DZone

Exactly-Once Semantics With Apache Kafka

by Himani Arora MVB · Oct. 16, 18 · Big Data Zone · Tutorial

Kafka's exactly once semantics was recently introduced with the version which enabled the message being delivered exactly once to the end consumer even if the producer retries to send the messages.

This major release raised many eyebrows in the community as people believed that this was not mathematically possible in distributed systems. Jay Kreps, co-founder of Confluent and co-creator of Apache Kafka, explained its possibility and how is it achieved in Kafka in this post.

In this blog, we will be discussing how can one take advantage of the exactly once message semantics provided by Kafka.

Overview of Different Message Delivery Semantics Provided by Apache Kafka

"At most once-messages may be lost but are never redelivered."

In this case, the producer does not retry to send the message when an ACK times out or returns an error, thus the message might end up not being written to the Kafka topic, and hence not delivered to the consumer.

"At least once-messages are never lost but may be redelivered."

In this case, the producer tried to resend the message if the ACK times out or receives an error, assuming that the message was not written to the Kafka topic.

" Exactly once — this is what people actually want, each message is delivered once and only once."

In this case, even if a producer tries to resend a message, it leads to the message being delivered exactly once to the end consumer.

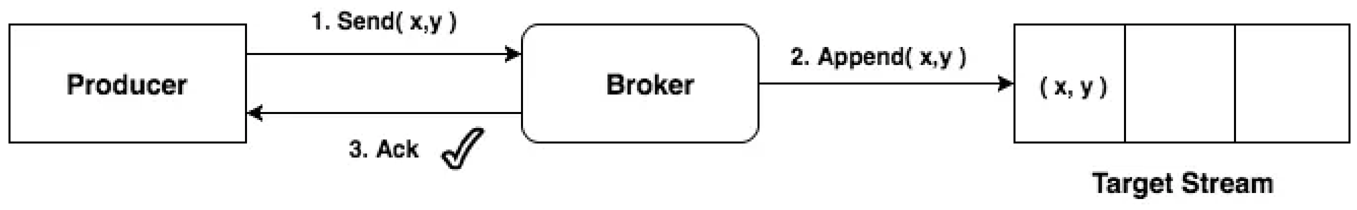
Exactly-once semantics are the most desirable guarantee and require cooperation between the messaging system itself and the application producing and consuming the messages.

For instance, if, after consuming a message successfully, you rewind your Kafka consumer to a previous offset, you will receive all the messages from that offset to the latest one, all over again. This shows why the messaging system and the client application must cooperate to make exactly-once semantics happen.

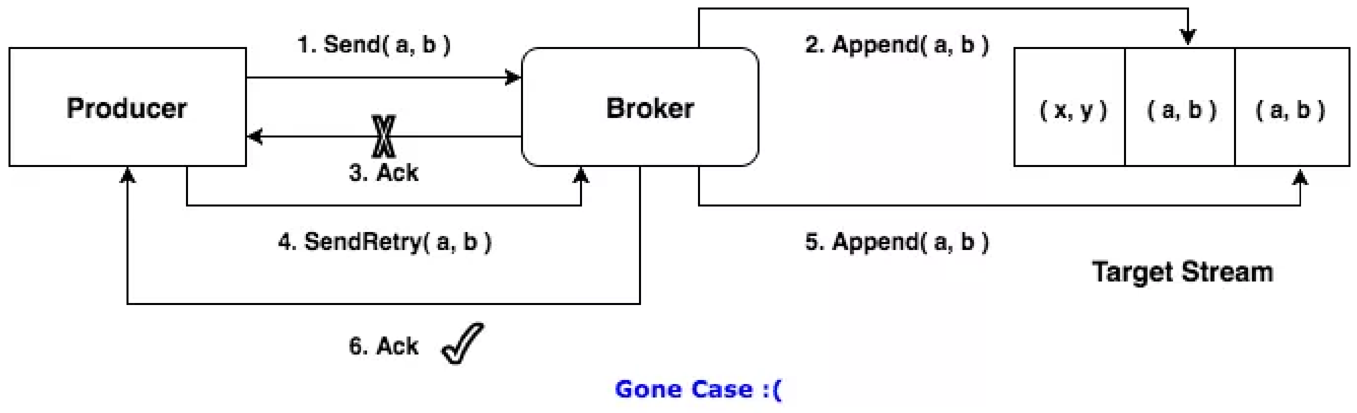
Why Use the Exactly-Once Semantics of Kafka?

We know that at-least-once guarantees that every message will be persisted at least once, without any data loss, but this may cause duplicates in the stream.

For example, if the broker failed right before it sent the ACK, but after the message was successfully written to the Kafka topic, this retry will lead to the message being written twice and hence delivered more than once to the end consumer.



Favorable Case :)



Acknowledgement failed, led to message duplication

In the new exactly-once semantics, Kafka's processing semantics guarantee delivery of the message to the end consumer exactly once. This has been strengthened by introducing:

- Idemptotent producers
- Atomic transactions

Idempotent Producer

An **idempotent** *operation is an operation that can be performed many times without causing a different effect than if the operation was only performed once.*

Now, in Kafka, the producer sends operations that can be made idempotent, so that if an error occurs which causes a producer retry, the same message which is sent by the producer multiple times will only be written once to the logs on the maintained Kafka broker.

Idempotent producers ensure that messages are delivered exactly once to a particular topic partition during the lifetime of a single producer.

To turn on this feature and get exactly-once semantics per partition — meaning no duplicates, no data loss, and in-order semantics — configure your producer with the following property:

```
enable.idempotence=true
```

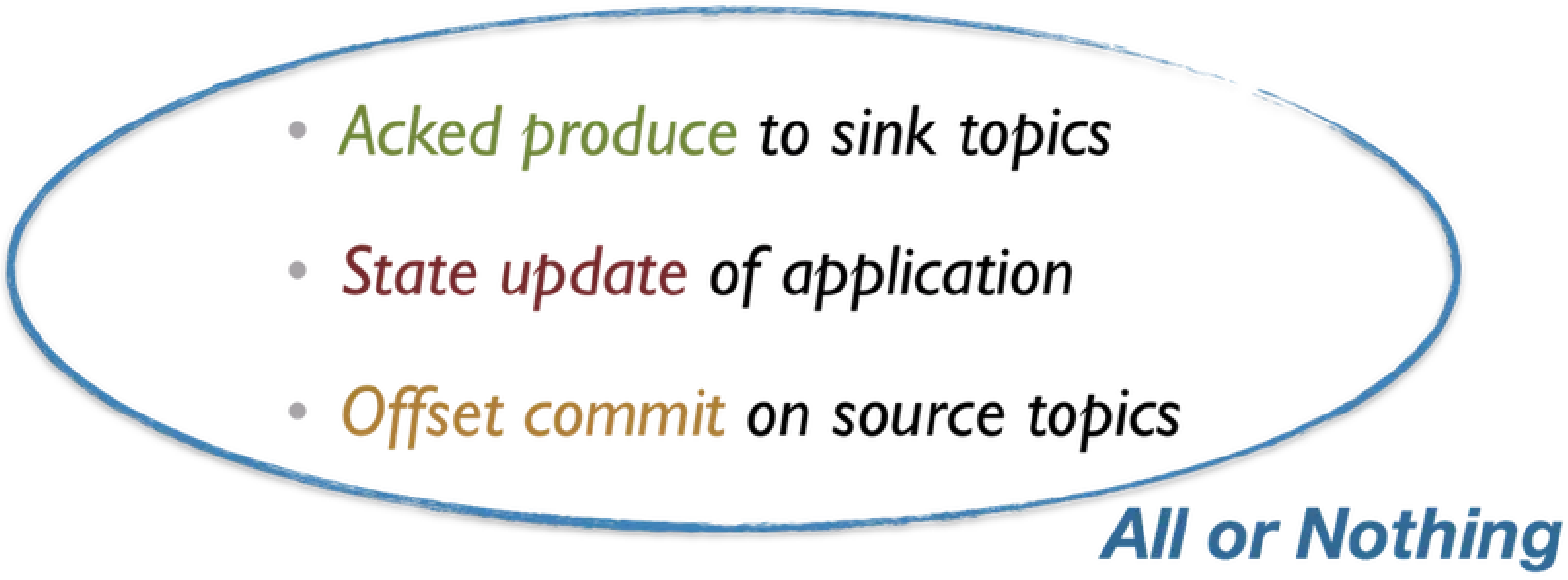
With this feature turned on, each producer gets a unique id (PID), and each message is sent together with a sequence number. When either the broker or the connection fails, and the producer tried to resend the message, it will only be accepted if the sequence number of that message is one more than the one last message.

However, if the producer fails and restarts, it will get a new PID. Hence, the idempotency is guaranteed for only a **single producer session**.

Atomic Transactions

Kafka now supports atomic writes across multiple partitions through the new transactions API. This allows a producer to send a batch of messages to multiple partitions such that either all the messages in the batch are visible to all the consumers or none are ever visible to any consumer.

It allows you to commit your consumer offsets in the same transaction along with the data you have processed, thereby allowing end-to-end exactly-once semantics.


Acked produce to sink topics
State update of application
Offset commit on source topics
All or Nothing

Below is an example snippet that describes how can you send messages atomically to a set of topic partitions using the new Producer API:

```
{
    producer.initTransactions();
    try{
        producer.beginTransaction();
            producer.send(record0);
            producer.send(record1);
            producer.sendOffsetsToTxn(…);
            producer.commitTransaction();
        } catch( ProducerFencedException e) {
            producer.close();
        } catch( KafkaException e ) {
            producer.abortTransaction();
        }
    }
```

Consumers

To use transactions, you need to configure the Consumer to use the right **isolation.level** and use the new Producer APIs. There are now two new isolation levels in Kafka consumer:

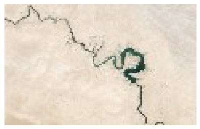
1. **read_committed**: Read both kinds of messages (those that are not part of a transaction and that are) after the transaction is committed.
2. **read_uncommitted**: Read all messages in offset order without waiting for transactions to be committed. This option is similar to the current semantics of a Kafka consumer.

Also, the **transactional.id** property must be set to a unique ID in the producer config. This unique ID is needed to provide continuity of transactional state across application restarts.

References

- Confluent’s blog on exactly once semantics
- Transactions in Apache Kafka
- Image source for comparison between favorable and gone cases of at least once semantics

Like This Article? Read More From DZone



Applying Kafka Streams to the Purchase Transaction Flow

Self-Learning Kafka Streams With Scala (Part 2)

Self-Learning Kafka Streams With Scala (Part 1)

Free DZone Refcard

Software Usage Analytics for Data-Driven Development

Topics: BIG DATA , KAFKA , BIG DATA ANALYSIS , STREAM COMPUTING , TUTORIAL