bien, nous allons enfin finaliser helpers.py dans un premier temps.

Dans un deuxieme temps nous allons finaliser setup\_logging.py, error\_logging.py et arbitrage\_instance\_metrics.py.

Dans un troisième temps nous allons finaliser et/ou établir les fichiers transversaux config.yaml, prometheus.yml, grafana\_dashboard.json, pairs\_data.json, trade\_data.json, metrics.json, execution\_results.json, performance\_metrics.py

Dans un quatrieme temps nous allons travailler sur la représentation du pipeline et des différents workflow dans grafana pour une représentation sous forme de grid, soit une extension du diagramme du processus global.

Ensuite nous entrerons dans un cinquieme temps sur l'établissement de grafana et l'exploitation de prometheus.

Dans un sixieme temps nous aborderons l’intégration de chatgpt dans le pipeline, en guise de superviseur lors du déploiement du pipeline, dans le cas où des erreurs de scripts entravent le déploiement du pipeline au sein de digital ocean à travers une conteneurisation de l’ensemble et pipelinisé via Apache Airflow. J’ai besoin d’un moniteur renvoyant les messages d’erreurs à l’image du terminal de vscode lors de l’execution du script, ou de powershell lors du docker-compose up.

Dans un septieme temps, on envisagera d’intégrer un modèle semblable à FreqAI au pipeline afin de s’ouvrir des possibilités plus vaste en termes de stratégie autre que l’arbitrage triangulaire et partir sur des stratégies de trading basées sur des modèles adaptatifs qui s'ajustent aux conditions changeantes du marché.

En 1/ établis la synthèse de ce déroulé.

En 2/ les codes fournit nécessitent-ils un complément pour tout ce qui attrait à la conception des moniteurs grafana, en plus de prometheus.yml je veux dire?.

En 3/ représente-moi le répertoire au complet suivant les scripts proposés inclut dans le pipeline.

Nous adopterons cette chronologie "Résumé de la Chronologie :

config.yaml (base de configuration globale).

prometheus.yml (surveillance et collecte des métriques).

grafana\_dashboard.json (visualisation des métriques).

pairs\_data.json (gestion des paires dynamiques).

trade\_data.json (données des ordres actifs).

metrics.json (sauvegarde des métriques locales).

execution\_results.json (archive des résultats d’exécution).

performance\_metrics.py (indicateurs avancés de performance).". Prend en compte qu'il y a 20 websockets et minimum 20 clé API mobilisées pour le workflow 1 seulement, sur 29 clés créées