Royaume du Maroc
Ministère de l'Enseignement Supérieur, de la Recherche Scientifique et de l'Innovation

École Nationale des Sciences Appliquées
ENSA – Tanger

Conception et implémentation d'un pipeline de logs distribué

Projet Big Data / Data Engineering

Module : Big Data

Réalisé par :
Abdelilah Benchine
Génie Informatique

Encadré par :
M. BADIR Hassan
Enseignant-Chercheur à l'ENSA de Tanger

Année universitaire : 2025 – 2026

Table des matières

Chapitre 1

Introduction au Big Data

Le Big Data désigne l'ensemble des technologies et méthodes permettant de stocker, traiter et analyser des volumes massifs de données générées à grande vitesse et sous des formes variées. Dans ce contexte, les architectures distribuées deviennent indispensables pour garantir performance, scalabilité et tolérance aux pannes.

1.1 Du traitement classique au Big Data

Données massives + Calcul distribué = Big Data

Contrairement aux systèmes traditionnels centralisés, le Big Data repose sur une distribution des données et des calculs sur plusieurs nœuds.

1.2 Caractéristiques du Big Data

Le Big Data est défini par les **5V** : Volume, Vélocité, Variété, Véracité et Valeur.

1.3 Comparaison

Critère	Classique	Big Data
Architecture	Centralisée	Distribuée
Traitement	Batch	Batch + Streaming
Scalabilité	Faible	Horizontale

Chapitre 2

Architecture générale du projet

Ce projet vise à mettre en œuvre un **pipeline Big Data complet** destiné au traitement et à l'analyse de logs applicatifs générés de manière continue. L'objectif principal de cette architecture est de simuler un environnement réel de Data Engineering, capable de gérer des flux de données à grande échelle, depuis leur ingestion jusqu'à leur analyse finale.

L'architecture adoptée repose sur une approche **modulaire et distribuée**, dans laquelle chaque composant joue un rôle bien défini. Cette séparation permet d'assurer la scalabilité du système, la tolérance aux pannes et la facilité de maintenance.

2.1 Pipeline global

Le pipeline Big Data implémenté dans ce projet est composé de cinq étapes principales, chacune correspondant à une phase clé du cycle de vie des données :

1. **Génération de logs** : Des logs applicatifs sont simulés à l'aide d'un producteur Python. Chaque log représente un événement HTTP (requête, statut, temps de réponse, etc.), généré à une fréquence élevée afin de reproduire un trafic réel.
2. **Ingestion des données avec Apache Kafka** : Les logs générés sont envoyés vers un topic Kafka nommé `logs_raw`. Kafka agit comme un système de messagerie distribué, assurant la réception fiable et ordonnée des flux de données en temps réel.
3. **Traitement en temps réel avec Spark Streaming** : Apache Spark Structured Streaming consomme les messages Kafka, applique des transformations (parsing JSON, filtrage, agrégation) et prépare les données pour leur stockage.
4. **Stockage distribué avec HDFS** : Les données traitées sont stockées dans le système de fichiers distribué HDFS sous forme de fichiers Parquet, organisés en partitions temporelles (date et heure).

5. **Analyse batch avec Apache Spark** : Une phase d'analyse batch permet d'agréger les données stockées afin de produire des indicateurs globaux (top endpoints, statistiques par heure, erreurs serveur, etc.).

Cette architecture garantit une séparation claire entre les traitements temps réel et les analyses différées, conformément aux bonnes pratiques du Big Data.

2.2 Déploiement de l'infrastructure

L'ensemble de l'infrastructure est déployé à l'aide de **Docker Compose**, ce qui permet de lancer et d'orchestrer facilement les différents services nécessaires au pipeline (Kafka, Spark, HDFS, etc.).

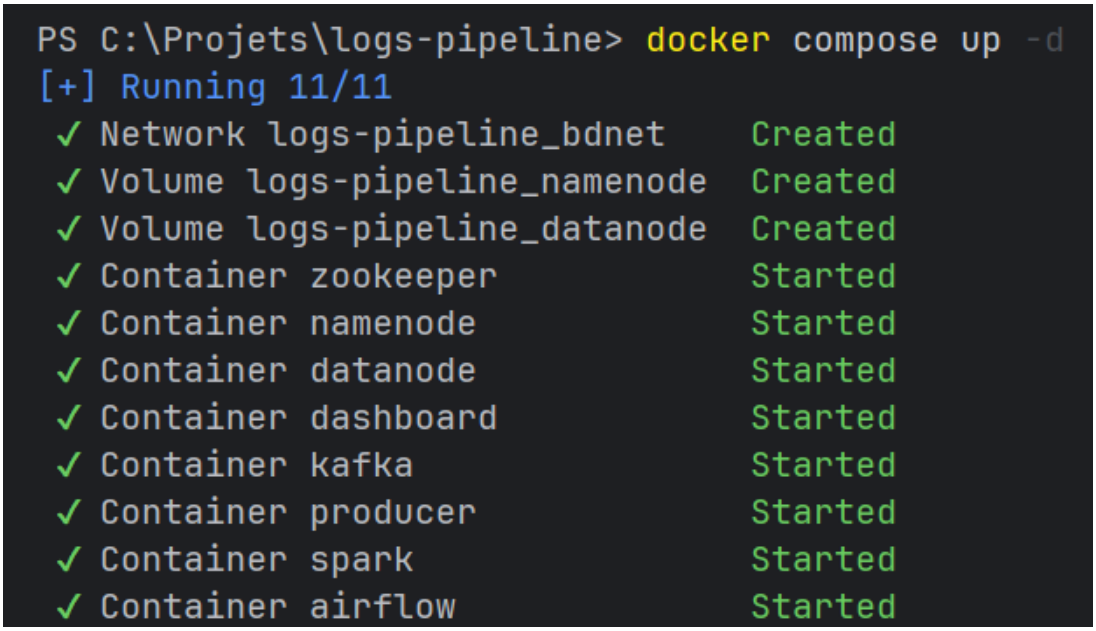

FIGURE 2.1 – Lancement de l'infrastructure Big Data via Docker Compose

La Figure ci-dessus illustre le démarrage de l'infrastructure à l'aide de la commande `docker compose up`. Cette approche garantit un environnement reproductible et isolé, indépendant du système hôte.

2.3 Services du pipeline en cours d'exécution

Une fois l'infrastructure lancée, il est possible de vérifier l'état des différents services du pipeline.

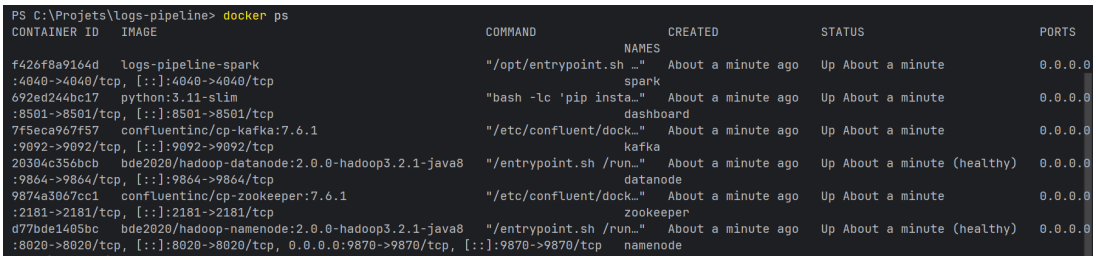

FIGURE 2.2 – Services Docker du pipeline Big Data en cours d'exécution

La Figure précédente montre que l'ensemble des conteneurs nécessaires au bon fonctionnement du pipeline sont actifs, notamment Kafka, Spark et HDFS. Cette étape constitue une vérification essentielle avant le lancement des traitements de données, car elle garantit la disponibilité de tous les composants de l'architecture.

—

Chapitre 3

Ingestion des données avec Apache Kafka

Dans ce projet, **Apache Kafka** est utilisé comme système de messagerie distribué pour assurer l'ingestion des logs applicatifs en temps réel. Kafka joue un rôle central dans l'architecture en servant de tampon entre la phase de génération des données et les traitements effectués par Spark.

Grâce à son architecture distribuée et à son modèle basé sur les topics, Kafka permet de gérer efficacement des flux de données continus tout en garantissant la fiabilité et la scalabilité du pipeline.

3.1 Création et gestion des topics

Dans Kafka, les données sont organisées au sein de **topics**, qui représentent des canaux logiques de communication. Pour ce projet, un topic dédié nommé `logs_raw` a été créé afin de stocker les logs bruts générés par l'application.

```
PS C:\Projets\logs-pipeline> docker exec -it kafka kafka-topics `
>>   --bootstrap-server localhost:9092 `
>>   --create `
>>   --topic logs_raw `
>>   --partitions 3 `
>>   --replication-factor 1
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues it is best to use either,
 but not both.
Created topic logs_raw.
```

FIGURE 3.1 – Création du topic Kafka `logs_raw`

La Figure ci-dessus illustre la création du topic `logs_raw`, avec une configuration adaptée à un flux de données continu (partitions multiples pour permettre le parallélisme).

Une fois le topic créé, il est nécessaire de vérifier son existence et sa configuration.

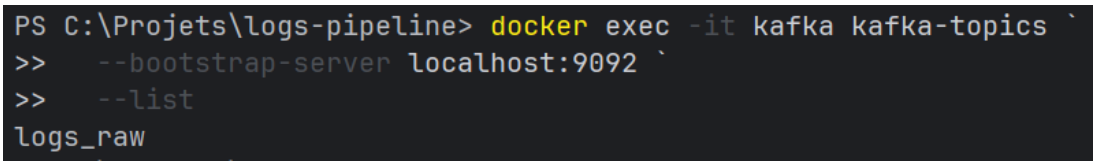

FIGURE 3.2 – Liste des topics Kafka disponibles

Cette étape permet de s'assurer que le topic est correctement enregistré dans le broker Kafka avant de lancer la production de messages.

3.2 Production et consommation des messages

La phase suivante consiste à produire des logs applicatifs et à les envoyer vers Kafka. Un **producer Python** est utilisé pour simuler la génération continue de logs HTTP, chacun représentant une requête vers l'application.

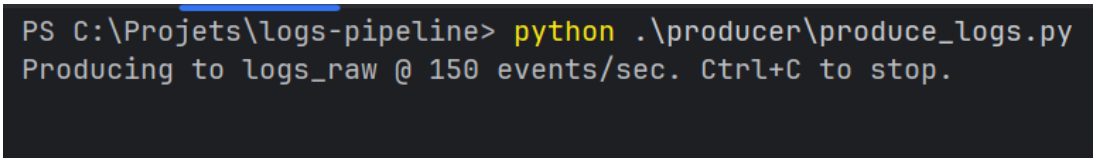

FIGURE 3.3 – Génération continue des logs par le producer Python

La Figure précédente montre le producteur en cours d'exécution, générant et envoyant des événements vers le topic `logs_raw` à une fréquence élevée, ce qui permet de simuler un trafic réel.

Afin de vérifier le bon fonctionnement de l'ingestion, un **consumer Kafka** est lancé pour lire les messages depuis le topic.

FIGURE 3.4 – Consommation des logs depuis le topic Kafka `logs_raw`

Cette consommation en temps réel permet de confirmer que les messages sont correctement produits, stockés et accessibles dans Kafka. Une fois cette étape validée, les données sont prêtes à être consommées par le moteur de traitement Spark Streaming.

—

Chapitre 4

Traitement en temps réel avec Spark Streaming

Dans cette phase du pipeline, **Apache Spark Structured Streaming** est utilisé pour traiter les logs applicatifs en temps réel dès leur arrivée dans Kafka. Ce mode de traitement permet d'analyser les données en continu, sans attendre leur stockage complet, ce qui est particulièrement adapté aux systèmes générant des flux permanents de données.

Spark Structured Streaming repose sur le moteur Spark SQL et offre une approche unifiée entre le traitement batch et le traitement streaming, facilitant ainsi le développement et la maintenance des applications Big Data.

4.1 Principe du traitement en continu

Le traitement en temps réel consiste à consommer les messages depuis Kafka, à les transformer et à les stocker progressivement dans HDFS. Chaque log est traité comme un événement individuel, tout en permettant des agrégations sur des fenêtres temporelles.

Les principales opérations réalisées lors de cette phase sont :

— Lecture des messages depuis le topic Kafka `logs_raw`
— Parsing des données JSON vers un format structuré
— Filtrage des enregistrements invalides ou incomplets
— Enrichissement et agrégation des données

4.2 Lancement du job Spark Streaming

Le job Spark Streaming est lancé à l'aide de la commande `spark-submit`, qui permet de soumettre une application Spark au cluster.

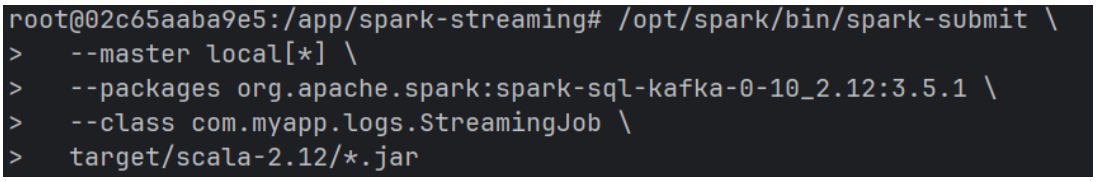

FIGURE 4.1 – Commande de lancement du job Spark Structured Streaming

La Figure ci-dessus illustre le lancement du job Spark Streaming, incluant la configuration du master, des dépendances Kafka et de la classe principale de l'application. Cette étape marque le début du traitement en temps réel des logs.

4.3 Exécution et suivi du traitement

Une fois le job lancé, Spark démarre la consommation des messages depuis Kafka et applique les transformations définies dans l'application.

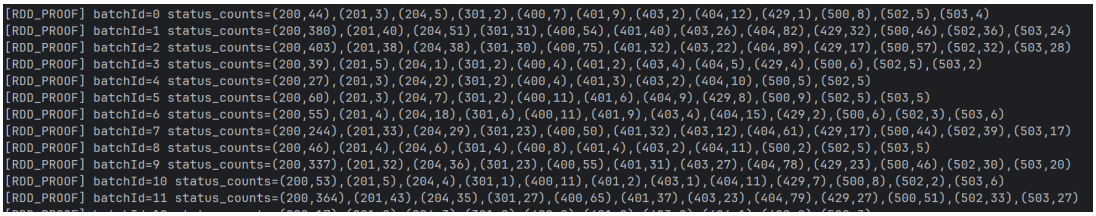

FIGURE 4.2 – Spark Structured Streaming en cours de traitement des logs

La Figure précédente montre le job Spark Streaming en cours d'exécution. Les logs affichés indiquent que les données sont correctement consommées depuis Kafka et traitées par Spark. Les résultats intermédiaires sont ensuite écrits de manière continue dans HDFS, garantissant la persistance des données et la tolérance aux pannes grâce au mécanisme de **checkpointing**.

Cette étape constitue le cœur du pipeline Big Data, reliant l'ingestion temps réel à la phase de stockage distribué.

—

Chapitre 5

Stockage distribué avec HDFS

Les données traitées sont stockées dans HDFS selon une structure partitionnée.

```
PS C:\Projets\logs-pipeline> docker exec -it namenode hdfs dfs -ls /datalake/logs
Found 2 items
drwxr-xr-x   - root supergroup          0 2025-12-24 17:49 /datalake/logs/_chk
drwxr-xr-x   - root supergroup          0 2025-12-24 18:13 /datalake/logs/curated
```

FIGURE 5.1 – Organisation du Data Lake dans HDFS

```
PS C:\Projets\logs-pipeline> docker exec -it namenode hdfs dfs -ls /datalake/logs/curated/dt=2025-12-24
Found 1 items
drwxr-xr-x   - root supergroup          0 2025-12-24 18:15 /datalake/logs/curated/dt=2025-12-24/hour=18
```

FIGURE 5.2 – Partitions temporelles des logs

```
PS C:\Projets\logs-pipeline> docker exec -it namenode hdfs dfs -find /datalake/logs/curated -name "*.parquet"
/datalake/logs/curated/dt=2025-12-24/hour=18/part-00004-e421d06f-9e94-4c0c-af2d-d0b792a7271b.c000.snappy.parquet
/datalake/logs/curated/dt=2025-12-24/hour=18/part-00008-31cc79a3-553d-49b7-98dc-c9679f1aa18a.c000.snappy.parquet
/datalake/logs/curated/dt=2025-12-24/hour=18/part-00011-ad3f9837-4225-4eb6-9bee-8707c253a483.c000.snappy.parquet
/datalake/logs/curated/dt=2025-12-24/hour=18/part-00012-6e77ea8b-bb97-4d78-9cfc-6cd515aa04bf.c000.snappy.parquet
/datalake/logs/curated/dt=2025-12-24/hour=18/part-00014-5dc11cba-2b68-496c-b36c-89c7bcf62ba9.c000.snappy.parquet
/datalake/logs/curated/dt=2025-12-24/hour=18/part-00018-85dcea88-fc35-4b4e-ba75-73cb9c56212f.c000.snappy.parquet
/datalake/logs/curated/dt=2025-12-24/hour=18/part-00019-ac353a38-18ca-4654-9bda-834cf587fc93.c000.snappy.parquet
/datalake/logs/curated/dt=2025-12-24/hour=18/part-00024-fdf07bac-0318-4275-af88-416c880d3253.c000.snappy.parquet
/datalake/logs/curated/dt=2025-12-24/hour=18/part-00030-7c402807-0c9f-46a3-b7e9-30d5f28a0f25.c000.snappy.parquet
```

FIGURE 5.3 – Fichiers Parquet générés

Chapitre 6

Traitement Batch et analyse

Après la phase de traitement en temps réel et le stockage des données dans HDFS, une étape de **traitement batch** est mise en place afin d'analyser l'ensemble des logs collectés sur une période donnée. Contrairement au streaming, le traitement batch s'effectue sur des données déjà persistées et permet de réaliser des agrégations globales plus complexes.

Cette phase vise à produire des indicateurs synthétiques exploitables pour l'analyse décisionnelle et l'observation du comportement global du système.

6.1 Objectifs du traitement batch

Le traitement batch permet notamment :

— d'agréger les données issues du Data Lake

— de calculer des statistiques globales sur l'ensemble des logs

— de produire des jeux de données analytiques prêts à être exploités

Dans ce projet, le traitement batch est implémenté à l'aide d'un job Spark dédié nommé `BatchAnalytics`.

6.2 Lancement du job BatchAnalytics

Le job batch est lancé via la commande `spark-submit`, en utilisant les fichiers Parquet générés par Spark Streaming comme source de données.

```
root@02c65aaba9e5:/app/spark-batch# /opt/spark/bin/spark-submit \
>    --master local[*] \
>    --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.1 \
>    --class com.myapp.logs.BatchAnalytics \
>    target/scala-2.12/*.jar
```

FIGURE 6.1 – Lancement du job Spark BatchAnalytics

La Figure ci-dessus illustre l'exécution du job `BatchAnalytics`. Celui-ci lit les données depuis le répertoire `/datalake/logs/curated` et calcule plusieurs indicateurs globaux, tels que :

— les chemins (*paths*) les plus sollicités
— les indicateurs par heure (trafic, erreurs, temps de réponse)
— les statistiques par hôte

6.3 Stockage des résultats analytiques

Les résultats du traitement batch sont écrits dans HDFS, dans un répertoire dédié `/datalake/logs/analytics`. Chaque type d'analyse est stocké dans un sous-dossier distinct afin de faciliter leur exploitation ultérieure.

```
PS C:\Projets\logs-pipeline> docker exec -it namenode hdfs dfs -ls /datalake/logs/analytics
Found 4 items
drwxr-xr-x   - root supergroup          0 2025-12-24 18:36 /datalake/logs/analytics/kpi_by_host
drwxr-xr-x   - root supergroup          0 2025-12-24 18:36 /datalake/logs/analytics/kpi_by_hour
drwxr-xr-x   - root supergroup          0 2025-12-24 18:36 /datalake/logs/analytics/top_paths
drwxr-xr-x   - root supergroup          0 2025-12-24 18:37 /datalake/logs/analytics/top_paths_csv
```

FIGURE 6.2 – Résultats analytiques stockés dans HDFS

Cette organisation permet une séparation claire entre les données brutes, les données traitées en streaming et les résultats analytiques finaux, conformément aux bonnes pratiques du Data Engineering.

6.4 Exploration et validation des résultats

Afin de vérifier et de visualiser les résultats produits par le traitement batch, l'outil `spark-shell` est utilisé. Il permet de charger directement les fichiers Parquet depuis HDFS et d'afficher leur contenu sous forme tabulaire.

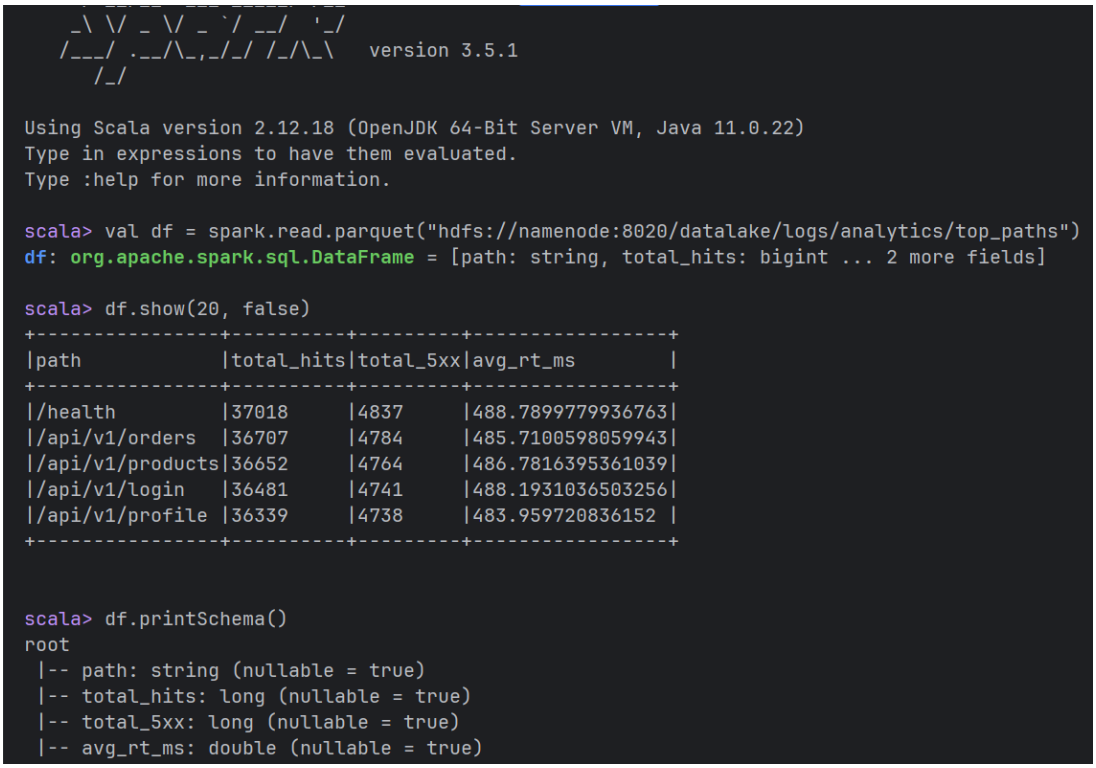

FIGURE 6.3 – Visualisation des résultats analytiques via spark-shell

La Figure précédente montre un exemple d'affichage des résultats du traitement batch, confirmant que les agrégations ont été correctement effectuées et que les données sont exploitables. Cette étape constitue une validation finale du pipeline Big Data, démontrant la cohérence et la fiabilité des résultats produits.

—

Conclusion et perspectives

Ce projet a permis de concevoir et de mettre en œuvre un **pipeline Big Data complet et fonctionnel**, couvrant l'ensemble du cycle de vie des données, depuis leur génération jusqu'à leur analyse finale. À travers ce travail, les principaux concepts du **Data Engineering** ont été concrètement appliqués, notamment l'ingestion de données en temps réel, le traitement distribué, le stockage à grande échelle et l'analyse batch.

L'architecture mise en place, basée sur des technologies largement utilisées dans l'industrie telles que **Apache Kafka**, **Apache Spark** et **HDFS**, illustre une approche moderne et scalable du traitement des données massives. L'utilisation de **Docker** a également permis de garantir un environnement reproductible et cohérent, facilitant le déploiement et l'orchestration des différents composants du pipeline.

Ce projet a ainsi permis de mieux comprendre les enjeux réels liés à la gestion des flux de données continus, à la tolérance aux pannes et à la structuration d'un Data Lake, tout en mettant en évidence l'importance de la séparation entre les traitements temps réel et batch.

Améliorations possibles

Plusieurs pistes d'amélioration peuvent être envisagées afin d'enrichir et d'industrialiser davantage ce pipeline Big Data :

- **Ajout d'une couche de visualisation** : L'intégration d'outils tels que Grafana, Apache Superset ou Power BI permettrait de visualiser les indicateurs analytiques et de faciliter l'aide à la décision.
- **Intégration de formats transactionnels avancés** : L'utilisation de technologies comme **Delta Lake** ou **Apache Iceberg** améliorerait la gestion des versions, la fiabilité des données et les performances des requêtes analytiques.
- **Déploiement sur un cloud public** : Un déploiement sur des plateformes telles qu'AWS, Azure ou GCP permettrait de bénéficier d'une scalabilité dynamique et de services managés pour le stockage et le calcul.
- **Mise en place d'un monitoring avancé** : L'ajout de mécanismes de supervision et d'alerting (Prometheus, Grafana) permettrait de surveiller l'état du pipeline en temps réel et d'anticiper les éventuelles défaillances.

Ces perspectives ouvrent la voie vers une solution Big Data plus robuste, plus performante et plus proche des architectures utilisées en milieu professionnel, constituant ainsi une base pertinente pour des projets futurs ou des travaux de fin d'études.