

TP2 KAFKA



RABHI Sofiene
30 novembre 2023



Introduction

Dans ce travail pratique, nous explorerons des concepts et des opérations clés liés à Apache Kafka, un système de messagerie distribué, et Apache NiFi, un outil de gestion de flux de données. Notre objectif est de développer une compréhension pratique de la gestion des messages et du traitement des flux de données en temps réel.

Apache Kafka est une plateforme de streaming de données open-source, utilisée pour construire des pipelines de données robustes, efficaces et en temps réel. Kafka permet la manipulation et le traitement de grands volumes de données avec une latence minimale. Il est largement utilisé pour la gestion de flux de données dans les architectures de microservices, le traitement d'événements et la collecte de données de télémétrie.

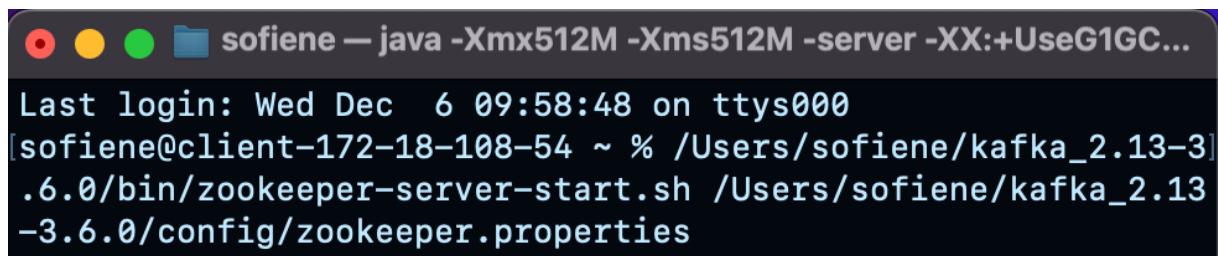
Apache NiFi, quant à lui, est un système automatisé pour le transfert et la gestion des données entre différents systèmes. Il fournit une interface utilisateur basée sur un navigateur pour la création, la surveillance et le contrôle de flux de données.

Ce TP comprend plusieurs étapes :

- 1) La configuration et la gestion d'un broker Kafka et de topics.
- 2) La manipulation de producteurs et de consommateurs dans Kafka.
- 3) La compréhension des groupes de consommateurs et leur comportement.
- 4) L'intégration de Apache NiFi pour lire et publier des messages dans Kafka.

Chaque étape sera accompagnée de captures d'écran et de commandes utilisées, formant ainsi un rapport détaillé et pratique sur le fonctionnement de Kafka et NiFi.

→ Exécution de Zookeeper



A terminal window showing the execution of Zookeeper. The command entered is: `sofiene — java -Xmx512M -Xms512M -server -XX:+UseG1GC...`. The output shows the user's last login information: `Last login: Wed Dec 6 09:58:48 on ttys000`, the command run: `[sofiene@client-172-18-108-54 ~ % /Users/sofiene/kafka_2.13-3]`, and the full command: `.6.0/bin/zookeeper-server-start.sh /Users/sofiene/kafka_2.13-3.6.0/config/zookeeper.properties`.

→ Lancement de Kafka

Pour initier l'utilisation de Kafka, il est essentiel de démarrer Zookeeper, car Kafka dépend de Zookeeper pour son fonctionnement, et ensuite on lance également Kafka.

 sofiene — java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:...
[sofiene@client-172-18-108-54 ~ % /Users/sofiene/kafka_2.13-3.6.0/bin/kafka-server-start.sh /Users/sofiene/kafka_2.13-3.6.0/config/server.properties

Nous avons déjà lancé un broker Kafka en exécutant la commande **kafka-server-start.sh** avec le fichier de configuration **server.properties**. Ce broker est le premier nœud de votre cluster Kafka. Kafka utilise Zookeeper pour gérer l'état du cluster et la coordination entre les brokers.

Nous allons ouvrir une autre fenêtre de terminal ou nous allons travaillerons. Nous garderons ses deux terminaux ouverts pour maintenir Kafka et Zookeeper en cours d'exécution.

→ Lancer un cluster de 3 brokers Kafka

Pour créer un cluster de 3 brokers Kafka, nous devons lancer deux autres instances de Kafka sur votre machine, chacune avec sa propre configuration. Pour cela nous allons modifier les fichiers de configuration :

 sofiene — -zsh — 60x14
Last login: Wed Dec 6 10:08:51 on ttys001
sofiene@client-172-18-108-54 ~ % cp /Users/sofiene/kafka_2.13-3.6.0/config/server.properties /Users/sofiene/kafka_2.13-3.6.0/config/server-1.properties
[cp /Users/sofiene/kafka_2.13-3.6.0/config/server.properties] /Users/sofiene/kafka_2.13-3.6.0/config/server-2.properties

Nous allons Configurer les nouveaux brokers, pour cela nous allons modifiez les fichiers **server-1.properties** et **server-2.properties** pour définir des **broker.id** uniques et des ports d'écoute différents. Utilisez les commandes suivantes pour effectuer ces modifications :

 sofiene — -zsh — 60x14
sofiene@client-172-18-108-54 ~ % echo "broker.id=1" >> /Users/sofiene/kafka_2.13-3.6.0/config/server-1.properties
echo "listeners=PLAINTEXT://:9093" >> /Users/sofiene/kafka_2.13-3.6.0/config/server-1.properties
echo "log.dirs=/tmp/kafka-logs-1" >> /Users/sofiene/kafka_2.13-3.6.0/config/server-1.properties

echo "broker.id=2" >> /Users/sofiene/kafka_2.13-3.6.0/config/server-2.properties
echo "listeners=PLAINTEXT://:9094" >> /Users/sofiene/kafka_2.13-3.6.0/config/server-2.properties
[echo "log.dirs=/tmp/kafka-logs-2" >> /Users/sofiene/kafka_2.13-3.6.0/config/server-2.properties

Nous ouvrons deux nouveaux terminaux. On lance le 2eme et 3eme broker avec :

Brokers 2

```
sofiene — java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:Ma...
Last login: Wed Dec 6 10:40:16 on ttys002
sofiene@client-172-18-108-54 ~ % /Users/sofiene/kafka_2.13-3.6.0/bin/kafka-server-start.sh /Users/sofiene/kafka_2.13-3.6.0/config/server-1.properties

[2023-12-06 10:48:59,854] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2023-12-06 10:48:59,986] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation (org.apache.zookeeper.common.X509Util)
[2023-12-06 10:49:00,025] INFO Registered signal handlers for TERM, INT, HUP (org.apache.kafka.common.utils.LoggingSignalHandler)
[2023-12-06 10:49:00,026] INFO starting (kafka.server.KafkaServer)
```

Broker 3

```
sofiene — java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:Ma...
sofiene@client-172-18-108-54 ~ % /Users/sofiene/kafka_2.13-3.6.0/bin/kafka-server-start.sh /Users/sofiene/kafka_2.13-3.6.0/config/server-2.properties

[2023-12-06 10:49:33,711] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2023-12-06 10:49:33,845] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation (org.apache.zookeeper.common.X509Util)
[2023-12-06 10:49:33,883] INFO Registered signal handlers for TERM, INT, HUP (org.apache.kafka.common.utils.LoggingSignalHandler)
[2023-12-06 10:49:33,884] INFO starting (kafka.server.KafkaServer)
```

II) Les Topics

TP_MIAGE_1

```
bin — -zsh — 62x15
Last login: Wed Dec 6 11:27:30 on ttys002
[sofiene@client-172-18-108-54 bin % ./kafka-topics.sh --create --topic TP_MIAGE_1 --partitions 3 --replication-factor 1 --bootstrap-server localhost:9092

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 TP_MIAGE_1.
```

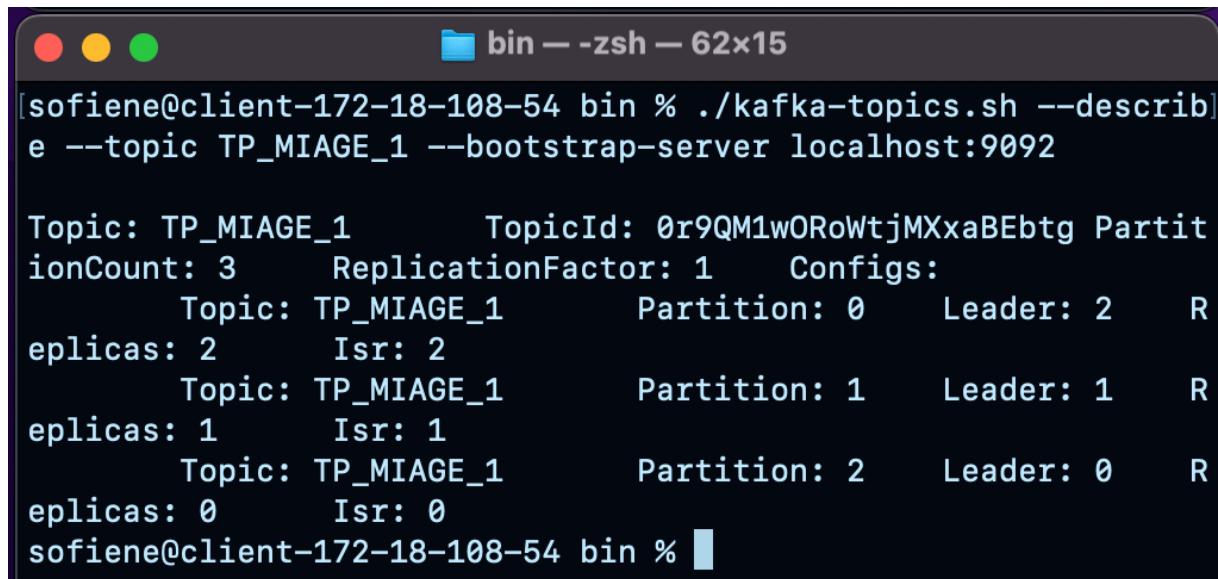
TP_MIAGE_1

```
[sofiene@client-172-18-108-54 bin % ./kafka-topics.sh --create --topic TP_MIAGE_2 --partitions 3 --replication-factor 1 --bootstrap-server localhost:9092

WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues i
```

Nous venons donc de créer 2 Topics comme demander, le message d'avertissement concernant les noms de topics avec un point (.) ou un trait de soulignement (_) est un conseil général pour éviter les problèmes potentiels dans les noms de métriques, mais cela ne devrait pas affecter votre cas d'utilisation actuel.

→ Afficher les informations du topic "TP_MIAGE_1"



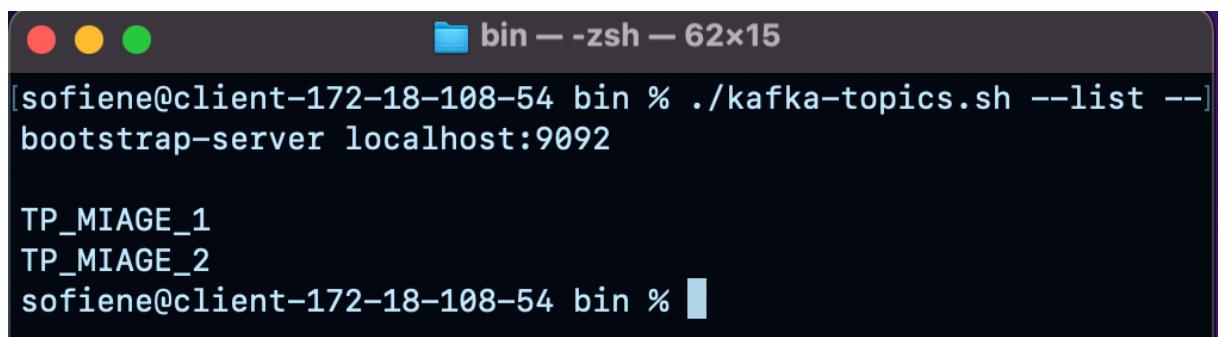
```
[sofiene@client-172-18-108-54 bin % ./kafka-topics.sh --describe --topic TP_MIAGE_1 --bootstrap-server localhost:9092

Topic: TP_MIAGE_1          TopicId: 0r9QM1wORoWtjMXxaBEbtg PartitionCount: 3      ReplicationFactor: 1      Configs:
          Topic: TP_MIAGE_1          Partition: 0      Leader: 2      Replicas: 2      Isr: 2
          Topic: TP_MIAGE_1          Partition: 1      Leader: 1      Replicas: 1      Isr: 1
          Topic: TP_MIAGE_1          Partition: 2      Leader: 0      Replicas: 0      Isr: 0
sofiene@client-172-18-108-54 bin % ]
```

Les informations du topic "TP_MIAGE_1" semblent correctes. Il a 3 partitions avec un facteur de réplication de 1, comme spécifié. Chaque partition a son propre leader et réplique, ce qui est essentiel pour la distribution et la résilience des données.

→ Lister les topics créés

Maintenant, passons à l'étape suivante qui consiste à lister tous les topics existants. On utilise la commande suivante :



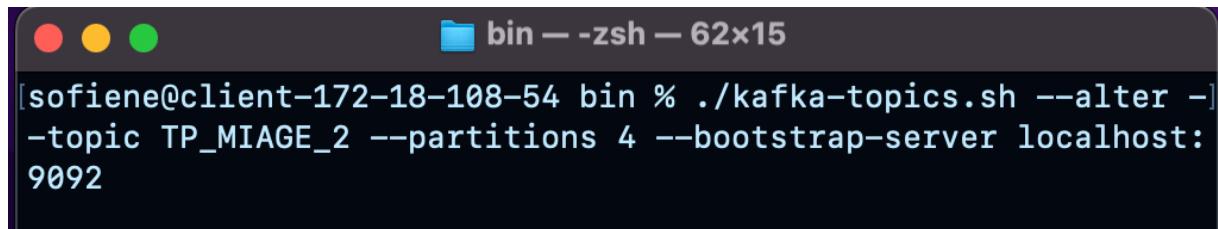
```
[sofiene@client-172-18-108-54 bin % ./kafka-topics.sh --list --bootstrap-server localhost:9092

TP_MIAGE_1
TP_MIAGE_2
sofiene@client-172-18-108-54 bin % ]
```

Les topics "TP_MIAGE_1" et "TP_MIAGE_2" ont été correctement listés. Nous pouvons maintenant passer à l'étape suivante.

→ Modifier le nombre de partitions du topic «TP_MIAGE_2» à 4

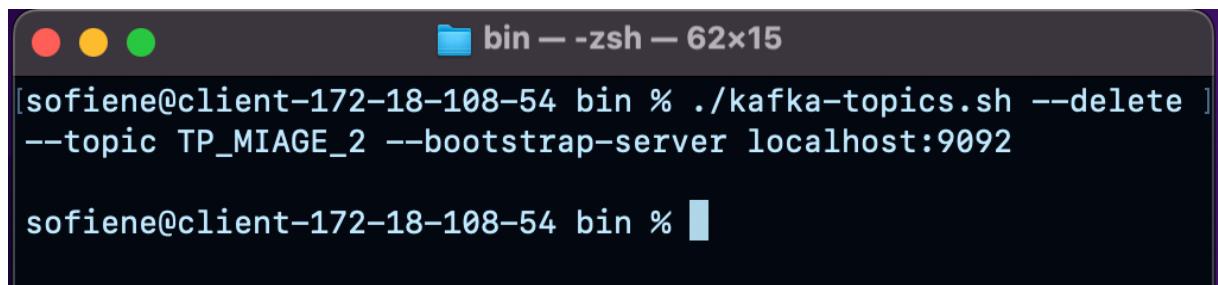
Pour modifier le nombre de partitions de "TP_MIAGE_2", utilisez la commande suivante :



```
bin — zsh — 62x15
[sofiene@client-172-18-108-54 bin % ./kafka-topics.sh --alter --topic TP_MIAGE_2 --partitions 4 --bootstrap-server localhost:9092
```

Cette commande augmente le nombre de partitions du topic "TP_MIAGE_2" à 4. C'est utile pour améliorer les performances en permettant une distribution plus large des données et une meilleure parallélisation.

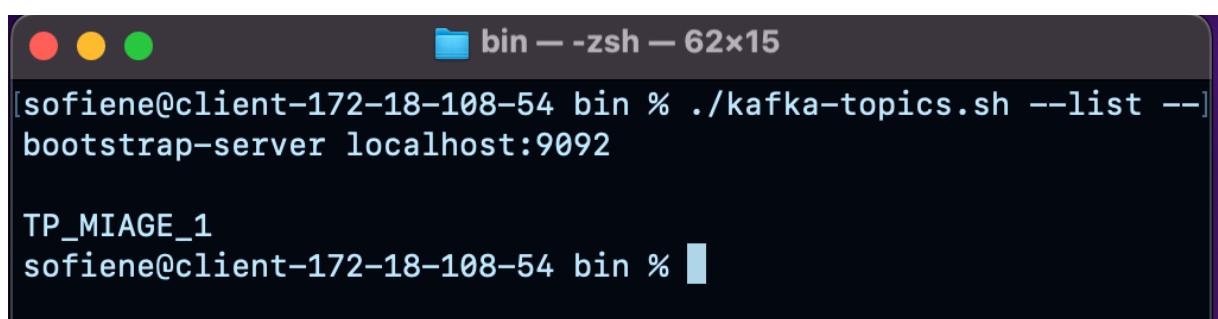
→ Supprimer le topic «TP_MIAGE_2»



```
bin — zsh — 62x15
[sofiene@client-172-18-108-54 bin % ./kafka-topics.sh --delete --topic TP_MIAGE_2 --bootstrap-server localhost:9092
sofiene@client-172-18-108-54 bin %
```

Cette commande supprimera le topic "TP_MIAGE_2" de notre instance Kafka. C'est une opération irréversible.

→ Afficher la liste des topics



```
bin — zsh — 62x15
[sofiene@client-172-18-108-54 bin % ./kafka-topics.sh --list --bootstrap-server localhost:9092
TP_MIAGE_1
sofiene@client-172-18-108-54 bin %
```

Cette commande affichera la liste de tous les topics existants dans notre instance Kafka. Après l'exécution de cette commande, on a donc uniquement le topic "TP_MIAGE_1" dans la liste, car la suppression du topic "TP_MIAGE_2" a bien été effectuée.

III) Le Producer

→ Créer un topic «TP_MIAGE_3 » avec un 1 seule partition

On exécute la commande suivante dans nôtres terminal Kafka :

```
bin — -zsh — 62x15
[sofiene@client-172-18-108-54 bin % ./kafka-topics.sh --create
--topic TP_MIAGE_3 --partitions 1 --replication-factor 1 --boo
tstrap-server localhost:9092

WARNING: Due to limitations in metric names, topics with a per
iod ('.') or underscore ('_') could collide. To avoid issues i
t is best to use either, but not both.
Created topic TP_MIAGE_3.
sofiene@client-172-18-108-54 bin %
```

Cette commande crée un nouveau topic nommé "TP_MIAGE_3" avec une seule partition et un facteur de réPLICATION de 1.

→Lancer un procduer sur le topic « TP_MIAGE_3 » →Publier des messages dans le topic

```
bin — java -Xmx512M -server -XX:+UseG1GC -XX:MaxGCPause...
[sofiene@client-172-18-108-54 bin % ./kafka-console-producer.sh
--broker-list localhost:9092 --topic TP_MIAGE_3

>Bonjour
>Je
>veut
>une
>bonne
>note
>svp
>
```

Cette commande ouvre une interface en ligne de commande où vous nous avons saisi des messages. Chaque ligne que nous entrerez sera envoyée comme un message distinct au topic "TP_MIAGE_3

→Afficher le nombre de messages dans le topic «TP_MIAGE_3»

Voici la commande pour démarrer un consommateur qui lira depuis le début du topic

```
bin -- zsh -- 62x15
[sofiene@client-172-18-108-54 bin % ./kafka-console-consumer.sh
--bootstrap-server localhost:9092 --topic TP_MIAGE_3 --from-beginning

Bonjour
Je
veut
une
bonne
note
svp
^CProcessed a total of 7 messages
sofiene@client-172-18-108-54 bin %
```

Pour voir le nombre de messages dans un topic, on utilise généralement un consommateur Kafka pour lire les messages. Cependant, Kafka ne stocke pas directement le nombre total de messages dans un topic. Néanmoins, on peut lancer un consommateur pour lire tous les messages et ainsi compter combien il y en a. ici dans notre exemple il y'a 7 message sur 7 lignes.

→Afficher le plus ancien offset présent dans le topic "TP_MIAGE_3"

Pour afficher les informations sur les offsets, nous utiliserons la commande **kafka-run-class.sh kafka.tools.GetOffsetShell**. Cette commande permet de voir les offsets pour chaque partition d'un topic. Comme "TP_MIAGE_3" a une seule partition, nous nous attendons à voir un seul offset. Voici la commande :

```
bin -- zsh -- 62x15
[sofiene@client-172-18-108-54 bin % ./kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list localhost:9092 --topic TP_MIAGE_3 --time -2

TP_MIAGE_3:0:0
sofiene@client-172-18-108-54 bin %
```

Cette commande affichera l'offset le plus ancien pour chaque partition du topic "TP_MIAGE_3". L'option **--time -2** est utilisée pour spécifier que nous voulons voir l'offset le plus ancien.

→Afficher le dernier offset du topic

Pour voir le dernier offset dans le topic, nous utiliserons la même commande que précédemment, mais avec l'option **--time -1**. Cette option permet de voir le dernier offset disponible pour chaque partition. Voici la commande à exécuter :

```
bin — zsh — 62x15
[sofiene@client-172-18-108-54 bin % ./kafka-run-class.sh kafka.
tools.GetOffsetShell --broker-list localhost:9092 --topic TP_M
IAGE_3 --time -1

TP_MIAGE_3:0:7
sofiene@client-172-18-108-54 bin %
```

→Publier des messages dans des partitions ciblés (utiliser les clés)

Pour publier des messages dans des partitions spécifiques en utilisant des clés, nous allons utiliser le `kafka-console-producer.sh` avec l'option `--property parse.key=true` et `--property key.separator=:`. Cette configuration permet de spécifier une clé pour chaque message, et Kafka utilisera cette clé pour déterminer dans quelle partition placer le message.

Voici la commande à exécuter :

```
bin — java -Xmx512M -server -XX:+UseG1GC -XX:MaxGCPause...
sofiene@client-172-18-108-54 bin % ./kafka-console-producer.sh
--broker-list localhost:9092 --topic TP_MIAGE_3 --property pa
rse.key=true --property key.separator=:

>etudiant1: bonjour
>etudiant2: salut sa va !!!!
>etudiant3: oui et toi
>
```

III) Les consumers

Pour lancer un consommateur qui lira les messages du topic "TP_MIAGE_3", on utilise la commande suivante :

```
bin — java -Xmx512M -server -XX:+UseG1GC -XX:MaxGCPause...
[sofiene@client-172-18-108-54 bin % ./kafka-console-consumer.sh]
--bootstrap-server localhost:9092 --topic TP_MIAGE_3
```

le consommateur Kafka est prêt à recevoir les nouveaux messages en temps réel du topic "TP_MIAGE_3".

Pour lancer un consommateur qui lira les messages du début, ajoutez l'option **--from-beginning** :

```
bin — java -Xmx512M -server -XX:+UseG1GC -XX:MaxGCPaus...
[sofiene@client-172-18-108-54 bin % ./kafka-console-consumer.s]
h --bootstrap-server localhost:9092 --topic TP_MIAGE_3 --from
--beginning

Bonjour
Je
veut
une
bonne
note
svp
Bonjour, ceci est un message pour le topic TP_MIAGE_3.
c'est très bien
il y aura une mise à jour du serveur le 23/12/23
bonjour
```

le consommateur Kafka a lu tous les messages déjà publiés sur le topic "TP_MIAGE_3" depuis le début, y compris les nouveaux messages que vous avez ajoutés.

→ Lancer un consommateur pour lire à partir d'un numéro d'offset particulier

Pour ce faire, nous aurons besoin d'ouvrir une troisième fenêtre de terminal et d'exécuter la commande suivante, en ajoutant le numéro d'offset à partir duquel nous voulons commencer la lecture (par exemple 2 ici sur le screen que nous avons:

```
● ● ● bin — java -Xmx512M -server -XX:+UseG1GC -XX:MaxGCPaus...
[sofiene@client-172-18-108-54 bin % ./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic TP_MIAGE_3 --offset 2 --partition 0

veut
une
bonne
note
svp
Bonjour, ceci est un message pour le topic TP_MIAGE_3.
c'est très bien
il y aura une mise à jour du serveur le 23/12/23
bonjour
salut sa va !!!! 
oui et toi
```

→ Arrêter les consumers et les producers

Pour arrêter un consumer ou un producer, on utilise simplement utiliser la combinaison de touches **Ctrl + C** dans les terminaux où ils sont en cours d'exécution.

V) Les groupe de Consumers

→ Créer un topic "TP_MIAGE_4" avec 3 partitions

Tout d'abord, nous devons créer un nouveau topic. En exécutent la commande suivante dans notre terminal Kafka (en étant dans le répertoire **bin** de Kafka) :

```
● ● ● bin — -zsh — 61x15
[sofiene@client-172-18-108-54 bin % ./kafka-topics.sh --create --topic TP_MIAGE_4 --partitions 3 --replication-factor 1 --bootstrap-server localhost:9092

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 TP_MIAGE_4.
sofiene@client-172-18-108-54 bin %
```

→ Créer un groupe de consumers sur la partition

Maintenant, nous allons créer un groupe de consumers. Cependant, il est important de noter que dans Kafka, les groupes de consumers ne sont pas "créés" via une commande spécifique. Au lieu de cela, ils sont définis lorsqu'un consumer commence à écouter un topic et se déclare membre d'un groupe.

Pour cette étape, vous allez lancer des consumers qui écouteront le topic **TP_MIAGE_4** et se déclareront membres d'un groupe spécifique. Cela sera fait en ouvrant plusieurs terminaux et en lançant un consumer dans chaque terminal.

→ Créer 3 consumers dans le group

Terminal 1

```
● ● ● bin — java -Xmx512M -server -XX:+UseG1GC -XX:MaxGCPause...
[sofiene@client-172-18-108-54 bin % ./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic TP_MIAGE_4 --group groupeMIAGE
```

Terminal 2

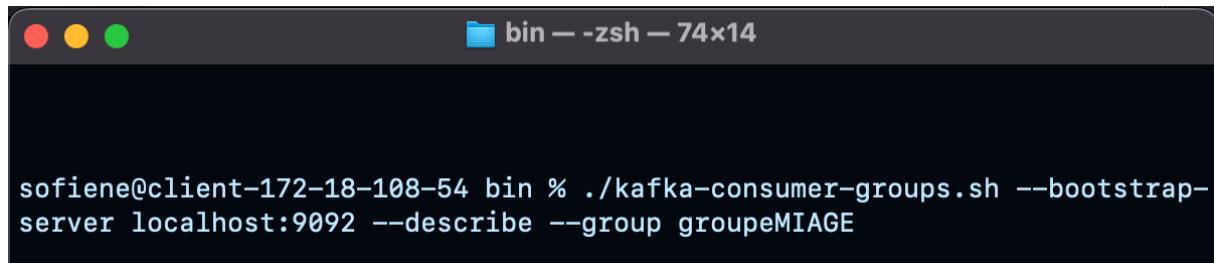
```
● ● ● bin — java -Xmx512M -server -XX:+UseG1GC -XX:MaxGCPause...
[sofiene@client-172-18-108-54 bin % ./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic TP_MIAGE_4 --group groupeMIAGE
```

Terminal 3

```
● ● ● bin — java -Xmx512M -server -XX:+UseG1GC -XX:MaxGCPause...
[sofiene@client-172-18-108-54 bin % ./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic TP_MIAGE_4 --group groupeMIAGE
```

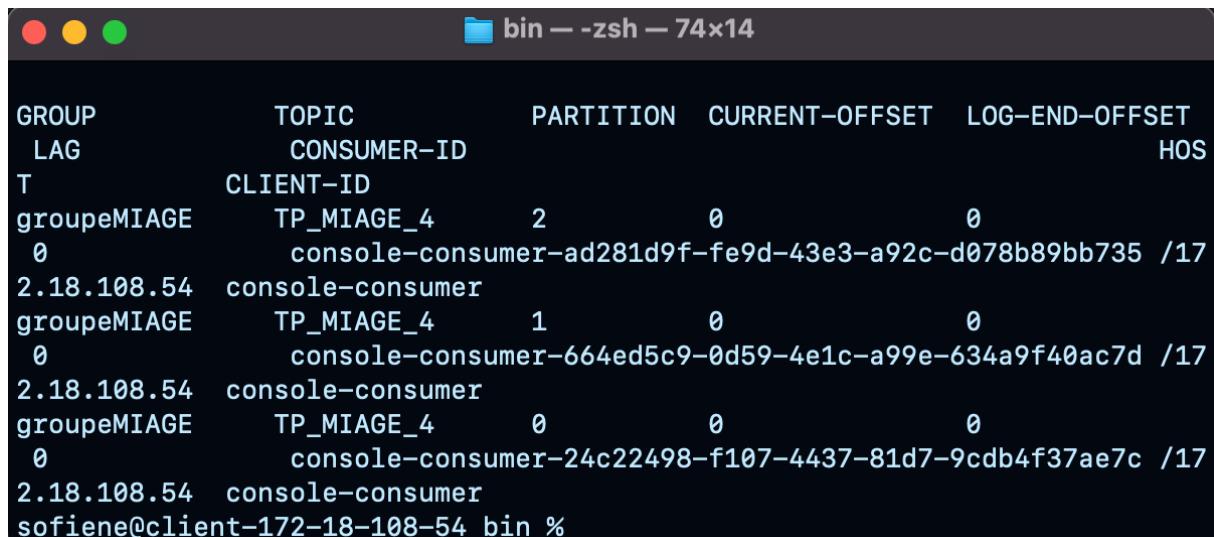
→ Afficher les membres du groupe

Pour afficher les informations sur les membres du groupe de consommateurs que nous avons créé, nous pouvons utiliser la commande suivante :



```
bin -- zsh -- 74x14
sofiene@client-172-18-108-54 bin % ./kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group groupeMIAGE
```

Cette commande va nous donner des détails sur le groupe de consommateurs "groupeMIAGE", y compris les membres du groupe, les topics auxquels ils sont abonnés, et la répartition des partitions entre eux.



GROUP	TOPIC	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	HOS
LAG	CONSUMER-ID				
T	CLIENT-ID				
groupeMIAGE	TP_MIAGE_4	2	0	0	
0	console-consumer-ad281d9f-fe9d-43e3-a92c-d078b89bb735			/17	
2.18.108.54	console-consumer				
groupeMIAGE	TP_MIAGE_4	1	0	0	
0	console-consumer-664ed5c9-0d59-4e1c-a99e-634a9f40ac7d			/17	
2.18.108.54	console-consumer				
groupeMIAGE	TP_MIAGE_4	0	0	0	
0	console-consumer-24c22498-f107-4437-81d7-9cdb4f37ae7c			/17	
2.18.108.54	console-consumer				
sofiene@client-172-18-108-54	bin %				

→ Lancer un producer sur le topic "TP_MIAGE_4" et publier 10 messages minimum

Maintenant, nous allons publier des messages sur le topic "TP_MIAGE_4". Pour cela, nous allons utiliser la commande suivante dans un nouveau terminal :

```
[sofiene@client-172-18-108-54 bin % ./kafka-console-producer.sh  
--broker-list localhost:9092 --topic TP_MIAGE_4  
  
>message 1  
>message 2  
>message 3  
>message 4  
>message 5  
>message 6  
>message 7
```

On obtient cela sur 1 seul des 3 terminaux consumer tous les messages ont été enregistrer sur un seul des 3 terminal pourquoi ?

```
[sofiene@client-172-18-108-54 bin % ./kafka-console-consumer.sh  
--bootstrap-server localhost:9092 --topic TP_MIAGE_4 --group  
groupeMIAGE  
  
message 1  
message 2  
message 3  
message 4  
message 5  
message 6  
message 7  
message 8  
message 9  
message 10  
bonjour
```

Dans nôtres cas, avec trois consommateurs dans le groupe "groupeMIAGE" et un topic "TP_MIAGE_4" ayant trois partitions, il est probable que chaque consommateur se soit vu attribuer une partition distincte. Lorsque nous avons publié nos messages, ils ont été répartis entre les partitions. Si tous les messages sont apparus sur un seul terminal de consommateur, cela suggère que tous les messages ont été envoyés à la même partition.

→ On décide d'utiliser des clés différentes pour nos messages et influencer la répartition des messages entre les partitions dans Kafka, nous avons utilisé la commande suivante pour lancer un producteur Kafka. Cette commande active **parse.key** et définit **key.separator** pour permettre l'envoi de messages avec des clés :

```
bin — java -Xmx512M -server -XX:+UseG1GC -XX:MaxGCPause...
[sofiene@client-172-18-108-54 bin % ./kafka-console-producer.sh
--broker-list localhost:9092 --topic TP_MIAGE_4 --property pa
rse.key=true --property key.separator=:

>etudiant: bonjour
>professeur: oui bonjour, vous avez besoin de qqchose
>etudiant: oui j'aiemrais avoir la correction du TD 2
>professeur: pas de souci je le met sur ecampus
>
```

Voici comment les messages ont été géré dans les différent consumer en fonction de leur clé :

Consumer 2

```
bin — java -Xmx512M -server -XX:+UseG1GC -XX:MaxGCPaus...
[sofiene@client-172-18-108-54 bin % ./kafka-console-consumer.s
h --bootstrap-server localhost:9092 --topic TP_MIAGE_4 --grou
p groupeMIAGE

bonjour
oui j'aiemrais avoir la correction du TD 2
[]
```

Consumer 2

```
bin — java -Xmx512M -server -XX:+UseG1GC -XX:MaxGCPaus...
[sofiene@client-172-18-108-54 bin % ./kafka-console-consumer.s
h --bootstrap-server localhost:9092 --topic TP_MIAGE_4 --grou
p groupeMIAGE

oui bonjour, vous avez besoin de qqchose
pas de souci je le met sur ecampus
[]
```

→ Qu'est-ce que vous remarquez ?

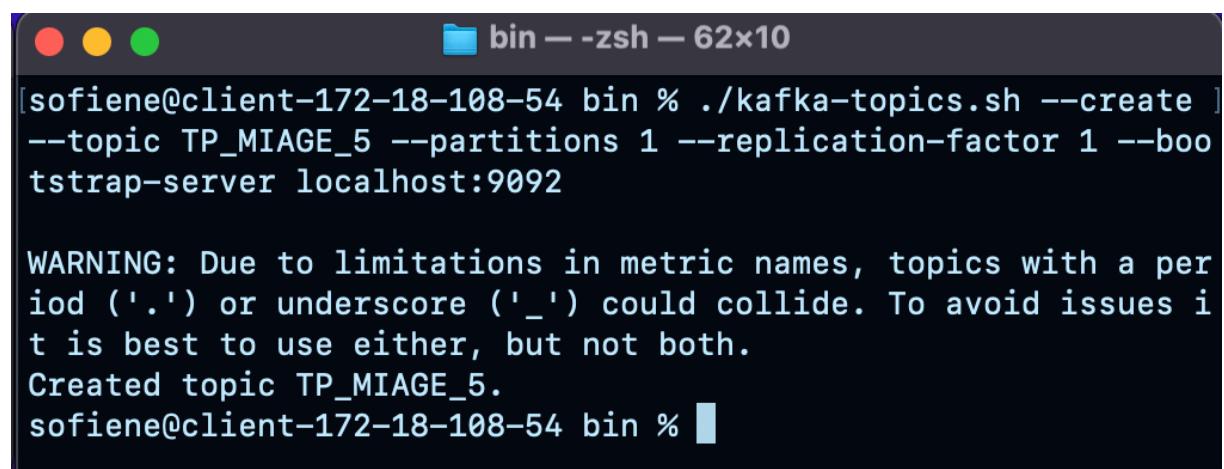
1. **Répartition Basée sur les Clés** : Les messages avec des clés différentes sont répartis entre les partitions de manière cohérente. Par exemple, tous les messages avec la clé "etudiant" vont dans une partition spécifique, tandis que ceux avec la clé "professeur" vont dans une autre.
2. **Consommation Groupée** : Les consommateurs dans un groupe de consommateurs (**groupeMIAGE** dans nos cas) reçoivent des messages de différentes partitions. Chaque consommateur du groupe se voit attribuer une ou plusieurs partitions du topic, et il ne consomme que les messages de ces partitions.
3. **Équilibre de Charge** : L'utilisation de clés aide à équilibrer la charge entre les partitions. Cela permet une consommation plus efficace des messages, car les consommateurs travaillent en parallèle sur différentes partitions.
4. **Garantie de l'Ordre des Messages par Clé** : Kafka garantit que les messages envoyés avec la même clé (par exemple, tous les messages de "etudiant") sont consommés dans l'ordre dans lequel ils ont été produits.
5. **Distribution des Messages** : Vous avez peut-être remarqué que les messages ne sont pas uniformément répartis si vous n'utilisez qu'un nombre limité de clés. Par exemple, si vous avez trois partitions mais seulement deux clés, une des partitions pourrait ne recevoir aucun message.

En résumé, l'utilisation de clés dans Kafka permet une meilleure gestion et une consommation efficace des messages dans un système de messagerie distribué, tout en garantissant l'ordre des messages pour chaque clé.

VI) NIFI et KAFKA

→ Créer un topic "TP_MIAGE_5"

Comme précédemment, dans notre terminal Kafka, on exécute la commande suivante pour créer le topic :



```
sofiene@client-172-18-108-54 bin % ./kafka-topics.sh --create --topic TP_MIAGE_5 --partitions 1 --replication-factor 1 --bootstrap-server localhost:9092

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 TP_MIAGE_5.
sofiene@client-172-18-108-54 bin %
```

→ Lire le fichier log nifi ligne par ligne



Voici à quoi ressemble notre première étape sur nifi afin de lire les fichier log et de les split ligne par ligne come demander dans le TP, voici les paramètre quant au processus GetFile afin de lire le fichier voulue dans les log, et le processus SplitText pour lire de façon ligne par ligne voici les préréglage des deux processor ci-dessous :

GetFile

Configure Processor | GetFile 1.22.0

Stopped

SETTINGS	SCHEDULING	PROPERTIES	RELATIONSHIPS	COMMENTS																										
Required field																														
<table border="1"> <thead> <tr> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>Input Directory</td> <td>✓ /Users/sofiene/Documents/nifi-1.22.0/logs</td> </tr> <tr> <td>File Filter</td> <td>✓ nifi-app.log</td> </tr> <tr> <td>Path Filter</td> <td>✓ No value set</td> </tr> <tr> <td>Batch Size</td> <td>✓ 10</td> </tr> <tr> <td>Keep Source File</td> <td>✓ true</td> </tr> <tr> <td>Recurse Subdirectories</td> <td>✓ true</td> </tr> <tr> <td>Polling Interval</td> <td>✓ 0 sec</td> </tr> <tr> <td>Ignore Hidden Files</td> <td>✓ true</td> </tr> <tr> <td>Minimum File Age</td> <td>✓ 0 sec</td> </tr> <tr> <td>Maximum File Age</td> <td>✓ No value set</td> </tr> <tr> <td>Minimum File Size</td> <td>✓ 0 B</td> </tr> <tr> <td>Maximum File Size</td> <td>✓ No value set</td> </tr> </tbody> </table>					Property	Value	Input Directory	✓ /Users/sofiene/Documents/nifi-1.22.0/logs	File Filter	✓ nifi-app.log	Path Filter	✓ No value set	Batch Size	✓ 10	Keep Source File	✓ true	Recurse Subdirectories	✓ true	Polling Interval	✓ 0 sec	Ignore Hidden Files	✓ true	Minimum File Age	✓ 0 sec	Maximum File Age	✓ No value set	Minimum File Size	✓ 0 B	Maximum File Size	✓ No value set
Property	Value																													
Input Directory	✓ /Users/sofiene/Documents/nifi-1.22.0/logs																													
File Filter	✓ nifi-app.log																													
Path Filter	✓ No value set																													
Batch Size	✓ 10																													
Keep Source File	✓ true																													
Recurse Subdirectories	✓ true																													
Polling Interval	✓ 0 sec																													
Ignore Hidden Files	✓ true																													
Minimum File Age	✓ 0 sec																													
Maximum File Age	✓ No value set																													
Minimum File Size	✓ 0 B																													
Maximum File Size	✓ No value set																													
<input checked="" type="checkbox"/> <input type="button" value="+"/>																														
<input type="button" value="CANCEL"/> <input type="button" value="APPLY"/>																														

SplitText

Configure Processor | SplitText 1.22.0

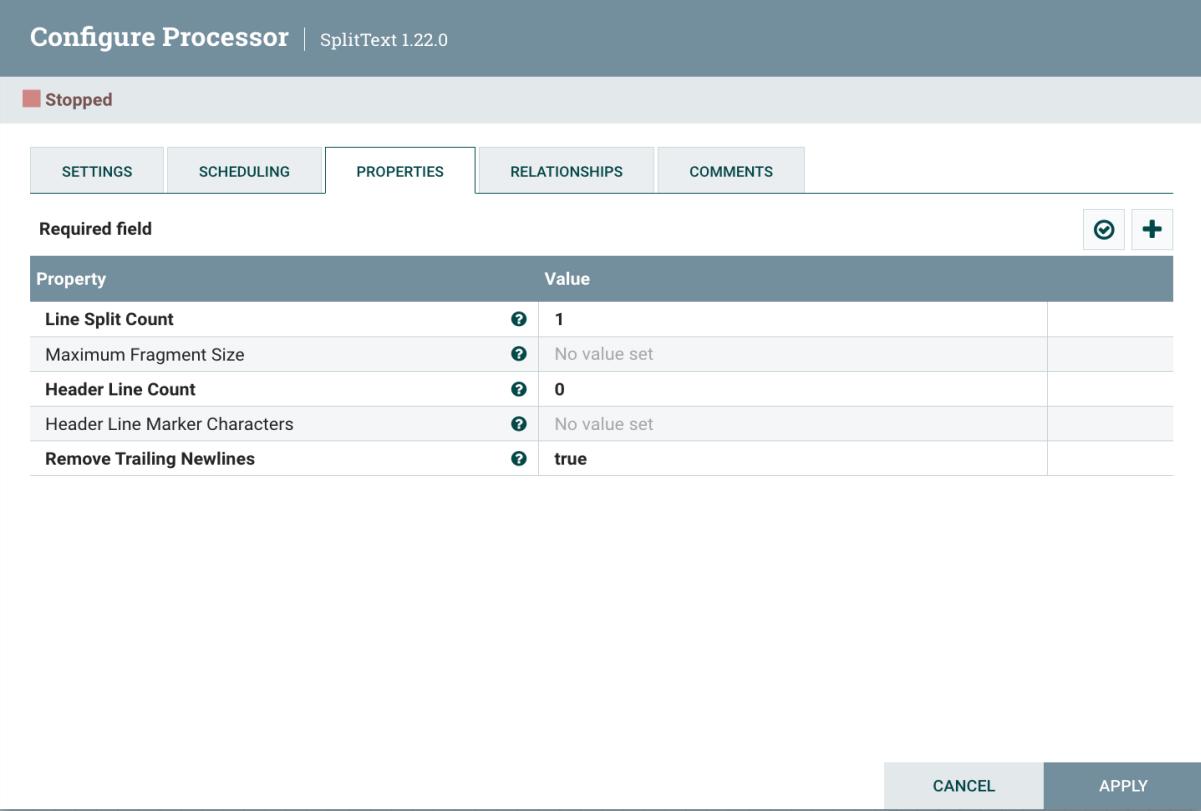
Stopped

SETTINGS SCHEDULING PROPERTIES RELATIONSHIPS COMMENTS

Required field

Property	Value
Line Split Count	1
Maximum Fragment Size	No value set
Header Line Count	0
Header Line Marker Characters	No value set
Remove Trailing Newlines	true

CANCEL APPLY



→ Publier chaque ligne dans le topic "TP_MIAGE_5"

Pour cela, vous nous allons avoir besoin d'utiliser un processeur PublishKafka dans NiFi. Nous avons configuré ce processeur avec les informations de notre serveur Kafka et le nom du topic où publier les messages. Voici le paramétrage ci-dessous :

Configure Processor | PublishKafka_2_0 1.22.0

Stopped

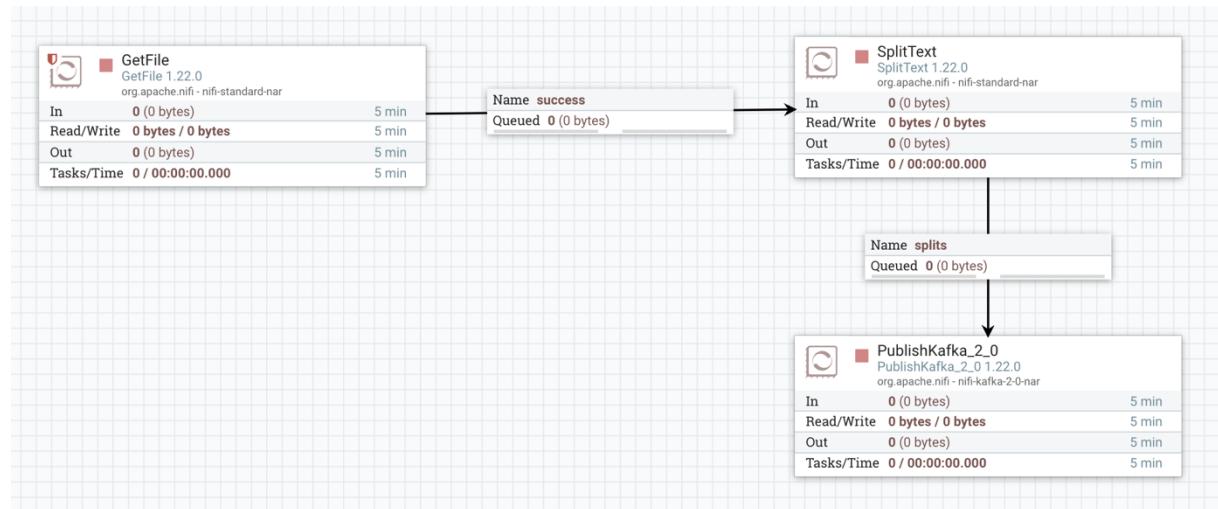
SETTINGS SCHEDULING PROPERTIES RELATIONSHIPS COMMENTS

Required field

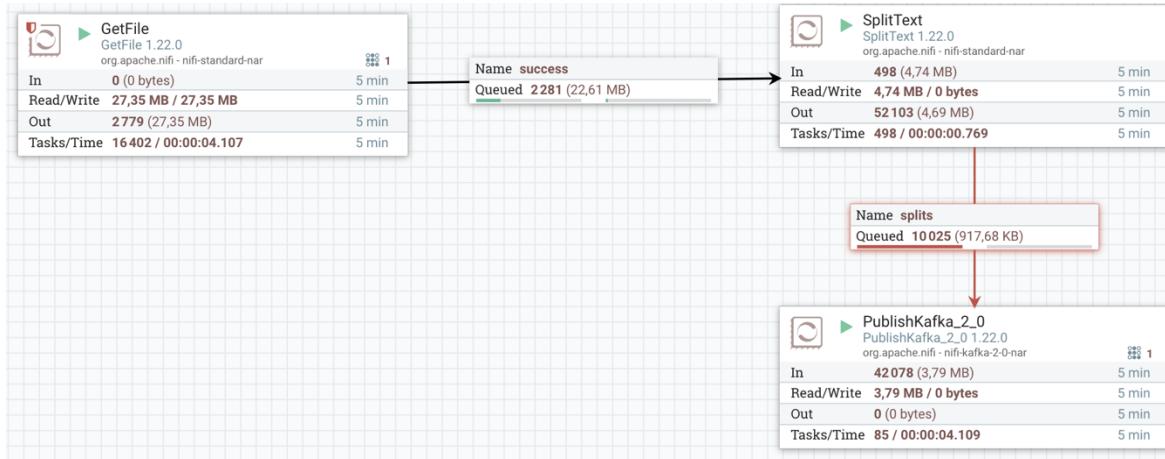
Property	Value
Kafka Brokers	localhost:9092
Security Protocol	PLAINTEXT
SASL Mechanism	GSSAPI
Kerberos Service Name	No value set
Kerberos Credentials Service	No value set
Kerberos Principal	No value set
Kerberos Keytab	No value set
SSL Context Service	No value set
Topic Name	TP_MIAGE_5
Delivery Guarantee	Guarantee Replicated Delivery
Failure Strategy	Route to Failure
Use Transactions	true

CANCEL APPLY

On a donc la vue suivante qui représente tous nos processors avant d'avoir exécuter :



Après l'exécution de notre groupe, on voit bien les données qui ont été chargées,分裂, puis publiées sur notre Kafka. Nous allons vérifier cela juste après :



splits						
Displaying 100 of 10,161 (1.60 MB)						
Position	UUID	Filename	File Size	Queued Duration	Lineage Duration	Penalized
1	601c49f-4192-46ee-aef3-2c4874d703b	nifi-app.log	156.0 bytes	00:00:01.358	00:00:34.008	No
2	2a93a03-c670-aab8-86ad-7934a79407b	nifi-app.log	188.0 bytes	00:00:01.358	00:00:34.008	No
3	703019c1-c1e3-41fb-bee7-47d12e7a8bc	nifi-app.log	180.0 bytes	00:00:01.358	00:00:34.008	No
4	099786e4-c8f5-4acd-845a-12176e63010	nifi-app.log	178.0 bytes	00:00:01.358	00:00:34.008	No
5	90d61b26-e425-4242-ab8d-bf50969484e	nifi-app.log	122.00 bytes	00:00:01.358	00:00:34.008	No
6	6f586e1b-1648-47de-9a38-522e3f38ead	nifi-app.log	11.00 bytes	00:00:01.358	00:00:34.008	No
7	8a5f2f76-7069-4c0a-91ce-568d46fb010	nifi-app.log	19.00 bytes	00:00:01.358	00:00:34.008	No
8	325734c5-7e43-478e-b5b5-5a54c3502c13	nifi-app.log	37.00 bytes	00:00:01.358	00:00:34.008	No
9	408b9839-7b33-46a0-9878-e121408ee407	nifi-app.log	25.00 bytes	00:00:01.358	00:00:34.008	No
10	9ff32ab6-e270-4721-96a6-05434c460ac	nifi-app.log	13.00 bytes	00:00:01.358	00:00:34.008	No
11	09a789e2-bae1-49de-82e7-55884ca500	nifi-app.log	24.00 bytes	00:00:01.358	00:00:34.008	No
12	11fdff322-5310-4a3c-a831-23499feee8	nifi-app.log	33.00 bytes	00:00:01.358	00:00:34.008	No
13	33d3f46e-9c79-414b-918d-281e28744885	nifi-app.log	27.00 bytes	00:00:01.358	00:00:34.008	No
14	a6e998b0-9fb0-4f5d-b934-5c054c45434	nifi-app.log	25.00 bytes	00:00:01.358	00:00:34.008	No
15	d6795270-f7e9-41e1-9d8a-001f7cde552a	nifi-app.log	81.00 bytes	00:00:01.358	00:00:34.008	No
16	4c511dbd-a8c3-4e9b-a997-a95a09f150a8	nifi-app.log	14.00 bytes	00:00:01.358	00:00:34.008	No
17	be07a8057-1b4d-4f70-8c1b-5b0d5eb14c5	nifi-app.log	20.00 bytes	00:00:01.358	00:00:34.008	No
18	bc5e6533-fc88-40b9-9250-e1b628744885	nifi-app.log	42.00 bytes	00:00:01.358	00:00:34.008	No
19	bdb4a11c-d0c0-4b80-a7cd-44562650337c	nifi-app.log	27.00 bytes	00:00:01.358	00:00:34.008	No
20	0217afe1-e202-424d-849d-2ca5bcb7fb6	nifi-app.log	29.00 bytes	00:00:01.358	00:00:34.008	No
21	1a5b0cc0-658f-44de-bb7c-400ca024682	nifi-app.log	22.00 bytes	00:00:01.358	00:00:34.008	No
22	a10d4c94-d9fe-4447-a55c-4cd0a3299fd	nifi-app.log	24.00 bytes	00:00:01.358	00:00:34.008	No
23	c6ae05e9-0979-43a2-b9ef-b9f9cccd418c	nifi-app.log	31.00 bytes	00:00:01.358	00:00:34.008	No
24	5d9cbe4b-77d8-4ccf-ad58-67eb2720170	nifi-app.log	33.00 bytes	00:00:01.358	00:00:34.008	No
25	b3f7a530-129a-464a-8578-c6b5cc0ed527	nifi-app.log	89.00 bytes	00:00:01.358	00:00:34.008	No
26	0a23c76d-f3f3-4337-9c32-9c452e135f	nifi-app.log	29.00 bytes	00:00:01.358	00:00:34.008	No
27	0ce88713-48aa-44d7-a9cb-45c130566ab	nifi-app.log	32.00 bytes	00:00:01.358	00:00:34.008	No
28	75f8874f-c415-4961-aa8b-03b20e6266c	nifi-app.log	26.00 bytes	00:00:01.358	00:00:34.008	No
29	94fa23ec-702d-4ca5-8c57-5325869e1136	nifi-app.log	27.00 bytes	00:00:01.358	00:00:34.008	No
30	8fdcfe3a-1d72-4128-873a-5206cb236ea	nifi-app.log	12.00 bytes	00:00:01.358	00:00:34.008	No

SUCCESS						
Displaying 100 of 2,064 (41.80 MB)						
Position	UUID	Filename	File Size	Queued Duration	Lineage Duration	Penalized
1	f3af0e3c-3d97-4ee5-89b9-7f09c1b044	nifi-app.log	20.38 KB	00:00:49.424	00:00:49.425	No
2	d393a03-c629-4c25-b20a-zea9af809c9	nifi-app.log	20.38 KB	00:00:49.423	00:00:49.424	No
3	eb6049df-e911-4360-9f08-078d27dbbc6	nifi-app.log	20.38 KB	00:00:49.422	00:00:49.423	No
4	49582424-a334-ba6-b8d5-0724fb16bc	nifi-app.log	20.38 KB	00:00:49.421	00:00:49.422	No
5	825486b5-1c73-45a6-8eb1-9e6a4817966	nifi-app.log	20.38 KB	00:00:49.420	00:00:49.421	No
6	fb28f90a-4a49-4927-ba0d-012a023a1	nifi-app.log	20.38 KB	00:00:49.419	00:00:49.420	No
7	37bf43c0-3c46-443e-9998-01e89f56793	nifi-app.log	20.38 KB	00:00:49.418	00:00:49.419	No
8	55a2ffdf-b88e-4c01-84d0-7e1892d73f78	nifi-app.log	20.38 KB	00:00:49.417	00:00:49.418	No
9	8c9b8c3a-2509-4d79-9ab0-5496658a3c0	nifi-app.log	20.38 KB	00:00:49.416	00:00:49.416	No
10	9de3002b-c084-4564-89b0-4ea45b275c	nifi-app.log	20.38 KB	00:00:49.414	00:00:49.415	No
11	66814625-464f-4b08-8317-2e7f6b31	nifi-app.log	20.38 KB	00:00:49.414	00:00:49.414	No
12	36171d4-e133-40b8-9767-2e7f6d50b0	nifi-app.log	20.38 KB	00:00:49.413	00:00:49.413	No
13	d43966ab-0ad4-40d7-8393-ede7-cf57b27	nifi-app.log	20.38 KB	00:00:49.412	00:00:49.412	No
14	da04d4b8-96a3-4770-9884-c487fe23af	nifi-app.log	20.38 KB	00:00:49.411	00:00:49.411	No
15	1edc3b25-b06b-497b-81c7-c5f4ea046c2	nifi-app.log	20.38 KB	00:00:49.410	00:00:49.410	No
16	568ec0d2-d814-421a-836a-d7758b119807	nifi-app.log	20.38 KB	00:00:49.409	00:00:49.409	No
17	05c4b246-00b0-49e8-82c4-53592b172900	nifi-app.log	20.38 KB	00:00:49.407	00:00:49.408	No
18	63c1dc41-0ba2-4195-9b94-5f7a5178490	nifi-app.log	20.38 KB	00:00:49.406	00:00:49.406	No
19	bde32fb0-3d41-46e8-b5c4-5a501e89b30	nifi-app.log	20.38 KB	00:00:49.405	00:00:49.405	No
20	5b2c8872-3b2e-4486-95e6-be208221172d	nifi-app.log	20.38 KB	00:00:49.404	00:00:49.404	No
21	87fac117-775c-4af9-8415-7e779b9516	nifi-app.log	20.38 KB	00:00:49.403	00:00:49.403	No
22	c2f4e6c4-3fe3-4ba8-939f-9fa87fe6400	nifi-app.log	20.38 KB	00:00:49.402	00:00:49.402	No
23	47fbfe7b-dac5-4e88-b126-5c85922599	nifi-app.log	20.38 KB	00:00:49.401	00:00:49.401	No
24	5c4e9907-517d-4c22-9605-d579be75df4	nifi-app.log	20.38 KB	00:00:49.400	00:00:49.400	No
25	df375bcd-8b9d-4990-6381-1a13d57a1c	nifi-app.log	20.38 KB	00:00:49.398	00:00:49.399	No
26	9742a6a-7201-4d3d-9cc5-e7b253fb3f7	nifi-app.log	20.38 KB	00:00:49.393	00:00:49.394	No
27	ff944an6-bfb4-4e0b-a683-1a21b6f114	nifi-app.log	20.38 KB	00:00:49.392	00:00:49.393	No
28	c59b4fe6-6152-408b-bd01-b54c1217092	nifi-app.log	20.38 KB	00:00:49.391	00:00:49.391	No
29	a39298ba-692c-41c6-b36d-b76e213d4e8	nifi-app.log	20.38 KB	00:00:49.390	00:00:49.390	No
30	8d719e75-7317-4e5e-87ed-68d9c70ee5e3	nifi-app.log	20.38 KB	00:00:49.389	00:00:49.389	No

→ Lire le topic "TP_MIAGE_5" depuis le début

Pour lire le topic "TP_MIAGE_5" depuis le début à l'aide de Kafka, nous utilisons le consommateur Kafka en mode console. La commande suivante démarrera le consommateur Kafka et lira les messages depuis le début (from-beginning) du topic "TP_MIAGE_5":

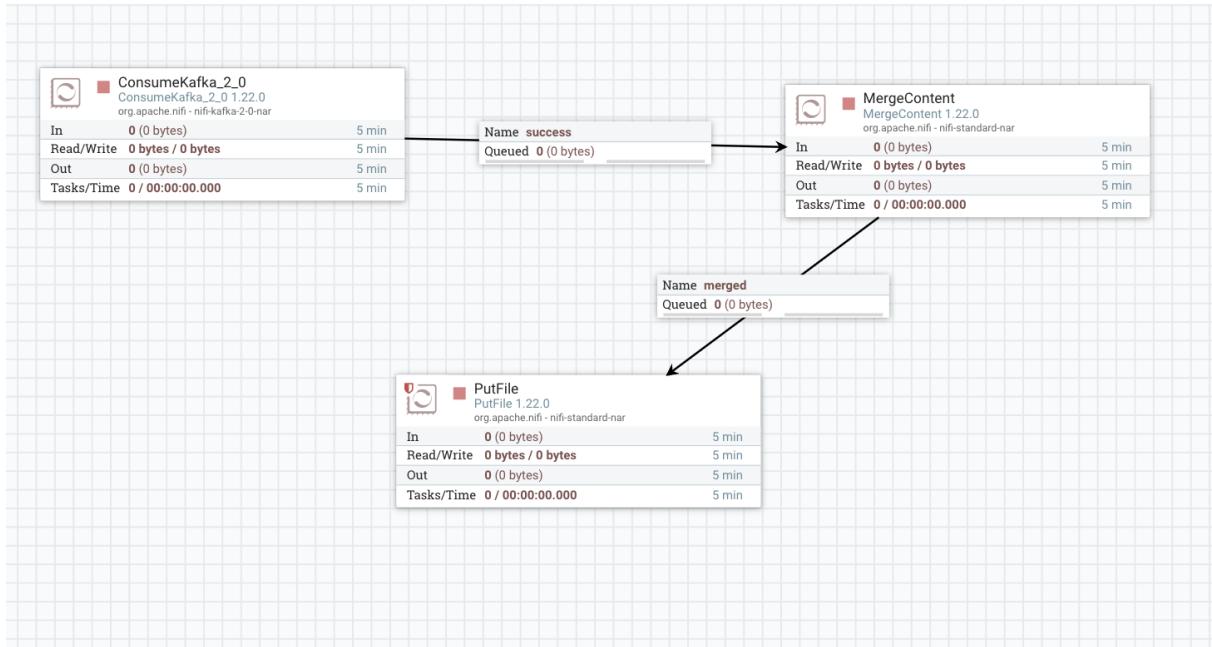
```
bin — java -Xmx512M -server -XX:+UseG1GC -XX:MaxGCPauseMillis=...
```

```
sofiene@client-172-18-108-54 bin % ./kafka-console-consumer.sh --bo
otstrap-server localhost:9092 --topic TP_MIAGE_5 --from-beginning

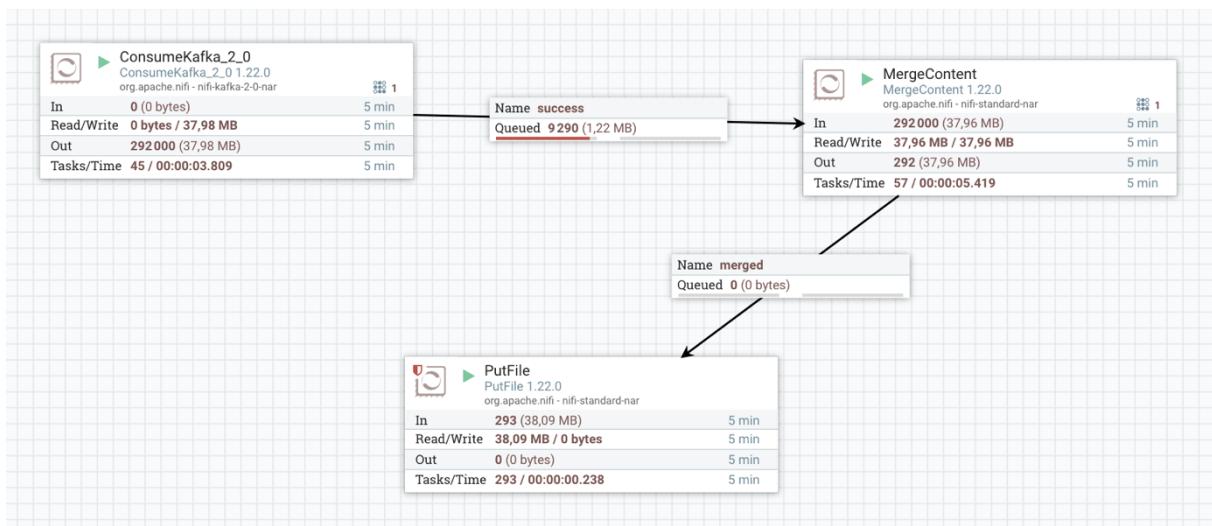
2023-12-06 15:00:02,408 INFO [Cleanup Archive for default] o.a.n.c.
repository.FileSystemRepository Successfully deleted 0 files (0 byt
es) from archive
2023-12-06 15:00:02,411 INFO [Cleanup Archive for default] o.a.n.c.
repository.FileSystemRepository Archive cleanup completed for conta
iner default; will now allow writing to this container. Bytes used
= 286,96 GB, bytes free = 173,47 GB, capacity = 460,43 GB
2023-12-06 15:00:17,562 INFO [pool-7-thread-1] o.a.n.c.r.WriteAhead
FlowFileRepository Initiating checkpoint of FlowFile Repository
2023-12-06 15:00:17,564 INFO [pool-7-thread-1] o.a.n.c.r.WriteAhead
FlowFileRepository Successfully checkpointed FlowFile Repository wi
th 2 records in 0 milliseconds
2023-12-06 15:00:28,098 INFO [Write-Ahead Local State Provider Main
tenance] org.wali.MinimalLockingWriteAheadLog org.wali.MinimalLocki
ngWriteAheadLog@7def08fc checkpointed with 61 Records and 0 Swap Fi
les in 18 milliseconds (Stop-the-world time = 9 milliseconds, Clear
Edit Logs time = 5 millis), max Transaction ID 794
2023-12-06 15:00:37,569 INFO [pool-7-thread-1] o.a.n.c.r.WriteAhead
FlowFileRepository Initiating checkpoint of FlowFile Repository
2023-12-06 15:00:37,569 INFO [pool-7-thread-1] o.a.n.c.r.WriteAhead
FlowFileRepository Successfully checkpointed FlowFile Repository wi
th 2 records in 0 milliseconds
2023-12-06 15:00:57,572 INFO [pool-7-thread-1] o.a.n.c.r.WriteAhead
FlowFileRepository Initiating checkpoint of FlowFile Repository
2023-12-06 15:00:57,573 INFO [pool-7-thread-1] o.a.n.c.r.WriteAhead
FlowFileRepository Successfully checkpointed FlowFile Repository wi
th 2 records in 0 milliseconds
2023-12-06 15:01:02,438 INFO [Cleanup Archive for default] o.a.n.c.
```

→ Compacter pendant 1mn les flowfiles lus et écrivez le résultat dans fichier texte dans un répertoire sur le disque local.

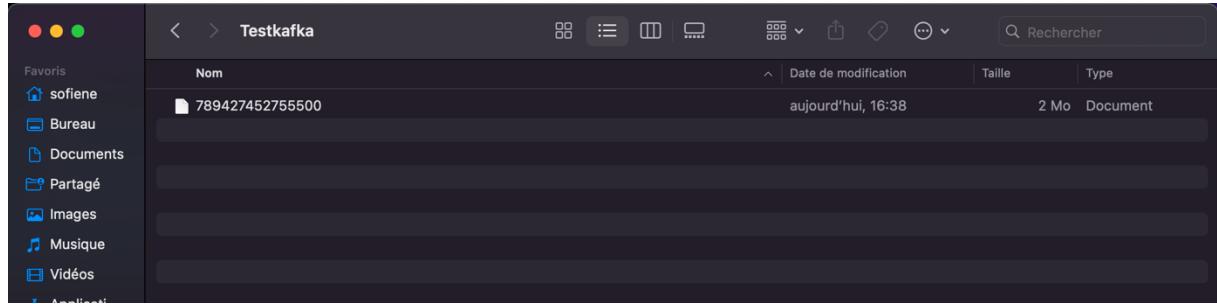
Pour cela, On utilise le processus MergeContent pour fusionner et compacter, et le processus consumeKafka pour récolter les données directement de Kafka. En fin, nous utilisons PutFile pour donner le chemin du répertoire où déposer le fichier compacter.



Une fois lancé on a la vue suivante :



Au bout d'une minute on peut arrêter le processus, et vérifier si on a bien 1 seul fichier qui regroupe toutes les lignes, comme ci-dessous :



Et on voit bien ici que notre fichier txt contient toute les lignes stockées dans Kafka :

```
789427452755500
2023-12-06 15:03:18,058 INFO [NiFi Web Server-143] o.a.n.controller.StandardProcessorNode
Starting GetFile[id=3f3ce800-018c-1000-76e4-b9556c10789a]2023-12-06 15:03:18,060 INFO [NiFi Web
Server-143] o.a.n.c.s.StandardProcessScheduler Starting SplitText[id=3f43fda5-018c-1000-
aed1-5b2422d23ac7]2023-12-06 15:03:18,060 INFO [NiFi Web Server-143]
o.a.n.controller.StandardProcessorNode Starting SplitText[id=3f43fda5-018c-1000-
aed1-5b2422d23ac7]2023-12-06 15:03:18,061 INFO [NiFi Web Server-143]
o.a.n.c.s.StandardProcessScheduler Starting
PublishKafka_2_0[id=3f546274-018c-1000-3adb-238db5d5f010]2023-12-06 15:03:18,061 INFO [NiFi Web
Server-143] o.a.n.controller.StandardProcessorNode Starting
PublishKafka_2_0[id=3f546274-018c-1000-3adb-238db5d5f010]2023-12-06 15:03:18,071 INFO [Timer-
Driven Process Thread-10] o.a.n.c.s.TimerDrivenSchedulingAgent Scheduled
PublishKafka_2_0[id=3f546274-018c-1000-3adb-238db5d5f010] to run with 1 threads2023-12-06
15:03:18,071 INFO [Timer-Driven Process Thread-8] o.a.n.c.s.TimerDrivenSchedulingAgent Scheduled
SplitText[id=3f43fda5-018c-1000-aed1-5b2422d23ac7] to run with 1 threads2023-12-06 15:03:18,071
INFO [Timer-Driven Process Thread-2] o.a.n.c.s.TimerDrivenSchedulingAgent Scheduled
GetFile[id=3f3ce800-018c-1000-76e4-b9556c10789a] to run with 1 threads2023-12-06 15:03:18,153
INFO [Timer-Driven Process Thread-3] o.a.k.clients.producer.ProducerConfig ProducerConfig
values: acks = all      batch.size = 16384      bootstrap.servers = [localhost:9092]
buffer.memory = 33554432      client.id =      compression.type = none      connections.max.idle.ms
= 540000      enable.idempotence = false      interceptor.classes = []      key.serializer =
class org.apache.kafka.common.serialization.ByteArraySerializer linger.ms = 0      max.block.ms =
5000      max.in.flight.requests.per.connection = 5      max.request.size = 1048576
metadata.max.age.ms = 300000      metric.reporters = []      metrics.num.samples = 2
metrics.recording.level = INFO      metrics.sample.window.ms = 30000      partitioner.class =
class org.apache.kafka.clients.producer.internals.DefaultPartitioner      receive.buffer.bytes =
32768      reconnect.backoff.max.ms = 1000      reconnect.backoff.ms = 50      request.timeout.ms =
30000      retries = 0      retry.backoff.ms = 100      sasl.client.callback.handler.class = null
sasl.jaas.config = null      sasl.kerberos.kinit.cmd = /usr/bin/kinit
sasl.kerberos.min.time.before.relogin = 60000      sasl.kerberos.service.name = null
sasl.kerberos.ticket.renew.jitter = 0.05      sasl.kerberos.ticket.renew.window.factor = 0.8
```

VI) Conclusion

Au cours de ce TP, nous avons configuré et manipulé un environnement Kafka en créant et gérant des topics, en produisant et consommant des messages via la ligne de commande et NiFi. Nous avons expérimenté les fonctionnalités avancées comme la gestion des groupes de consommateurs et la compaction des logs dans NiFi, ce qui a renforcé notre compréhension des flux de données et de la gestion des événements en temps réel. Ce TP a illustré l'importance de Kafka dans le traitement de données distribuées et la puissance de NiFi pour l'orchestration des flux de données.