        break
```



```
async def main():
    async with websockets.connect("wss://stream.binance.com:9443/ws") as
websocket:

        await subscribe(websocket)

        # Mettre en place une boucle d'attente de 10 secondes
        # et d'arrêt après 1 minute (6 boucles)
        data_list = []
        for i in range(6):
            await receive(websocket, data_list)
            time.sleep(10)

        # Écrire les données dans un fichier JSON
        with open("BTCUSDT_data_stream.json", "w") as json_file:
            json.dump(data_list, json_file)
```



```
if __name__ == "__main__":
```

```
asyncio.run(main())
```

Le code ci-dessus contient 3 fonctions. Une fonction **main** qui est appelé au début, et qui crée un 'websocket' (canal de communication ouvert) avec Binance. Ensuite une fonction **subscribe** envoie une demande à Binance pour les informations du ticker 'BTCUSDT'. Ensuite, dans la fonction **main**, une boucle initie la fonction **receive** 6 fois, à dix secondes d'intervalle. Cette fonction **receive** incrémente une liste '**data_list**' avec les informations reçues. Enfin, le tout est exporté dans un fichier JSON à la fin.

7. Enregistrez les données dans un format approprié, tel que JSON.

Vu dans les deux fichiers de récupération de données précédents, les données sont exportés sous le format JSON.

Vous trouverez dans le dossier « **livrable_sprint_1_collecte** » un exemple de chacun des JSON exporté avec nos deux méthodes de collecte :

- ticker_data_hist → résultat du script de collecte historique
- BTCUSDT_data_stream → résultat du script de collecte stream