**Feuille de route d’exploitation**  
[BTC\_ETH\_streaming\_data](https://github.com/gaetancorin/btc_eth_streaming_data)

Réalisé par Gaëtan Corin

Introduction :

Le projet de récupération de données de crypto-monnaies BTC et ETH en données temps réels nécessite des actions de maintenance afin d’assurer la viabilité et pérennité de l’application. Ce document permet de lister ces différentes tâches de maintenances.

Contexte :

Le projet contient 6 services déployables en Docker de manière séparée. Il est donc parfaitement possible de travailler sur un seul service de manières indépendantes, ou bien de le tester de manière séparée lors des phases de maintenances.

Les services sont les suivants:

* Postgres

- Postgres\_external

- Postgres\_admin

* Mailhog
* Monitoring

- Docker exporter

- Prometheus

- Grafana

* Spark

- Spark-master

- Spark-worker-1

- Spark-worker-2

* Airflow

- Postgres

- Redis

- Airflow-apiserver

- Airflow-scheduler

- Airflow-dag-processor

- Airflow-worker

- Airflow-trigerrer

- Airflow-init

- Airflow-cli

* Ingestion

Maintenance :

Afin de maintenir en condition opérationnelle l’infrastructure DATA de ce projet, voici les opérations a mettres en places:

De manière constant:

* **Avoir son téléphone ainsi que les notifications activés reliés à l'email d’alerte**.  
  Ainsi, cela permet d’être directement alerté lors d’un problème d’ingestion de données (avec Ingestion et Postgres), un problème de transformation de données (avec Airflow, Spark et Postgres), ou un problème de monitoring (avec Monitoring qui détecte un manque de CPU, de mémoire, ou de container éteint).

De manières journalières:

* **Aller voir dans l’email de production les différents emails d’alerte reçus** (dans le cas où une notification de potentiel email d’alerte n’ai pas été vu).
* **Vérifier l’historique des dags Airflow** afin d’assurer le bon fonctionnement
* **Vérifier la consommation de CPUs, de mémoires disponible et l’état des containers sur les 24 dernières heures avec Grafana.**

De manières hebdomadaires:

* **Créer un dump de la base de données Postgres, et sauvegarder le fichier dans un cloud.**
* **Lancer en local les tests unitaires du service Ingestion** afin d’assurer le bon fonctionnement de l’ingestion.
* **Lancer/Mettre à jour le programme “Orchestrator”afin de simuler un déploiement en local.**
* **Réaliser au minimum un commit afin de faire fonctionner la CI/CD et vérifier le bon fonctionnement et la maintenance de la pipeline.**
* **Vérifier la pertinence des données des indicateurs en faisant fonctionner le fichier notebook permettant de créer des graphiques et d’avoir une visualisation globale.**

De manières mensuel:

* **Monter en versions les dépendances de chaque service et vérifier au bon fonctionnement une fois mise à jour (en utilisant le programme “Orchestrator” en local).**
* **Vérifier l’utilisation de minutes utilisées sur github actions** afin de ne passer à la version payant si on dépasse les 2000 minutes d’utilisation.
* **Contrôler l’utilisation des serveurs qui héberge l’application.**

De manières annuel:

* **Renouveler le nom de domaine**

Points de vigilances :

Il est nécessaire d’être vigilant sur certains points qui peuvent devenir critique pour l’application:

* La dépendance à la librairie YFinance pour la récupération des données peut être problématique sur le long terme. Il serait pertinent de trouver une autre sources de données en cas de panne de Yfinance
* L’application fait tourner 19 containers docker en même temps. Une vigilance spécifique doit être dédiée sur l’utilisation de la quantité de CPUs et de mémoire afin de ne pas tomber en saturation.

Aide au développement :

Des outils ont été spécifiquement développés pour faciliter la maintenance et le débogage de l’application:

* Le programme “Orchestrator.ps1” permet de lancer l’ensemble des 19 containers Docker en local afin de pouvoir tester directement l’interaction des containers entre eux de manière simple et rapide.
* Le programme “Clean-orchestrator.ps1” permet de supprimer l’ensemble des 19 containers Docker en local, ainsi que leurs volumes, afin de pouvoir facilement nettoyer l’espace de travail après un test.
* Dans chaque service, il existe un fichier “help\_{nom\_du\_service}.txt” avec des explications claires de comment lancer le service de manière individuel, de comment travailler dessus, et de comment le supprimer. Cela permet de ne modifier que le service en question lors d’un travail effectué sur un service, ce qui améliore grandement la phase de tests lors de modifications.