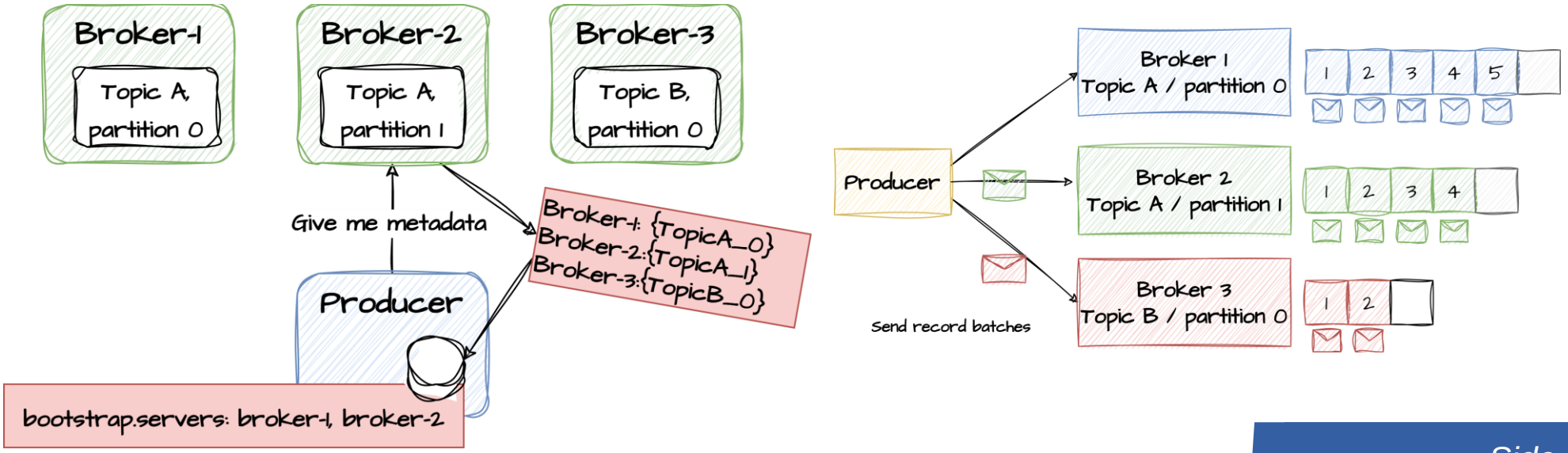
Lund&Bendsen
developing developers

Producer

Bootstrapping

Lund&Bendsen
developing developers

During initialization, the producer must specify a bootstrap URL – a list of brokers in the cluster and potentially a producer id along with a serialization class. One of the brokers in the bootstrap list is contacted and returns information of how partitions, leaders etc is distributed in the cluster. This information is cached in the client and only renewed if a repartitioning event happens. With this bootstrap information in place, the producer can now target the right brokers when producing messages.



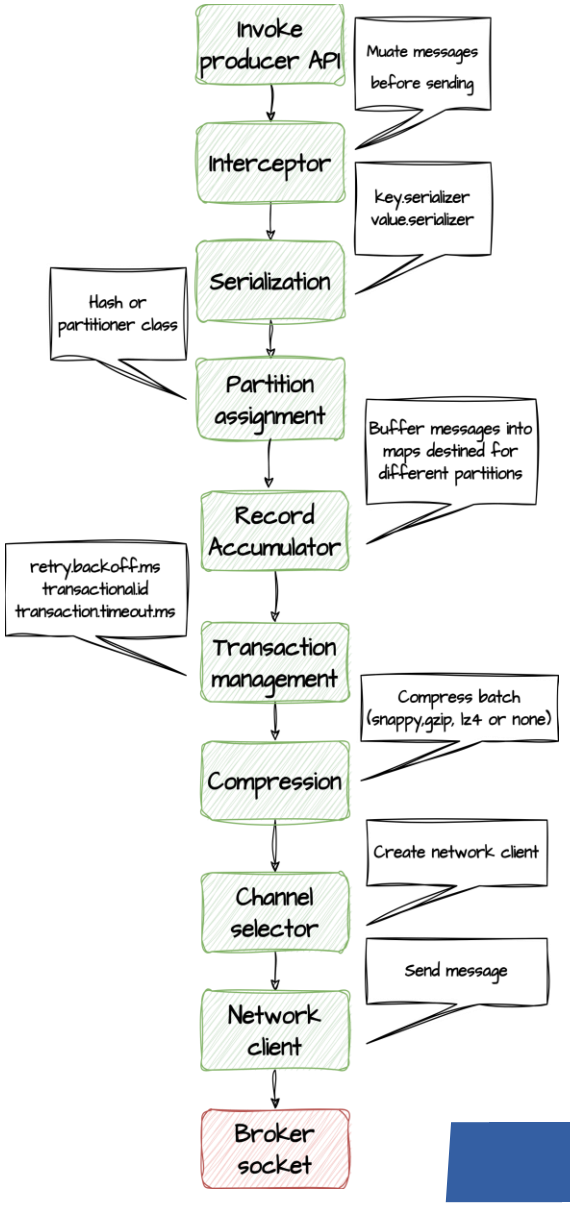
Sending messages

Lund&Bendsen
developing developers

Producers run through a number of steps before a record is successfully sent.

Records are being serialized to a binary format and a partitioning strategy is used to assign each record to a partition. Partitioning happens in order to distribute the write load on all brokers in the cluster.If a record key is provided the partition is determined based on `Hash(key) % number_of_partitions` otherwise the partitioning is round-robin.

Further down the stack, transaction management and resend logic is handled and finally the batch is compressed before sending to the right partition leader.



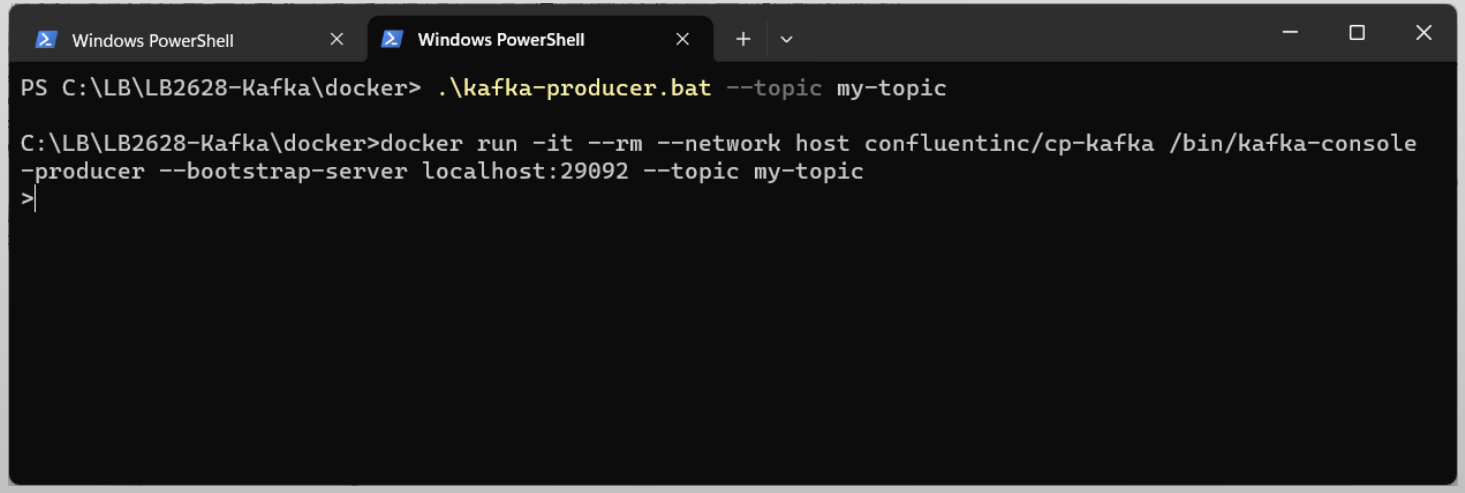
Exercise

Lund&Bendsen
developing developers

5.1

Create producer

- Open powershell and navigate to <project-folder>/docker
- Type **.\kafka-producer.bat --topic my-topic**
- You should now see a prompt >
- This allows you to produce test messages and have them sent to the topic
- Type some string**s** and hit enter



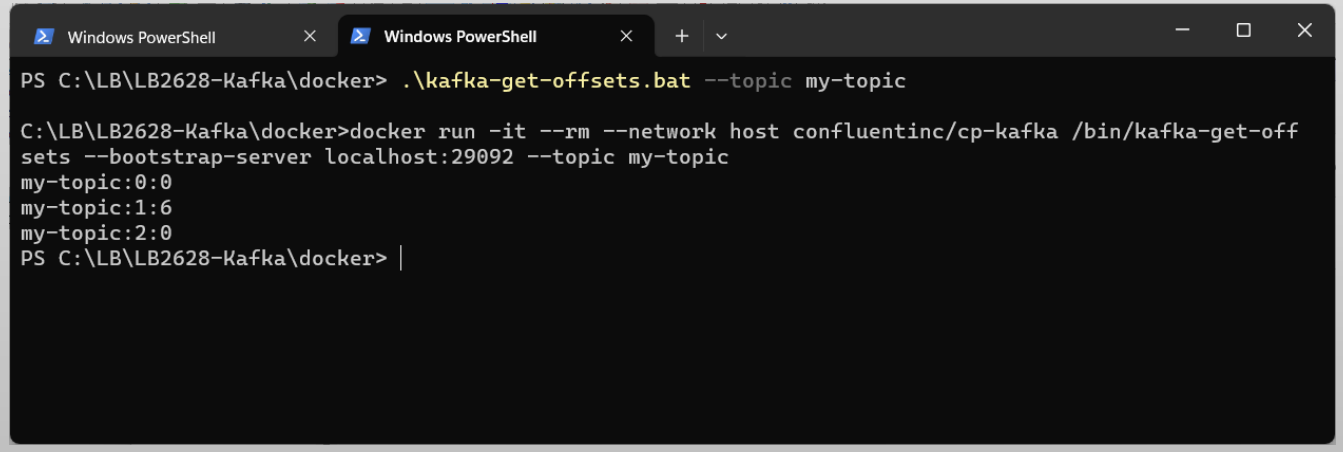
Exercise

Lund&Bendsen
developing developers

5.2

See offsets

- Open a new powershell and navigate to <project-folder>/docker
- Type **.\kafka-get-offsets.bat --topic my-topic**
- This will show the partitions for that topic and their current offset
- Open AKHQ.IO to see the produced messages



Exercise

Lund&Bendsen
developing developers

Create producer sending keys and values

- Open powershell and navigate to <project-folder>/docker
- Delete topic by typing **.\kafka-topics.bat --topic my-topic --delete**
- Create new topic with 3 partitions. Type **.\kafka-topics.bat --topic my-topic --partitions 3 --replication-factor 2 --create**
- Type **.\kafka-producer.bat --topic my-topic --property parse.key=true --property key.separator=":"**
- You should now see a prompt >
- This allows you to produce test messages and have them sent to the topic passing keys and values as *key*:*value*
- Type some string**s** and hit enter, like *test-key:test-value*. Use different keys
- Check AKHQ.IO that the messages have reached different partitions

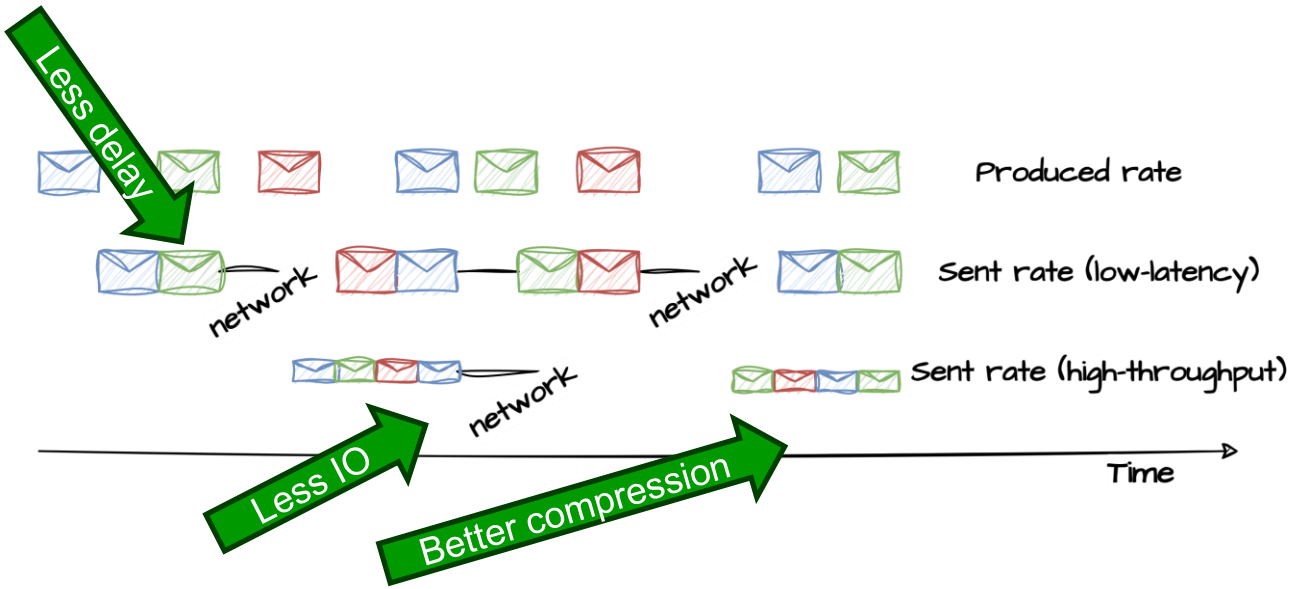
Latency vs throughput

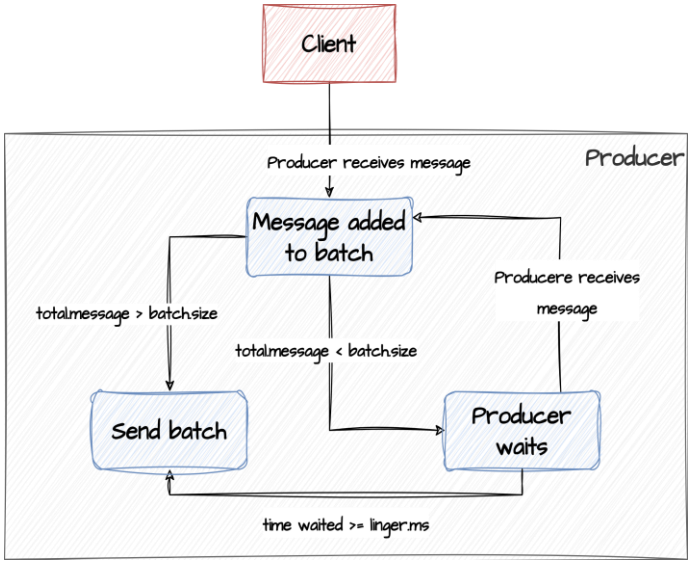
Lund&Bendsen
developing developers

All components in the Kafka setup play an active role when it comes to performance optimizations. Besides distributing the load among the leaders and batching and compressing the payload, the producer can be configured to do high-throughput or low-latency. The below settings are used for that.

Linger.ms represents the amount of time a Kafka producer will wait before sending a batch of messages. If a producer receives multiple messages in a short time frame, it can choose to wait for a brief period before sending them as a batch. This is done to optimize network and resource usage.

Batch.size is the maximum amount of data, in bytes, that a Kafka producer will include in a single batch of messages. When the batch size is reached or the linger.ms time elapses, the producer will send the batch to the Kafka broker.



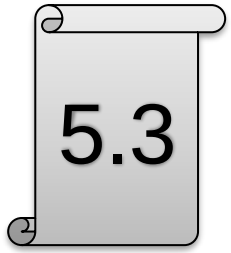


Exercise

Lund&Bendsen
developing developers

Change linger.ms

- Open akhq.io and navigate to topics->my-topic->details->live tail
- Open a new PowerShell and navigate to <project-folder>/docker
- Type **docker run -it --rm --network host confluentinc/cp-kafka /bin/kafka-producer-perf-test --producer-props bootstrap.servers=localhost:29092 linger.ms=100 batch.size=100000 --topic my-topic --num-records 1000 --record-size 2 --throughput 2**
- This will start a producer that produces 1000 small records at the rate of 2 rec/sec.
- Open AKHQ.IO to see the produced messages for the topic in live-tail. Explain what you see (messages should arrive in chunks)
- Stop the producer and change linger.ms to 3000
- Type **docker run -it --rm --network host confluentinc/cp-kafka /bin/kafka-producer-perf-test --producer-props bootstrap.servers=localhost:29092 linger.ms=3000 batch.size=100000 --topic my-topic --num-records 1000 --record-size 2 --throughput 2**
- Watch the live-tail. You should see a difference now.



Acknowledgement

Ack: 0 (fire-and-forget)

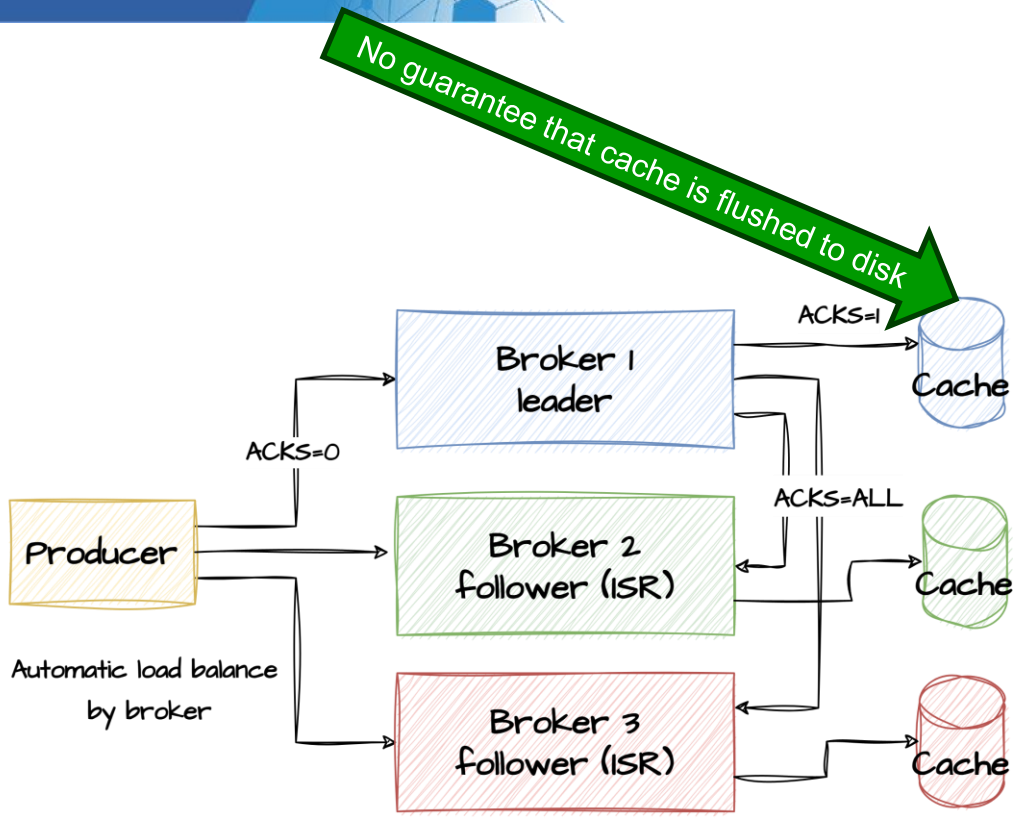
The producer won't wait for any acknowledgment from the leader.

Acks: 1

The leader will append the message to its own log and. Then it returns the acknowledgment to the producer .

Acks: all (safe)

The leader will append the message to its own log and wait for acknowledgments from all in-sync replicas. Then it returns the acknowledgment to the producer

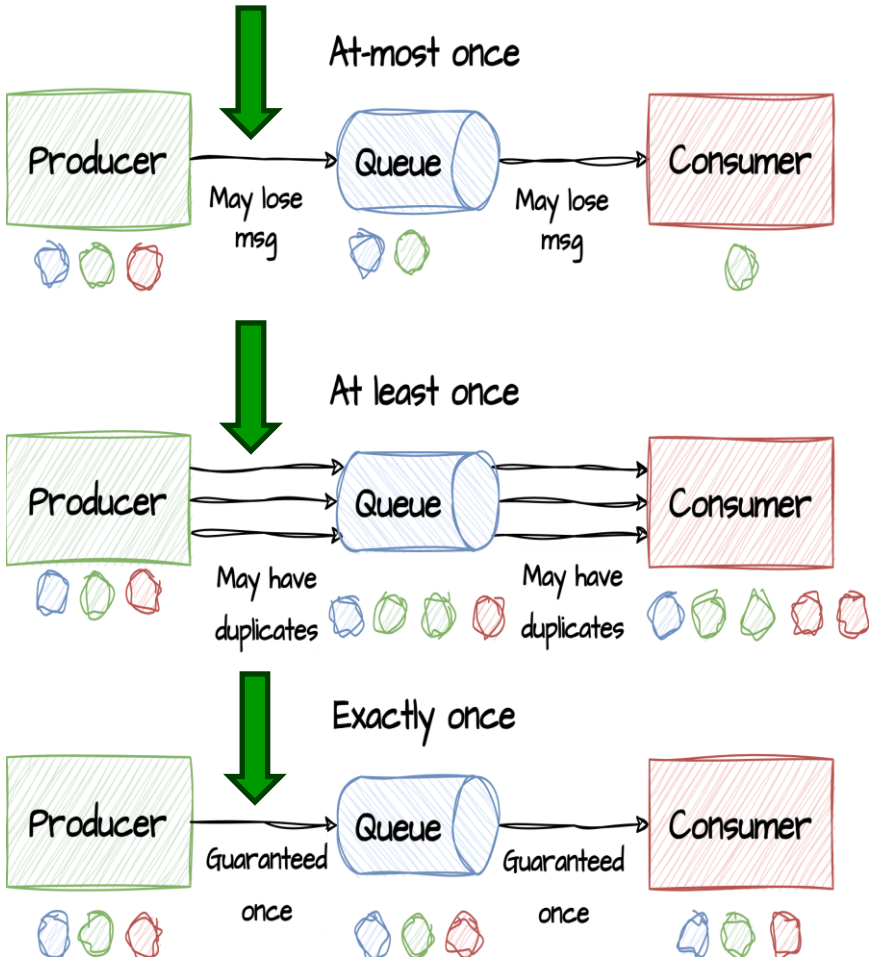


ACKS	Description
0	Producer don't wait, possible loss of data
1	Producer wait for leader acknowledgement
ALL	Producer wait for ISR acknowledgement

Delivery guarantee

Lund&Bendsen
developing developers

- **At most once** - Messages can be sent asynchronously in a “fire and forget” way (act:0), meaning the producer does not wait for any acknowledgement that messages were received, or for more latency, but less risk of message loss, the producer can wait for acknowledgment from the leader broker.
- **At least once** - If a producer expects but fails to receive a response indicating that a message was committed, it will resend the message. This provides at-least-once delivery semantics since the message may be written to the log again during resending if the original request had succeeded. The Kafka producer provides an **idempotent** option that guarantees that resending a message will not result in duplicate entries in the log.
- **Exactly once** - Producers can request acknowledgment that messages were received and successfully replicated, and if it resends a message, it resends with idempotency, meaning existing messages are overwritten rather than duplicated.

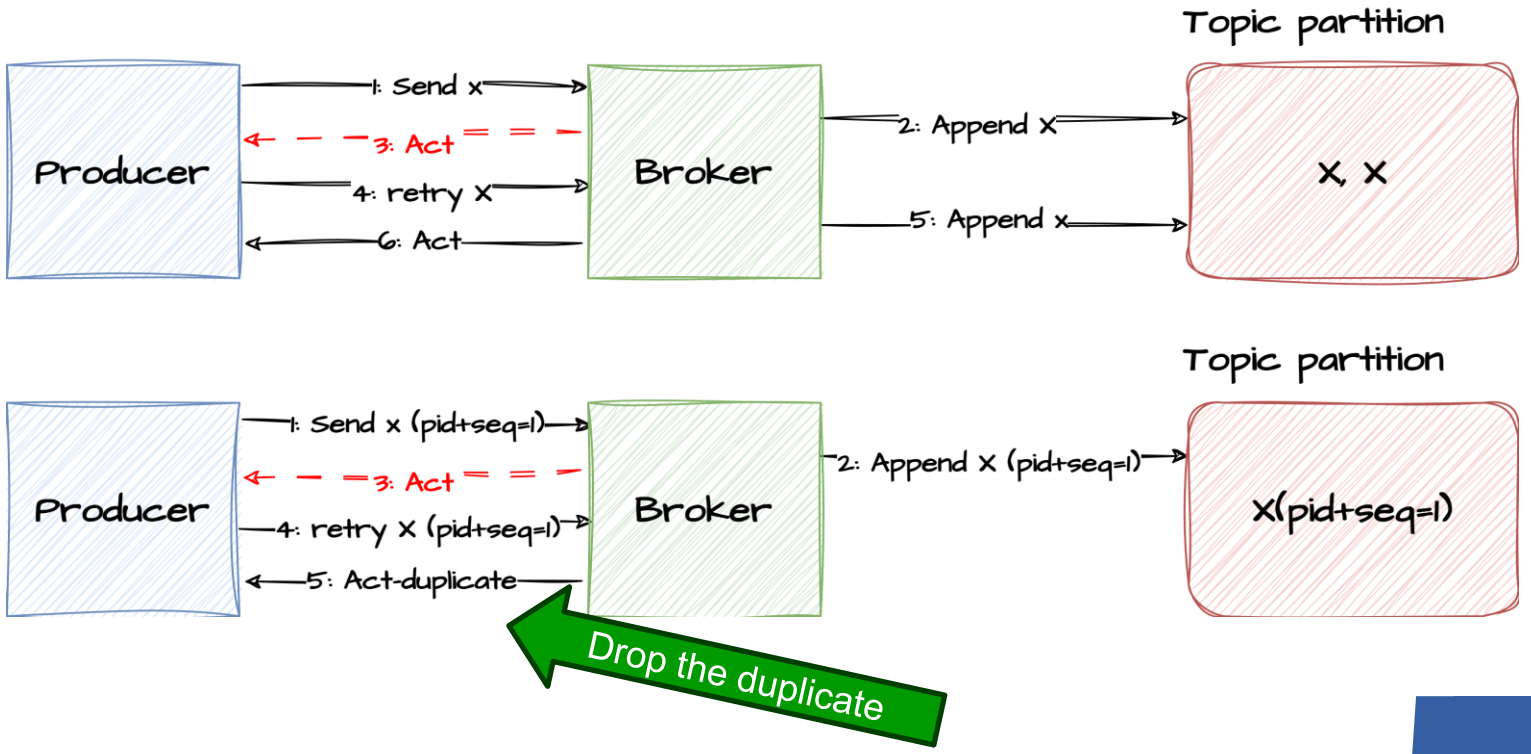


Idemponent producer

Lund&Bendsen
developing developers

An idempotent producer setup ensures that a message is written to a partition only once. This is achieved in combination with the partition leader. All messages get a monotonically created sequence-number that is stored along with the message in the outbound buffer. Whenever the producer resends a message, the broker can determine if the message has been sent and successfully processed previously.

This makes the producer idempotent since the same message can be sent multiple times without changing the result..

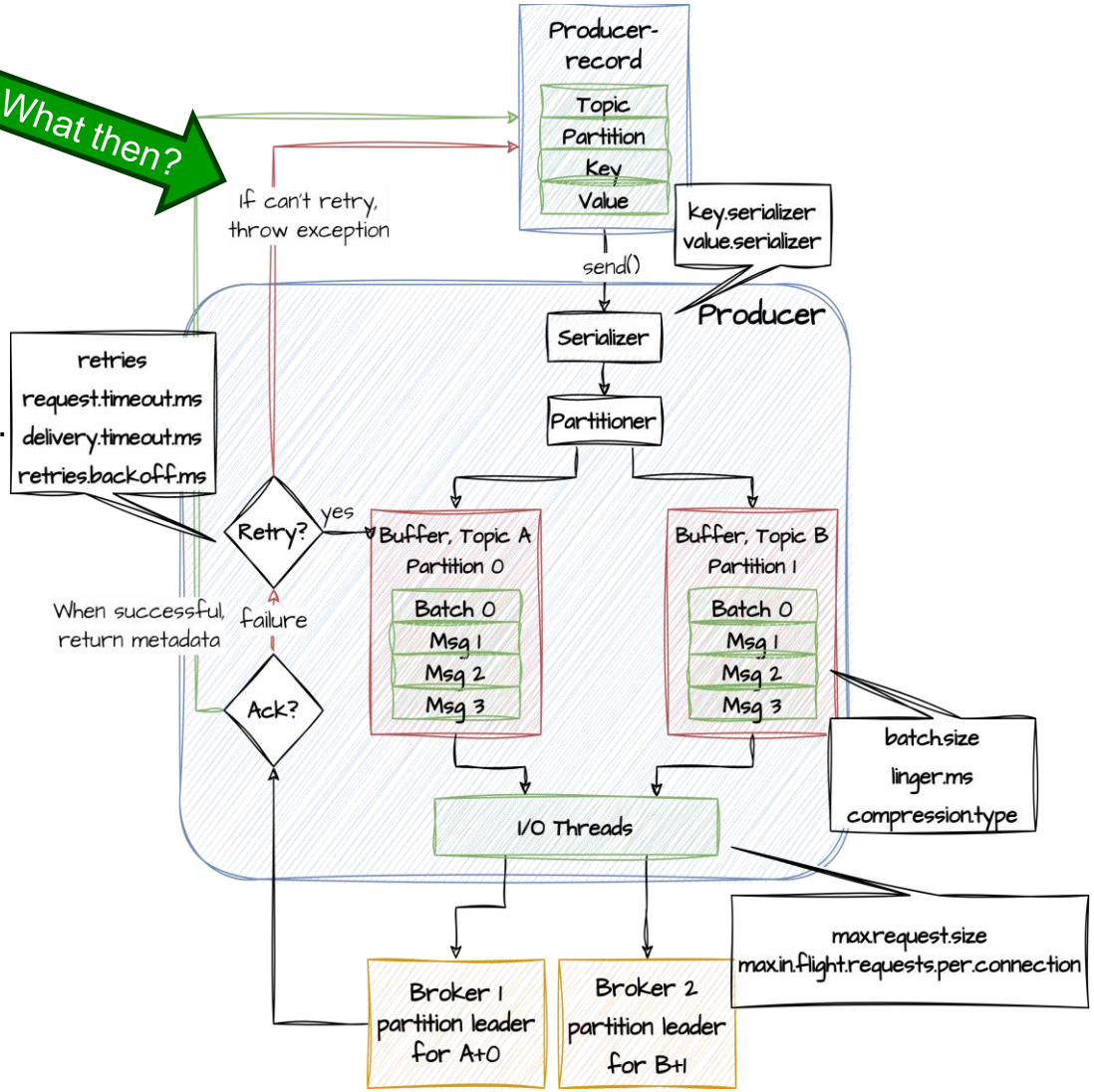


Producer internal

Lund&Bendsen
developing developers

1. `Send()` is called and the message is serialized.
2. Which partition the message should be routed to.
3. One buffer for each partition and each buffer can hold many batches of messages grouped for each partition.
4. I/O threads pick up these batches and sent them over to the brokers.
5. Messages are in-flight from the client to the brokers.
6. Messages are written on disk and sent to the followers for replication.
7. `RecordMetadata` response is sent back to the client.
8. If a failure occurred without receiving an ACK, check if message retry is enabled; if so, we need to resend it.

The client receives the response.



Producer configuration

Lund&Bendsen
developing developers

The producer polls for a batch of messages from the batch queue, one batch per partition.

The producer groups the batch based on the leader broker.

The producer sends the grouped batch to the leader broker.

A batch is ready when one of the following is true:

batch.size is reached. Note: Larger batches typically have better compression ratios and higher throughput, but they have higher latency.

linger.ms (time-based batching threshold) is reached. Note: There is no simple guideilne for setting linger.ms values; you should test settings on specific use cases. For small events (100 bytes or less), this setting does not appear to have much impact.

Producer configuration

Lund&Bendsen
developing developers

compression.type

Compression is an important part of a producer's work, and the speed of different compression types differs a lot.

To specify compression type, use the compression.type property. It accepts standard compression codecs ('gzip', 'snappy', 'lz4'), as well as 'uncompressed' (the default, equivalent to no compression), and 'producer' (uses the compression codec set by the producer).

More batching leads to more efficient compression.

acks

The acks setting specifies acknowledgments that the producer requires the leader to receive before considering a request complete. This setting defines the durability level for the producer.

0 No Guarantee. The producer does not wait for acknowledgment from the server.

1 Leader writes the record to its local log, and responds without awaiting full acknowledgment from all followers.

-1 Leader waits for the full set of in-sync replicas (ISRs) to acknowledge the record. This guarantees that the record is not lost as long as at least one IRS is active.

Producer configuration

Lund&Bendsen
developing developers

enable.idempotence

When set to ‘true’, the producer will ensure that exactly one copy of each message is written in the stream. If ‘false’, producer retries due to broker failures, etc., may write duplicates of the retried message in the stream

Delivery.timeout.ms Producer won’t retry forever. Records will fail if they cannot be delivered within the specified timeout

Retry.backoff.ms

Number of milliseconds the producer will retry between retries.

max.in.flight.requests.per.connection

Number of batches that can be processed simultaneously for the same partition. If sending the first batch fails, the producer will proceed to the next and retry the first later. Having a number higher than 1 can potentially change the order of sending