Formation 4SH

Kafka - Fundamentals

Exercise Book

Version 1.2

Introduction

Introdution

- Repository : https://github.com/4sh/kafka-training
- Exécutez la commande docker-compose pull afin de télécharger les images nécessaires
- Au début de chaque exercice, vous devrez démarrer les containers nécessaires avec la commande docker-compose up
- Si a un moment vous souhaitez réinitialiser votre cluster, vous pouvez exécuter la commande docker-compose down et refaire le docker-compose up du début du TP
- Control Center => http://localhost:9021

TP 1 - Fundamentals

TP1 - Use Command-Line Tools

Produce and Consume messages

Kafka possède des utilitaires pour produire et lire des messages depuis un **Topic**.

1. Ouvrez un terminal

2. Démarrez le cluster **Kafka**

```
$ docker-compose up -d training-zookeeper training-kafka control-center
```

3. Exécutez la commande suivante pour vous connecter au **Broker**

```
$ docker exec -it kafka bash
```

Produce and Consume messages

4. Avant de commencer, il faut créer un topic en utilisant la commande kafka-topics. Exécutez la commande suivante pour afficher la documentation

```
$ kafka-topics
```

5. Exécutez maintenant la commande suivante pour créer le premier Topic

```
$ kafka-topics --bootstrap-server training-kafka:29092 \
      --create \
      --partitions 1 \
      --replication-factor 1 \
      --topic testing
```

Le **Topic** est créé avec 1 **Partition** et un replication factor de 1. Il était possible d'activer l'auto-creation des Topics qui aurait pu éviter d'exécuter la commande ci-dessus. L'auto-creation des Topics est fortement déconseillée en production !

TP1 - Use Command-Line Tools

Produce and Consume messages

6. Passons maintenant à la publication de **Messages** dans le **Topic** que nous venons de créer :

```
$ kafka-console-producer
```

Cela va vous afficher les paramètres de la commande

7. Exécutez maintenant la commande suivante :

```
$ kafka-console-producer --bootstrap-server training-kafka:29092 --topic testing
```

Cela va vous ouvrir un prompt. Saisissez plusieurs messages.

Produce and Consume messages

8. Nous allons maintenant utiliser un **Consumer** pour lire les **Messages** que nous venons de produire. Dans un nouveau terminal, exécutez la commande suivante :

```
$ kafka-console-consumer
```

Cela va vous afficher les paramètres de la commande.

9. Exécutez maintenant la commande suivante :

```
$ kafka-console-consumer \
      --bootstrap-server training-kafka:29092 \
      --from-beginning \
      --topic testing
```

Vous devriez commencer à voir des **Messages** apparaître.
Dans le terminal du **Producer**, vous pouvez continuer à saisir des **Messages**. Vous devriez les voir arriver dans le terminal du **Consumer** rapidement.

Une fois que vous avez terminé, fermez les **Producer** et **Consumer** avec Ctrl+C

TP1 - Use Command-Line Tools

Record Keys

Par défaut, les commandes kafka-console-producer et kafka-console-consumer considèrent que les **Keys** sont nulles.

1. Relancez le Producer avec les arguments suivants :

```
$ kafka-console-producer \
      --bootstrap-server training-kafka:29092 \
      --topic testing \
      --property parse.key=true \
      --property key.separator=,
```

2. Saisissez quelques **Messages** comme par exemple :

```
> 1,my first record
> 2,another record
> 3,Kafka is cool
```

Record Keys

3. Lancez maintenant le Consumer avec des paramètres supplémentaires pour afficher les **Keys** :

```
$ kafka-console-consumer \
      --bootstrap-server training-kafka:29092 \
      --from-beginning \
      --topic testing \
      --property print.key=true
```

4. Les Messages devraient s'afficher de la manière suivante :

```
null  some data
null  more data
null  final data
1     my first record
2     another record
3     Kafka is cool
```

Les **Messages** avec une **Key** nulle sont ceux que vous avez créés au début.

TP1 - Use Command-Line Tools

Zookeeper Shell

1. Les données **Kafka** qui sont dans **Zookeeper** peuvent être visualisées avec la commande zookeeper-shell :

```
$ zookeeper-shell training-zookeeper
Connecting to zookeeper
Welcome to ZooKeeper!
JLine support is disabled
WATCHER::
WatchedEvent state:SyncConnected type:None path:null
```

2. Depuis le shell **Zookeeper**, exécutez la commande ls / pour voir la structure des répertoires dans Zookeeper

```
ls /
[admin, brokers, cluster, config, consumers, controller,
controller_epoch, isr_change_notification, latest_producer_id_block,
log_dir_event_notification, zookeeper]
```

Zookeeper Shell

3. Saisissez ls /brokers pour afficher le niveau suivant dans la structure des répertoires :

```
ls /brokers
[ids, seqid, topics]
```

4. Saisissez ls /brokers/ids pour afficher la liste des IDs des **Brokers** du cluster :

```
ls /brokers/ids
[1]
```

5. Saisissez get /brokers/ids/1 pour afficher les métadatas du **Broker** 1 :

```
get /brokers/ids/1
{"listener_security_protocol_map":{"PLAINTEXT":"PLAINTEXT"},"endpoint
s":["PLAINTEXT://kafka:9092"],"jmx_port":-
1,"host":"kafka","timestamp":"1581126250804","port":9092,"version":4}
```

Zookeeper Shell

6. Saisissez `get /brokers/topics/testing/partitions/0/state` pour afficher les métadatas de la **Partition** 0 du **Topic** `testing` :

```
get /brokers/topics/testing/partitions/0/state
{"controller_epoch":1,"leader":101,"version":1,"leader_epoch":0,"isr"
:[101]}
```

7. Appuyez sur `Ctrl+D` pour quitter le shell **Zookeeper**

TP1 - Use Command-Line Tools

TP 2 - Produce Messages to Kafka

TP2 - Produce Messages to Kafka

Produce Messages

L'objectif de ce TP est de créer un **Producer**. Ce **Producer** va lire un fichier qui contient des latng et les envoyer dans le **Topic** driver-positions. L'application boucle à l'infini sur le fichier CSV.
L'ID du driver est envoyé en clé, et la latng est envoyée en value (une simple string avec une virgule entre la latitude et la longitude).

Key	Value
driver-1	47.5952,-122.3316

1. Démarrez le cluster **Kafka**

```
$ docker-compose up -d training-zookeeper training-kafka control-center create-topics
```

Notez la présence du container create-topics . Il crée tous les **Topics** qui serviront aux différents TPs et se termine.

Produce Messages

2. Ouvrez le projet dans IntelliJ et ouvrez la classe labs/tp2-producer/src/main/java/clients/Producer.java :

3. Il y a plusieurs TODO dans la classe qu'il faut compléter pour finaliser l'implémentation

4. N'hésitez pas à lancer à tout moment le Producer avec la configuration TP2 - Run Producer

5. Une fois que vous avez fini l'implémentation, vous devriez avoir ce genre d'ouput dans la console

```
Starting Java producer.
Sent Key:driver-1 Value:47.618579,-122.355081
Sent Key:driver-1 Value:47.618577152452055,-122.35520620652974
Sent Key:driver-1 Value:47.61857902704408,-122.35507321130525
Sent Key:driver-1 Value:47.618579488930855,-122.35494018791431
Sent Key:driver-1 Value:47.61857995081763,-122.35480716452278
...
```

6. Vous pouvez utiliser la commande kafka-console-consumer pour visualiser les Messages qui ont été publiés dans le Topic driver-positions :

Produce Messages

6. Vous pouvez utiliser la commande kafka-console-consumer pour visualiser les Messages qui ont été publiés dans le Topic driver-positions :

```
$ kafka-console-consumer --bootstrap-server training-kafka:9092 \
      --topic driver-positions \
      --property print.key=true \
      --from-beginning
```

7. Si nécessaire, vous trouverez la solution ici solutions/tp2-producer-solution/src/main/java/clients/Producer.java :

8. Lorsque votre Producer comment la production de **Messages** n'hésitez pas à ouvrir Control Center et afin de voir les **Messages** arriver.

Bonus : Vous pouvez retirer le Thread.sleep() et tuner le batching du **Producer** afin de produire un très grand volume de Messages. ⚠️ Ne laissez pas le **Producer** produire trop longtemps de cette manière, vous pourriez être surpris du volume de données qui va être généré.

TP2 - Produce Messages to Kafka

TP 3 - Consume Messages from Kafka

TP3 - Consume Messages from Kafka

L'objectif de ce TP est de créer un **Consumer**. Ce **Consumer** va lire les Messages qui ont été publiés précédemment dans le **Topic** driver-positions.

1. Démarrez le cluster **Kafka**

```
$ docker-compose up -d training-zookeeper training-kafka control-center create-topics
```

2. Ouvrez la classe labs/tp3-consumer/src/main/java/clients/Consumer.java

3. Il y a plusieurs TODO dans la classe qu'il faut compléter pour finaliser l'implémentation

4. N'hésitez pas à lancer à tout moment le Producer avec la configuration TP3 - Run Consumer

TP3 - Consume Messages from Kafka

5. Une fois que vous avez fini l'implémentation, vous devriez avoir ce genre d'ouput dans la console

```
Starting Java Consumer.
Key:driver-1 Value:47.618579,-122.355081 [partition 1]
Key:driver-1 Value:47.618577152452055,-122.35520620652974 [partition1][offset 0]
Key:driver-1 Value:47.61857902704408,-122.35507321130525 [partition1][offset 1]
Key:driver-1 Value:47.618579488930855,-122.35494018791431 [partition1][offset 2]
Key:driver-1 Value:47.61857995081763,-122.35480716452278 [partition1][offset 3]
...
```

6. Si nécessaire, vous trouverez la solution ici solutions/tp3-consumer-solution/src/main/java/clients/Consumer.java

7. Lorsque votre Consumer débute la consommation des **Messages** n'hésitez pas à ouvrir Control Center et constatez la diminution du lag

TP3 - Consume Messages from Kafka

TP 4 - Schema Management

TP 4 - Schema Management

L'objectif de ce TP est de modifier notre **Producer** pour écrire dans le **Topic** driver-positions-avro au format Avro.

1. Démarrez le cluster **Kafka**

```
$ docker-compose up -d training-zookeeper training-kafka control-center schema-registry
create-topics
```

2. Inspected le schéma suivant : labs/tp4-producer-avro/src/main/avro/position_value.avsc.

3. Ouvrez la classe labs/tp4-producer-avro/src/main/java/clients/Producer.java

Vous constaterez qu'il y a des erreurs sur l'import de la classe PositionValue. Il faut que vous lanciez une compilation pour que le plugin Maven avro-maven-plugin génère la classe à partir du schéma.

Vous pouvez utiliser la configuration TP4 - Compile Producer afin de lancer la compilation.

TP 4 - Schema Management

4. Ensuite, vous pouvez démarrer le Producer avec la configuration TP4 - Run Producer . Vous devriez voir les logs suivants dans la console :

```
Connected to the target VM, address: '127.0.0.1:33737', transport: 'socket'
Starting Java Avro producer.
Sent Key:driver-1 Latitude:47.60855272900047 Latitude:-122.3351861842602
Sent Key:driver-1 Latitude:47.60857966448788 Latitude:-122.3352112142356
Sent Key:driver-1 Latitude:47.6086171168696 Latitude:-122.335245921525
```

5. Connectez vous au container du Schema Registry avec la commande suivante :

```
$ docker exec -it schema-registry bash
```

TP 4 - Schema Management

6. Exécutez ensuite la commande suivante afin de visualiser les **Messages** qui sont créés dans le **Topic** driver-positions-avro :

```
$ kafka-avro-console-consumer --bootstrap-server training-kafka:29092 \
      --property schema.registry.url=http://schema-registry:8081 \
      --topic driver-positions-avro --property print.key=true \
      --key-deserializer=org.apache.kafka.common.serialization.StringDeserializer \
      --from-beginning
```

7. Vous devriez voir des **Messages** apparaître dans la console :

```
driver-1    {"latitude":47.60855272900047,"longitude":-122.3351861842602}
driver-1    {"latitude":47.60857966448788,"longitude":-122.3352112142356}
driver-1    {"latitude":47.6086171168696,"longitude":-122.335245921525}
driver-1    {"latitude":47.60864062123478,"longitude":-122.335267358756}
driver-1    {"latitude":47.60867273496408,"longitude":-122.3352965444222}
…
```

TP 4 - Schema Management

8. Faites de même avec la classe labs/tp4-consumer-avro/src/main/java/clients/Consumer.java et démarrez le Consumer avec la configuration TP4 - Run Consumer

9. Tout comme précédemment, vous devriez voir des **Messages** apparaître dans la console :

```
driver-1    {"latitude":47.60855272900047,"longitude":-122.3351861842602}
driver-1    {"latitude":47.60857966448788,"longitude":-122.3352112142356}
driver-1    {"latitude":47.6086171168696,"longitude":-122.335245921525}
driver-1    {"latitude":47.60864062123478,"longitude":-122.335267358756}
driver-1    {"latitude":47.60867273496408,"longitude":-122.3352965444222}
…
```

10. **Bonus** : essayez de faire évoluer le schéma côté Producer en renommant puis rajoutant un champ. Pensez à bien faire une compilation avant de relancer le Producer.

Vous devriez avoir des erreurs, notamment si vous renommez un champ. Essayez de faire un changement qui soit **BACKWARD** compatible

TP2 - Produce Messages to Kafka

TP 5 - Stream Processing with Kafka Streams

TP 5 - Stream Processing

L'objectif de ce TP est d'effectuer des opérations Kafka Streams stateless sur le Topic driver-positions-kstreams-avro et d'envoyer le résultat dans le Topic driver-positions-string-avro.

1. Démarrez le cluster **Kafka**

```
$ docker-compose up -d training-zookeeper training-kafka control-center schema-registry
create-topics
```

2. Inspected les schémas dans le répertoire : labs/tp5-streams-avro/src/main/avro/

3. Ouvrez la classe labs/tp5-streams-avro/src/main/java/clients/StreamsApp.java

Il y a plusieurs TODO dans la classe qu'il faut compléter pour finaliser l'implémentation

4. Lancez les configurations suivantes en parallèle afin de générer des positions pour plusieurs drivers en même temps : TP5 - Run Producer - Driver 1 , TP5 - Run Producer - Driver 2 et TP5 - Run Producer - Driver 3

TP 5 - Stream Processing

5. Vous devriez voir apparaître dans votre console les logs que vous avez ajouté dans la méthode peek()

6. Exécutez ensuite la commande suivante afin de visualiser les **Messages** qui sont créés dans le **Topic** driver-positions-string-avro :

```
$ kafka-avro-console-consumer --bootstrap-server training-kafka:29092 \
      --property schema.registry.url=http://schema-registry:8081 \
      --topic driver-positions-string-avro --property print.key=true \
      --key-deserializer=org.apache.kafka.common.serialization.StringDeserializer \
      --from-beginning
```

7. Vous devriez voir des **Messages** apparaître dans la console :

```
driver-3    {"latitude":47.60869232006395,"longitude":-122.3353153594997,"positionString":"Latitude: 47.60869232006395, Longitude: -122.3353153594997"}
driver-1    {"latitude":47.60956975866959,"longitude":-122.3361153285211,"positionString":"Latitude: 47.60956975866959, Longitude: -122.3361153285211"}
driver-3    {"latitude":47.60869519052452,"longitude":-122.3353213183663,"positionString":"Latitude: 47.60869519052452, Longitude: -122.3353213183663"}
driver-1    {"latitude":47.60959596541749,"longitude":-122.3361412328432,"positionString":"Latitude: 47.60959596541749, Longitude: -122.3361412328432"}
driver-3    {"latitude":47.60870240392993,"longitude":-122.3353255266773,"positionString":"Latitude: 47.60870240392993, Longitude: -122.3353255266773"}
driver-1    {"latitude":47.6096065444621,"longitude":-122.3361475992628,"positionString":"Latitude: 47.6096065444621, Longitude: -122.3361475992628"}
driver-3    {"latitude":47.6087046538951,"longitude":-122.3353272995542,"positionString":"Latitude: 47.6087046538951, Longitude: -122.3353272995542"}
```

TP2 - Produce Messages to Kafka

TP 6 - Event Streaming Apps with ksqlDB

TP 6 - ksqlDB

L'objectif de ce TP est de publier dans un nouveau **Topic** des données enrichies du **Topic** driver-positions-ksql-avro à l'aide de **ksqlDB**.

1. Démarrez le cluster **Kafka**

```
$ docker-compose up -d
```

2. Les containers ksql-producer-1 ksql-producer-2 et ksql-producer-3 produisent des données dans le **Topic** driver-positions-ksql-avro.

3. Connectez vous ensuite au CLI **ksqlDB** :

```
$ docker exec -it ksql-cli ksql http://ksqldb-server:8088
```

4. Si vous êtes bien connecté, vous devriez voir le message suivant apparaître :

```
                  ===========================================
                  =                                         =
                  =    _            _ ____    ____         =
                  =   | | _____  __ _| |  _ \  | __ )        =
                  =   | |/ / __|/ _` | | | | | |  _ \        =
                  =   |   <\__ \ (_| | | |_| | | |_) |       =
                  =   |_|\_\___/\__, |_|____/  |____/        =
                  =                |_|                      =
                  =    The Database purpose-built           =
                  =    for stream processing apps            =
                  ===========================================

Copyright 2017-2022 Confluent Inc.

CLI v7.3.1, Server v7.3.1 located at http://ksqldb-server:8088
Server Status: RUNNING

Having trouble? Type 'help' (case-insensitive) for a rundown of how things work!

ksql>
```

5. Depuis un nouveau terminal, connectez vous au **Broker** :

```
$ docker exec -it training-kafka bash
```

6. Ensuite, il faut produire des messages dans le **Topic** driver-profiles-ksql qui vont servir à faire des jointures avec le Topic driver-positions-ksql-avro :

```
$ kafka-console-producer \
      --bootstrap-server training-kafka:29092 \
      --topic driver-profiles-ksql \
      --property parse.key=true \
      --property key.separator=':'
```

7. Créez 3 messages afin d'avoir un profil pour chaque driver :

```
driver-1:Jean|Bonbeurre|Renault|Clio
driver-2:Jacky|Tuning|Citroën|2CV
driver-3:Manu|Cronma|Ferrari|Enzo
```

8. Vous pouvez ensuite quitter kafka-console-producer en appuyant sur Ctrl+C

9. Vous pouvez maintenant revenir sur votre terminal avec le CLI **ksqlDB**. Commencez par configurer la propriété auto.offset.reset à earliest. Cela va indiquer à **ksqlDB** de se placer au début des **Topics** lorsque l'on va exécuter des requêtes :

```
ksql> SET 'auto.offset.reset' = 'earliest';
```

10. Il faut maintenant créer une **Table** dans ksqlDB à partir du **Topic** driver-profiles-ksql :

```
ksql> CREATE TABLE DRIVER (driverkey VARCHAR PRIMARY KEY, firstname VARCHAR, lastname VARCHAR,
make VARCHAR, model VARCHAR)
     WITH (KAFKA_TOPIC='driver-profiles-ksql', VALUE_FORMAT='delimited', VALUE_DELIMITER='|');
```

11. Vérifiez qu'il y a bien des données dans la **Table** DRIVER :

```
ksql> SELECT * FROM DRIVER EMIT CHANGES;
+-----------------+-----------------+-----------------+-----------------+-----------------+
|DRIVERKEY        |FIRSTNAME        |LASTNAME         |MAKE             |MODEL            |
+-----------------+-----------------+-----------------+-----------------+-----------------+
|driver-2         |Jacky            |Tuning           |Citroën          |2CV              |
|driver-1         |Jean             |Bonbeurre        |Renault          |Clio             |
|driver-3         |Manu             |Cronma           |Ferrari          |Enzo             |
```

Vous pouvez ensuite arrêter la requête en appuyant sur Ctrl+C.

12. Créez un Stream depuis le Topic driver-positions-ksql-avro :

```
ksql> CREATE STREAM DRIVERPOSITIONS (driverkey VARCHAR KEY, latitude DOUBLE, longitude DOUBLE)
      WITH (KAFKA_TOPIC='driver-positions-ksql-avro', VALUE_FORMAT='avro');
```

13. Il faut maintenant joindre le **Stream** DRIVERPOSITIONS avec la **Table** DRIVER

```
ksql> CREATE STREAM DRIVERPOSITIONSAUGMENTED
WITH (kafka_topic='driver-augmented-avro', value_format='avro')
AS
SELECT
      DRIVERPOSITIONS.driverkey AS driverkey,
      DRIVERPOSITIONS.latitude,
      DRIVERPOSITIONS.longitude,
      DRIVER.firstname,
      DRIVER.lastname,
      DRIVER.make,
      DRIVER.model
FROM DRIVERPOSITIONS
LEFT JOIN DRIVER on DRIVERPOSITIONS.driverkey = DRIVER.driverkey
EMIT CHANGES;
```

14. Observez les messages qui sont publiés dans le **Stream** DRIVERPOSITIONSAUGMENTED avec la commande suivante :

```
ksql> SELECT * FROM DRIVERPOSITIONSAUGMENTED EMIT CHANGES;
```

TP 6 - ksqlDB

4SH

www.4sh.fr

linkedin.com/company/4sh-france/

@4sh_france

Bordeaux

6-8 avenue des satellites
33185 Le Haillan

+33 (0)9 63 28 62 73