Laboratoire pratique : Clés de message Kafka et décalage

Temps estimé nécessaire : **40** minutes

Introduction

Dans ce laboratoire, vous conserverez les flux de messages ajoutés aux partitions dans les rubriques d'un courtier Kafka triés dans leur ordre de publication et leur état d'origine à l'aide d'un décalage et de groupes.

Objectifs

Après avoir terminé ce laboratoire, vous serez capable de :

- Utilisez les clés de message pour conserver les flux de messages triés dans leur état et leur ordre de publication d'origine
- Utiliser le décalage du consommateur pour contrôler et suivre les positions séquentielles des messages dans les partitions de rubrique

À propos de Skills Network Cloud IDE

L'IDE Cloud de Skills Network (basé sur Theia et Docker) fournit un environnement pour les travaux pratiques liés aux cours et aux projets. Theia est un IDE (environnement de développement intégré) open source qui peut être exécuté sur des ordinateurs de bureau ou dans le cloud. Pour réaliser ce laboratoire, nous utiliserons l'IDE Cloud basé sur Theia exécuté dans un conteneur Docker.

Avis important concernant cet environnement de laboratoire

Veuillez noter que les sessions de cet environnement de laboratoire ne sont pas permanentes. Un nouvel environnement est créé pour vous à chaque fois que vous vous connectez à ce laboratoire et toutes les données que vous avez pu enregistrer lors d'une session précédente seront perdues. Pour éviter de perdre vos données, prévoyez de terminer ces laboratoires en une seule session.

Exercice 1 : Télécharger et extraire Kafka

1. Ouvrez un nouveau terminal en cliquant sur la barre de menu et en sélectionnant **Terminal** -> **Nouveau terminal** .

   Cela ouvrira un nouveau terminal en bas de l'écran.

Exécutez les commandes ci-dessous sur le terminal nouvellement ouvert.

   Remarque : Vous pouvez copier le code en cliquant sur le petit bouton copier en bas à droite du code ci-dessous puis le coller, où vous le souhaitez.

2. Téléchargez Kafka en exécutant la commande ci-dessous :

```
1
```

```
wget https://downloads.apache.org/kafka/3.8.0/kafka_2.13-3.8.0.tgz
```

Copié! Exécuté!

3. Extrayez Kafka du fichier zip en exécutant la commande ci-dessous.

```
1
```

```
tar -xzf kafka_2.13-3.8.0.tgz
```

Copié! Exécuté!

   Remarque : cette commande crée un répertoire nommé `kafka_2.13-3.8.0`dans le répertoire courant.

Exercice 2 : Configurer KRaft et démarrer le serveur

1. Changer de `kafka_2.13-3.8.0`répertoire.

```
1
```

```
cd kafka_2.13-3.8.0
```

Copié! Exécuté!

2. Générez un UUID de cluster qui identifiera de manière unique le cluster Kafka.

```
1
```

```
KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"
```

Copié! Exécuté!

Cet ID de cluster sera utilisé par le contrôleur KRaft.

3. KRaft nécessite que les répertoires de journaux soient configurés. Exécutez la commande suivante pour configurer les répertoires de journaux en transmettant l'ID de cluster.

```
1
```

```
1. bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c config/kraft/server.properties
```

Copié! Exécuté!

4. Maintenant que KRaft est configuré, vous pouvez démarrer le serveur Kafka en exécutant la commande suivante.

```
1. 1
```

```
1. bin/kafka-server-start.sh config/kraft/server.properties
```

Copié! Exécuté!

Vous pouvez être sûr que cela a commencé lorsque vous voyez un résultat comme celui-ci :

Exercice 3 : Créer un sujet et un producteur pour le traitement des transactions bancaires aux distributeurs automatiques de billets

Ensuite, vous allez créer un `bankbranch`sujet pour traiter les messages provenant des distributeurs automatiques de billets des agences bancaires.

Supposons que les messages proviennent du guichet automatique sous la forme d'un simple objet JSON, comprenant un identifiant de guichet automatique et un identifiant de transaction comme dans l'exemple suivant :

```
{"atmid": 1, "transid": 100}
```

Pour traiter les messages ATM, vous devez d'abord créer un nouveau sujet appelé `bankbranch`.

1. Ouvrez un nouveau terminal et accédez au `kafka_2.13-3.8.0`répertoire.

```
1. 1
```

```
1. cd kafka_2.13-3.8.0
```

Copié! Exécuté!

2. Créez une nouvelle rubrique à l'aide de l' `--topic`argument portant le nom `bankbranch`. Pour simplifier la configuration de la rubrique et mieux expliquer le fonctionnement de la clé de message et du décalage du consommateur, vous spécifiez l' `--partitions 2`argument pour créer deux partitions pour cette rubrique. Pour comparer les différences, vous pouvez essayer d'autres `partitions`paramètres pour cette rubrique.

```
1. 1
```

```
1. bin/kafka-topics.sh --create --topic bankbranch --partitions 2 --bootstrap-server localhost:9092
```

Copié! Exécuté!

3. Répertoriez tous les sujets pour vérifier s'ils `bankbranch`ont été créés avec succès.

```
1. 1
```

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

Copié! Exécuté!

4. Vous pouvez également utiliser la `--describe`commande pour vérifier les détails du sujet `bankbranch`.

```
1. 1
```

```
1. bin/kafka-topics.sh --bootstrap-server localhost:9092 --describe --topic bankbranch
```

Copié! Exécuté!

Vous pouvez afficher les `bankbranch`deux partitions, `Partition 0`et `Partition 1`. Si aucune clé de message n'est spécifiée, les messages seront publiés sur ces deux partitions dans une séquence alternée comme suit :

`Partition 0`-> `Partition 1`-> `Partition 0`-> `Partition 1`…

Ensuite, vous pouvez créer un producteur pour publier des messages de transaction ATM.

5. Exécutez la commande suivante dans la même fenêtre de terminal avec les détails du sujet pour créer un producteur pour le sujet `bankbranch`.

```
1. 1
```

```
1. bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic bankbranch
```

Copié! Exécuté!

6. Pour produire les messages, recherchez l' >icône et ajoutez les messages ATM suivants après l'icône :

```
1. 1
```

```
1. {"atmid": 1, "transid": 100}
```

Copié! Exécuté!

```
1. 1
```

```
1. {"atmid": 1, "transid": 101}
```

Copié! Exécuté!

```
1
```

```
{"atmid": 2, "transid": 200}
```

Copié! Exécuté!

```
1
```

```
{"atmid": 1, "transid": 102}
```

Copié! Exécuté!

```
1
```

```
{"atmid": 2, "transid": 201}
```

Copié! Exécuté!

Ensuite, créez un consommateur dans une nouvelle fenêtre de terminal pour consommer ces cinq nouveaux messages.

7. Ouvrez un nouveau terminal et accédez au `kafka_2.13-3.8.0`répertoire.

```
1
```

```
cd kafka_2.13-3.8.0
```

Copié! Exécuté!

8. Ensuite, demandez à un nouveau consommateur de s'abonner au `bankbranch`sujet :

```
1
```

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic bankbranch --from-beginning
```

Copié! Exécuté!

Vous devriez pouvoir visualiser les cinq nouveaux messages que vous avez publiés. Toutefois, les messages peuvent ne pas être consommés dans le même ordre que celui dans lequel ils ont été publiés. En règle générale, vous devrez conserver les messages consommés triés dans leur ordre de publication d'origine, en particulier pour les cas d'utilisation critiques, tels que les transactions financières.

Exercice 4 : Produire et consommer avec des clés de messages

Dans cette étape, vous utiliserez des clés de message pour garantir que les messages avec la même clé sont consommés dans le même ordre que celui dans lequel ils ont été publiés. Dans le back-end, les messages avec la même clé sont publiés dans la même partition et seront toujours consommés par le même consommateur. Ainsi, l'ordre de publication d'origine est conservé côté consommateur.

À ce stade, vous devriez avoir les trois terminaux suivants ouverts dans Cloud IDE :

- Terminal du serveur Kafka
- Terminal producteur
- Terminal consommateur

Au cours des étapes suivantes, vous basculerez fréquemment entre ces terminaux.

- Tout d'abord, accédez au terminal consommateur et arrêtez le consommateur en utilisant `Ctrl`+ `C` (Windows) ou `COMMAND`+ `.`(Mac).
- Ensuite, passez au terminal Producteur et arrêtez le producteur précédent.

Vous allez maintenant démarrer un nouveau producteur et un nouveau consommateur à l'aide de clés de message. Vous pouvez démarrer un nouveau producteur avec les options de clé de message suivantes :

- `--property parse.key=true`pour que le producteur analyse les clés des messages
- `--property key.separator=`: define the key separator to be the `:` character, so our message with key now looks like the following key-value pair example:
  - `1:{"atmid": 1, "transid": 102}`
    Here, the message key is `1`, which also corresponds to the ATM ID, and the value is the transaction JSON object, `{"atmid": 1, "transid": 102}`.

1. Start a new producer with the message key enabled.

```
1
```

```
bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic bankbranch --property parse.key=true --property key.separato
```

Copied! Executed!

2. Once you see the `>` symbol, you can start to produce the following messages, where you define each key to match the ATM ID for each message:

```
1
```

```
1:{"atmid": 1, "transid": 103}
```

Copied! Executed!

```
1
```

```
1:{"atmid": 1, "transid": 104}
```

Copied! Executed!

```
1
```

```
2:{"atmid": 2, "transid": 202}
```

Copied! Executed!

```
1
```

```
2:{"atmid": 2, "transid": 203}
```

Copied! Executed!

```
1
```

```
1:{"atmid": 1, "transid": 105}
```

Copied! Executed!

3. Next, switch to the consumer terminal again and start a new consumer with `--property print.key=true` and `--property key.separator=:` arguments to print the keys.

```
1
```

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic bankbranch --from-beginning --property print.key=true --prop
```

Copied! Executed!

Now, you should see that messages with the same key are being consumed in the same order (for example: `trans102 -> trans103 -> trans104`) as they were published.

Each topic partition maintains its message queue, and new messages are enqueued (appended to the end of the queue) as they are published to the partition. Once consumed, the earliest messages are dequeued and no longer available for consumption.

Recall that with two partitions and no message keys specified, the transaction messages were published to the two partitions in rotation:

- Partition 0: `[{"atmid": 1, "transid": 100}, {"atmid": 2, "transid": 200}, {"atmid": 2, "transid": 201}]`
- Partition 1: `[{"atmid": 1, "transid": 101}, {"atmid": 1, "transid": 102}]`

As you can see, the transaction messages from `atm1` and `atm2` got scattered across both partitions. It would be difficult to unravel this and consume messages from one ATM in the same order as they were published.

However, with the message key specified as the `atmid` value, the messages from the two ATMs will look like the following:

- Partition 0: `[{"atmid": 1, "transid": 103}, {"atmid": 1, "transid": 104}, {"atmid": 1, "transid": 105}]`
- Partition 1: `[{"atmid": 2, "transid": 202}, {"atmid": 2, "transid": 203}]`

Messages with the same key will always be published to the same partition so that their published order will be preserved within the message queue of each partition.

As such, you can keep the states or orders of the transactions for each ATM.

Exercise 5: Consumer offset

Topic partitions keep published messages in a sequence, such as a list. Message offset indicates a message's position in the sequence. For example, the offset of an empty Partition 0 of `bankbranch` is `0`, and if you publish the first message to the partition, its offset will be `1`.

Using offsets in the consumer, you can specify the starting position for message consumption, such as from the beginning to retrieve all messages or from some later point to retrieve only the latest messages.

Consumer group

In addition, you normally group related consumers together as a consumer group.

For example, you may want to create a consumer for each ATM in the bank and manage all ATM-related consumers together in a group.

So let's see how to create a consumer group, which is actually very easy with the `--group` argument.

1. In the consumer terminal, stop the previous consumer if it is still running.
2. Run the following command to create a new consumer within a consumer group called `atm-app`:

```
1
```

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic bankbranch --group atm-app
```

Copied! Executed!

After the consumer within the `atm-app` consumer group is started, you should not expect any messages to be consumed. This is because the offsets for both partitions have already reached the end. In other words, previous consumers have already consumed all messages and therefore queued them.

You can verify that by checking consumer group details.

3. Stop the consumer.
4. Show the details of the consumer group `atm-app`:

```
1
```

```
bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group atm-app
```

Copied! Executed!

Now you should see the offset information for the topic `bankbranch`:

Recall that you have published `10` messages in total, and you can see the `CURRENT-OFFSET` column of partition 1 and partition 0 add up to 10 messages.

The `LOG-END-OFFSET`column indicates the last offset or the end of the sequence. Thus, both partitions have reached the end of their queues and no more messages are available for consumption.

Meanwhile, you can check the `LAG` column which represents the number of unconsumed messages for each partition.

Currently, it is `0` for all partitions, as expected.

Next, let's produce more messages and see how the offsets change.

5. Switch to the previous producer terminal and publish two more messages:

```
1. 1
```

```
1. 1:{"atmid": 1, "transid": 106}
```

Copied! Executed!

```
1. 1
```

```
1. 2:{"atmid": 2, "transid": 204}
```

Copied! Executed!

6. Switch back to the consumer terminal and check the consumer group details again.

```
1. 1
```

```
1. bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group atm-app
```

Copied! Executed!

You should see that both offsets have been increased by 1, and the `LAG` columns for both partitions have become `1`. It means you have one new message for each partition to be consumed.

7. Start the consumer again and see whether the two new messages will be consumed.

```
1. 1
```

```
1. bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic bankbranch --group atm-app
```

Copied! Executed!

Both partitions have reached the end once again.

Reset offset

Next, let's look at how you can set the partitions to consume the messages again from the beginning through resetting offset.

You can reset the index with the `--reset-offsets` argument.

First, let's try resetting the offset to the earliest position (beginning) using `--reset-offsets --to-earliest`.

1. Stop the previous consumer if it is still running, and run the following command to reset the offset.

```
1. 1
```

```
1. bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092  --topic bankbranch --group atm-app --reset-offsets --to-earliest --e
```

Copied! Executed!

Now, the offsets have been set to 0 (the beginning).

2. Start the consumer again:

```
1. 1
```

```
1. bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic bankbranch --group atm-app
```

Copied! Executed!

You should see that all 12 messages are consumed and that all offsets have reached the partition ends again.

You can reset the offset to any position. For example, let's reset the offset so that you only consume the last two messages.

3. Stop the previous consumer.

4. Shift the offset to the left by two using `--reset-offsets --shift-by -2`:

```
1. 1
```

```
1. bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092  --topic bankbranch --group atm-app --reset-offsets --shift-by -2 --e
```

Copied! Executed!

5. If you run the consumer again, you should see that you consumed four messages, 2 for each partition:

```
1. 1
```

```
1. bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic bankbranch --group atm-app
```

Copied! Executed!

6. Stop your producer, consumer and the Kafka server.

Authors

Lavanya T S

Other Contributors

Yan Luo