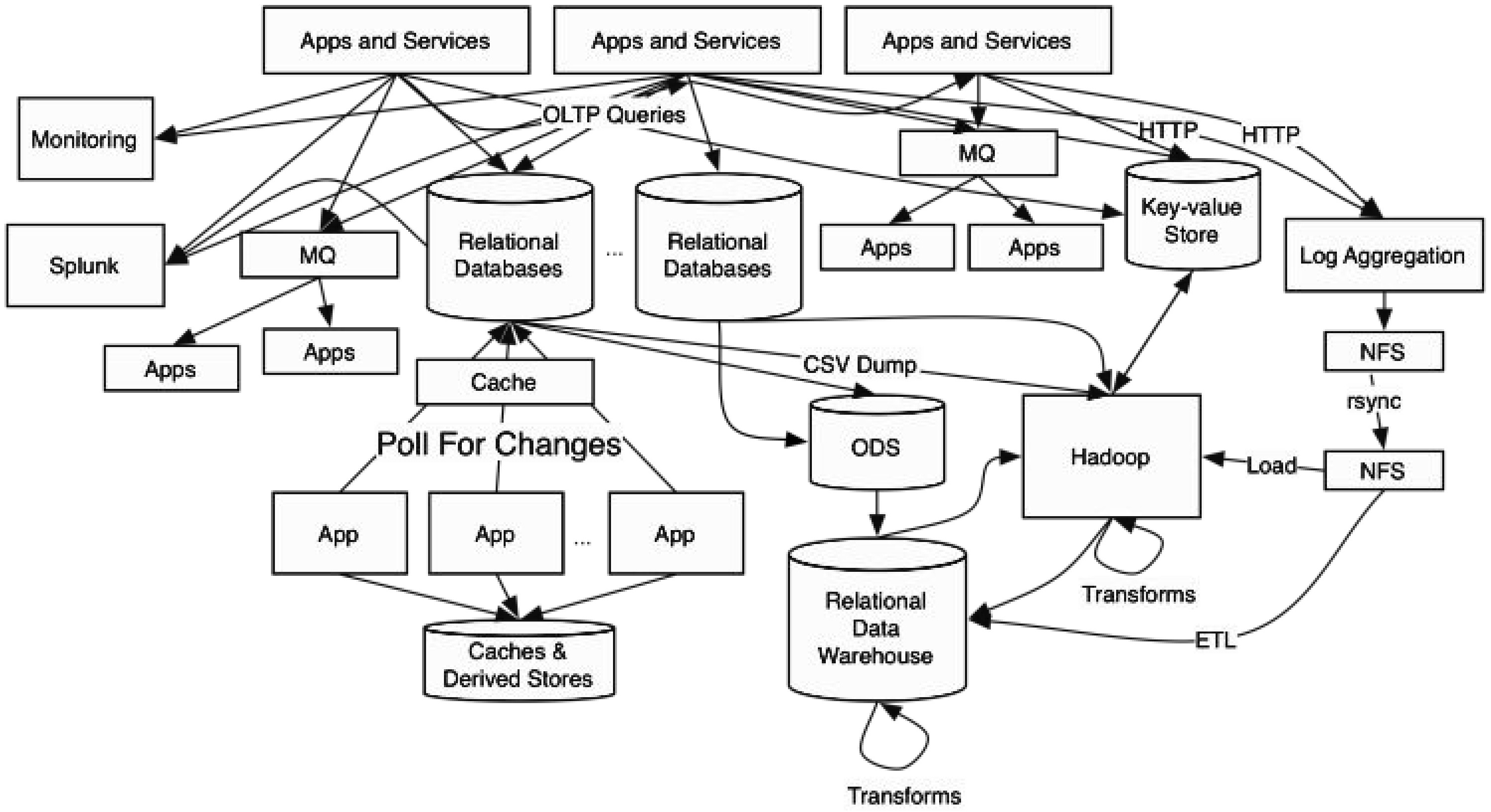
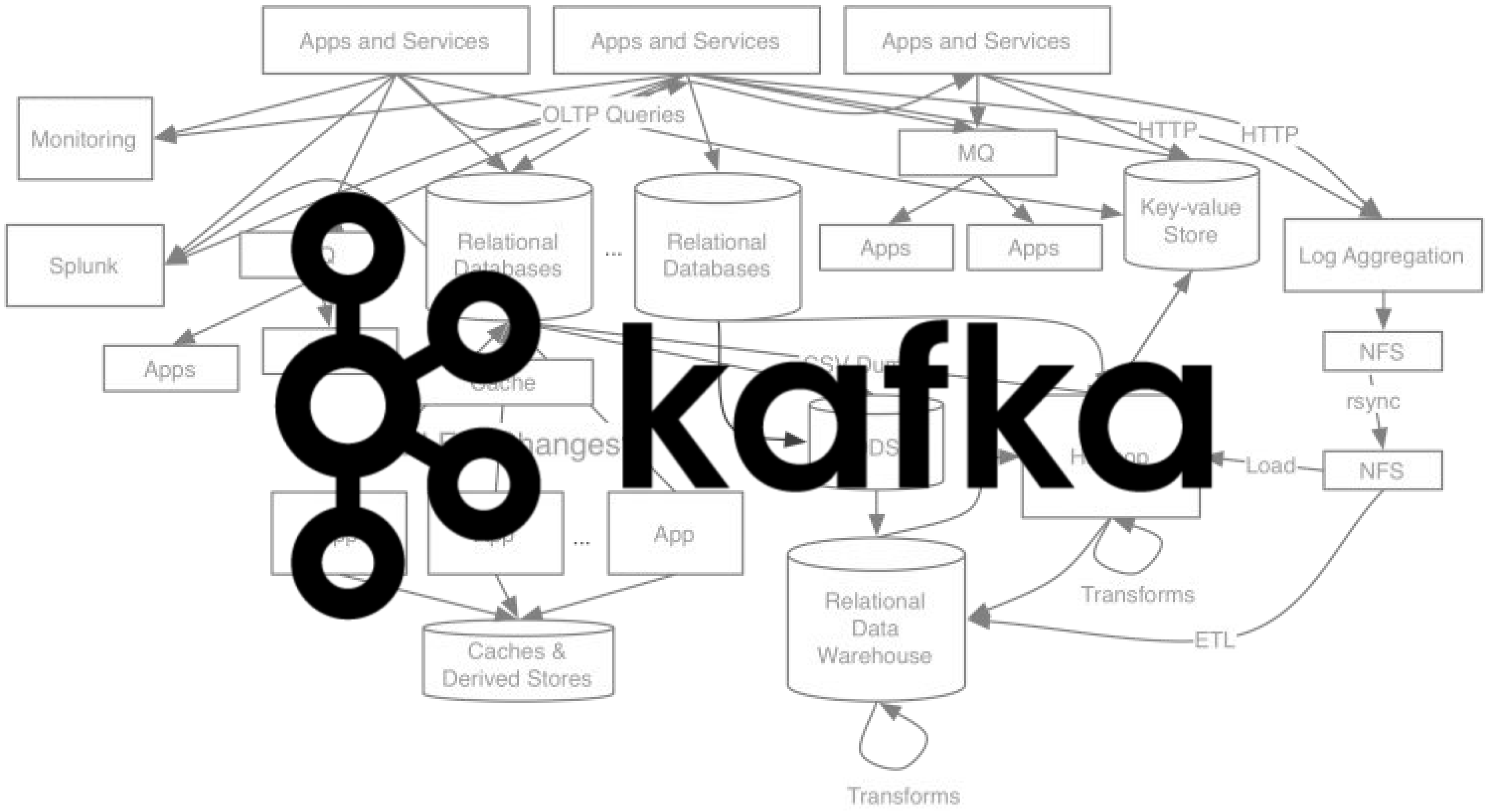
APACHE kafka™
A distributed streaming platform

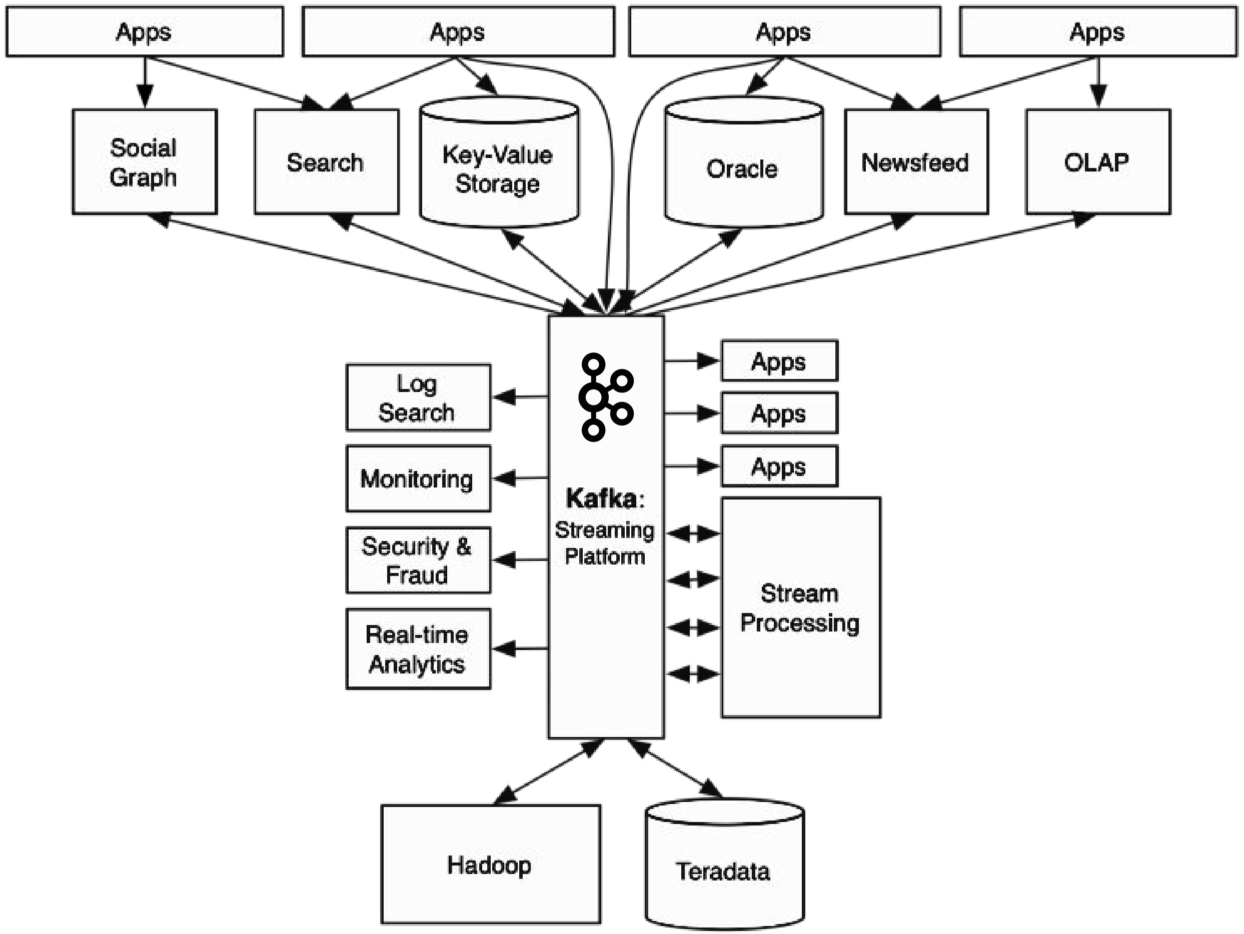
Aleš Kopecký - Martin Vrábel - Norbert Fabián

What is Apache Kafka?

- Distributed streaming platform
- Originated in 2010 at **LinkedIn**, in 2011 open sourced at **Apache** and now managed by **Confluent** group
- Written in **Java** and **Scala**
- Fast, scalable, distributed, fault tolerance
- Real-time streaming data pipelines between systems/applications
- Real-time reaction or transformation of streams of data
- Uses **Apache Zookeeper**
    - as a distributed store,
    - for distributed configuration service,
    - for synchronization between nodes


Apps and Services
Apps and Services
Apps and Services
Monitoring
OLTP Queries
HTTP
HTTP
MQ
Splunk
MQ
Relational Databases
...
Relational Databases
Apps
Apps
Key-value Store
Log Aggregation
Apps
Apps
Cache
CSV Dump
NFS
rsync
Poll For Changes
ODS
Hadoop
Load
NFS
App
App
...
App
Relational Data Warehouse
Transforms
ETL
Caches & Derived Stores
Transforms


Apps and Services
Apps and Services
Apps and Services
Monitoring
OLTP Queries
HTTP
HTTP
MQ
Splunk
Relational Databases
...
Relational Databases
Apps
Apps
Key-value Store
Log Aggregation
Apps
Cache
NFS
rsync
Load
NFS
App
...
App
Transforms
Relational Data Warehouse
ETL
Caches & Derived Stores
Transforms
kafka


Apps
Apps
Apps
Apps
Social Graph
Search
Key-Value Storage
Oracle
Newsfeed
OLAP
Kafka: Streaming Platform
Log Search
Monitoring
Security & Fraud
Real-time Analytics
Apps
Apps
Apps
Stream Processing
Hadoop
Teradata

Used by

NETFLIX

Spotify®

UBER

PayPal

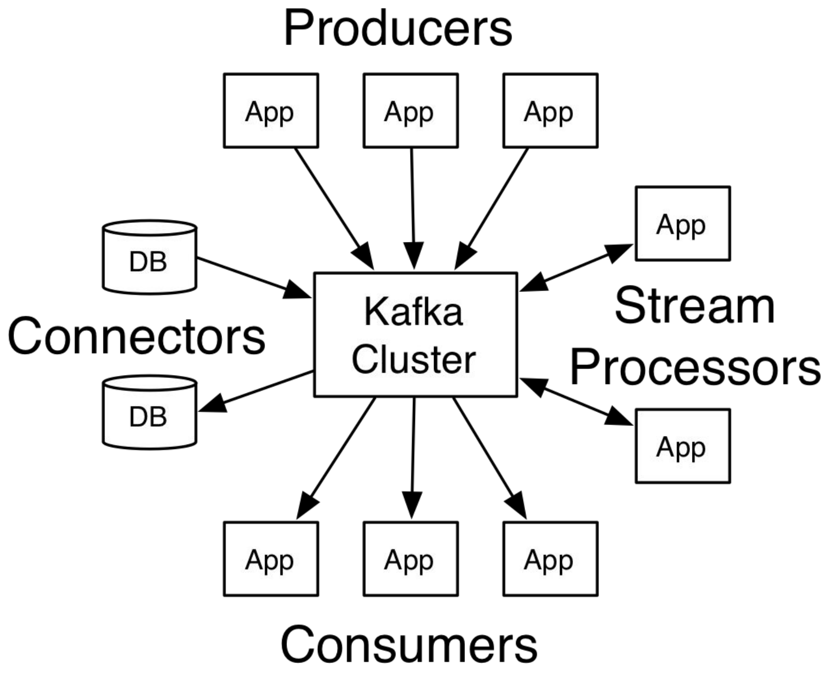
Walmart

ebay

cisco

Terminology

- **Topics**: stored streams of records
- **Producers**: publishing stream of records to topics
- **Consumers**: subscribing topics and processing records
- **Stream processor**: consume - transform - produce

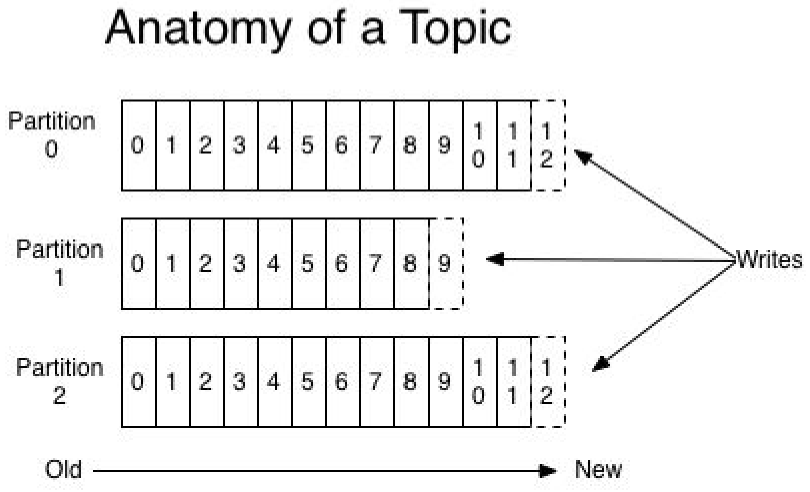


Topic

- Category
- Multisubscriber (0 - n consumers)
- Partitioned log
- Retention period

Partition

- Ordered sequence of records
- Record has sequential ID - offset

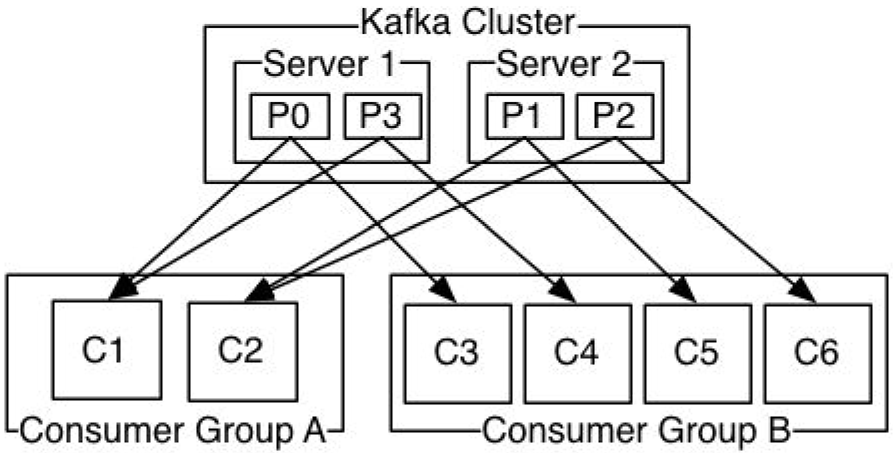


Producers

- Publishing data to topics (choosing the partition)
- Responsible for choosing which record to assign to which partition within the topic
- Round-robin (balance load)
- Semantic partition function (based on keys in records)

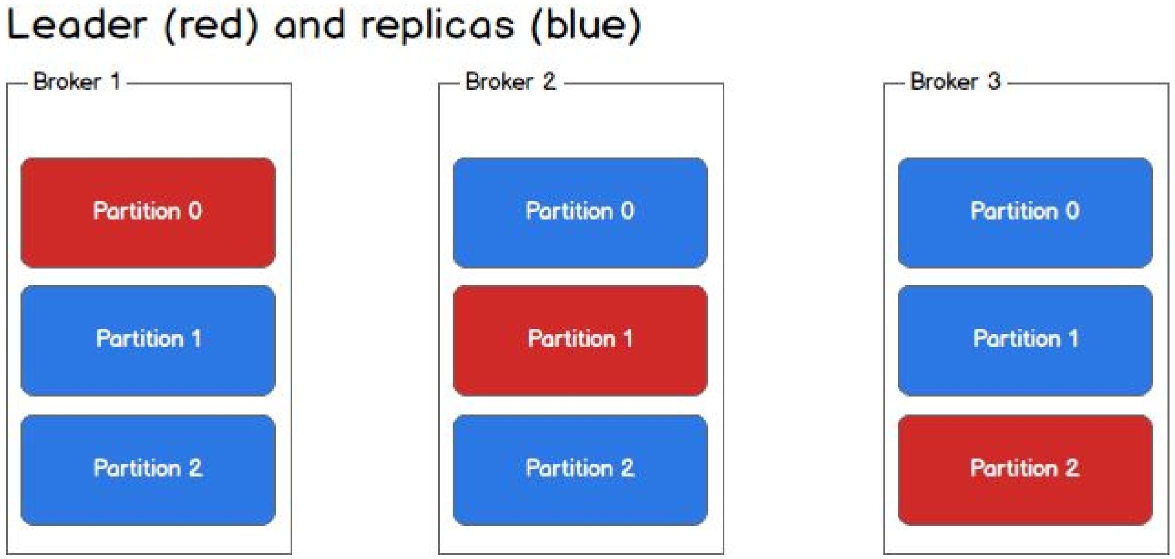
Consumer groups

- Generalize queuing and publish-subscribe
- Consist of consumer instances
- Record from topic is delivered to on consumer instance from a group



Replication

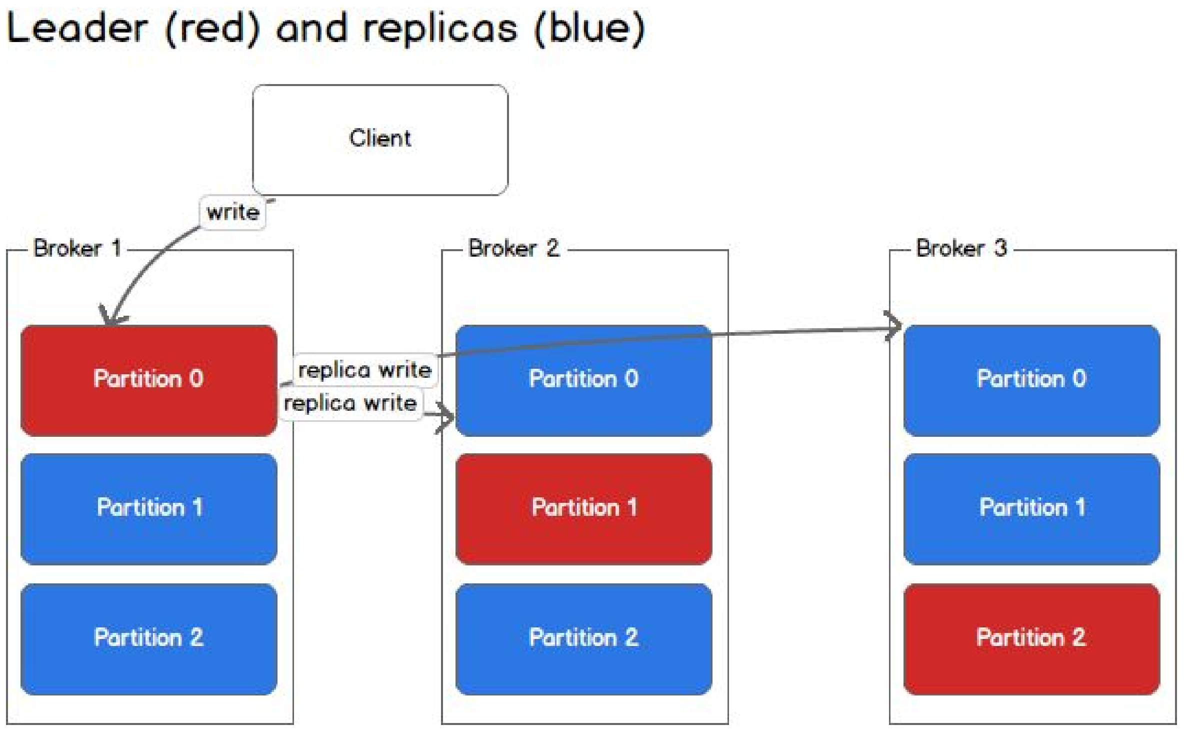
- Configurable, based on data importance.
- Automated replica management.



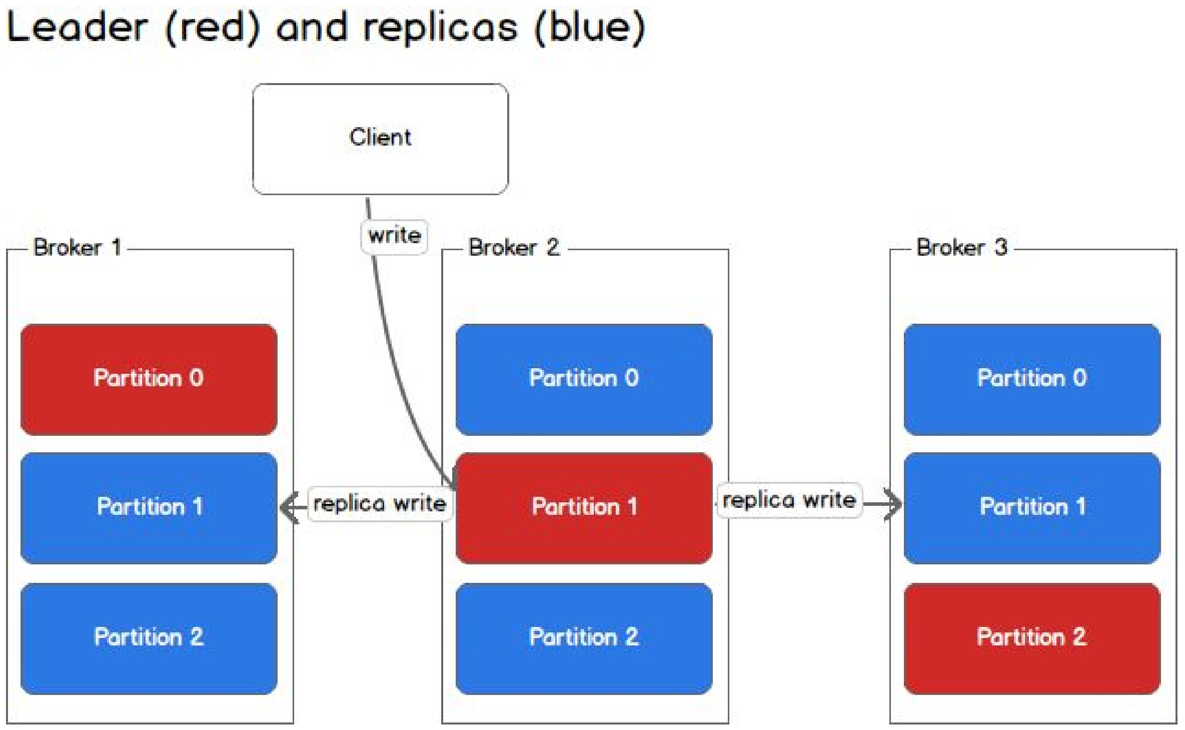
Replication

- Since each machine is responsible for each write, throughput of the system as a whole is increased.
- When communicating with a Kafka cluster, all messages are sent to the partition’s leader.
- The leader is responsible for writing the message to its own in sync replica and for propagating the message to additional replicas on different brokers.
- Each replica acknowledges that they have received the message and can now be called in sync.

Replication



Replication



Handling Failures

replica fails

- When replica dies, nothing happens.
- When leader dies last, there are **2 solutions**:
    - Wait for leader to back up and as the replicas are brought back online they will be made in sync with the leader.
    - Elect the first broker to come back up as the new leader => all data written between the time where this broker went down and when it was elected the new leader will be lost.

Handling Failures

leader fails

- The Kafka controller will detect the loss of the leader and elect a new leader from the pool of in sync replicas.
- This may take a few seconds and result in LeaderNotAvailable errors from the client.
- Producers and consumers must handle this situation on their own.

Consistency as a Kafka Client

PRODUCERS:

- wait for all in sync **replicas** to **acknowledge** the message
- wait for only the **leader** to **acknowledge** the message
- **do not wait** for acknowledgement

CONSUMERS:

- receive each message **at most once**
- receive each message **at least once**
- receive each message **exactly once**

Consumers

receive each message at most once

- Consumer reads data from a partition, commits the offset that it has read, and then processes the message.
- If the consumer crashes between committing the offset and processing the message it will restart from the next offset without ever having processed the message.
- This would lead to potentially undesirable message loss.

Consumers

receive each message at least once

- For at least once delivery, the consumer reads data from a partition, processes the message, and then commits the offset of the message it has processed.
- This leads to duplicate messages in downstream systems but no data loss.

Consumers

receive each message exactly once

- Exactly once delivery is guaranteed by having the consumer process a message and commit the output of the message along with the offset to a transactional system.
- If the consumer crashes it can re-read the last transaction committed and resume processing from there.
- This leads to no data loss and no data duplication.
- In practice however, exactly once delivery implies significantly decreasing the throughput of the system as each message and offset is committed as a transaction.

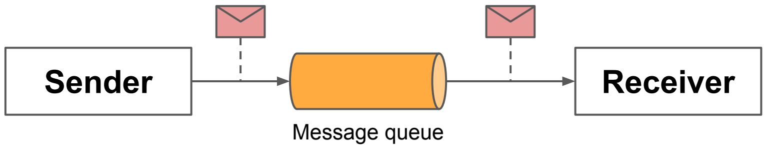
Performance

- highly influenced by clients
- benchmark comparison↗ of Apache Kafka and RabbitMQ

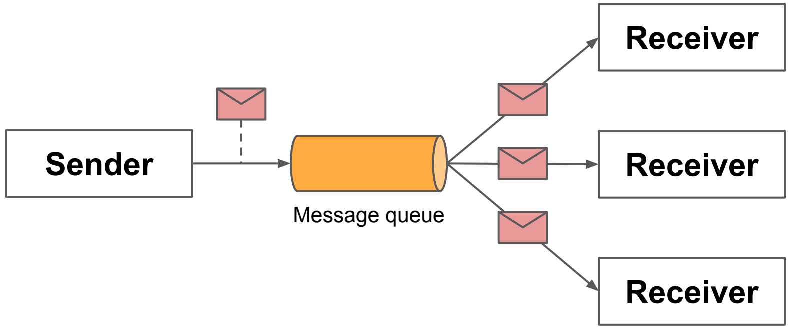
Kafka

as messaging system

→ 2 models:



Queuing



Publish-subscribe system

Kafka

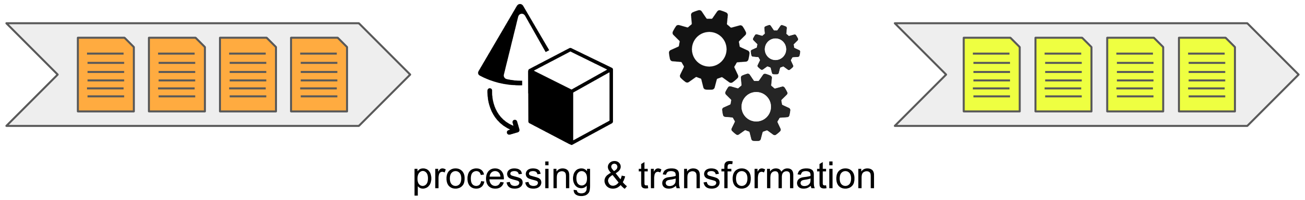
as storage system

- data written to disk
- replicated for fault-tolerance
- scaling of disk structures
- producer waits for acknowledgement of write
- special purpose distributed filesystem

Kafka

for stream processing

- usage of pipelining



- from simple processing with producer/consumer API to complex transformations:

  Streams API

  ⇒ built-in powerful and lightweight library for stream data processing and analyzing, available from v0.10

Use cases

- Messaging
  - replacement for message broker (ActiveMQ, RabbitMQ...)
- Website activity tracking
  - real-time processing and monitoring of page views, searches, user interactions..
- Metrics
  - produce centralized feeds of statistical data from distributed applications
- Log Aggregation
  - abstraction for lower-latency processing of logs from multiple data sources
- Microservices
- Event Sourcing
  - style of application design where state changes are logged as sequence of records
- Commit Log
  - external commit log from distributed system with replication between nodes

Demo

Questions?

APACHE kafka™
A distributed streaming platform

Thank you for the attention

Aleš Kopecký - Martin Vrábel - Norbert Fabián