

TP Atelier Big Data

Valentin KOUN, Julie MONTEIRO & Mehdi BOURAS

Lien du git : https://github.com/Julilaa/TP_BigDATA_M1

Introduction	1
Architecture globale du projet	2
Préparation et Mise en Place de l'Environnement	3
Mise en place de l'environnement	5
Développement des Scripts	6
1. Création des Topics Kafka et Envoi des Données (Producer Kafka)	6
2. Lecture des Données avec Spark Streaming (Consumer)	8
3. Transformations des Données	9
4. Sauvegarde des Données dans MinIO	11
5. Lecture et Validation des Données (optionnel)	12
Conclusion	14

Introduction

Dans ce projet, nous avons exploré le domaine du Big Data à travers une simulation de flux de transactions financières, depuis leur création jusqu'à leur traitement et leur stockage. L'objectif était de comprendre comment gérer des données en temps réel et d'acquérir des bases solides sur les outils clés du Big Data.

Nous avons utilisé Kafka pour gérer le flux de données, Conduktor pour superviser et visualiser ce flux, et Spark pour transformer et nettoyer les données. Ces données ont ensuite été sauvegardées dans MinIO dans un format adapté au traitement de gros volumes. Enfin, tous nos scripts ont été développés et exécutés avec Python, en utilisant l'environnement intégré Pycharm de la suite JetBrains, qui a grandement facilité le processus.

Architecture globale du projet

Le projet se structure autour de plusieurs étapes fondamentales qui illustrent tout le cycle de vie des données en temps réel, depuis leur création jusqu'à leur analyse. La première étape consiste à produire des données simulées grâce à un producteur Kafka. Ces données, générées de manière aléatoire, représentent des transactions contenant des informations variées, telles que des montants, des adresses ou des moyens de paiement. Cette phase permet d'imiter un flux continu de données, tel qu'un système de capteurs en temps réel.

Une fois les données produites, elles sont transformées à l'aide de Spark Streaming. Cette étape est cruciale pour normaliser et enrichir les données. Parmi les opérations réalisées, on peut citer la conversion des devises, le formatage des dates et la suppression des erreurs ou des champs incomplets. L'objectif est de préparer les données pour qu'elles soient prêtes à être utilisées dans des analyses ou des visualisations.

Après transformation, les données sont sauvegardées dans un espace de stockage, ici un bucket MinIO, au format Parquet. Ce format a été choisi pour son efficacité dans la manipulation de gros volumes de données et sa compatibilité avec de nombreux outils Big Data. Cette étape garantit la pérennité et la facilité d'accès aux données pour des usages ultérieurs.

Enfin, une dernière étape optionnelle consiste à lire ces données transformées et stockées dans MinIO, toujours avec Spark, pour en vérifier l'intégrité. Cette vérification nous permet de nous assurer que toutes les étapes du traitement se sont déroulées correctement et que les données sont prêtes pour une exploitation future.

Préparation et Mise en Place de l'Environnement

Les outils essentiels :

Dans cette première étape, nous avons mis en place les outils requis pour le bon déroulement du projet, conformément aux consignes fournies. L'installation a été réalisée sous Windows, et chaque logiciel a été configuré pour garantir leur compatibilité et leur intégration. Voici les principaux outils et leurs versions :

- **Apache Spark 3.5.3** : Utilisé pour traiter et transformer les flux de données en temps réel. Spark est au cœur de l'architecture Big Data du projet.
- **Hadoop 3.3.6** : Nécessaire pour fournir les bibliothèques de gestion des fichiers distribués, notamment pour Spark.
- **JDK 8 (Java Development Kit)** : Pré-requis indispensable pour le fonctionnement de Spark et Hadoop.
- **Docker Desktop 4.23.0** : Utilisé pour déployer les conteneurs nécessaires, notamment pour Kafka et MinIO.
- **Conduktor 3.16.2** : Employé pour superviser les topics Kafka, visualiser les messages en temps réel et diagnostiquer les éventuels problèmes de flux.
- **Miniconda avec Python 3.9** : Permettant de créer un environnement Python isolé pour installer les dépendances nécessaires comme pyspark et kafka-python.
- **PyCharm 2024.3.1** : IDE de développement utilisé pour coder et exécuter les scripts Python, offrant un environnement de travail adapté et performant.

Chaque outil a été installé et configuré pour répondre aux besoins du projet, en suivant les prérequis techniques et les consignes du sujet.

Variables d'environnement :

Dans le cadre de la mise en place des outils, il a été nécessaire de configurer certaines variables d'environnement système sur notre machine Windows afin d'assurer leur bon fonctionnement. Parmi celles-ci, nous avons défini les variables suivantes :

- **SPARK_HOME** : Pointant vers C:\Spark\spark-3.5.3-bin-hadoop3, cette variable permet à Spark de localiser ses fichiers essentiels.
- **HADOOP_HOME** : Configurée sur le même chemin que Spark, C:\Spark\spark-3.5.3-bin-hadoop3, cette variable est utilisée par Spark pour accéder aux dépendances Hadoop nécessaires.
- **JAVA_HOME** : Configurée sur le chemin du JDK utilisé (C:\Zulu\zulu-21), cette variable est indispensable pour Spark et Hadoop, qui reposent sur Java pour fonctionner.

SPARK_HOME	C:\Spark\spark-3.5.3-bin-hadoop3\
HADOOP_HOME	C:\Spark\spark-3.5.3-bin-hadoop3\
JAVA_HOME	C:\Zulu\zulu-21

Ces variables ont également été ajoutées à la variable système PATH, incluant leurs sous-dossiers \bin, afin de permettre leur reconnaissance dans l'ensemble des outils et scripts utilisés. Par exemple, nous avons ajouté %SPARK_HOME%\bin pour permettre à Spark de fonctionner directement dans le terminal ou depuis l'IDE.

```
%SPARK_HOME%\bin
```

```
%JAVA_HOME%\bin
```

```
%HADOOP_HOME%\bin
```

Compatibilité Java :

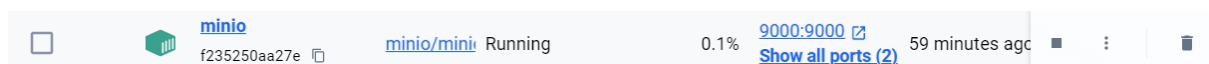
Un autre point important a été la version de Java installée. Spark et Hadoop nécessitent une version compatible avec leurs bibliothèques. Initialement, une version 21 du JDK était installée, ce qui a provoqué des erreurs d'exécution. Après analyse, nous avons installé le JDK 8 (via Zulu), qui correspond aux pré requis pour ces outils. Cela a immédiatement résolu les problèmes rencontrés lors des premiers tests.

Mise en place de l'environnement

Déploiement des conteneurs avec Docker

Pour faciliter le déploiement des différents services nécessaires au projet, nous avons utilisé Docker. Cet outil nous a permis de configurer rapidement des conteneurs pour les services tels que Kafka, Zookeeper, et MinIO. Le fichier **docker-compose.yml** fourni dans les ressources du projet a servi de base pour orchestrer le démarrage des conteneurs. Grâce à la commande **docker-compose up -d**, les services ont été correctement démarrés en arrière-plan.

L'utilisation de Docker a simplifié la gestion des dépendances et évité les conflits de configuration entre les outils. Nous avons pu vérifier que les conteneurs étaient bien en cours d'exécution à l'aide de la commande **docker ps**, qui liste les conteneurs actifs.



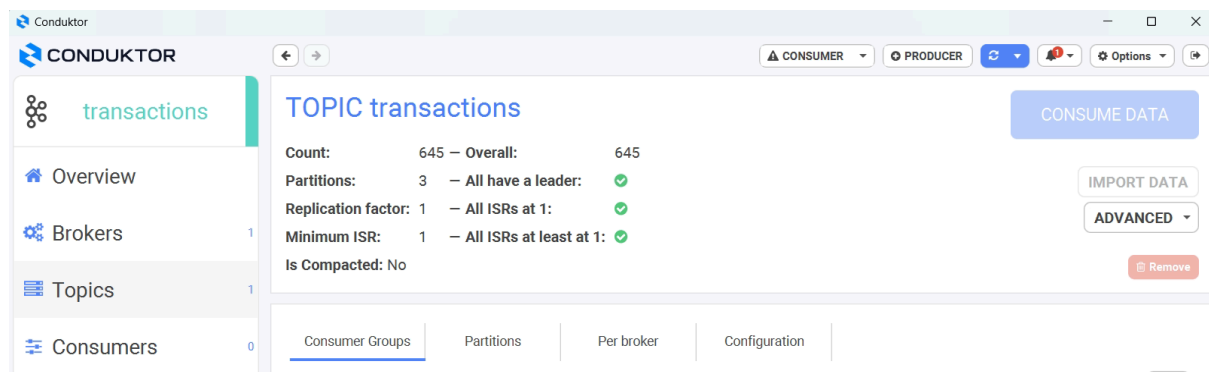
Développement des Scripts

1. Création des Topics Kafka et Envoi des Données (Producer Kafka)

La première étape de cette phase a consisté à configurer un topic Kafka via Conduktor et à développer un script pour simuler et envoyer des transactions fictives.

Création des Topics avec Conduktor

Avec Conduktor, un topic nommé **transactions** a été créé. Ce topic est essentiel pour le projet puisqu'il sert de point d'entrée aux données générées. Conduktor a permis une configuration simple et rapide grâce à son interface intuitive. Une fois le topic créé, il a été configuré pour recevoir et gérer en temps réel les messages envoyés par le producer Kafka. Le topic agit comme un espace tampon, facilitant la transmission fluide des données vers les étapes de traitement suivantes.



Développement du Producer Kafka

Le script Python **producer.py** a été conçu pour générer des transactions fictives, les convertir en JSON, et les publier sur le topic Kafka **transactions**. Les données simulées par le producer incluent divers champs tels que :

- **ID de transaction** : Identifiant unique pour chaque opération.
- **Type de transaction** : Par exemple, achat, transfert, remboursement.
- **Montant et devise** : Les montants en USD, qui seront ensuite convertis en EUR.
- **Date et lieu** : Informations temporelles et géographiques associées à la transaction.
- **Moyen de paiement** : Carte de crédit, espèces, ou virement.
- **Détails de la transaction** : Produit, quantité, prix unitaire.
- **Informations utilisateur** : Nom, adresse, email, et identifiant unique.

Grâce à Kafka, ces données générées ont été publiées sur le topic en temps réel, prêtes à être consommées par Spark Streaming dans la prochaine étape.

```
3', 'lieu': 'Rue Fontaine, Lyon', 'moyen_paiement': 'erreur', 'details': {'produit': 'Produit15', 'quantite': 6, 'prix_unitaire': 151.43}, 'utilisateur': {'id_utilisateur': 'User246', 'nom': 'Utilisateur581', 'adresse': '893 Rue Marceaux, Nantes', 'email': 'utilisateur896@example.com'}}
Transaction envoyée : {'id_transaction': '597b1e4-5a47-47cd-9061-76a5ecc4a4f6', 'type_transaction': 'remboursement', 'montant': 623.11, 'devise': 'USD', 'date': '2024-12-13T20:02:35.085759', 'lieu': 'Rue Saint-Michel, Marseille', 'moyen_paiement': 'carte_de_credit', 'details': {'produit': 'Produit33', 'quantite': 6, 'prix_unitaire': 122.32}, 'utilisateur': {'id_utilisateur': 'User904', 'nom': 'Utilisateur222', 'adresse': '454 Rue de la Grande Armée, Nice', 'email': 'utilisateur785@example.com'}}
Transaction envoyée : {'id_transaction': '2b76b241-d1d4-4f84-b3d3-629fc280e7fd', 'type_transaction': 'transfert', 'montant': 782.12, 'devise': 'USD', 'date': '2024-12-13T20:02:36.088499', 'lieu': 'Rue de Bretagne, Montpellier', 'moyen_paiement': 'especes', 'details': {'produit': 'Produit86', 'quantite': 2, 'prix_unitaire': 166.94}, 'utilisateur': {'id_utilisateur': 'User569', 'nom': 'Utilisateur231', 'adresse': '75 Rue Marceaux, Nantes', 'email': 'utilisateur993@example.com'}}
Arrêt du producteur Kafka.
PS C:\EPSI\M1\Atelier Big Data\tp_BigDATA\Python>
```


2. Lecture des Données avec Spark Streaming (Consumer)

Pour consommer les données publiées en temps réel sur le topic Kafka **transactions**, un script Python nommé **consumer.py** a été développé en utilisant Spark Streaming. Cette étape permet de lire les messages JSON du topic, de les structurer selon le schéma défini dans le producteur, et de préparer les données pour des traitements et analyses ultérieurs.

Fonctionnement du Script Consumer

Le script **consumer.py** s'appuie sur Spark Streaming pour :

1. **Écouter et lire les messages Kafka**

Les données JSON sont extraites du topic **transactions**, et chaque message est lu en tant que chaîne de caractères.

2. **Structuration des données**

Les messages JSON suivent le même schéma défini dans le producer, ce qui garantit la cohérence des champs.

3. **Validation et visualisation initiale**

Les données sont affichées directement dans la console pour valider que tous les champs sont correctement extraits et que la connexion au topic fonctionne comme attendu.

```
Consuming from Topic: transactions
2024-12-13 20:02:19 # (null)
{"id_transaction": "c3e014d0-e115-4462-b4a8-8f6f9f23e070", "type_transaction": "achat", "montant": 119.39, "devise": "USD", "date": "2024-12-13T20:02:19.943281", "lieu": "Rue Gustave Courbet, Toulouse", "moyen_paiement": "virement_bancaire", "detail...
2024-12-13 20:02:18 # (null)
{"id_transaction": "728495d0-e778-4f7c-bbf2-c07731f938c6", "type_transaction": "achat", "montant": 763.7, "devise": "USD", "date": "2024-12-13T20:02:18.931449", "lieu": "Rue Gambetta, Bordeaux", "moyen_paiement": "erreur", "details": {"produit": "Pro...
2024-12-13 20:02:17 # (null)
{"id_transaction": "7478fe3b-6a3b-4377-9512-cc6ada8577d1", "type_transaction": "transfert", "montant": 254.93, "devise": "USD", "date": "2024-12-13T20:02:17.923589", "lieu": "Rue du Faubourg Saint-Antoine, Saint-\u00c9tienne", "moyen_paiement": "vire...
2024-12-13 20:02:16 # (null)
{"id_transaction": "345a684c-aa4b-47da-9fc1-89fe8512bc22", "type_transaction": "transfert", "montant": 617.93, "devise": "USD", "date": "2024-12-13T20:02:16.919405", "lieu": "None, Le Havre", "moyen_paiement": "especes", "details": {"produit": "Produ...
2024-12-13 20:02:15 # (null)
{"id_transaction": "164fbf0a-9835-4a25-b50b-16491781ce0b", "type_transaction": "transfert", "montant": 201.18, "devise": "USD", "date": "2024-12-13T20:02:15.907220", "lieu": "Rue de la R\u00e9publique, Marseille", "moyen_paiement": "virement_bancaire...
2024-12-13 20:02:14 # (null)
{"id_transaction": "1e5c57b9-e50c-4c2a-a4d4-e63410018706", "type_transaction": "remboursement", "montant": 955.91, "devise": "USD", "date": "2024-12-13T20:02:14.892633", "lieu": "Rue de la Villette, Toulon", "moyen_paiement": "carte_de_credit", "deta...

> SHOW PARTITIONS 6 records consumed (2,8 kB, 7s) CLEAR STOP!
```

3. Transformations des Données

Après avoir consommé les données en temps réel depuis le topic Kafka **transactions**, plusieurs transformations ont été appliquées dans le script **consumer.py** pour garantir leur qualité, leur cohérence, et leur pertinence pour les étapes d'analyse et de sauvegarde.

Étapes de Transformation

1. **Conversion des montants en EUR** : Les montants, initialement exprimés en USD, ont été convertis en EUR en appliquant un taux de conversion fixe de **1 USD = 0.85 EUR**. Une nouvelle colonne **montant_eur** a été ajoutée pour conserver les valeurs converties.
2. **Ajout du fuseau horaire** : Une colonne **timezone** a été intégrée aux données, définissant le fuseau horaire **UTC** pour toutes les transactions. Cette normalisation facilite les analyses temporelles à l'échelle globale.
3. **Nettoyage des données : Exclusion des transactions erronées** : Les transactions où le moyen de paiement est égal à **"erreur"** ont été filtrées et **Suppression des valeurs critiques manquantes** : Les transactions sans **adresse utilisateur** ont également été exclues pour garantir des données complètes.
4. **Formatage des dates** : Les dates, initialement au format texte, ont été transformées en type **timestamp** pour les rendre manipulables dans des calculs temporels ou des visualisations. Le format utilisé est **yyyy-MM-dd'T'HH:mm:ss**, conforme aux standards ISO.

Ces transformations assurent que les données sont propres, cohérentes et prêtes à être exploitées pour des analyses ou des visualisations avancées. La conversion des montants garantit une homogénéité dans l'analyse, tandis que l'ajout du fuseau horaire uniformise la gestion des informations temporelles. Enfin, le nettoyage des données, par l'exclusion des anomalies et des valeurs manquantes, permet d'éviter les biais dans les résultats et d'assurer leur fiabilité.

```
# Définir le schéma des données JSON envoyées par le producteur
schema = StructType([
    StructField(name: "id_transaction", StringType(), nullable: True),
    StructField(name: "type_transaction", StringType(), nullable: True),
    StructField(name: "montant", FloatType(), nullable: True),
    StructField(name: "devise", StringType(), nullable: True),
    StructField(name: "date", StringType(), nullable: True),
    StructField(name: "lieu", StringType(), nullable: True),
    StructField(name: "moyen_paiement", StringType(), nullable: True),
    StructField(name: "details", StructType([
        StructField(name: "produit", StringType(), nullable: True),
        StructField(name: "quantite", FloatType(), nullable: True),
        StructField(name: "prix_unitaire", FloatType(), nullable: True)
    ]), nullable: True),
    StructField(name: "utilisateur", StructType([
        StructField(name: "id_utilisateur", StringType(), nullable: True),
        StructField(name: "nom", StringType(), nullable: True),
        StructField(name: "adresse", StringType(), nullable: True),
        StructField(name: "email", StringType(), nullable: True)
    ]), nullable: True)
])
```

```
# Appliquer les transformations demandées
transformed_df = transactions_df.withColumn(colName: "montant_eur", col("montant") * lit(0.85)) \
    .withColumn(colName: "timezone", lit("UTC")) \
    .withColumn(colName: "date", to_timestamp(col("date"), "yyyy-MM-dd'T'HH:mm:ss")) \
    .filter(col("moyen_paiement") != "erreur") \
    .filter(col("utilisateur.adresse").isNotNull())
```












4. Sauvegarde des Données dans MinIO

Une fois les transformations terminées, les données ont été sauvegardées dans un bucket MinIO nommé **bigdata** au format Parquet. Ce format a été sélectionné pour ses performances optimales en termes de compression, de gestion des données volumineuses, et pour sa compatibilité avec les systèmes Big Data.

Dans le script **consumer.py**, l'écriture des données dans MinIO a été réalisée en configurant les paramètres d'accès, notamment les clés d'identification, l'URL du point de terminaison, et le bucket cible. Les fichiers Parquet générés permettent de stocker efficacement les données structurées et transformées, tout en offrant une facilité d'intégration dans des étapes d'analyse ultérieures.

La réussite de cette étape a été vérifiée en consultant l'interface de MinIO. Nous avons confirmé la présence des fichiers Parquet dans le bucket **bigdata** et validé qu'ils contenaient les colonnes attendues après transformation.

```
# Écrire les données transformées dans MinIO au format Parquet
query = transformed_df.writeStream \
    .outputMode("append") \
    .format("parquet") \
    .option( key: "path", value: "s3a://bigdata/transactions") \
    .option( key: "checkpointLocation", value: "s3a://bigdata/checkpoints/transactions") \
    .start()
```

	bigdata	Created on: Fri, Dec 13 2024 12:33:02 (GMT+1)	Access: PUBLIC	91.0 B - 5 Objects	Rewind ↺	Refresh ↻	Upload ⬆
bigdata / transactions							
Create new path ↗							
<input type="checkbox"/>	Name	Last Modified	Size				
<input type="checkbox"/>	 _spark_metadata		-				
<input type="checkbox"/>	 part-00000-05037be3-1856-475f-8223-76bb24a2eae2-c000.snap...	Today, 20:34	5.3 KiB				
<input type="checkbox"/>	 part-00000-09b20d00-1812-4b36-b688-95fcc5e25e62-c000.snap...	Today, 20:34	5.3 KiB				
<input type="checkbox"/>	 part-00000-0be30687-fc10-4b10-8192-bd3188551f38-c000.snapp...	Today, 20:34	5.3 KiB				
<input type="checkbox"/>	 part-00000-0c129aca-af7c-4537-a858-c264915288a2-c000.snapp...	Today, 20:34	5.3 KiB				
<input type="checkbox"/>	 part-00000-0ca7b95a-df31-48d9-889f-10291df03e80-c000.sna...	Today, 20:34	5.4 KiB				
<input type="checkbox"/>	 part-00000-15b2e3c3-2e16-4eae-b92f-be73b27aa18d-c000.snapp...	Today, 20:34	5.2 KiB				
<input type="checkbox"/>	 part-00000-17cb87b2-efa0-4fec-a498-ceb5894b1d73-c000.snapp...	Today, 20:34	5.3 KiB				
<input type="checkbox"/>	 part-00000-25b40194-8d36-4daa-ae26-503dce675443-c000.sna...	Today, 20:34	5.2 KiB				
<input type="checkbox"/>	 part-00000-2f553cd2-c7c2-4d17-9e50-1339fa47ccec-c000.snapp...	Today, 20:34	5.4 KiB				

Name:

part-00000-05037be3-1856-475f-
8223-76bb24a2eae2-
c000.snappy.parquet

5. Lecture et Validation des Données (optionnel)

Une fois les données enregistrées dans MinIO au format Parquet, une étape de validation a été réalisée à l'aide de **Spark** pour s'assurer de l'intégrité et de la conformité des transformations effectuées.

Lecture des fichiers Parquet depuis MinIO

Grâce à **Spark**, les fichiers stockés dans MinIO ont été chargés en utilisant le connecteur **S3A**. Cette lecture à l'aide du script **reader.py** permet de vérifier que les données sont bien accessibles et que l'écriture s'est correctement déroulée.

Vérification des données transformées

- **Validation des transformations appliquées**, notamment :
 - La conversion des montants en **EUR**.
 - Le **formatage correct des dates**.
 - L'exclusion des transactions erronées.
 - La présence du fuseau horaire **UTC**.

[id_transaction]	[type_transaction]	[montant]	[devise]	[date]	[lieu]	[moyen_paiement]	[details]
[d1989df0-186a-461a-b5e5-e349e26d0e6a]	[transfert]	[986.96]	[USD]	[2024-12-13 20:35:00]	[Rue du Faubourg Saint-Antoine, Paris]	[carte_de_credit]	[{Produit33, 8.0, 140.96}]
[bad3c226-e76f-436e-b742-595cb2156307]	[achat]	[413.31]	[USD]	[2024-12-13 20:34:51]	[Rue du Faubourg Saint-Antoine, Le Havre]	[carte_de_credit]	[{Produit37, 5.0, 47.75}]
[291e90ad-35ac-4aaa-880f-0f382611b416]	[remboursement]	[916.86]	[USD]	[2024-12-13 20:34:22]	[Rue de Luxembourg, Saint-Étienne]	[virement_bancaire]	[{Produit82, 7.0, 109.98}]
[96f3887b-1602-4e4f-a313-4c1597825996]	[achat]	[242.99]	[USD]	[2024-12-13 20:34:33]	[Rue Gustave Courbet , Toulouse]	[virement_bancaire]	[{Produit92, 4.0, 95.31}]
[f18d8118-650e-4ea4-aaa2-6a0f663a6a6b]	[achat]	[889.28]	[USD]	[2024-12-13 20:34:44]	[Rue de la République, Saint-Étienne]	[lespeces]	[{Produit39, 1.0, 158.41}]
[413ff73b-eac6-4699-b89e-8955649311fb]	[achat]	[435.74]	[USD]	[2024-12-13 20:34:41]	[Rue de Luxembourg, Strasbourg]	[virement_bancaire]	[{Produit4, 6.0, 141.84}]
[d114492c-ba69-44a9-b1a5-0b273193fdef]	[remboursement]	[54.15]	[USD]	[2024-12-13 20:34:03]	[Rue Gustave Courbet , Reims]	[carte_de_credit]	[{Produit48, 5.0, 29.19}]
[9f21c564-9451-467a-88b3-0d08dc325c2a]	[achat]	[938.03]	[USD]	[2024-12-13 20:34:59]	[Rue de la Grande Armée, Toulon]	[lespeces]	[{Produit65, 1.0, 120.12}]
[a0dc94a6-7343-406b-af36-c14388b1b847]	[transfert]	[903.42]	[USD]	[2024-12-13 20:34:55]	[Rue Zinedine Zidane, Strasbourg]	[carte_de_credit]	[{Produit82, 6.0, 115.55}]
[47a82f46-e13f-4a2b-b639-cc671060fea2]	[remboursement]	[57.2]	[USD]	[2024-12-13 20:34:05]	[Rue de la République, Rennes]	[virement_bancaire]	[{Produit91, 4.0, 29.68}]
[0fb84df9-fbbf-45b6-bab4-cb5f367606b7]	[achat]	[437.62]	[USD]	[2024-12-13 20:35:05]	[Rue de la Villette, Strasbourg]	[lespeces]	[{Produit97, 4.0, 32.29}]
[fe453b49-c400-40e9-ac4b-f6f302a0ff8f]	[remboursement]	[374.3]	[USD]	[2024-12-13 20:34:01]	[Rue de Bretagne, Paris]	[carte_de_credit]	[{Produit64, 1.0, 162.62}]
[e758bc92-0961-404d-86f0-d01b5fbcc1d1]	[transfert]	[888.07]	[USD]	[2024-12-13 20:34:46]	[Rue de Luxembourg, Montpellier]	[lespeces]	[{Produit47, 2.0, 83.59}]
[1aad0b4e-e30e-42d1-9ec8-ac71e1266c5e]	[remboursement]	[175.88]	[USD]	[2024-12-13 20:34:12]	[Rue Fontaine, Paris]	[virement_bancaire]	[{Produit2, 3.0, 175.07}]
[f2fd6941-ad81-4d9c-9f75-8ed3ad41a020]	[achat]	[779.75]	[USD]	[2024-12-13 20:34:17]	[Rue de la Villette, Rennes]	[carte_de_credit]	[{Produit9, 2.0, 141.75}]
[3521b9ee-a3a4-4aa7-8258-9da094951107]	[transfert]	[869.35]	[USD]	[2024-12-13 20:34:57]	[Rue Gustave Courbet , Toulon]	[virement_bancaire]	[{Produit7, 7.0, 32.58}]
[f27bd8bd-8910-41d4-9ca2-1425ea718b02]	[remboursement]	[202.68]	[USD]	[2024-12-13 20:34:30]	[Rue de Luxembourg, Lille]	[virement_bancaire]	[{Produit82, 2.0, 102.95}]
[32844b78-c7ae-4208-9a24-e79469a647bc]	[achat]	[410.8]	[USD]	[2024-12-13 20:34:14]	[Rue Manceaux, Nice]	[virement_bancaire]	[{Produit30, 5.0, 19.43}]
[d753ca82-8a2f-4441-94c7-967986be88a8]	[achat]	[800.72]	[USD]	[2024-12-13 20:34:04]	[Rue de la Grande Armée, None]	[lespeces]	[{Produit13, 3.0, 62.24}]
[8fd18de8-2575-4e05-8a10-bd9254626be5]	[transfert]	[565.18]	[USD]	[2024-12-13 20:34:53]	[Rue de Bretagne, Toulon]	[virement_bancaire]	[{Produit86, 3.0, 149.45}]
[4bdc7340-925d-4eee-a60f-9b211c3dca28]	[transfert]	[14.01]	[USD]	[2024-12-13 20:33:58]	[Rue Zinedine Zidane, Montpellier]	[lespeces]	[{Produit20, 8.0, 52.0}]

utilisateur	montant_eur	timezone
{User253, Utilisateur477, 861 Rue de Luxembourg, Saint-Étienne, utilisateur909@example.com}	838.9160186767577	UTC
{User575, Utilisateur95, 964 Rue Saint-Michel, Saint-Étienne, utilisateur710@example.com}	351.31349792480466	UTC
{User698, Utilisateur193, 739 Rue Zinedine Zidane, Paris, utilisateur694@example.com}	779.3309875488281	UTC
{User270, Utilisateur586, 202 Rue Gustave Courbet, Saint-Étienne, utilisateur443@example.com}	206.54150466918944	UTC
{User288, Utilisateur5, 134 Rue du Faubourg Saint-Antoine, Nantes, utilisateur472@example.com}	755.8880249023438	UTC
{User201, Utilisateur984, 710 Rue Saint-Michel, Saint-Étienne, utilisateur416@example.com}	370.3789916992187	UTC
{User615, Utilisateur853, 411 Rue de Bretagne, Strasbourg, utilisateur439@example.com}	46.027501296997066	UTC
{User300, Utilisateur311, 737 Rue du Faubourg Saint-Antoine, Nantes, utilisateur675@example.com}	797.3255249023438	UTC
{User564, Utilisateur182, 440 Rue Gambetta, Montpellier, utilisateur438@example.com}	767.9069854736327	UTC
{User228, Utilisateur592, 387 Rue Saint-Michel, Reims, utilisateur7@example.com}	48.62000064849853	UTC
{User375, Utilisateur866, 136 Rue Gustave Courbet, Saint-Étienne, utilisateur66@example.com}	371.97699584960935	UTC
{User675, Utilisateur858, 455 Rue Saint-Michel, Bordeaux, utilisateur838@example.com}	318.1549896240234	UTC
{User790, Utilisateur371, 723 Rue Zinedine Zidane, Le Havre, utilisateur387@example.com}	754.859506225586	UTC
{User182, Utilisateur936, 34 Rue de Luxembourg, Le Havre, utilisateur866@example.com}	149.4980041503906	UTC
{User54, Utilisateur539, 208 Rue de la Grande Armée, Paris, utilisateur123@example.com}	662.7875	UTC
{User101, Utilisateur72, 625 Rue de la Pompe, Paris, utilisateur492@example.com}	738.9474792480469	UTC
{User350, Utilisateur514, 30 Rue de Bretagne, None, utilisateur359@example.com}	172.27799377441406	UTC
{User848, Utilisateur568, 418 Rue de la République, Marseille, utilisateur758@example.com}	349.17998962402345	UTC
{User350, Utilisateur622, 341 Rue de Luxembourg, Montpellier, utilisateur427@example.com}	680.6119750976562	UTC
{User887, Utilisateur492, 864 Rue Gambetta, Marseille, utilisateur329@example.com}	480.402993774414	UTC
{User783, Utilisateur695, 984 Rue de Luxembourg, Lyon, utilisateur162@example.com}	11.90850019454956	UTC

Conclusion

Ce projet a permis de concevoir une chaîne complète de traitement de données en temps réel, de leur génération jusqu'à leur stockage et validation. Grâce à **Kafka**, **Spark Streaming** et **MinIO**, nous avons pu manipuler des flux de transactions, appliquer des transformations essentielles et sauvegarder les données dans un format optimisé.

Ce travail nous a permis d'approfondir l'utilisation de ces outils et de mieux comprendre les enjeux du traitement de données en temps réel. Malgré quelques défis techniques liés aux versions des outils et à la configuration des environnements, les résultats obtenus confirment la bonne ingestion, transformation et exploitation des données.