



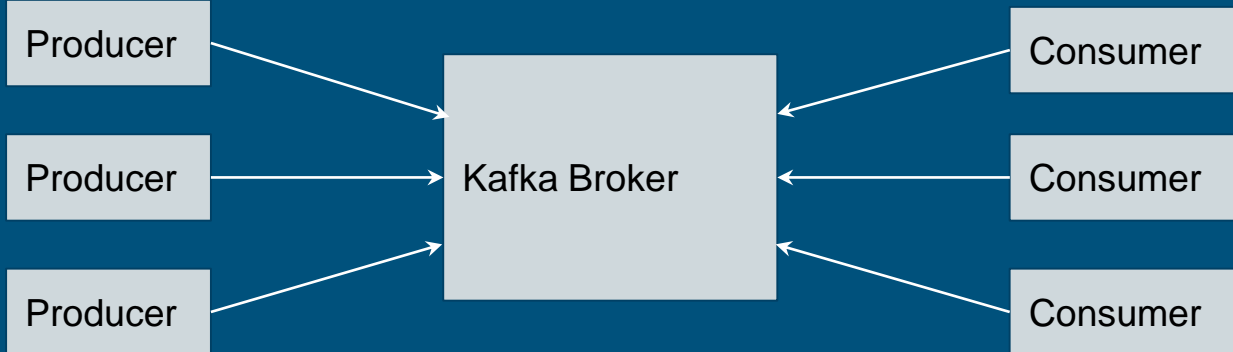
Kafka



A distributed streaming engine



Pub-Sub Architecture

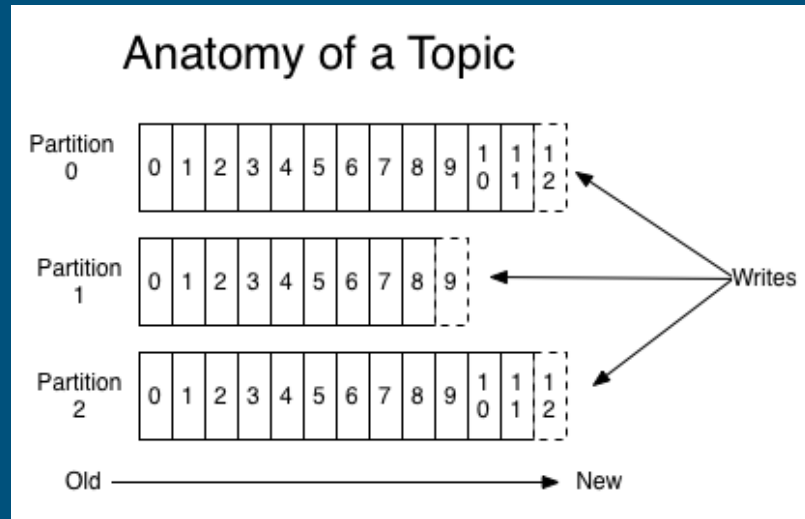


APIs

- **Producer**
 - Push data into a topic
- **Consumer**
 - Pull data from a topic
- **Streaming**
 - Producer + Consumer API with inbuilt common functionalities
- **Connector**
 - Connect with any other system to push or pull data

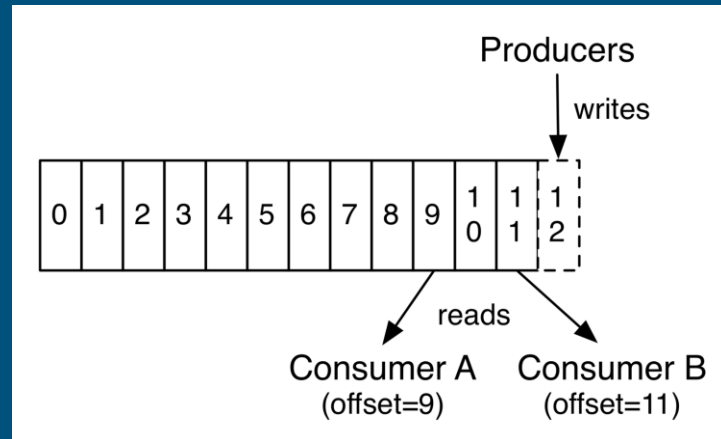
Topic

- Abstractions over stream of records
- Or in other word, its a name tag of data stream
- Kafka maintains a partitioned log for each topic
- Each topic has multiple partitions within it



Partition

- Structured record holder that is
 - Ordered
 - Immutable
 - Append only
- Each record has a unique sequence id
- Persists all record no matter consumed or not for a specified retention period
- Consumer can manage offset to read data from a partition
- Hence, consumer can go and join any time
- Producer has choice to push individual record to a specific partition



Benefits of Partitions

- Allows log to scale beyond capacity of a single node
- Provides unit of parallelism

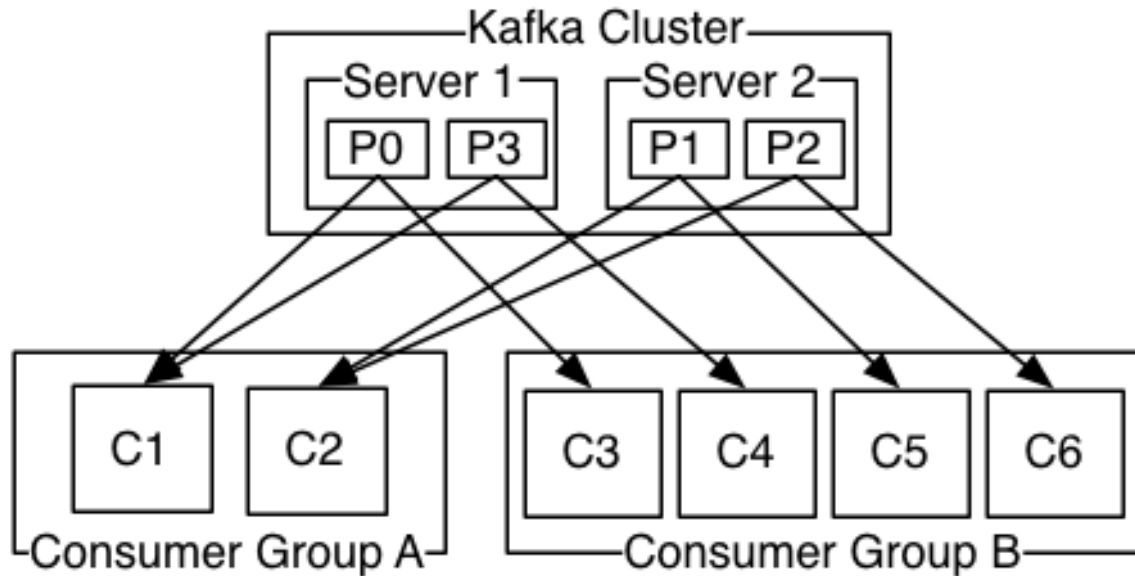
Fault tolerance

- Each partition is replicated (configurable) across multiple nodes to achieve fault tolerance
- One server will work as Leader and rest as followers
- Leader will be entertaining all read-write requests
- Followers will just make sure to be in sync with leader
- One of the followers will push himself as leader in case existing leader crashes
- Hence, a topic with N replication factor can tolerate N-1 failures
- Quick question, who elect new leader?

But what about Consumer?

- Consumers can form a group
- Kafka will deliver each message to at least one of the consumer within a group
- Do we need to have all consumers in same process?
 - No, they can be in different processes or even machines
- What if all consumers are in same group
 - Kafka will do the load balancing
- And if all are in different groups
 - Simple! broadcasts each record to all of them

Message delivery



What if a message fails to deliver

- Various message delivery semantics
 - a. Exactly once
 - b. At most once
 - c. At least once: This is what Kafka follows

More clarification on load balancing

- Partitions will be divided equally among #consumers in a group
 - If new consumer joins then it will grab some from existing members
 - And if one fails then kafka will distribute equally among remaining
- Wait! What about order of records then?
 - Total order will be maintained within partition but not between partitions

Ordering in Distributed System

- FIFO Ordering
 - If event m was generated before m' then m will be delivered before m'
- Causal Ordering
 - If m has happened before m' then m will be delivered before m'
- Total Ordering
 - No matter when m and m' were generated, but if one **correct process** receives m before m' then all the correct processes should receive m before m'

Kafka is just for processing?

- Not really! It can be used for distributed storage too
- How?

Use cases

- Messaging
- Website activity tracking
- Log aggregation
- Stream processing
- And many more...

Reference

- <https://stackoverflow.com/questions/44014975/kafka-consumer-api-vs-stream-api>
- <https://kafka.apache.org/intro>