EPSI : école privée des sciences informatiques

DEPARTEMENT INFORMATIQUE

Première Année

ATL-BigData

Rapport de projet

Rapport réalisé par :

BAZIZ Ilhem

LEKHAL Massinissa

BELABBAS Madani Wassim

Responsable de l'UE :

Hedi Boufaden

Année universitaire : 2024/2025

Introduction :

Dans le cadre de notre formation en Big Data, nous avons mené à bien un projet visant à concevoir et déployer une architecture de traitement de données en temps réel à l'aide de Kafka et Spark. Ce projet s'inscrit dans une approche pratique et immersive, avec pour objectif de maîtriser les différentes étapes de gestion des flux de données, depuis leur ingestion jusqu'à leur exploitation finale.

Le projet consistait à développer une solution de streaming permettant de traiter des données financières en temps réel. Pour ce faire, nous avons mis en œuvre une architecture reposant sur les principes suivants :

- L'utilisation de Kafka pour la gestion et la transmission des flux de données.
- La transformation et le stockage des données en DataFrame avec Spark Streaming.
- L'application de traitements analytiques tels que la conversion monétaire, la gestion des fuseaux horaires et la validation des données.
- Le stockage des résultats transformés au format Parquet sur Minio pour une exploitation ultérieure.

Ce document retrace les différentes étapes du projet, de la mise en place des composants techniques à l'élaboration d'un pipeline de traitement performant et optimisé.

Objectifs principaux :

Ce projet vise à démontrer l'intégration de plusieurs technologies pour gérer, traiter et analyser des flux de données en temps réel dans un environnement distribué.

Les objectifs principaux sont :

1. **Gestion des transactions avec Kafka :**

    - Implémenter un Producer Kafka pour envoyer des messages au topic "transaction", sous forme de données JSON, avec des informations telles que l'identifiant de la transaction, le type de transaction, le montant, la devise, la date, le lieu, et d'autres détails pertinents.

2. **Consommation des données en temps réel avec Spark Streaming :**

    - Utiliser Spark Streaming pour consommer les messages du topic Kafka en temps réel et les analyser.
    - Utiliser Spark SQL pour transformer et structurer ces données.

3. **Stockage et analyse des données** :

    - Les données traitées seront stockées dans un système de fichiers compatible avec S3, tel que MinIO, en utilisant le format Parquet pour une analyse ultérieure.

4. **Gestion des dépendances et configuration du projet** :

    - Mettre en place un environnement de développement avec **Akka**, **Kafka**, **Spark**, et d'autres dépendances nécessaires pour un traitement efficace des données.
    - Configurer **SBT** pour gérer les dépendances et les versions des bibliothèques utilisé

Lancement du projet

Méthodologie

Partie 1 : Installation des composants

1. *Docker*

Docker est utilisé pour faciliter le déploiement et la gestion des services requis. Après installation, la commande suivante permet de démarrer les services nécessaires :
docker-compose up -d # Windows

2. *Conduktor*

Conduktor est utilisé pour gérer Apache Kafka. Il est disponible pour tous les OS via : Conduktor

3. Spark

Spark est utilisé pour le traitement des flux de données en temps réel. Il est installé via : Installation Spark.

- Pré-requis : JDK ou OpenJDK 8.

4. *IntelliJ IDEA*

L'IDE IntelliJ de JetBrains a été utilisé pour développer le projet en Scala. : JetBrains.

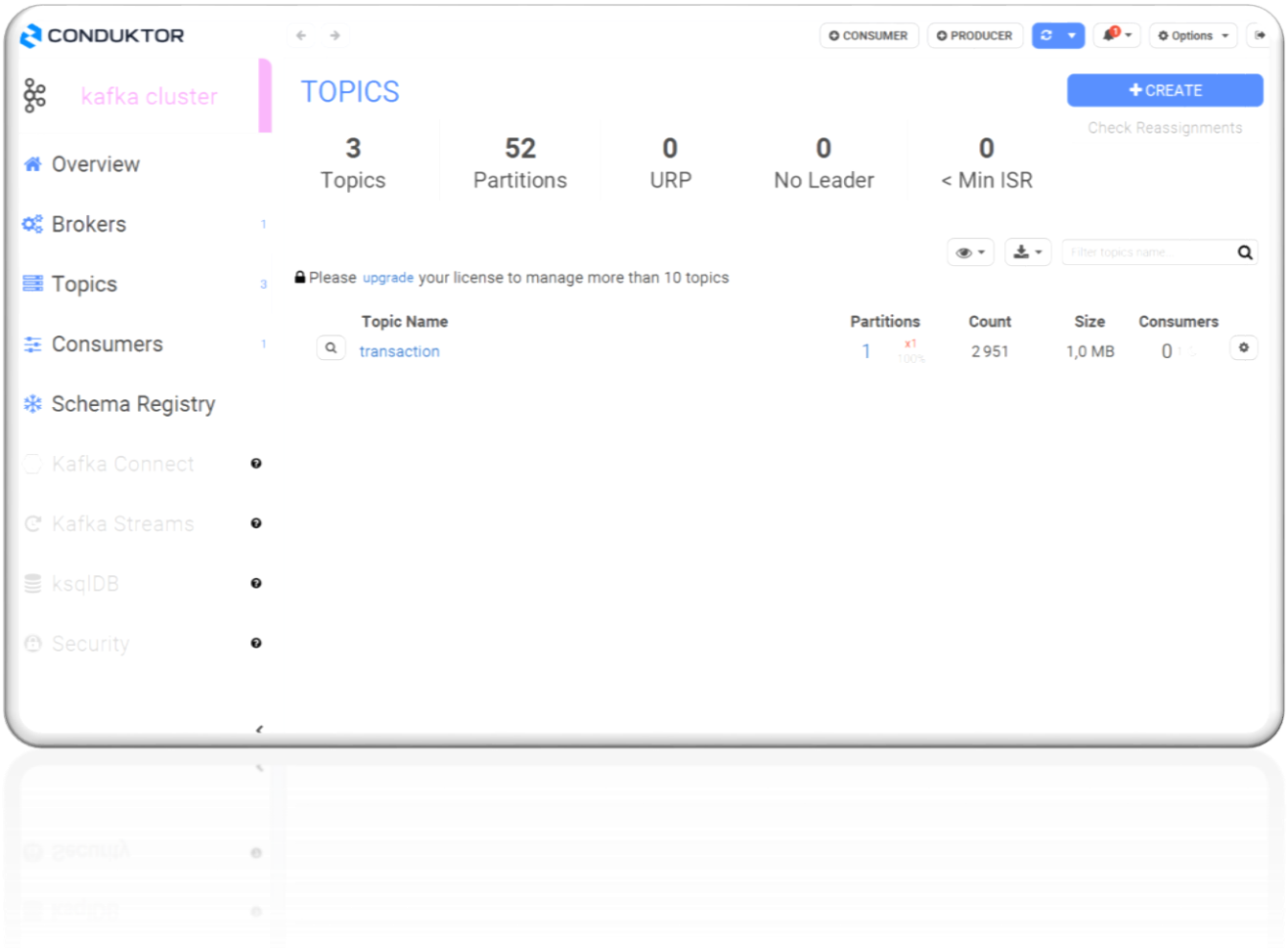
Partie 2 : Mise en place du projet

. Développement en Scala

Le projet a été développé en Scala avec Kafka Producer et Kafka Consumer. IntelliJ IDEA a été utilisé pour l'écriture du code.

1. Développement du Producer Kafka

- Le **Producer Kafka** (Kafkaproducer.scala) a été implémenté en utilisant **Akka Streams** et **Kafka** pour envoyer des messages au topic "transaction".
- Le message représente une transaction sous forme de JSON, avec des attributs tels que (idTransaction, typeTransaction, montant, devise, date, lieu, moyenPaiement ..etc).
- La sérialisation des objets Transaction en JSON est effectuée par un sérialiseur personnalisé utilisant la bibliothèque **Gson**.
- Chaque transaction est générée de **manière aléatoire**, avec des données simulant des achats, remboursements ou transferts, ainsi que des informations comme le montant, le type de paiement, et les détails de l'utilisateur.
- Les transactions sont envoyées toutes les secondes au **cluster Kafka local** via un producteur Kafka configuré pour se connecter à **localhost:9092**.
- Le code permet ainsi d'envoyer périodiquement des messages de transactions, qui peuvent être visualisés dans le topic "transaction" du cluster Kafka local.



2. Développement du Consumer Kafka

- Le consommateur Kafka(KafkaConsumer.scala) développé avec **Apache Spark Streaming**, lit en temps réel les transactions envoyées au topic Kafka **"transaction"**.
- Après réception des messages, ceux-ci sont convertis du format binaire en JSON et structurés avec un schéma défini.
- Les données sont ensuite traitées en continu avec Spark, et un nombre limité de messages (20) est traité à chaque exécution.
- Les résultats sont écrits en **format Parquet** sur un stockage MinIO (simulant S3), avec un mécanisme de **checkpointing** pour assurer la fiabilité du traitement.
- Une fois le traitement effectué, les données sont lues depuis le stockage Parquet pour valider leur écriture.

Ce traitement permet d'assurer une gestion efficace des données en temps réel tout en garantissant leur intégrité et leur stockage durable.

3. Dépendances du Projet

Les dépendances du projet sont gérées via **sbt**, le système de build Scala. Le fichier **build.sbt** contient les bibliothèques nécessaires pour intégrer **Apache Spark**, **Kafka**, **Spark** :

1. **Spark :**
   - **spark-core, spark-sql, spark-streaming** : Bibliothèques pour le traitement distribué et en temps réel des données avec Apache Spark.
   - spark-sql-kafka-0-10 : Pour connecter Spark avec Kafka afin de consommer et traiter des flux de données en temps réel.
2. **Hadoop :**
   - **hadoop-aws :** Utilisé pour interagir avec des services de stockage compatibles avec S3, comme MinIO.
3. **Akka :**
   - **akka-actor, akka-stream, akka-stream-kafka :** Utilisés pour gérer les flux de données Kafka de manière asynchrone avec Akka Streams**.**
4. **Kafka :**
   - **kafka-clients :** Pour l'intégration avec Kafka et la gestion des producteurs/consommateurs de messages.
5. **Autres :**
   - **gson :** Pour la sérialisation et désérialisation des objets JSON.
   - **logback-classic :** Pour la gestion des logs, compatible avec Java 8.

Toutes ces dépendances permettent de mettre en place un système de traitement de données en temps réel avec Kafka, Spark, et Akka, garantissant ainsi un flux de données fluide et fiable.