Kafka

Topics:

- a stream of data
- like a table in database
- identified by topic name
- can be any kind of message (jason, binary, etc)
- a sequence of message is called a data stream
- cannot query topics
- producer – write data to topic consumer – read data from topic

partitions:

- topic split in partitions
- message in partition are ordered
- offset: incremental id

Example:

- truck: send (truck id + position)
- topic = trucks_gps

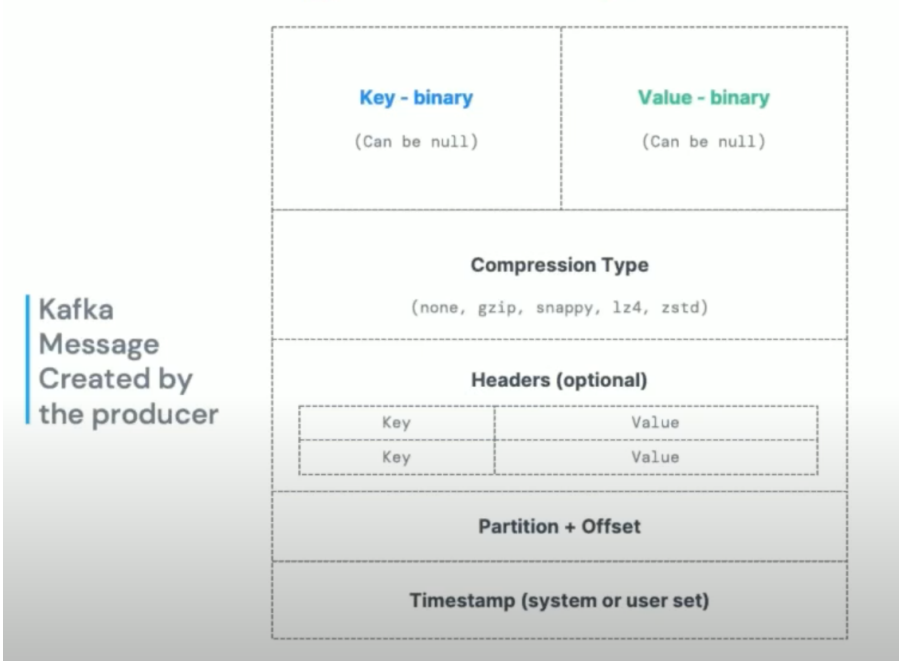
- data are immutable
- data keep for a limited time
- offset are not re-used even if previous message have been deleted
- order is only guaranteed within a partition
- data is assigned randomly to a partition unless a key is provided

Producer:

- know write to which partition
- kafka broker failures, producer will automatically recover

Message key:

- key = null → data sent round robin(partition0, partition1, partition2)
- same key go to same partition
- key是通过hashing来决定map到哪个partition的



Message Serializer:

- message type is bytes
- serialize means: objects/data to bytes
- work on both key and value
- common serializer: string, int, float, Avro, Protobuf

- serialization / deserialization type must not change during a topic lifecycle
-

Consumers:

- read from topic
- consumer automatically know which broker to red
- consumer know how to recover

Consumer Deserializer:

- bytes to objects/data
- used on value and key
- common deserializers: string, int, float, Avro, Protobuf
- serialization / deserialization type must not change during a topic lifecycle

Consumer Groups:

- consumers read data as a consumer group
- each consumer within a group read from exclusive partition
- we can have multiple groups consume on same topic
- distinct consumer group: use consumer property “group.id”

Consumer offset:

- kafka stores the offset at which a consumer group has been reading
- the offsets committed are in kafka topic

- When a consumer in a group has processed data received from Kafka, it should be **periodically** committing the offsets (the Kafka broker will write to `__consumer_offsets`, not the group itself)
- If a consumer dies, it will be able to read back from where it left off thanks to the committed consumer offsets!

Delivery semantics for consumers:

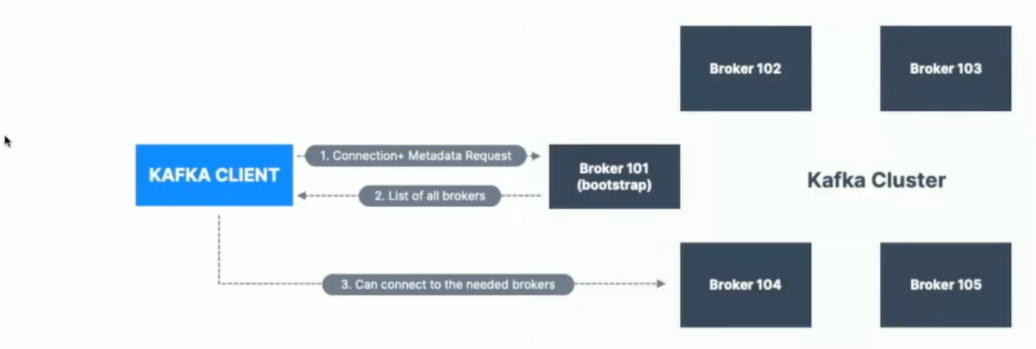
- At least once
  - offset commit after message processed
  - 如果出错，可以重新读
  - might result in duplicate processing of messages. Need to make sure processing again the message won't impact system
- At most once
  - offsets are committed as soon as message received (收到就commit)
  - 出错，message lost
- Exactly once
  - for Kafka => Kafka workflows: use the Transactional API (easy with Kafka Streams API)
  - for Kafka => External System workflows: use an idempotent consumer

Kafka Brokers:

- A cluster include lot's of brokers
- broker identified with ID(integer)
- each broker contains certain topic partitions
- 连上一个broker,就会连上他所在的整个cluster
- at least 3 brokers
-

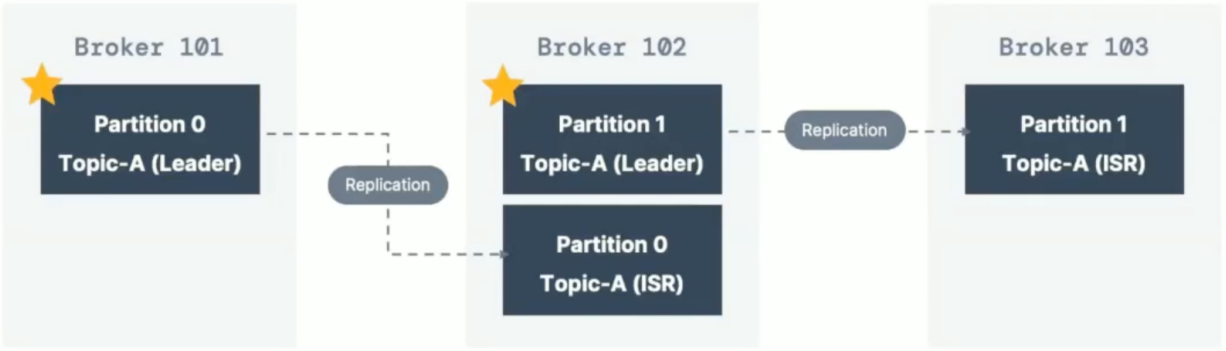
Kafka Broker Discovery

- Every Kafka broker is also called a “bootstrap server”
- That means that **you only need to connect to one broker**, and the Kafka clients will know how to be connected to the entire cluster (smart clients)



Replication:

- replication usually 2 or 3
- one broker down, another broker can serve the data
- only one broker can be the leader of a partition



-
- producer write to the leader, consumer read from the leader
- since Kafka 2.4, consumer can read from closest replica
    - can improve latency, also decrease network costs if using cloud

Producer Acknowledgements (acks):

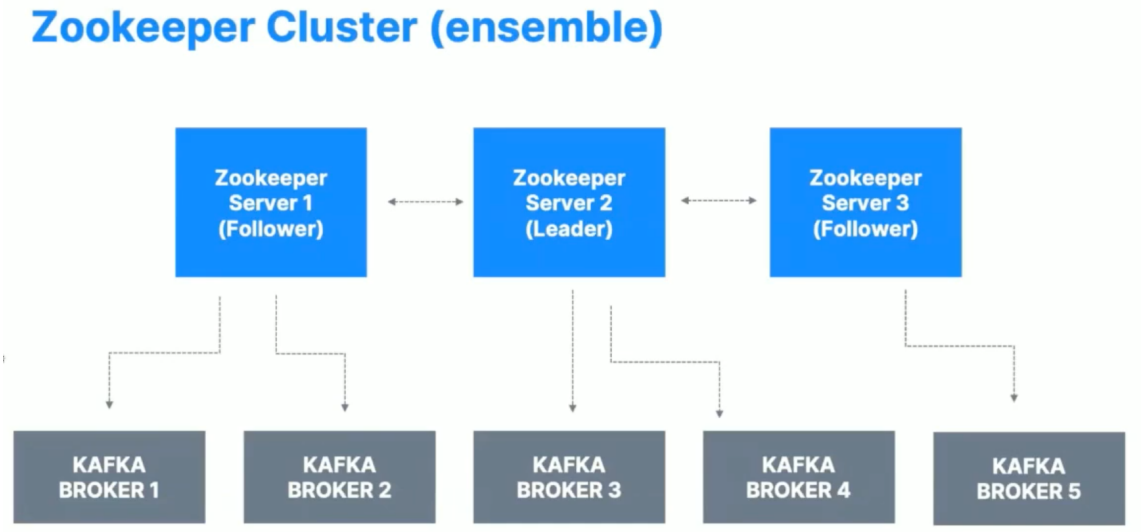
Producer choose to receive acknowledgement of data writes:

- acks = 0: producer won't wait for acknowledgement (possible data loss)
- acks = 1: wait for leader (limit data loss)
- acks = all: leader + all replicate (no data loss)

Kafka topic durability:

- replicate = 3, 那就可以忍受2个broker loss

Zookeeper:

- zookeeper manage brokers (keep list of brokers)
- select leader for partition
- send notification to kafka in case of changes(new topic, broker down, broker come up, delete topic, etc)
- kafka2.0x can't work without zookeeper
- kafka3.0x cn work without zookeeper – using kafka raft instead
- kafka4.0x no zookeeppper
- operate with odd number of servers(1, 3, 5, 7, 通常不会>7)
- zookeeper也有leader
- 

Should you use Zookeeper?

- With Kafka Brokers?
  - Yes, until Kafka 4.0 is out while waiting for Kafka without Zookeeper to be production-ready
- With Kafka Clients?
  - Over time, the Kafka clients and CLI have been migrated to leverage the brokers as a connection endpoint instead of Zookeeper
  - Since Kafka 0.10, consumers store offset in Kafka and Zookeeper and must not connect to Zookeeper as it is deprecated
  - Since Kafka 2.2, the `kafka-topics.sh` CLI command references Kafka brokers and not Zookeeper for topic management (creation, deletion, etc...) and the Zookeeper CLI argument is deprecated.
  - All the APIs and commands that were previously leveraging Zookeeper are migrated to use Kafka instead, so that when clusters are migrated to be without Zookeeper, the change is invisible to clients.
  - Zookeeper is also less secure than Kafka, and therefore Zookeeper ports should only be opened to allow traffic from Kafka brokers, and not Kafka clients

Why remove zookeeper:

by removing, kafka can:

- scale to millions of partitions, easier to maintain and setup
- improve stability, easier to monitor, support and administer
- single security model for whole system
- single process to start with kafka
- faster controller shutdown and recover time