

# Compte Rendu Lab3

## Apache Kafka

Nom et Prénom : AMRANI Hamza

Filière : IDF

## I. Installation et Démarrage de Kafka

### 1. Démarrage du Cluster Hadoop

Démarrer les conteneurs Docker :

```
docker start hadoop-master hadoop-slave1 hadoop-slave2
```

Accéder au conteneur master :

```
docker exec -it hadoop-master bash
```

### 2. Démarrage de Hadoop et Kafka

Lancer les démons YARN et HDFS :

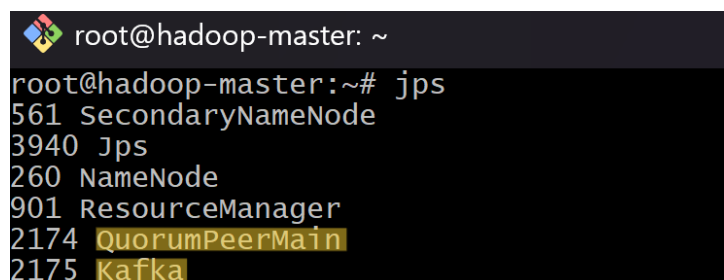
```
./start-hadoop.sh
```

Lancer Kafka et Zookeeper :

```
./start-kafka-zookeeper.sh
```

Vérifier que les démons sont démarrés :

```
jps
```



```
root@hadoop-master: ~  
root@hadoop-master:~# jps  
561 SecondaryNameNode  
3940 Jps  
260 NameNode  
901 ResourceManager  
2174 QuorumPeerMain  
2175 kafka
```

- **QuorumPeerMain (2174)** : Processus Zookeeper (coordination du cluster Kafka)
- **Kafka (2175)** : Processus broker Kafka (le serveur Kafka lui-même)

Le cluster Hadoop (HDFS + YARN) et Kafka avec Zookeeper sont correctement démarrés.

## II. Première Utilisation d'Apache Kafka

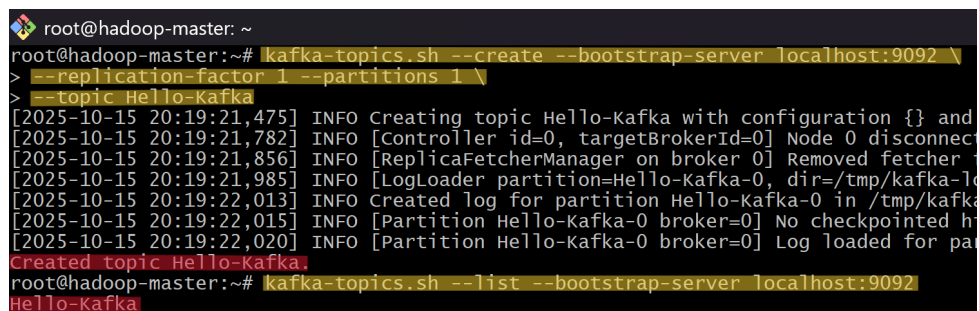
### 1. Création d'un Topic

Créer un nouveau topic appelé "Hello-Kafka" :

```
kafka-topics.sh --create --bootstrap-server localhost:9092 \
--replication-factor 1 --partitions 1 \
--topic Hello-Kafka
```

Afficher la liste des topics existants :

```
kafka-topics.sh --list --bootstrap-server localhost:9092
```

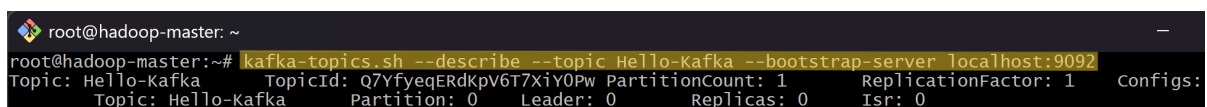


```
root@hadoop-master: ~
root@hadoop-master:~# kafka-topics.sh --create --bootstrap-server localhost:9092 \
> --replication-factor 1 --partitions 1 \
> --topic Hello-Kafka
[2025-10-15 20:19:21,475] INFO Creating topic Hello-Kafka with configuration {} and
[2025-10-15 20:19:21,782] INFO [Controller id=0, targetBrokerId=0] Node 0 disconnect
[2025-10-15 20:19:21,856] INFO [ReplicaFetcherManager on broker 0] Removed fetcher
[2025-10-15 20:19:21,985] INFO [LogLoader partition=Hello-Kafka-0, dir=/tmp/kafka-logs
[2025-10-15 20:19:22,013] INFO Created log for partition Hello-Kafka-0 in /tmp/kafka
[2025-10-15 20:19:22,015] INFO [Partition Hello-Kafka-0 broker=0] No checkpointed h
[2025-10-15 20:19:22,020] INFO [Partition Hello-Kafka-0 broker=0] Log loaded for pa
Created topic Hello-Kafka.
root@hadoop-master:~# kafka-topics.sh --list --bootstrap-server localhost:9092
Hello-Kafka
```

### 2. Description d'un Topic

Afficher les détails du topic créé :

```
kafka-topics.sh --describe --topic Hello-Kafka \
--bootstrap-server localhost:9092
```

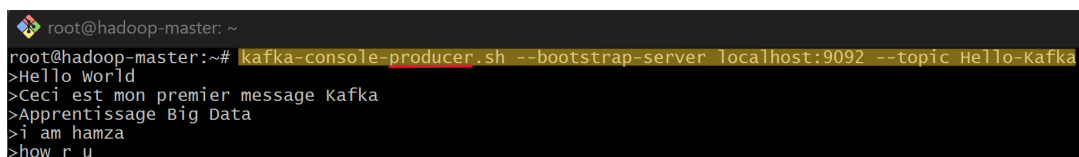


```
root@hadoop-master: ~
root@hadoop-master:~# kafka-topics.sh --describe --topic Hello-Kafka --bootstrap-server localhost:9092
Topic: Hello-Kafka      TopicId: Q7YfyegERdKpV6T7XiY0Pw PartitionCount: 1      ReplicationFactor: 1      Configs:
Topic: Hello-Kafka      Partition: 0      Leader: 0      Replicas: 0      Isr: 0
```

### 3. Écrire des Événements dans un Topic (Terminal 1)

Exécuter le producer de la console pour écrire des événements :

```
kafka-console-producer.sh --bootstrap-server localhost:9092 \
--topic Hello-Kafka
```

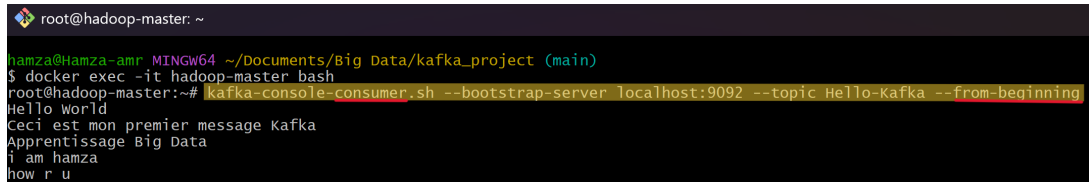


```
root@hadoop-master: ~
root@hadoop-master:~# kafka-console-producer.sh --bootstrap-server localhost:9092 --topic Hello-Kafka
>Hello world
>Ceci est mon premier message Kafka
>Apprentissage Big Data
>i am hamza
>how r u
```

## 4. Lire des Événements (Terminal 2)

dans une autre session de terminal en exécuter le consumer de la console :

```
kafka-console-consumer.sh --bootstrap-server localhost:9092 \
--topic Hello-Kafka --from-beginning
```

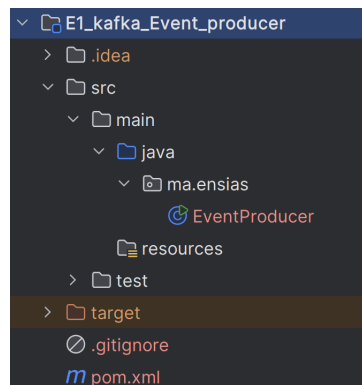


```
root@hadoop-master: ~
hamza@Hamza-amr MINGW64 ~/Documents/Big Data/kafka_project (main)
$ docker exec -it hadoop-master bash
root@hadoop-master:~# kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic Hello-Kafka --from-beginning
Hello World
Ceci est mon premier message Kafka
Apprentissage Big Data
i am hamza
how r u
```

## III. Création d'une Application Kafka

### 1. Structure du projet

Projet 1 : E1\_kafka\_Event\_Producer



### 2. Création du Producer (Terminal 1)

Créer la classe EventProducer.java qui envoie 10 messages au topic Kafka.  
Compiler le projet avec Maven :

```
mvn clean package
```



```
hamza@Hamza-amr MINGW64 ~/Documents/Big Data/kafka_project/E1_kafka_Event_producer (main)
$ mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] -----< ma.ensias:E1_kafka_Event_producer >-----
[INFO] Building E1_kafka_Event_producer 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 5.049 s
[INFO] Finished at: 2025-10-15T22:55:06+02:00
[INFO]
```

Copier le JAR dans le dossier partagé et l'exécuter :

```
java -jar /shared_volume/kafka/kafka-producer-app-jar-with-dependencies.jar Hello-Kafka
```

```
root@hadoop-master: ~
root@hadoop-master:~# java -jar /shared_volume/kafka-producer-app-jar-with-dependencies.jar Hello-Kafka
[main] INFO org.apache.kafka.clients.producer.ProducerConfig - Idempotence will be disabled because retries is set to 0.
[main] INFO org.apache.kafka.clients.producer.ProducerConfig - ProducerConfig values:
Message envoyé avec succès
[main] INFO org.apache.kafka.clients.producer.KafkaProducer - [Producer clientId=producer-1] Closing the Kafka producer with timeoutMillis = 9223372036854775807
[main] INFO org.apache.kafka.common.metrics.Metrics - Metrics scheduler closed
[main] INFO org.apache.kafka.common.metrics.Metrics - Closing reporter org.apache.kafka.common.metrics.JmxReporter
[main] INFO org.apache.kafka.common.metrics.Metrics - Metrics reporters closed
[main] INFO org.apache.kafka.common.utils.AppInfoParser - App info kafka.producer for producer-1 unregistered
```

Vérifier avec le consumer de la console :

```
kafka-console-consumer.sh --bootstrap-server localhost:9092 \
--topic Hello-Kafka --from-beginning
```

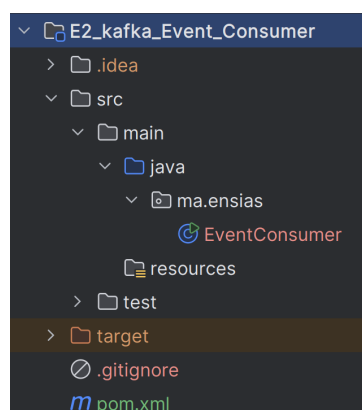
```
root@hadoop-master:~# kafka-console-consumer.sh --bootstrap-server localhost:9092 \
--topic Hello-Kafka --from-beginning
[2025-10-15 21:02:27,791] INFO [GroupCoordinator 0]: Dynamic member with unknown member
[2025-10-15 21:02:27,802] INFO [GroupCoordinator 0]: Preparing to rebalance group console-consumer
[2025-10-15 21:02:27,807] INFO [GroupCoordinator 0]: Stabilized group console-consumer
[2025-10-15 21:02:27,826] INFO [GroupCoordinator 0]: Assignment received from leader c
Hello World
Ceci est mon premier message Kafka
Apprentissage Big Data
i am hamza
how r u
0
1
2
3
4
5
6
7
8
9
```

### 3. Création du Consumer (Terminal 2)

Créer la classe `EventConsumer.java` qui lit les enregistrements du topic.

#### a) Structure du projet

Projet 2 : E2\_kafka\_Event\_Consumer



Compiler le projet avec Maven :

```
mvn clean package
```

```
hamza@Hamza-amr MINGW64 ~/Documents/Big Data/kafka_project/E2_kafka_Event_Consumer (main)
$ mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] -----< ma.ensias:E2_kafka_Event_Consumer >-----
[INFO] Building E2_kafka_Event_Consumer 1.0-SNAPSHOT
[INFO]    from pom.xml
[INFO] -----[ jar ]-----
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time:  5.771 s
[INFO] Finished at: 2025-10-15T23:12:32+02:00
[INFO] -----
```

Exécuter le consumer :

```
java -jar /shared_volume/kafka/kafka-consumer-app-jar-with-dependencies.jar Hello-Kafka
```

```
root@hadoop-master: ~
root@hadoop-master:~# java -jar /shared_volume/kafka-consumer-app-jar-with-dependencies.jar Hello-Kafka
[main] INFO org.apache.kafka.clients.consumer.ConsumerConfig - ConsumerConfig values:
allow.auto.create.topics = true
[main] INFO org.apache.kafka.clients.consumer.internals.ConsumerCoordinator - [Consumer clientId=consumer-test-1, groupId=
[main] INFO org.apache.kafka.clients.consumer.internals.ConsumerCoordinator - [Consumer clientId=consumer-test-1, groupId=
[main] INFO org.apache.kafka.clients.consumer.internals.SubscriptionState - [Consumer clientId=consumer-test-1, groupId=t
fsetEpoch=Optional.empty, currentLeader=LeaderAndEpoch{leader=Optional[hadoop-master:9092 (id: 0 rack: null)], epoch=0}}.
offset = 26, key = 0, value = 0
offset = 27, key = 1, value = 1
offset = 28, key = 2, value = 2
offset = 29, key = 3, value = 3
offset = 30, key = 4, value = 4
offset = 31, key = 5, value = 5
offset = 32, key = 6, value = 6
offset = 33, key = 7, value = 7
offset = 34, key = 8, value = 8
offset = 35, key = 9, value = 9
```

Le consumer affiche les messages avec leur offset, clé et valeur.

## IV. Kafka Connect

### 1. Objectif

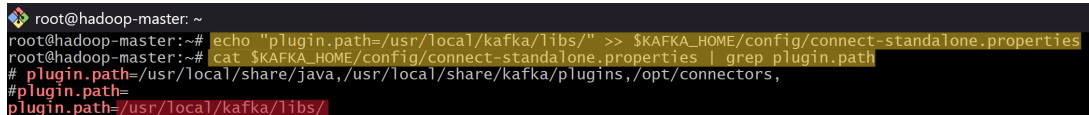
Cette section vise à mettre en œuvre Kafka Connect pour créer un pipeline de données qui importe des données depuis un fichier source vers un topic Kafka, puis exporte ces données vers un fichier destination (sink).

## 2. Configuration des Connecteurs

### a) Configuration du plugin path

Modifier le fichier de configuration pour ajouter le plugin path :

```
echo "plugin.path=/usr/local/kafka/libs/" >> $KAFKA_HOME/config/
connect-standalone.properties
```



```
root@hadoop-master: ~
root@hadoop-master:~# echo "plugin.path=/usr/local/kafka/libs/" >> $KAFKA_HOME/config/connect-standalone.properties
root@hadoop-master:~# cat $KAFKA_HOME/config/connect-standalone.properties | grep plugin.path
# plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/connectors,
#plugin.path=
plugin.path=/usr/local/kafka/libs/
```

Cette commande ajoute le chemin où Kafka pourra localiser les classes des connecteurs nécessaires.

### b) Configuration des fichiers de connecteurs

Les fichiers de configuration des connecteurs sont créés dans \$KAFKA\_HOME/config/ :

**Fichier connect-file-source.properties**

**Fichier connect-file-sink.properties**

### c) Vérification des fichiers de configuration

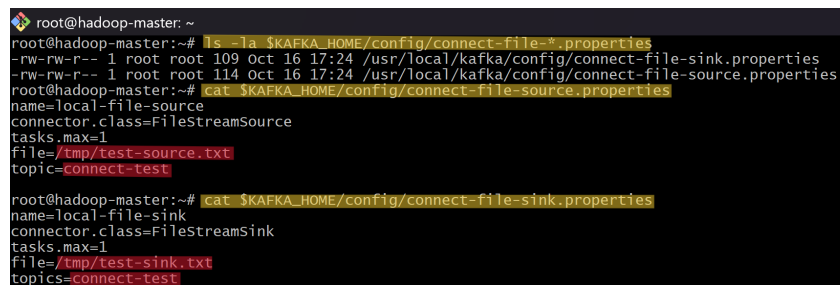
Lister les fichiers de configuration des connecteurs :

```
ls -la $KAFKA_HOME/config/connect-file-*.properties
```

Afficher le contenu des fichiers source et sink connector :

```
cat $KAFKA_HOME/config/connect-file-source.properties
```

```
cat $KAFKA_HOME/config/connect-file-sink.properties
```



```
root@hadoop-master: ~
root@hadoop-master:~# ls -la $KAFKA_HOME/config/connect-file-*.properties
-rw-rw-r-- 1 root root 109 Oct 16 17:24 /usr/local/kafka/config/connect-file-sink.properties
-rw-rw-r-- 1 root root 114 Oct 16 17:24 /usr/local/kafka/config/connect-file-source.properties
root@hadoop-master:~# cat $KAFKA_HOME/config/connect-file-source.properties
name=local-file-source
connector.class=FileStreamSource
tasks.max=1
file=/tmp/test-source.txt
topic=connect-test

root@hadoop-master:~# cat $KAFKA_HOME/config/connect-file-sink.properties
name=local-file-sink
connector.class=FileStreamSink
tasks.max=1
file=/tmp/test-sink.txt
topics=connect-test
```

## 3. Création du Topic Connect

Créer un nouveau topic pour Kafka Connect :

```
kafka-topics.sh --create --bootstrap-server localhost:9092 \
--replication-factor 1 --partitions 1 \
--topic connect-topic
```

```

root@hadoop-master: ~
root@hadoop-master:~# kafka-topics.sh --create --bootstrap-server localhost:9092 \
> --replication-factor 1 --partitions 1 \
> --topic connect-topic
[2025-10-16 17:14:04,502] INFO Creating topic connect-topic with configuration {} and
[2025-10-16 17:14:04,639] INFO [Controller id=0, targetBrokerId=0] Node 0 disconnecte
[2025-10-16 17:14:04,651] INFO [ReplicaFetcherManager on broker 0] Removed fetcher fo
[2025-10-16 17:14:04,666] INFO [LogLoader partition=connect-topic-0, dir=/tmp/kafka-l
[2025-10-16 17:14:04,668] INFO Created log for partition connect-topic-0 in /tmp/kafk
[2025-10-16 17:14:04,670] INFO [Partition connect-topic-0 broker=0] No checkpointed h
[2025-10-16 17:14:04,670] INFO [Partition connect-topic-0 broker=0] Log loaded for pa
Created topic connect-topic.
root@hadoop-master:~# kafka-topics.sh --list --bootstrap-server localhost:9092
Hello-Kafka
__consumer_offsets
connect-topic

```

Le topic connect-topic est créé avec succès.

## 4. Création du Fichier Source

(**Terminale 1**) Dans le terminal principal du conteneur hadoop-master, créer le fichier source avec des données initiales :

```

echo "Bonjour Kafka" > /tmp/test-source.txt
echo "Bienvenue dans le monde du streaming" >> /tmp/test-source.
txt

```

```

root@hadoop-master: ~
root@hadoop-master:~# echo "Bonjour Kafka" > /tmp/test-source.txt
root@hadoop-master:~# echo "Bienvenue dans le monde du streaming" >> /tmp/test-source.txt
root@hadoop-master:~# cat /tmp/test-source.txt
Bonjour Kafka
Bienvenue dans le monde du streaming

```

Le contenu du fichier est vérifié avec la commande `cat /tmp/test-source.txt`.

## 5. Démarrage de Kafka Connect

(**Terminale 1**) Ajouter des données au fichier source et Lancer Kafka Connect en mode standalone avec les deux connecteurs :

```

$KAFKA_HOME/bin/connect-standalone.sh \
$KAFKA_HOME/config/connect-standalone.properties \
$KAFKA_HOME/config/connect-file-source.properties \
$KAFKA_HOME/config/connect-file-sink.properties

```

Une fois lancé, Kafka Connect affiche plusieurs logs importants :

- Création du topic connect-topic avec la configuration spécifiée
- Déconnexion et reconnexion du contrôleur (Controller id=0)
- Gestion des répliques par le ReplicaFetcherManager
- Chargement des logs pour la partition connect-topic-0
- Création des logs dans le répertoire /tmp/kafka-logs

```
root@hadoop-master: ~  
root@hadoop-master:~# echo "Message 1" >> /tmp/test-source.txt  
root@hadoop-master:~# echo "Message 2" >> /tmp/test-source.txt  
root@hadoop-master:~# echo "Message 3" >> /tmp/test-source.txt  
root@hadoop-master:~# cat /tmp/test-source.txt  
Bonjour Kafka  
Bienvenue dans le monde du streaming  
Message 1  
Message 2  
Message 3  
root@hadoop-master:~# $KAFKA_HOME/bin/connect-standalone.sh \  
> $KAFKA_HOME/config/connect-standalone.properties \  
> $KAFKA_HOME/config/connect-file-source.properties \  
> $KAFKA_HOME/config/connect-file-sink.properties
```

## 6. Vérification du Processus

(Terminale 2) Vérifier que le processus ConnectStandalone est en cours d'exécution :

```
jps | grep Connect
```

```
root@hadoop-master: ~  
root@hadoop-master:~# jps | grep Connect  
12497 ConnectStandalone
```

## 7. Vérification du Fichier Destination

Visualiser le contenu du fichier de destination :

```
cat /tmp/test-sink.txt
```

```
root@hadoop-master: ~  
root@hadoop-master:~# cat /tmp/test-sink.txt  
Bonjour Kafka  
Bienvenue dans le monde du streaming  
Message 1  
Message 2  
Message 3
```

## 8. Test de Streaming en Temps Réel

Cette section démontre la capacité de Kafka Connect à propager les données en temps réel à travers le pipeline.



### a) Configuration de l'environnement de test

Pour observer le streaming en temps réel, nous utilisons deux terminaux simultanément :

- **Terminal 1 (en haut)** : Consumer Kafka en mode écoute continue
- **Terminal 2 (en bas)** : Terminal de commande pour modifier le fichier source

Dans le **Terminal 1**, lancer le consumer en mode continu :

```
kafka-console-consumer.sh --bootstrap-server localhost:9092 \
  --topic connect-test --from-beginning
```

Le consumer affiche immédiatement les messages déjà présents dans le topic au format JSON.

À ce stade, le Terminal 1 reste actif et attend de nouveaux messages.

### b) Ajout de données en temps réel

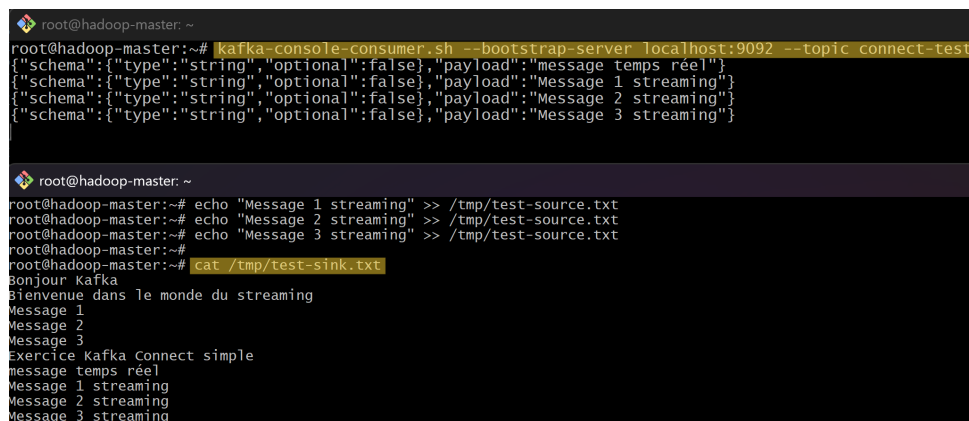
Dans le **Terminal 2**, ajouter de nouvelles lignes au fichier source :

```
echo "Message x streaming" >> /tmp/test-source.txt
```

**Observation immédiate dans le Terminal 1** : Les nouveaux messages apparaissent instantanément dans le consumer, démontrant le streaming en temps réel !

Vérifier ensuite le contenu du fichier sink :

```
cat /tmp/test-sink.txt
```



The screenshot shows two terminal windows on a Linux system. The top window (Terminal 1) shows the output of the `kafka-console-consumer.sh` command, displaying JSON messages from the `connect-test` topic. The bottom window (Terminal 2) shows the execution of `echo` commands to write messages to `/tmp/test-source.txt` and the `cat` command to read from `/tmp/test-sink.txt`. The output of `cat` shows the messages as they are received by the Sink Connector, demonstrating real-time streaming.

## 9. Analyse et Conclusion

Le pipeline Kafka Connect fonctionne correctement :

1. Le **Source Connector** lit continuellement le fichier `test-source.txt`
2. Les données sont publiées dans le topic `connect-topic`
3. Le **Sink Connector** consomme les messages et les écrit dans `test-sink.txt`
4. La propagation est quasi-instantanée, démontrant la capacité de streaming en temps réel de Kafka

Cette expérimentation illustre l'un des cas d'usage principaux de Kafka Connect : l'intégration de systèmes externes (fichiers, bases de données, etc.) avec Kafka sans écrire de code personnalisé.

## V. Application WordCount avec Kafka Streams

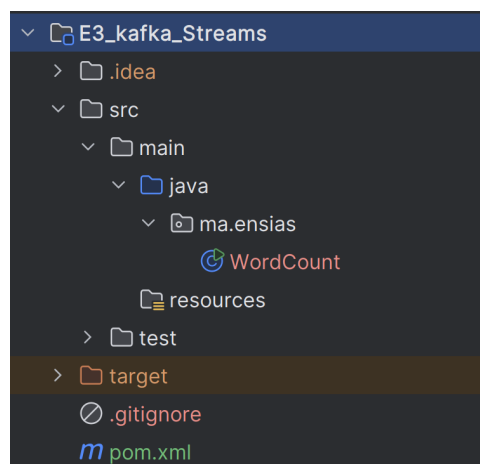
### 1. Objectif

Cette section vise à développer une application Kafka Streams qui lit des phrases depuis un topic Kafka (input-topic), compte la fréquence des mots, puis envoie les résultats vers un autre topic (output-topic).

### 2. Création de l'Application WordCount

#### Projet 3 : E3\_kafka\_Streams

Créer la classe `WordCountApp.java` qui implémente la logique de comptage des mots.



Compiler le projet avec Maven :

```
mvn clean package
```

### 3. Création des Topics

#### a) Création du topic input-topic

Créer le topic qui recevra les phrases à analyser :

```
kafka-topics.sh --create --bootstrap-server localhost:9092 \
  --replication-factor 1 --partitions 1 \
  --topic input-topic
```

#### b) Création du topic output-topic

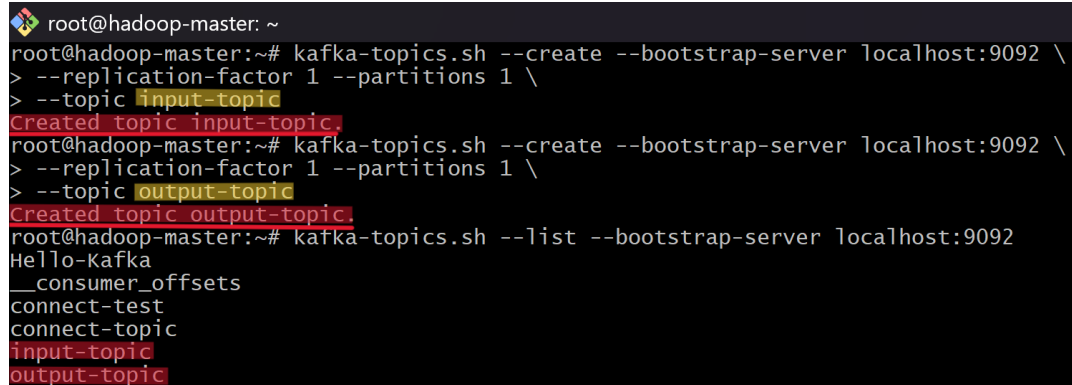
Créer le topic qui stockera les résultats du comptage de mots :

```
kafka-topics.sh --create --bootstrap-server localhost:9092 \
  --replication-factor 1 --partitions 1 \
  --topic output-topic
```

### c) Vérification des topics créés

Lister tous les topics existants pour confirmer la création :

```
kafka-topics.sh --list --bootstrap-server localhost:9092
```



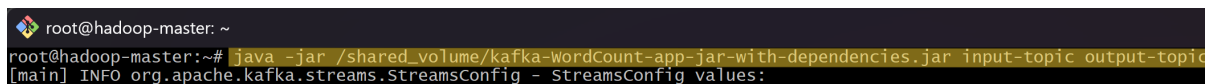
```
root@hadoop-master: ~
root@hadoop-master:~# kafka-topics.sh --create --bootstrap-server localhost:9092 \
> --replication-factor 1 --partitions 1 \
> --topic input-topic
Created topic input-topic.
root@hadoop-master:~# kafka-topics.sh --create --bootstrap-server localhost:9092 \
> --replication-factor 1 --partitions 1 \
> --topic output-topic
Created topic output-topic.
root@hadoop-master:~# kafka-topics.sh --list --bootstrap-server localhost:9092
Hello-Kafka
__consumer_offsets
connect-test
connect-topic
input-topic
output-topic
```

- input-topic : Topic d'entrée pour l'application Word Count (créé)
- output-topic : Topic de sortie pour les résultats du Word Count (créé)

## 4. Exécution

Lancer l'application WordCount :

```
java -jar /shared_volume/kafka/kafka-wordcount-app-jar-with-dependencies.jar input-topic output-topic
```



```
root@hadoop-master: ~
root@hadoop-master:~# java -jar /shared_volume/kafka-wordcount-app-jar-with-dependencies.jar input-topic output-topic
[main] INFO org.apache.kafka.streams.StreamsConfig - StreamsConfig values:
```

## 5. Test de l'Application

Dans un terminal, produire des messages :

```
kafka-console-producer.sh --bootstrap-server localhost:9092 \
--topic input-topic
```

Dans un autre terminal, consommer les résultats :

```
kafka-console-consumer.sh --topic output-topic --from-beginning \
--bootstrap-server localhost:9092 --property print.key=true
```

```

root@hadoop-master: ~
root@hadoop-master:~# kafka-console-consumer.sh --topic output-topic --from-beginning \
> --bootstrap-server localhost:9092 \
> --property print.key=true \
> --property key.separator=":"
world:1
hello:2
kafka:1
kafka:2
streams:1
is:1
powerful:1
hello:3
world:2
again:1

root@hadoop-master: ~
root@hadoop-master:~# kafka-console-producer.sh --bootstrap-server localhost:9092 \
> --topic input-topic
>hello world
>hello kafka
>kafka streams is powerful
>hello world again
>

```

Les résultats affichent chaque mot avec son comptage en temps réel.

## VI. Cluster Kafka Multi-Brokers

### 1. Configuration des Brokers

Pour mettre en place un cluster Kafka avec plusieurs brokers, nous avons créé deux fichiers de configuration supplémentaires à partir du fichier `server.properties` de base.

**Création des fichiers de configuration :**

```

cp $KAFKA_HOME/config/server.properties $KAFKA_HOME/config/
server-one.properties
cp $KAFKA_HOME/config/server.properties $KAFKA_HOME/config/
server-two.properties

```

```

root@hadoop-master: ~
root@hadoop-master:~# cp $KAFKA_HOME/config/server.properties $KAFKA_HOME/config/server-one.properties
root@hadoop-master:~# cp $KAFKA_HOME/config/server.properties $KAFKA_HOME/config/server-two.properties
root@hadoop-master:~# ls -la $KAFKA_HOME/config/server-*
-rw-r--r-- 1 root root 6896 Oct 17 09:00 /usr/local/kafka/config/server-one.properties
-rw-r--r-- 1 root root 6896 Oct 17 09:00 /usr/local/kafka/config/server-two.properties

```

**Configuration du Broker 1 (server-one.properties) :**

```

root@hadoop-master: ~
root@hadoop-master:~# vim $KAFKA_HOME/config/server-one.properties
root@hadoop-master:~# grep -E "broker.id|listeners|log.dirs" $KAFKA_HOME/config/server-one.properties
broker.id=1
listeners=PLAINTEXT://localhost:9093
log.dirs=/tmp/kafka-logs-1

```

**Configuration du Broker 2 (server-two.properties) :**

```

root@hadoop-master: ~
root@hadoop-master:~# vim $KAFKA_HOME/config/server-two.properties
root@hadoop-master:~# grep -E "broker.id|listeners|log.dirs" $KAFKA_HOME/config/server-two.properties
broker.id=2
listeners=PLAINTEXT://localhost:9094
log.dirs=/tmp/kafka-logs-2

```

## 2. Création des Répertoires de Logs

Avant de démarrer les brokers, nous avons créé les répertoires nécessaires pour stocker les logs Kafka :

```
mkdir -p /tmp/kafka-logs-1
mkdir -p /tmp/kafka-logs-2
```

```
root@hadoop-master: ~
root@hadoop-master:~# mkdir -p /tmp/kafka-logs-1
root@hadoop-master:~# mkdir -p /tmp/kafka-logs-2
root@hadoop-master:~# ls -ld /tmp/kafka-logs-*
drwxr-xr-x 2 root root 4096 oct 17 09:08 /tmp/kafka-logs-1
drwxr-xr-x 2 root root 4096 oct 17 09:08 /tmp/kafka-logs-2
```

La vérification avec `ls -ld` confirme que les deux répertoires ont été créés avec les permissions appropriées.

## 3. Démarrage des Brokers

Les deux nouveaux brokers ont été démarrés en parallèle dans des terminaux séparés :

**Terminal 1 - Broker 1 :**

```
$KAFKA_HOME/bin/kafka-server-start.sh $KAFKA_HOME/config/server-one.properties
```

**Terminal 2 - Broker 2 :**

```
$KAFKA_HOME/bin/kafka-server-start.sh $KAFKA_HOME/config/server-two.properties
```

```
root@hadoop-master: ~
root@hadoop-master:~# $KAFKA_HOME/bin/kafka-server-start.sh $KAFKA_HOME/config/server-two.properties
root@hadoop-master: ~
root@hadoop-master:~# $KAFKA_HOME/bin/kafka-server-start.sh $KAFKA_HOME/config/server-one.properties
```

## 4. Vérification du Cluster

Après le démarrage, nous avons vérifié que les trois brokers Kafka (le broker initial sur le port 9092 plus les deux nouveaux sur les ports 9093 et 9094) sont actifs :

```
jps | grep Kafka
```

```
root@hadoop-master: ~
root@hadoop-master:~# jps | grep Kafka
21832 Kafka
21371 Kafka
2175 Kafka
```

La sortie montre trois processus Kafka en cours d'exécution, confirmant que notre cluster multi-brokers est opérationnel.

## 5. Création d'un Topic Répliqué

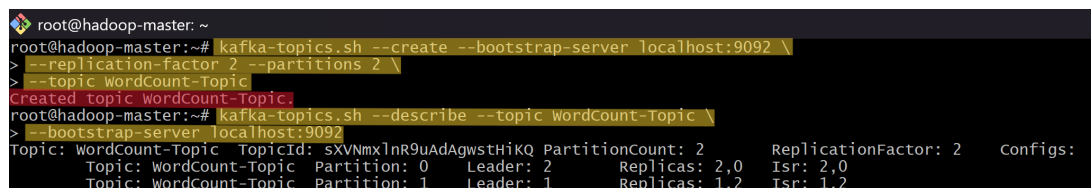
Nous avons créé un topic nommé `WordCount-Topic` avec un facteur de réplication de 2 et 2 partitions pour profiter de la haute disponibilité offerte par le cluster :

```
kafka-topics.sh --create --bootstrap-server localhost:9092 \
--replication-factor 2 --partitions 2 \
--topic WordCount-Topic
```

### a) Vérification de la Réplication

Pour vérifier la configuration du topic et sa réplication sur les différents brokers, nous avons utilisé la commande `describe` :

```
kafka-topics.sh --describe --topic WordCount-Topic \
--bootstrap-server localhost:9092
```



```
root@hadoop-master: ~
root@hadoop-master:~# kafka-topics.sh --create --bootstrap-server localhost:9092 \
--replication-factor 2 --partitions 2 \
--topic WordCount-Topic
Created topic WordCount-Topic.
root@hadoop-master:~# kafka-topics.sh --describe --topic WordCount-Topic \
--bootstrap-server localhost:9092
Topic: WordCount-Topic TopicId: sXVNmXlnR9uAdAgwstHiKQ PartitionCount: 2 ReplicationFactor: 2 Configs:
Topic: WordCount-Topic Partition: 0 Leader: 2 Replicas: 2,0 Isr: 2,0
Topic: WordCount-Topic Partition: 1 Leader: 1 Replicas: 1,2 Isr: 1,2
```

Le message *"Created topic WordCount-Topic"* confirme la création réussie du topic. La sortie montre :

1. Le topic possède 2 partitions
2. Chaque partition est répliquée sur 2 brokers différents

## 6. Application WordCount Interactive

### a) Architecture de l'Application

Nous avons développé une application `WordCount` interactive composée de deux classes Java :

- **WordProducer (E4\_kafka\_producer)** : Lit du texte depuis le clavier, découpe les phrases en mots, et envoie chaque mot individuellement dans le topic Kafka
- **WordCountConsumer (E4\_kafka\_consumer)** : Consomme les messages du topic et affiche la fréquence de chaque mot en temps réel dans une structure de données en mémoire

Les deux applications se connectent au cluster Kafka en utilisant tous les brokers disponibles pour garantir la résilience :

```
props.put("bootstrap.servers", "localhost:9092,localhost:9093,localhost:9094");
```

## b) Compilation et Packaging

Deux JARs distincts ont été créés avec Maven, chacun configuré avec sa propre classe principale :

```
mvn clean package
```

Les JARs résultants ont été copiés dans le volume partagé pour être accessibles depuis le conteneur :

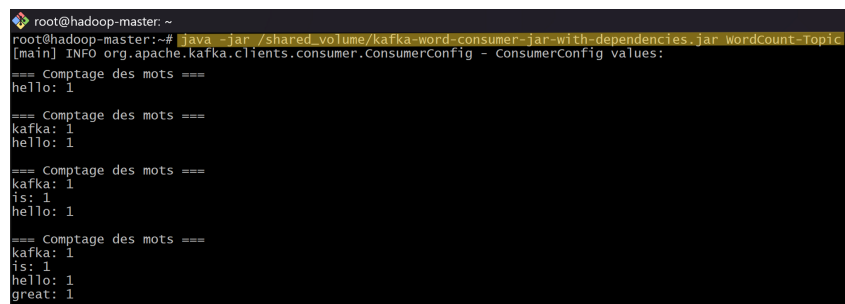
- kafka-word-producer-jar-with-dependencies.jar
- kafka-word-consumer-jar-with-dependencies.jar

## c) Exécution de l'Application WordCount

### Terminal 1 - Consumer :

Le consumer a été démarré en premier pour être prêt à recevoir les messages :

```
java -jar /shared_volume/kafka/kafka-word-consumer-jar-with-  
dependencies.jar \  
WordCount-Topic
```



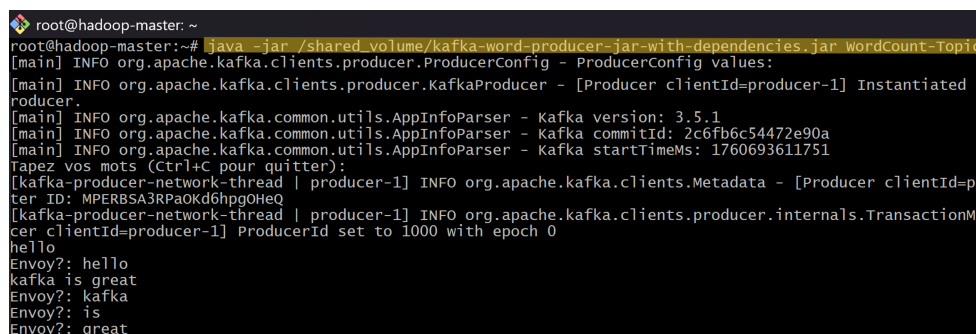
```
root@hadoop-master: ~  
root@hadoop-master:~# java -jar /shared_volume/kafka-word-consumer-jar-with-dependencies.jar WordCount-Topic  
[main] INFO org.apache.kafka.clients.consumer.ConsumerConfig - ConsumerConfig values:  
=== Comptage des mots ===  
hello: 1  
  
=== Comptage des mots ===  
kafka: 1  
hello: 1  
  
=== Comptage des mots ===  
kafka: 1  
is: 1  
hello: 1  
  
=== Comptage des mots ===  
kafka: 1  
is: 1  
hello: 1  
great: 1
```

La Figure montre le consumer en cours d'exécution, affichant les comptages de mots au fur et à mesure qu'ils arrivent.

### Terminal 2 - Producer :

Le producer a été lancé et attend la saisie utilisateur :

```
java -jar /shared_volume/kafka/kafka-word-producer-jar-with-  
dependencies.jar \  
WordCount-Topic
```



```
root@hadoop-master: ~  
root@hadoop-master:~# java -jar /shared_volume/kafka-word-producer-jar-with-dependencies.jar WordCount-Topic  
[main] INFO org.apache.kafka.clients.producer.ProducerConfig - ProducerConfig values:  
[main] INFO org.apache.kafka.clients.producer.KafkaProducer - [Producer clientId=producer-1] Instantiated producer.  
[main] INFO org.apache.kafka.common.utils.AppInfoParser - kafka version: 3.5.1  
[main] INFO org.apache.kafka.common.utils.AppInfoParser - kafka commitId: 2c6fb6c54472e90a  
[main] INFO org.apache.kafka.common.utils.AppInfoParser - kafka startTimeMs: 1760693611751  
Tapez vos mots (Ctrl+C pour quitter):  
[kafka-producer-network-thread | producer-1] INFO org.apache.kafka.clients.Metadata - [Producer clientId=producer-1] Metadata for kafka: 1760693611751  
[kafka-producer-network-thread | producer-1] INFO org.apache.kafka.clients.producer.internals.TransactionManager - [Producer clientId=producer-1] ProducerId set to 1000 with epoch 0  
hello  
Envoy?: hello  
kafka is great  
Envoy?: kafka  
Envoy?: is  
Envoy?: great
```

La Figure illustre le producer en action. L'utilisateur a saisi la phrase "hello kafka is great" et chaque mot a été envoyé individuellement au topic Kafka.