KAFKA

# Kafka et les groupes de consommateurs

Les groupes de consommateurs de Kafka ne vous disent rien ? Ils vont pourtant devenir inévitables à l'avenir alors autant les découvrir dès maintenant !

FRANÇOIS SARRADIN
14 MARS 2019 • 5 MIN READ



Kafka est l'un des systèmes orientés messages les plus performants existant actuellement. Il se base sur une approche producteurs-consommateurs communiquant à travers des topics. Afin d'aider à mettre en place diverses stratégies d'ingestion de données, Kafka donne la possibilité de regrouper des consommateurs qui vont se partager les messages de topics auxquels ils ont souscript.



apparaît, si vous oubliez de fournir le paramètre `group.id` :

Ça vaut le coup d'en savoir plus sur la notion de groupe de consommateurs dans Kafka.

# Généralité

Dans Kafka, un **groupe** représente un **ensemble de consommateurs partageant les mêmes offsets,** et donc aussi les mêmes partitions et les mêmes topics. En réalité, il faut voir un groupe de consommateur comme un seul et unique consommateur logique, ayant la capacité de scaler.

Néanmoins, à un instant donné les différentes partitions d'un topic vont être réparties sur les différents consommateurs du groupe sans qu'il ait d'accès concurrent (ie. il ne peut pas y avoir deux consommateurs du même groupe sur une même partition). Si le groupe contient plus de consommateurs que de partitions, certains consommateurs ne seront pas utilisés.

Un groupe est identifié par un groupId, qui est une simple chaîne de caractères. Il est déclaré au niveau des consommateurs et il est géré au niveau brokers.

On peut utiliser la notion de groupe pour coordonner des consommateurs de différentes manières :



analytics lorsqu'on a besoin de sortir différentes mesures ou différents KPI.

Si vous utilisez la plateforme Confluent, vous verrez l'ensemble des groupes disponibles en cliquant sur "Consumer lag". Sinon, vous pouvez utiliser la commande `kafka-consumer-groups` avec l'option `--list` .

```
$ ./bin/kafka-consumer-groups \
  --bootstrap-server localhost:9092 \
  --list
```

Ce qui donne, par exemple :

```
my_group
_confluent-controlcenter-5-1-2-1-command
_confluent-controlcenter-5-1-2-1
group_641bd1a6-c275-42a8-b90b-26f92f25cb38
```

Vous pouvez aussi utiliser la classe `AdminClient` dans l'API kafka-client.



```
import scala.collection.JavaConverters._
```

```
  Map(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG -> brokers)
val adminClient: AdminClient    = AdminClient.create(mapAsJavaMap(config))

for (group <- adminClient.listConsumerGroups().all().get().asScala) {
  println(s"${group.groupId()}")
}

adminClient.close()
```

# Création d'un groupe

La création d'un groupe est déclenchée par l'opération `poll()` au niveau du consommateur. Il faut donc créer un consommateur, le faire souscrire à un topic et tenter une récupération de message pour pouvoir créer un groupe. Si le groupId n'existe pas dans Kafka, il y aura une sélection au niveau des brokers à partir du *hashcode* du group ID, pour décider du broker qui deviendra le **coordinateur** du groupe. Le coordinateur est responsable de la gestion du groupe, de ses membres, du suivi de son cycle de vie et d'assigner les partitions.

Par contre, il n'y a pas d'autres possibilités pour créer un groupe, ni en ligne de commande, ni par l'API Kafka, ni par le Control Center de Confluent.

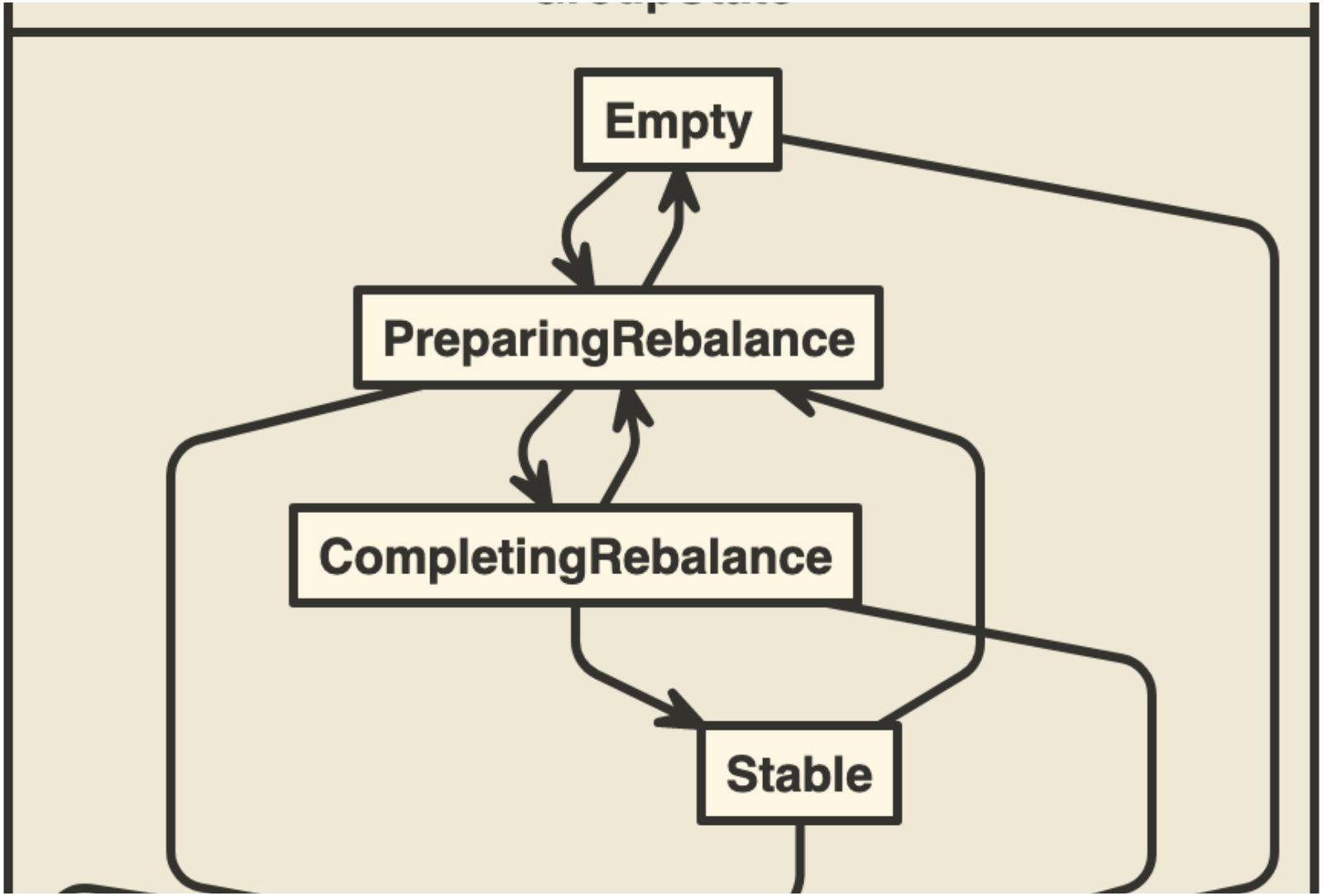


# Cycle de vie

- **Empty** : le groupe n'a plus de membre associé. Il est éligible à la suppression si aucun commit n'a été fait dans ce groupe. Il s'agit aussi de l'état initial d'un groupe lors de sa création, qui passe alors rapidement à l'état PreparingRebalance, puisque le membre responsable de la création du groupe va par suite demander de rejoindre le groupe.
- **Stable** : le groupe comporte un ou plusieurs membres et il n'y a pas de changement dans sa configuration.
- **PreparingRebalance** : un ou plusieurs membres rejoingnent le groupe ou sont retirés du groupe. Le groupe se prépare à un rééquilibrage des charges.
- **CompletingRebalance** (anciennement **AwaitingSync**) : le groupe est en attente de l'attribution d'un état de la part du leader du groupe.

En cas de problème rencontré avec un groupe, celui-ci passe alors à l'état **Dead**. Cet état est aussi atteint si un groupe est vide et si ses anciens membres n'ont pas fait de commit. Cet état apparaît enfin dans le cas où une partition associée au groupe disparait.

Voici le diagramme d'états-transitions que révèle le code source sur les états d'un groupe.



La notion de rééquilibrage ici sous-entend une rerépartition de la charge au sein d'un groupe et donc une rerépartition des partitions. Ce rééquilibrage a lieu lorsqu'un consommateur rejoint le groupe ou lorsqu'il le quitte. Il a aussi pour effet d'incrémenter le numéro `generationId`, qui représente un numéro de configuration (en terme de membres) du groupe. Ce numéro est très présent lorsqu'on regarde les logs des brokers Kafka.

```
INFO [GroupCoordinator 0]: Stabilized group my_group generation 2 (__consumer_offsets-14) (kafka.coordinat
INFO [GroupCoordinator 0]: Preparing to rebalance group my_group in state PreparingRebalance with old gene
INFO [GroupCoordinator 0]: Group my_group with generation 3 is now empty (__consumer_offsets-14) (kafka.co
INFO [GroupMetadataManager brokerId=0] Group my_group transitioned to Dead in generation 3 (kafka.coordina
INFO [GroupCoordinator 0]: Removed 1 offsets associated with deleted partitions: webclick-0. (kafka.coordi
```



Pour pouvoir rester dans un groupe, les membres doivent envoyer un signal de *heartbeat* au

`timeout.ms` - 10 secondes par défaut d'après le code source de Kafka). Si un membre n'a toujours rien envoyé malgré ce délai, il est viré du groupe et un rééquilibrage se met en place.

À tout moment, vous pouvez utiliser AdminClient de l'API Kafka, le Control Center (menu "Consumer lag") ou `kafka-consumer-groups` pour obtenir des informations sur un groupe. Par exemple :

```
$ ./bin/kafka-consumer-groups \
  --bootstrap-server localhost:9092 \
  --describe --group my_group
```

Ce qui donne, par exemple :

```
Consumer group 'my_group' has no active members.

TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID     HOST
webclick        0          20              50              30              -               -
webclick        1          0               30              30              -               -
```



La notion de *lag* correspond à la différence entre le dernier offset de message traité par le groupe pour

a aucun consommateur associé au groupe.

Avec l'option `--state` :

```
./bin/kafka-consumer-groups \
  --bootstrap-server localhost:9092 \
  --describe --group my_group \
  --state
```

Nous avons à la fois l'état du groupe ( `Empty` ici), le nombre de membres et la référence du broker qui sert de coordinateur.

```
Consumer group 'my_group' has no active members.

COORDINATOR (ID)          ASSIGNMENT-STRATEGY       STATE                #MEMBERS
192.168.0.40:9092 (0)                               Empty                0
```



La suppression d'un groupe peut être manuelle, en utilisant à nouveau la commande `kafka-consumer-grou`

```
$ ./bin/kafka-consumer-groups \
  --bootstrap-server localhost:9092 \
  --delete --group my_group
```

Elle peut se faire avec AdminClient de l'API Kafka. Par contre, le Control Center ne semble pas offrir cette possibilité.

Sinon, la suppression des groupes vides sans commit associé est orchestré par un scheduler interne. Les intervalles de vérification de ce scheduler dépendent du paramètre côté broker `offsets.retention.check.interval.ms` (par défaut 10 minutes d'après le code source de Kafka).

# Documentation

Pour plus d'information, vous pouvez aller sur cette page de Confluent qui décrit le fonctionnement des consommateurs et des groupes de consommateurs : Kafka Java Consumer - Confluent Platform.

Photographie par Ishan @seefromthesky sur Unsplash.

# Subscribe to Univalence blog

Get the latest posts delivered right to your inbox

youremail@example.com Subscribe

MORE IN **KAFKA**

SBT pour centraliser et homogénéiser la déclaration de services

4 juin 2020 – 5 min read


You've successfully subscribed to Univalence blog!

See all 6 posts →

Lors de mes débuts dans Spark, un collègue m'a dit la chose suivante : "N'utilise pas groupByKey, utilise plutôt reduceByKey". Je lui demande alors pourquoi l'un plutôt que

HARRISON CHENG
15 MARS 2019 • 4 MIN READ

Avez-vous déjà vu... un DataFrame Spark ayant deux colonnes ayant le même nom ? Et avez-vous essayé de lancer une requête SQL dessus ? Il est possible que ça soit le cas.

PHILIPPE HONG
8 MARS 2019 • 3 MIN READ