Une image contenant texte, Police, capture d’écran, Graphique

Le contenu généré par l’IA peut être incorrect.Ilhem BAZIZ – Ould Taleb Dyhia

Rapport :Architecture d'ingestion des données

Exercice 01 : Ingestion de CSV dans Apache Kafka via Kafka Connect

# ✅ Prérequis

- Apache Kafka installé  
- Kafka Connect installé  
- Java installé (vérifie avec `java -version`)  
- Ton fichier CSV prêt

# 1️⃣ Démarrage de Kafka et Zookeeper

Démarrer Zookeeper :

bin/zookeeper-server-start.sh config/zookeeper.properties

Une image contenant texte, capture d’écran, menu, Police

Le contenu généré par l’IA peut être incorrect.

Dans un autre terminal :

bin/kafka-server-start.sh config/server.properties

Une image contenant texte, capture d’écran, noir et blanc, monochrome

Le contenu généré par l’IA peut être incorrect.

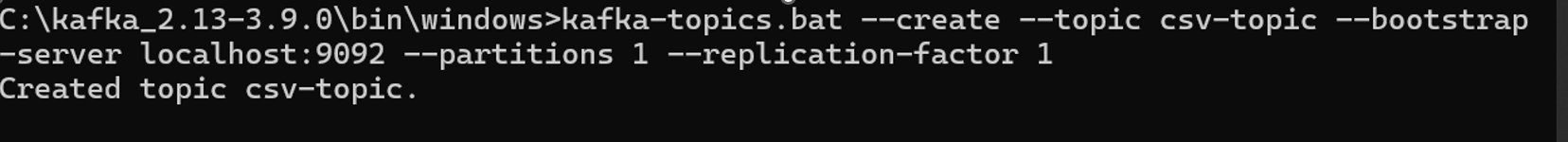
# 2️⃣ Création du fichier CSV

Crée un fichier CSV de test :

echo "id,nom,age" > /tmp/data.csv  
echo "1,Ilhem,24" >> /tmp/data.csv  
echo "2,Ali,30" >> /tmp/data.csv

# 3️⃣ Création du topic Kafka

bin/kafka-topics.sh --create --topic csv\_topic \  
 --bootstrap-server localhost:9092 \  
 --partitions 1 --replication-factor 1



# 4️⃣ Création du fichier de configuration Kafka Connect

Crée un fichier appelé file-source.properties :

nano file-source.properties

Ajoute ce contenu dedans :

name=file-source  
connector.class=FileStreamSource  
tasks.max=1  
file=/tmp/data.csv  
topic=csv\_topic

# 5️⃣ Lancer Kafka Connect en standalone

bin/connect-standalone.sh config/connect-standalone.properties file-source.properties

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

# 6️⃣ Vérification des messages dans Kafka

bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic csv\_topic --from-beginning

Résultat attendu :

id,nom,age  
1,Ilhem,24  
2,Ali,30

Exercice 02 : Configuration de la scalabilité et de la résilience de l’ingestion des données

**1. Apache Kafka**

**Scalabilité**

* **Partitionnement des topics** :  
  Chaque topic Kafka peut être divisé en plusieurs partitions. Cela permet une ingestion parallèle des données.
* **Ajout de producteurs et de consommateurs** :  
  Plusieurs producteurs peuvent envoyer des données à un même topic, et plusieurs consommateurs (dans un consumer group) peuvent traiter les messages en parallèle.
* **Répartition par clé** (key-based partitioning) :  
  Kafka peut envoyer les messages vers des partitions spécifiques selon une clé (ex. : ID utilisateur), assurant un bon équilibrage de charge.
* **Exemple de commande pour créer un topic scalable** :
* **kafka-topics.sh --create --topic nom-topic --bootstrap-server localhost:9092 --partitions 4 --replication-factor 2**
* **Résilience**
* **Réplication des partitions** :  
  Chaque partition est répliquée sur plusieurs brokers. En cas de panne, Kafka bascule vers un réplica.
* **Accusés de réception (acks)** :  
  En configurant acks=all, on s’assure que le message est bien écrit sur tous les réplicas avant confirmation.
* **Offsets et reprise automatique** :  
  Kafka garde une trace de l’emplacement de lecture (offset), ce qui permet à un consommateur de reprendre après une panne sans perte de données.

**2. Apache Flink**

* **Scalabilité**
* **Parallélisme configurable** :  
  Chaque tâche de traitement peut être exécutée en plusieurs instances simultanées.
* **Adaptation au nombre de partitions Kafka** :  
  Flink peut automatiquement créer des sous-tâches pour correspondre au nombre de partitions Kafka en entrée.
* **Résilience**
* **Checkpointing automatique** :  
  Flink prend des instantanés réguliers de l’état du job. En cas de crash, il redémarre à partir du dernier checkpoint.
* **State Backend persistant** :  
  L’état peut être sauvegardé sur disque avec RocksDB, ce qui le rend plus résistant aux pannes.
* **Haute disponibilité (High Availability)** :  
  Avec ZooKeeper, Flink peut relancer automatiquement les tâches et conserver la continuité du service.