

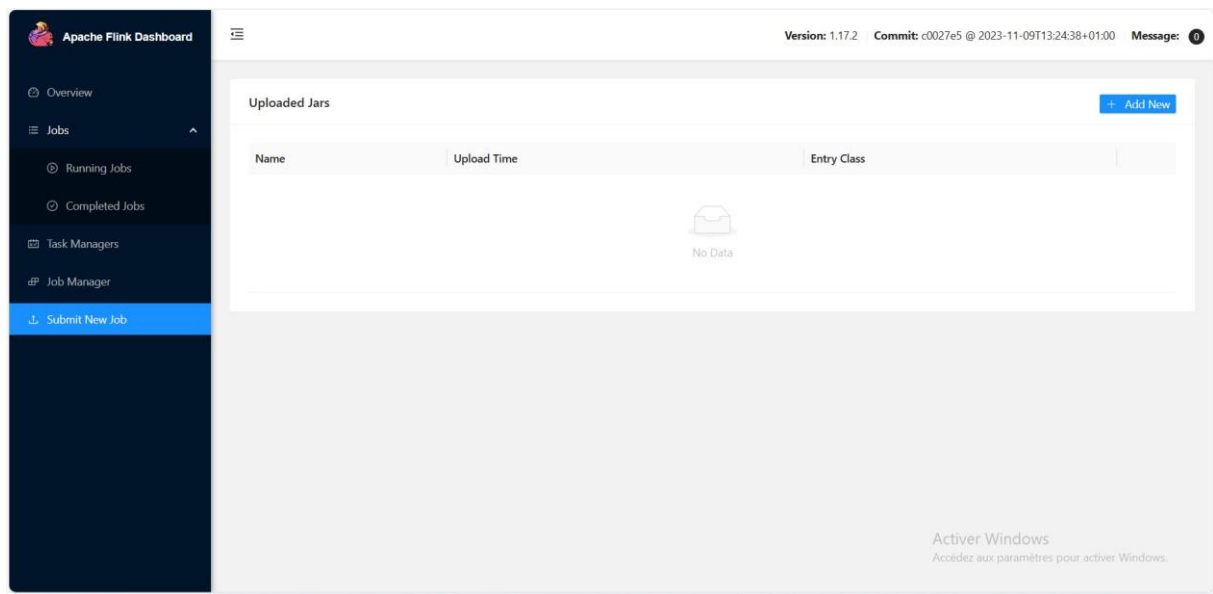
Rendu – Streaming Data Archi

Informations générales

- Réalisateurs : Bachiri Imane ,Babacar Sarr, Bouyad Zineb
- Sujet : Ingestion, scalabilité et résilience des données avec Kafka, Flink, Docker, Conduktor

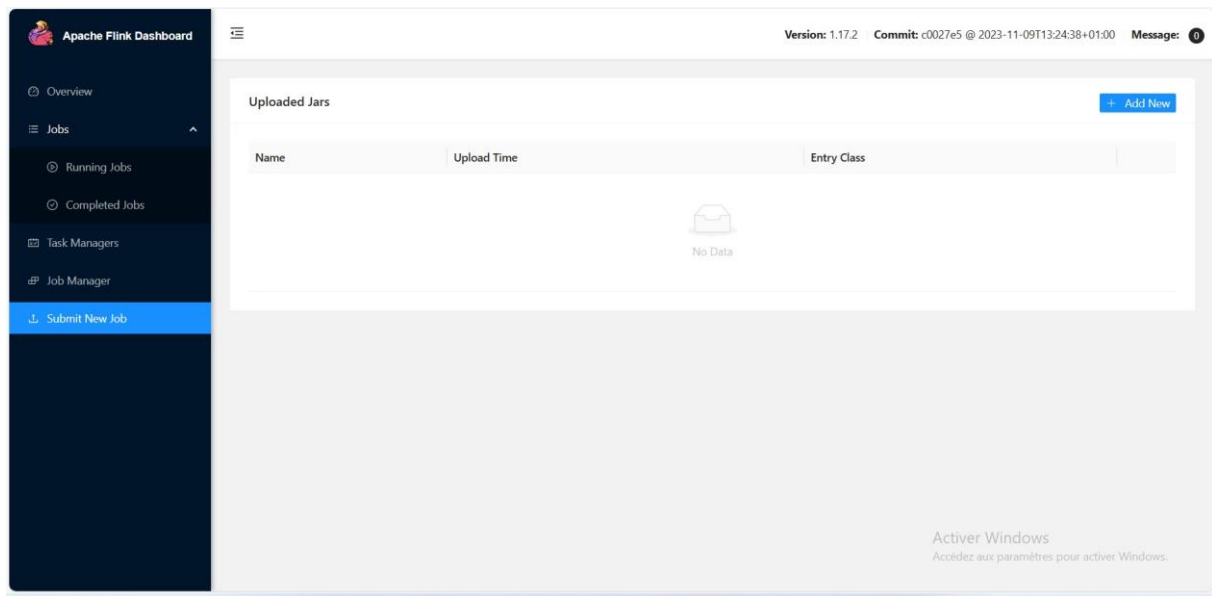
Captures de l'exécution

Capture 1 : Ajout d'un job Flink



Explication : Dans cette étape, nous avons ajouté un job Flink via l'interface Conduktor. Ce job est responsable du traitement des données en streaming en temps réel. Il va consommer les données envoyées par Kafka et effectuer les transformations nécessaires en fonction du traitement défini.

Capture 2 : Ajout du fichier .jar



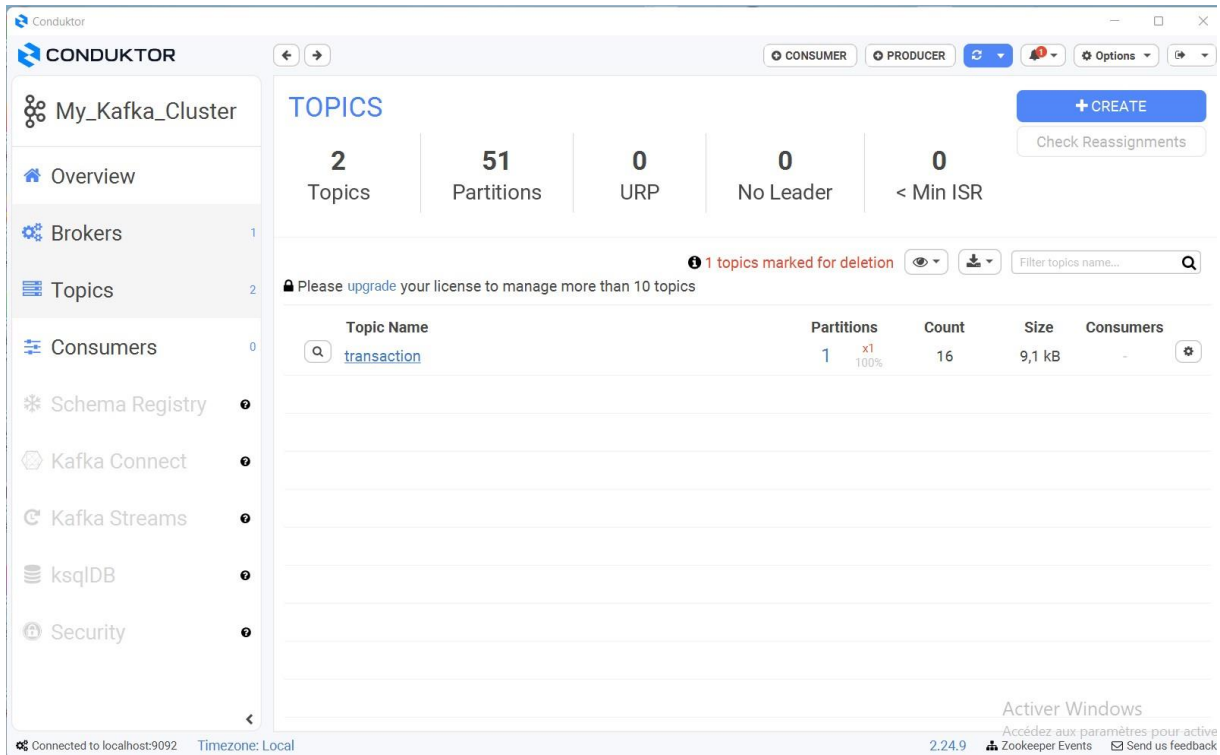
Explication : Ici, nous avons téléchargé et ajouté le fichier '.jar' contenant le code de notre job Flink. Ce fichier est exécuté par le cluster Flink pour effectuer les traitements définis dans le code Java.

Capture 3 : Consommation des données en temps réel avec Conduktor



Explication : Cette capture montre la consommation des données en temps réel avec Conduktor. Nous avons configuré un consommateur Kafka qui lit les messages en continu envoyés par Kafka, ce qui nous permet de suivre l'état de l'ingestion en temps réel.

Capture 4 : Création d'un cluster Kafka avec Conduktor



Explication : Ici, nous avons créé un cluster Kafka via l'interface de Conduktor. Ce cluster est utilisé pour gérer les topics Kafka et permettre l'ingestion et la consommation de messages en continu. Cette étape est cruciale pour assurer que les données circulent correctement entre les différentes étapes de l'architecture.

Capture 5 : Démarrage du cluster Flink

```
openjdk 21.0.6 2025-01-21
OpenJDK Runtime Environment (build 21.0.6+7-Ubuntu-124.04.1)
OpenJDK 64-Bit Server VM (build 21.0.6+7-Ubuntu-124.04.1, mixed mode, sharing)
babacar@BabacarSARR:/mnt$ cd /mnt/c/Users/Babacar/Downloads/flink-2.0.0-bin-scala_2.12/flink-2.0.0
in/start-cluster.shbabacar@BabacarSARR:/mnt/c/Users/Babacar/Downloads/flink-2.0.0-bin-scala_2.12/flink-2.0.0$
babacar@BabacarSARR:/mnt/c/Users/Babacar/Downloads/flink-2.0.0-bin-scala_2.12/flink-2.0.0$ ls
LICENSE NOTICE README.txt bin conf examples lib licenses log opt plugins
babacar@BabacarSARR:/mnt/c/Users/Babacar/Downloads/flink-2.0.0-bin-scala_2.12/flink-2.0.0$ cd bin
babacar@BabacarSARR:/mnt/c/Users/Babacar/Downloads/flink-2.0.0-bin-scala_2.12/flink-2.0.0/bin$ start-cluster.sh
start-cluster.sh: command not found
babacar@BabacarSARR:/mnt/c/Users/Babacar/Downloads/flink-2.0.0-bin-scala_2.12/flink-2.0.0/bin$ ./start-cluster.sh
Starting cluster.
Starting standalone session daemon on host BabacarSARR.
Starting taskexecutor daemon on host BabacarSARR.
babacar@BabacarSARR:/mnt/c/Users/Babacar/Downloads/flink-2.0.0-bin-scala_2.12/flink-2.0.0/bin$
```

Explication : Nous avons démarré un cluster Flink pour exécuter nos jobs de traitement de données. Ce cluster Flink est configuré pour lire les messages Kafka et effectuer des transformations en fonction de notre logique métier.

Capture 6 : Dockerisation de Flink (1)

<input type="checkbox"/>	python	-	-	-	37.92%	2 hours ago			
<input type="checkbox"/>	flink_taskmanag	ecad0b6d71c4	flink:1.17		36.48%	2 hours ago			
<input type="checkbox"/>	flink_jobmanag	ff26804de1bf	flink:1.17	8081:8081	1.44%	2 hours ago			

Explication : Nous avons dockerisé Flink pour pouvoir l'exécuter dans un environnement isolé et portable. Cela permet de s'assurer que l'environnement de production est exactement le même que l'environnement de développement, assurant ainsi la stabilité du système.

Capture 8 : Dockerisation de Kafka

<input type="checkbox"/>	python	-	-	-	37.92%	2 hours ago			
<input type="checkbox"/>	flink_taskmanag	ecad0b6d71c4	flink:1.17		36.48%	2 hours ago			
<input type="checkbox"/>	flink_jobmanag	ff26804de1bf	flink:1.17	8081:8081	1.44%	2 hours ago			

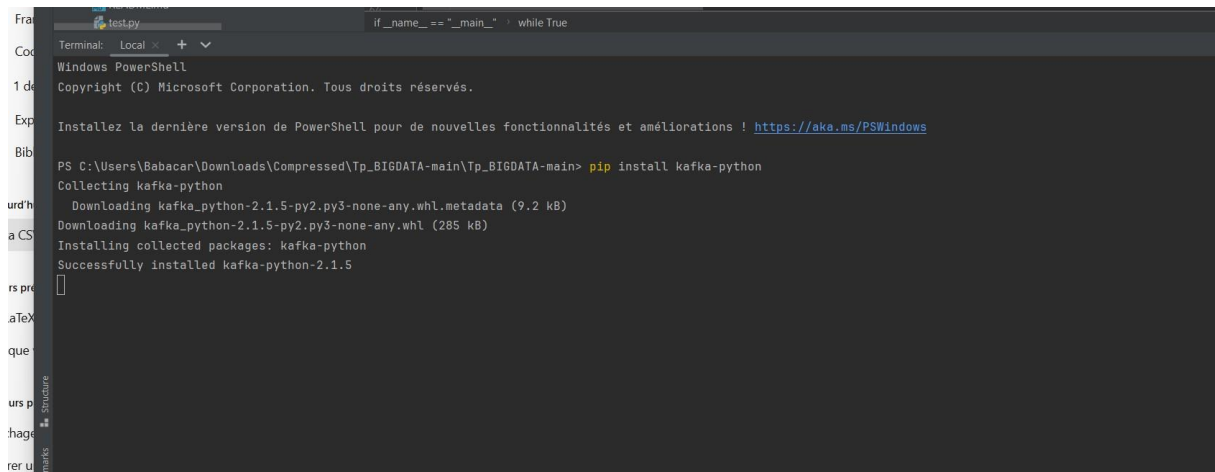
Explication : De la même manière, Kafka a été dockerisé pour faciliter sa gestion et son déploiement sur différentes machines ou environnements. Cela garantit une configuration cohérente et simplifie le **processus d'intégration continue et de déploiement**.

Capture 9 : Envoi des messages avec Kafka

```
Message envoyé : {"id_transaction": "c1a8f988-3a4e-4f6a-9e14-1f0fbd86e403", "type_transaction": "achat", "montant": 250.75, "devise": "USD", "date": "2025-04-17T10:30:00", "lieu": "Paris, Rue de Paris", "moyen_paiement": "carte_de_credit", "details": {"produit": "Ordinateur", "quantite": 1, "prixUnitaire": 250.75}, "utilisateur": {"idUtilisateur": "User001", "nom": "Alice Dubois", "adresse": "15 Rue de Lyon, Paris", "email": "alice@example.com"}}
Message envoyé : {"id_transaction": "eb2a59df-a06b-4635-80d6-9b37a1a70be4", "type_transaction": "transfert", "montant": 600.0, "devise": "USD", "date": "2025-04-17T11:00:00", "lieu": "Lyon, Rue de la R\u00e9publique", "moyen_paiement": "virement_bancaire", "details": {"produit": "Transfert bancaire", "quantite": 1, "prixUnitaire": 600.0}, "utilisateur": {"idUtilisateur": "User002", "nom": "Bruno Lefevre", "adresse": "42 Avenue du Rh\u00f4ne, Lyon", "email": "bruno@example.com"}}
Message envoyé : {"id_transaction": "ff91c1d7-2c92-4c4b-9a4f-f6f2329e55aa", "type_transaction": "remboursement", "montant": 75.2, "devise": "USD", "date": "2025-04-17T12:15:00", "lieu": "Marseille, Rue Saint-Michel", "moyen_paiement": "especes", "details": {"produit": "Livre", "quantite": 2, "prixUnitaire": 37.6}, "utilisateur": {"idUtilisateur": "User003", "nom": "Claire Martin", "adresse": "88 Boulevard Baille, Marseille", "email": "claire@example.com"}}
Message envoyé : {"id_transaction": "c1a8f988-3a4e-4f6a-9e14-1f0fbd86e403", "type_transaction": "achat", "montant": 250.75, "devise": "USD", "date": "2025-04-17T10:30:00", "lieu": "Paris, Rue de Paris", "moyen_paiement": "carte_de_credit", "details": {"produit": "Ordinateur", "quantite": 1, "prixUnitaire": 250.75}, "utilisateur": {"idUtilisateur": "User001", "nom": "Alice Dubois", "adresse": "15 Rue de Lyon, Paris", "email": "alice@example.com"}}
Message envoyé : {"id_transaction": "eb2a59df-a06b-4635-80d6-9b37a1a70be4", "type_transaction": "transfert", "montant": 600.0, "devise": "USD", "date": "2025-04-17T11:00:00", "lieu": "Lyon, Rue de la R\u00e9publique", "moyen_paiement": "virement_bancaire", "details": {"produit": "Transfert bancaire", "quantite": 1, "prixUnitaire": 600.0}, "utilisateur": {"idUtilisateur": "User002", "nom": "Bruno Lefevre", "adresse": "42 Avenue du Rh\u00f4ne, Lyon", "email": "bruno@example.com"}}
```

Explication : Cette capture montre l'envoi de messages vers Kafka, ce qui représente l'ingestion de données. Kafka est utilisé ici pour diffuser les données à plusieurs consommateurs (par exemple, Flink) en temps réel.

Capture 10 : Installation de la dépendance Kafka sur Python



```
test.py if __name__ == "__main__": while True
Terminal: Local + v
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Installez la dernière version de PowerShell pour de nouvelles fonctionnalités et améliorations ! https://aka.ms/PSWindows

PS C:\Users\Babacar\Downloads\Compressed\Tp_BIGDATA-main\Tp_BIGDATA-main> pip install kafka-python
Collecting kafka-python
  Downloading kafka_python-2.1.5-py2.py3-none-any.whl.metadata (9.2 kB)
  Downloading kafka_python-2.1.5-py2.py3-none-any.whl (285 kB)
Installing collected packages: kafka-python
Successfully installed kafka-python-2.1.5
```

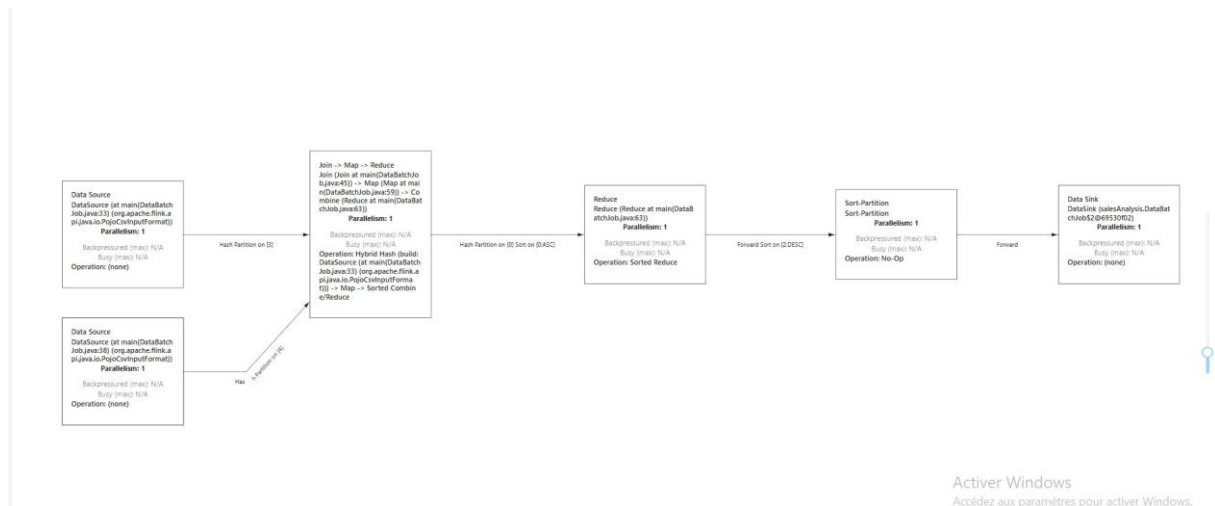
Explication : Nous avons installé la dépendance Kafka sur Python via pip pour permettre à notre application Python de consommer des données à partir de Kafka. Cette étape est nécessaire pour interagir avec le cluster Kafka depuis un programme Python et traiter les messages.

Capture 11 : Job complet

Job Name	Start Time	Duration	End Time	Tasks
Sales Analysis	2025-04-18 00:51:55	244ms	2025-04-18 00:51:55	

Explication : La capture montre un job complet exécuté sur Flink, où les données sont ingérées, traitées et envoyées vers un autre système ou stockage. Ce job est responsable de toutes les étapes de transformation des données.

Capture 12 : Plan de pipeline sur Flink



Explication : Cette capture montre le plan de pipeline dans Flink. Le pipeline représente les différentes étapes de traitement des données, allant de l'ingestion à la sortie, avec des opérations de transformation et de filtrage effectuées sur les données en streaming.

Capture 12 :

```

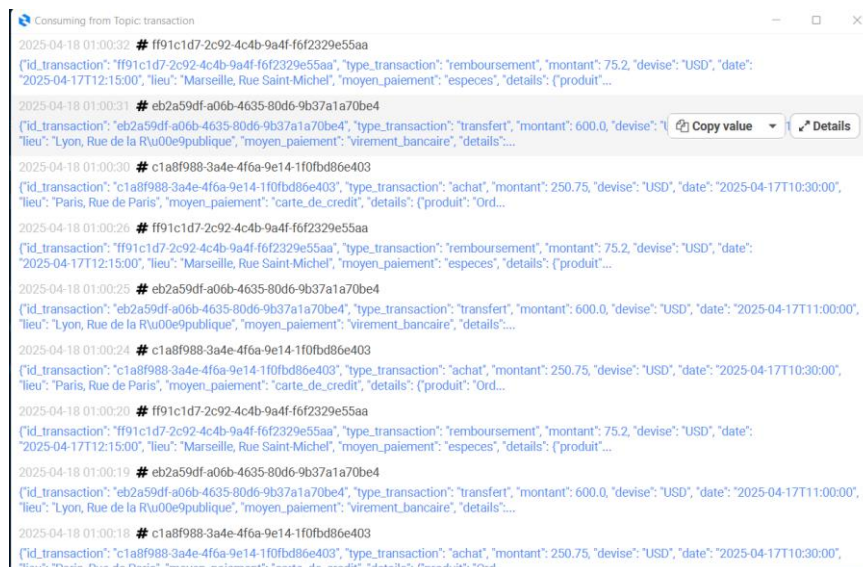
1  from confluent_kafka import Consumer, KafkaError
2  import json
3  import os
4  import csv
5
6  def write_to_csv(data, filename):
7      # Liste des colonnes attendues
8      fieldnames = [
9          "id_transaction", "type_transaction", "montant", "devise",
10         "date", "lieu", "moyen_paiement", "details", "utilisateur"
11     ]
12
13     # Vérifie si le fichier existe pour savoir si on écrit l'en-tête
14     file_exists = os.path.isfile(filename)
15
16     with open(filename, mode='a', newline='', encoding='utf-8') as csvfile:
17         writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
18
19         # Écrire l'en-tête si le fichier est nouveau
20         if not file_exists:
21             writer.writeheader()
22
23         # Convertir les champs complexes (dictionnaires) en chaînes JSON
24         data["details"] = json.dumps(data["details"], ensure_ascii=False)
25         data["utilisateur"] = json.dumps(data["utilisateur"], ensure_ascii=False)

```

Explication : Cette capture montre une fonction Python permettant d'écrire des données dans un fichier CSV. Le script est conçu pour enregistrer les données consommées depuis Kafka, en s'assurant que les champs complexes comme les dictionnaires sont convertis en chaînes JSON. Il vérifie également si le

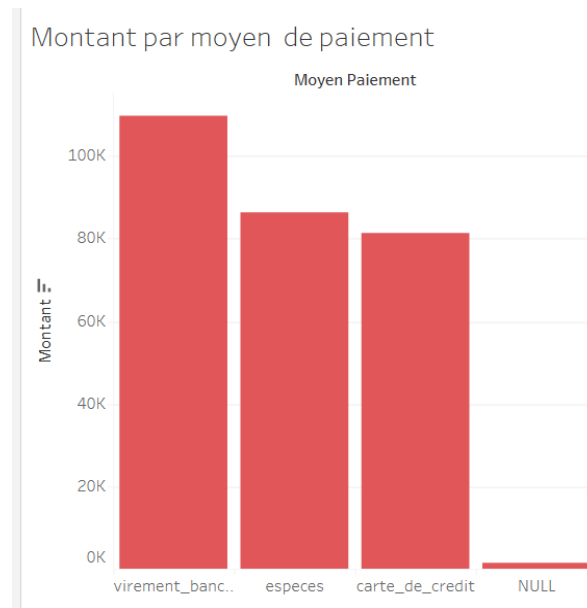
fichier existe afin d'écrire l'en-tête uniquement une fois. Ce processus permet de stocker des flux de données structurées de manière efficace pour une utilisation ultérieure.

Capture 13:



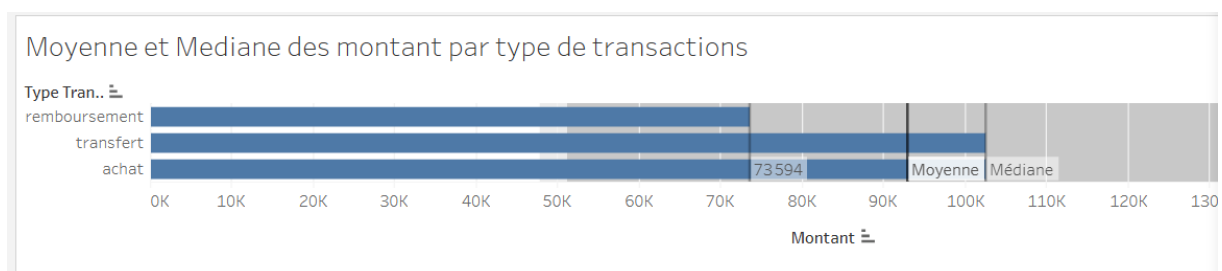
Cette capture montre la consommation de messages en temps réel depuis le topic Kafka nommé *transaction*. Chaque message correspond à une transaction, un identifiant unique et un contenu structuré au format JSON. Cela permet de visualiser en continu les données entrantes issues d'un flux Kafka, pour des traitements ou analyses ultérieurs.

Les différents graphiques sur Tableau



1. Montant par moyen de paiement

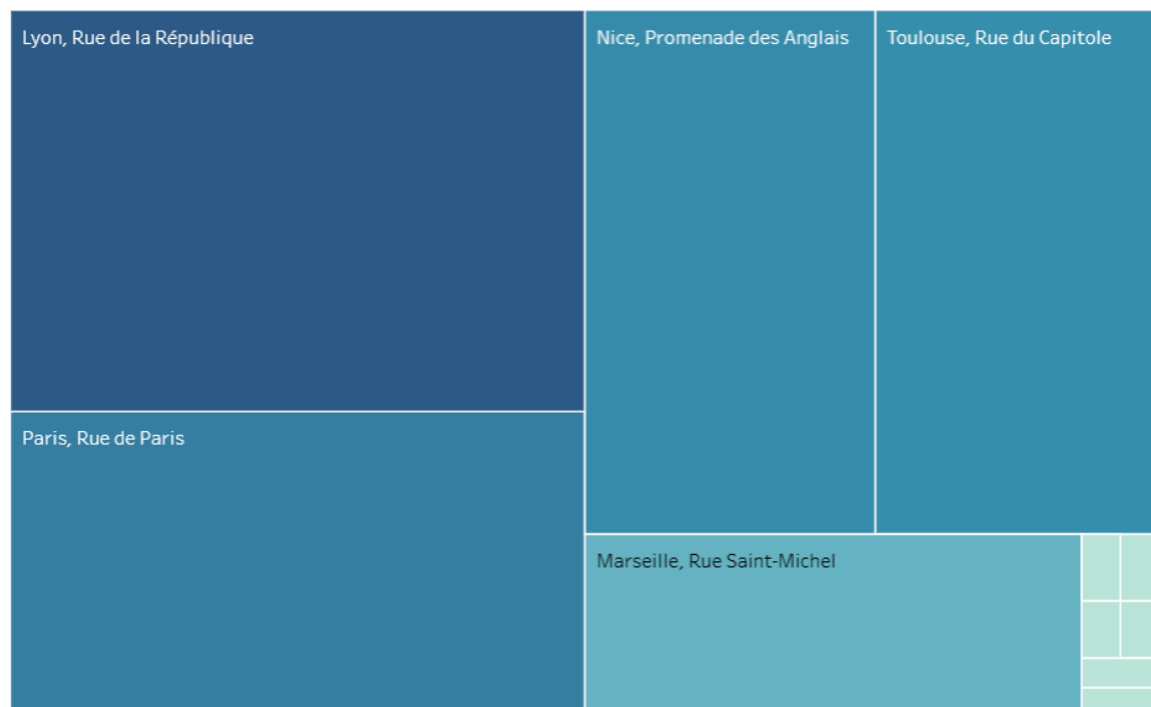
- Répartition des montants totaux par moyen de paiement (virement, espèces, carte de crédit).



Moyenne et Médiane des montants par type de transaction

- Compare la moyenne (73594) et la médiane (non précisée) pour différents types de transactions (remboursement, transfert, achat).

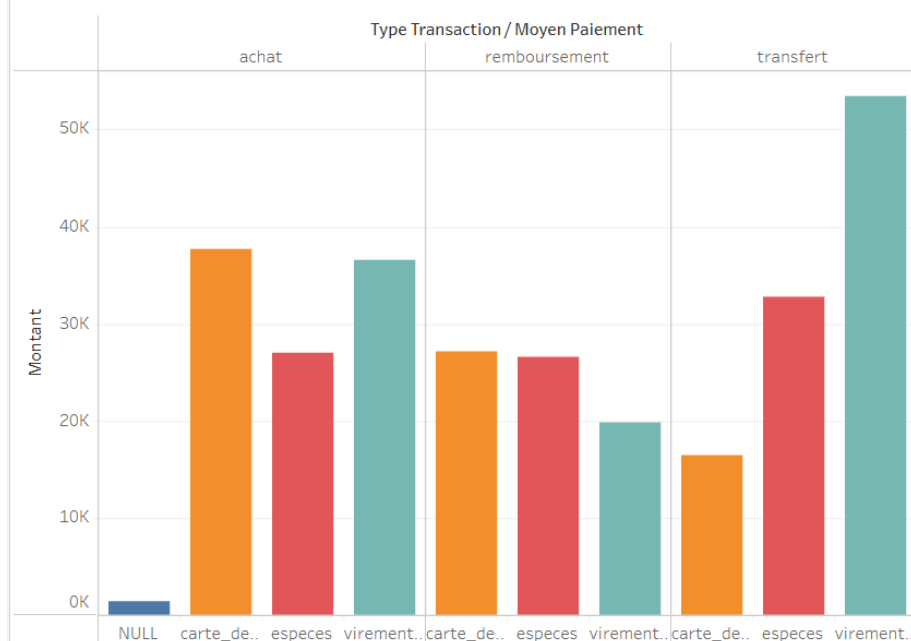
Somme des montant par lieu



Somme des montants par lieu

- Répartition des montants totaux par lieu (ex : Lyon, Nice, Paris, etc.).

Montant par type de transaction et moyen de paiement



Activer Win

Activité de la semaine

1. Montant par type de transaction et moyen de paiement

- Montants associés à des combinaisons de types de transaction (remboursement, transfert) et moyens de paiement (carte, espèces, virement).

Conclusion

Ce TP a permis la mise en place d'une architecture d'ingestion de données distribuée à l'aide de Kafka, Flink et Conduktor. L'orchestration via Docker, la consommation en temps réel et la visualisation du pipeline ont permis d'appréhender un système résilient et scalable.