Kafka Delivery Semantics

Let's explore what message delivery guarantees Kafka provides to its clients.

We'll cover the following

Producer delivery semantics#

As we know, a producer writes only to the leader broker, and the followers asynchronously replicate the data. How can a producer know that the data is successfully stored at the leader or that the followers are keeping up with the leader? Kafka offers three options to denote the number of brokers that must receive the record before the producer considers the write as successful:

- **Async**: Producer sends a message to Kafka and does not wait for acknowledgment from the server. This means that the write is considered successful the moment the request is sent out. This **fire-and-forget** approach gives the best performance as we can write data to Kafka at network speed, but no guarantee can be made that the server has received the record in this case.
- **Committed to Leader**: Producer waits for an acknowledgment from the leader. This ensures that the data is committed at the leader; it will be slower than the ‘Async’ option, as the data has to be written on disk on the leader. Under this scenario, the leader will respond

without waiting for acknowledgments from the followers. In this case, the record will be lost if the leader crashes immediately after acknowledging the producer but before the followers have replicated it.

- **Committed to Leader and Quorum**: Producer waits for an acknowledgment from the leader and the quorum. This means the leader will wait for the full set of in-sync replicas to acknowledge the record. This will be the slowest write but guarantees that the record will not be lost as long as at least one in-sync replica remains alive. This is the strongest available guarantee.

As we can see, the above options enable us to configure our preferred trade-off between durability and performance.

- If we would like to be sure that our records are safely stored in Kafka, we have to go with the last option – Committed to Leader and Quorum.
- If we value latency and throughput more than durability, we can choose one of the first two options. These options will have a greater chance of losing messages but will have better speed and throughput.

Consumer delivery semantics#

A consumer can read only those messages that have been written to a set of in-sync replicas. There are three ways of providing consistency to the consumer:

- **At-most-once** (Messages may be lost but are never redelivered): In this option, a message is delivered a maximum of one time only. Under this option, the consumer upon receiving a message, commit (or increment) the offset to the broker. Now, if the consumer crashes before fully consuming the message, that message will be lost, as when the consumer restarts, it will receive the next message from

the last committed offset.

- **At-least-once** (Messages are never lost but maybe redelivered): Under this option, a message might be delivered more than once, but no message should be lost. This scenario occurs when the consumer receives a message from Kafka, and it does not immediately commit the offset. Instead, it waits till it completes the processing. So, if the consumer crashes after processing the message but before committing the offset, it has to reread the message upon restart. Since, in this case, the consumer never committed the offset to the broker, the broker will redeliver the same message. Thus, duplicate message delivery could happen in such a scenario.
- **Exactly-once** (each message is delivered once and only once): It is very hard to achieve this unless the consumer is working with a transactional system. Under this option, the consumer puts the message processing and the offset increment in one transaction. This will ensure that the offset increment will happen only if the whole transaction is complete. If the consumer crashes while processing, the transaction will be rolled back, and the offset will not be incremented. When the consumer restarts, it can reread the message as it failed to process it last time. This option leads to no data duplication and no data loss but can lead to decreased throughput.

← Back
Controller Broker

Next →
Kafka Characteristics

Mark as Completed

Report an Issue