Even More on The Architecture of Kafka

In this lesson, we'll continue learning about the architecture of Kafka.

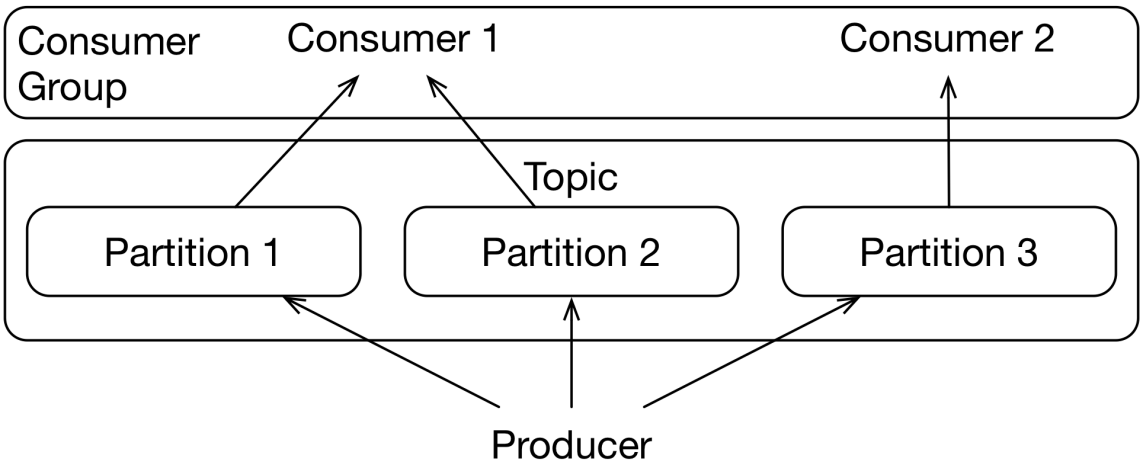
WE'LL COVER THE FOLLOWING

- Consumer groups
- Example
- Persistence
- Log compaction

Consumer groups

An event like `neworder42` should probably be processed only once by one instance of the invoicing microservice. Only one instance of a microservice should receive it, ensuring that only one invoice is written for this order. However, another instance of a microservice might work on `neworder21` in parallel.

Consumers are organized in consumer groups where each partition sends records to exactly one consumer in the consumer group. One consumer can be responsible for several partitions.



Consumer Groups and Partitions

Thus, a consumer receives the messages of one or multiple partitions.

Example

The drawing above shows an example. Consumer 1 receives the messages of partitions 1 and 2 and consumer 2 receives the messages of partition 3.

The invoicing microservice instances could be organized in a consumer group, ensuring that only one instance of the invoicing microservice processes each record.

When a consumer receives a message from a partition, it will also later receive all messages from the same partition. The order of messages per partition is also preserved meaning that records in different partitions can be handled in parallel, and at the same time the sequence of records in a single partition is guaranteed.

Therefore, the instance of the invoicing microservice that receives `neworder42` would also receive `updated42` if those records are sent to the same partition. So, the instance would be responsible for all events about the order 42.

Of course, this applies only if the mapping of consumers to partitions remains stable. For example, if new consumers are added to the consumer group for scaling, the mapping can change.

The new consumer would need to handle at least one partition that was previously handled by a different consumer.

The maximum number of consumers in a consumer group is equal to the number of partitions, because each consumer must be responsible for at least one partition. Ideally, there are more partitions than consumers so that we can add more consumers when scaling.

Consumers are always members of a consumer group. Therefore, they receive records sent only to their partitions. If each consumer is to receive all records from all partitions, then there must be a separate consumer group for each consumer with only one member.

Persistence

Kafka is a mixture of a messaging system and data storage solution. The

records in the partitions can be read by consumers and written by producers.

The default retention for records is seven days, but it can be changed. The records can also be saved permanently where the consumers merely store their offset.

A new consumer can therefore process all records that have ever been written by a producer in order to update its own state.

If a consumer is too slow to handle all records in a timely manner, Kafka stores them for quite a long time allowing the consumer to process the records later to keep up.

Log compaction

However, this means that Kafka has to store more and more data over time. Some records, however, eventually become irrelevant. If a customer has moved several times, you may only want to keep the last information about the last move as a record in Kafka. **Log compaction is used for this purpose**.

> All records with the same key are removed, except for the last one.

Therefore, the **choice of the key is very important** and must be considered from a domain logic point of view in order to have all the relevant records still available after log compaction.

Log compaction for the order topic would remove all events with the key `updated42` but the very last one. As a result, only the very last update to the order remains available in Kafka.

QUIZ

1 In a Kafka-based system, N consumers exist and M partitions exist.

Which of the following is NOT possible?

COMPLETED 0%

1 of 3

In the next lesson, we'll discuss events with Kafka.