codecentric

Kafka 101_

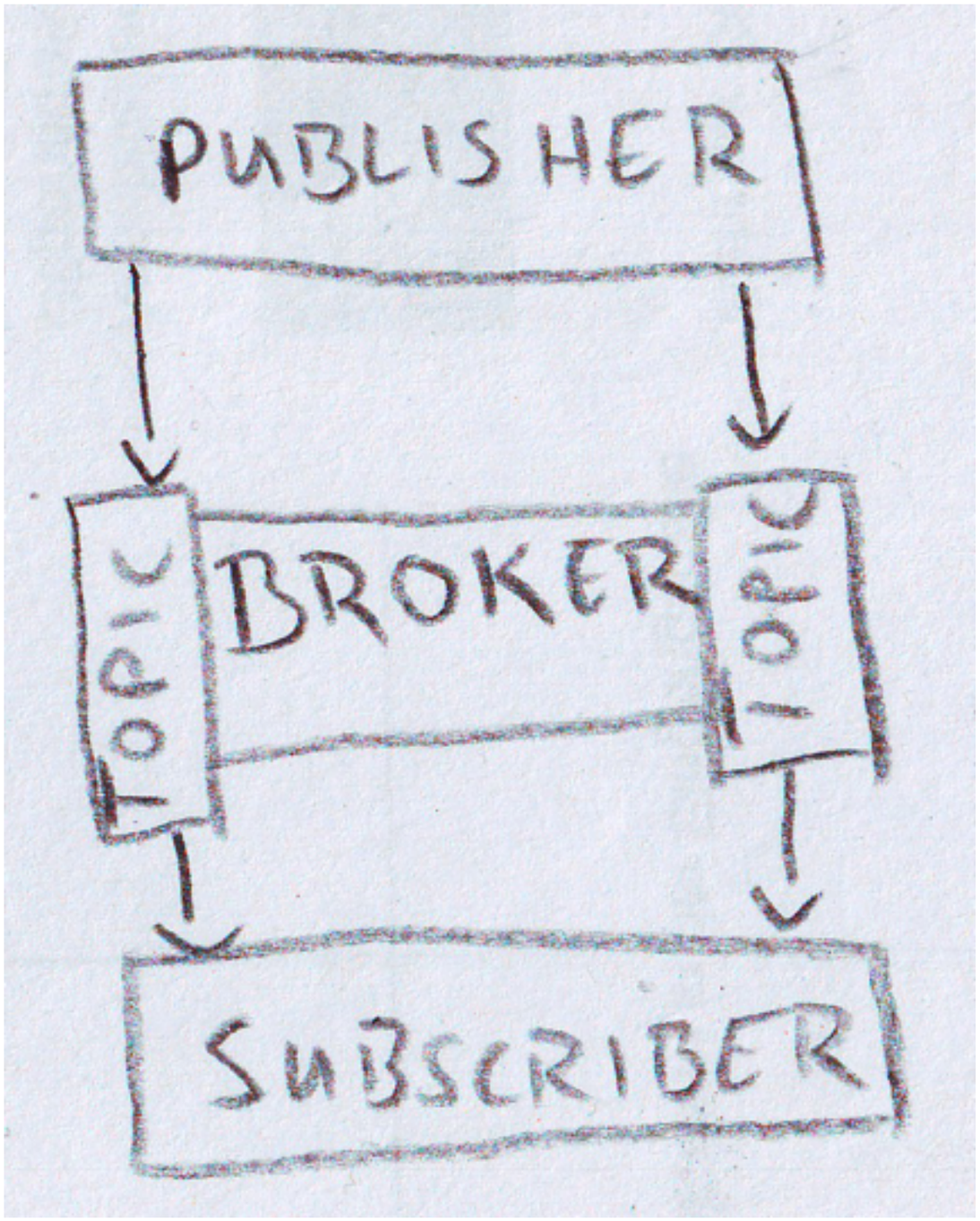
A Messaging Service

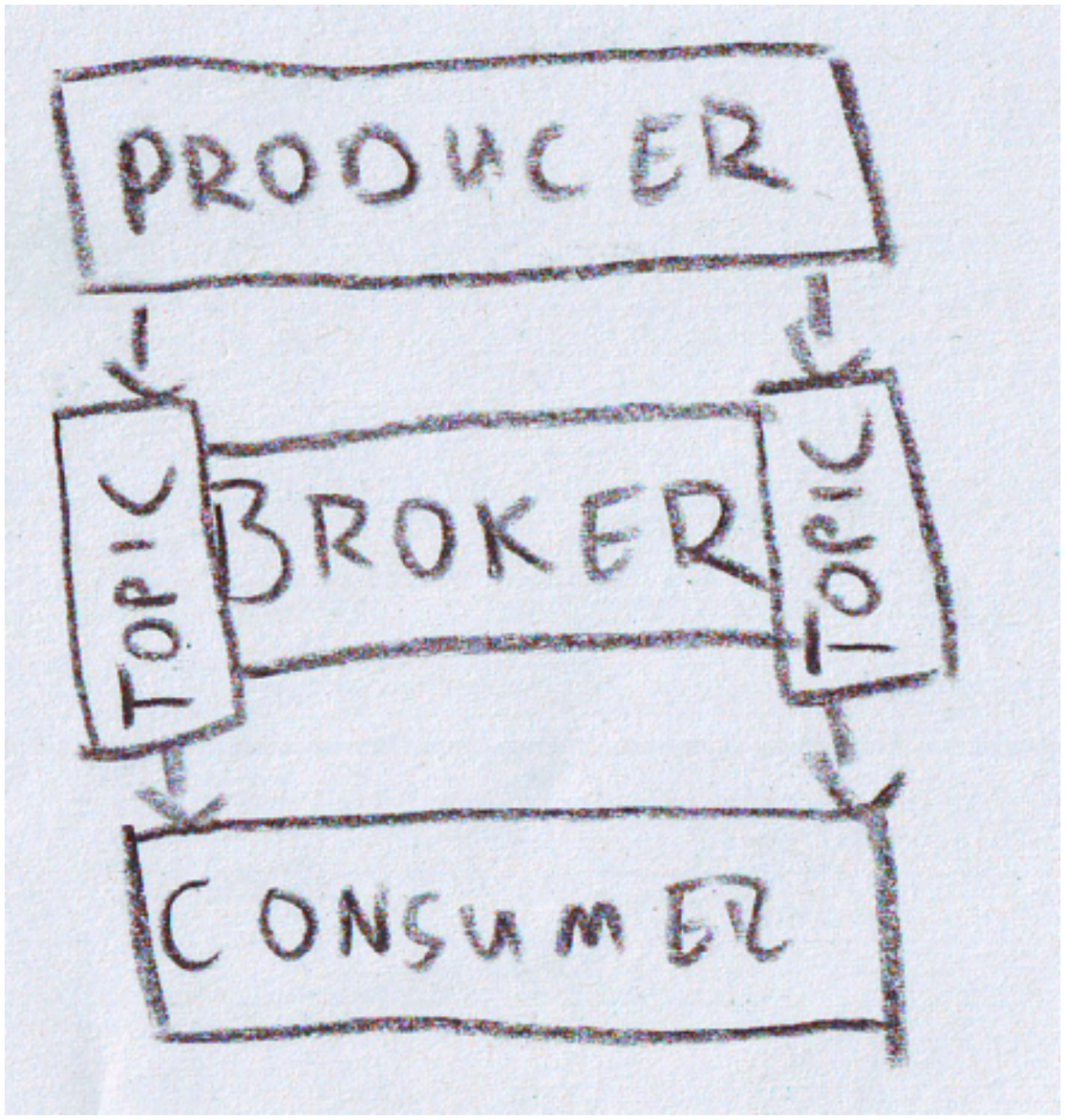
- Pub-Sub
- Distributed
- Replicated
- Commit Log based
- High Throughput
- Scalable

Where did it come from?_

- Developed at LinkedIn
- Open sourced in early 2011 at ASF
- Graduated from incubator in October 2012
- Original Developers formed Confluent in 2014

Pub-Sub Messaging





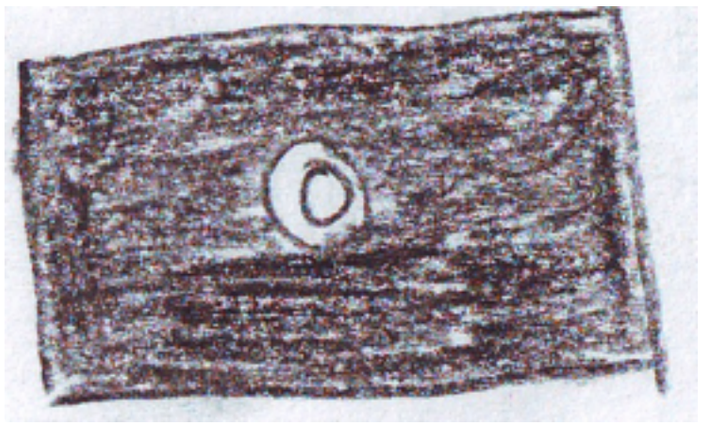
Messages

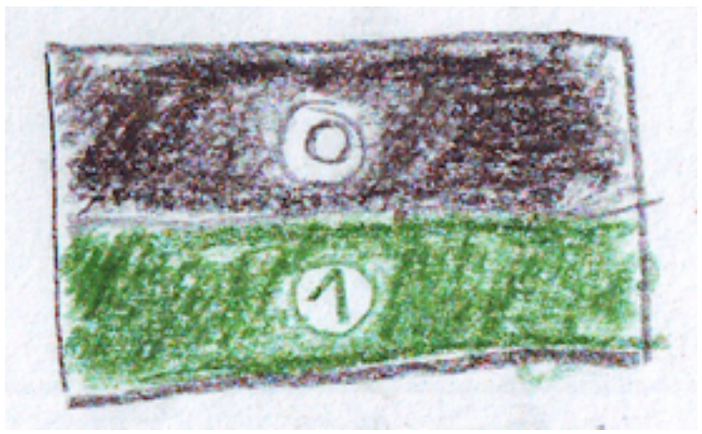
Key-Value Pairs

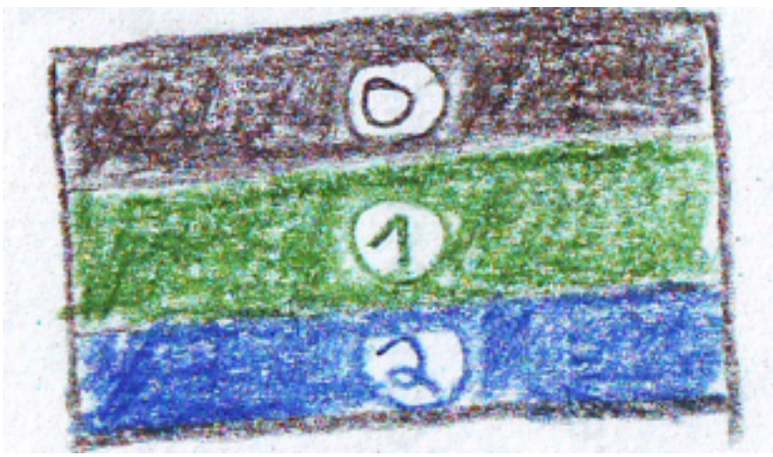
- Just bytes
- Can be zipped
- Arbitrary size
- Keys not necessarily unique

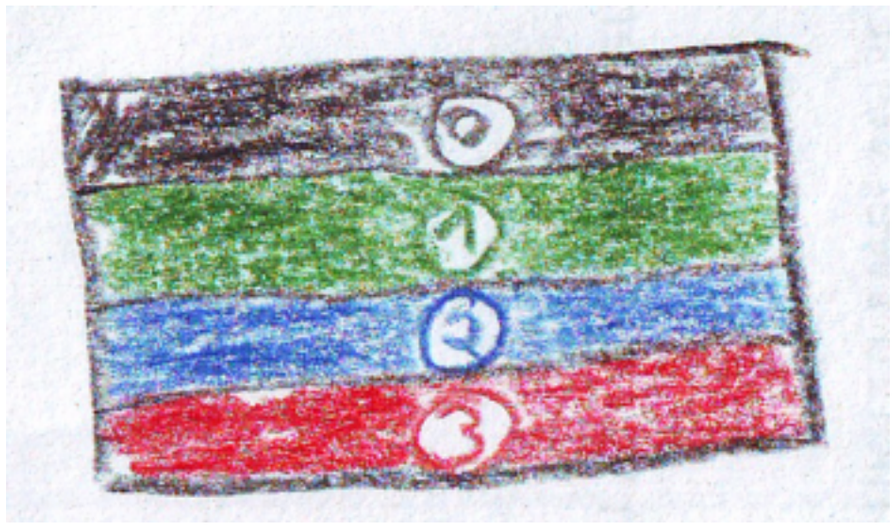
Topics, Partitions and Brokers

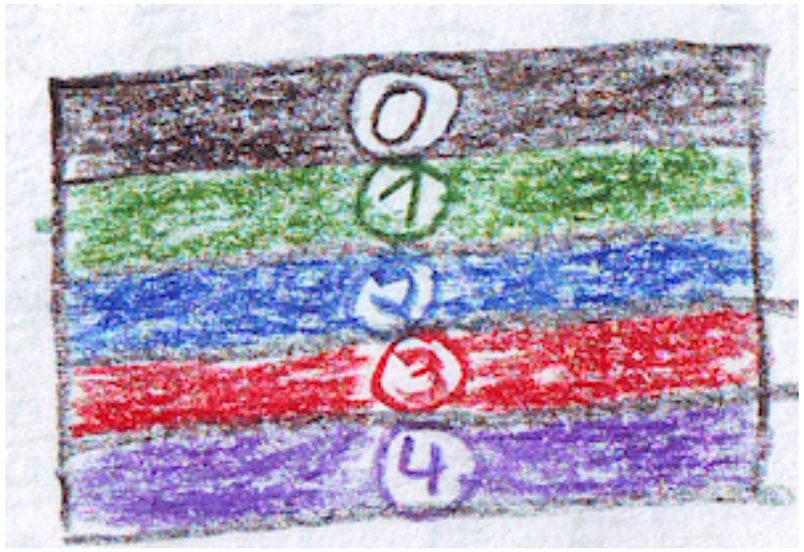
Topics are partitioned!



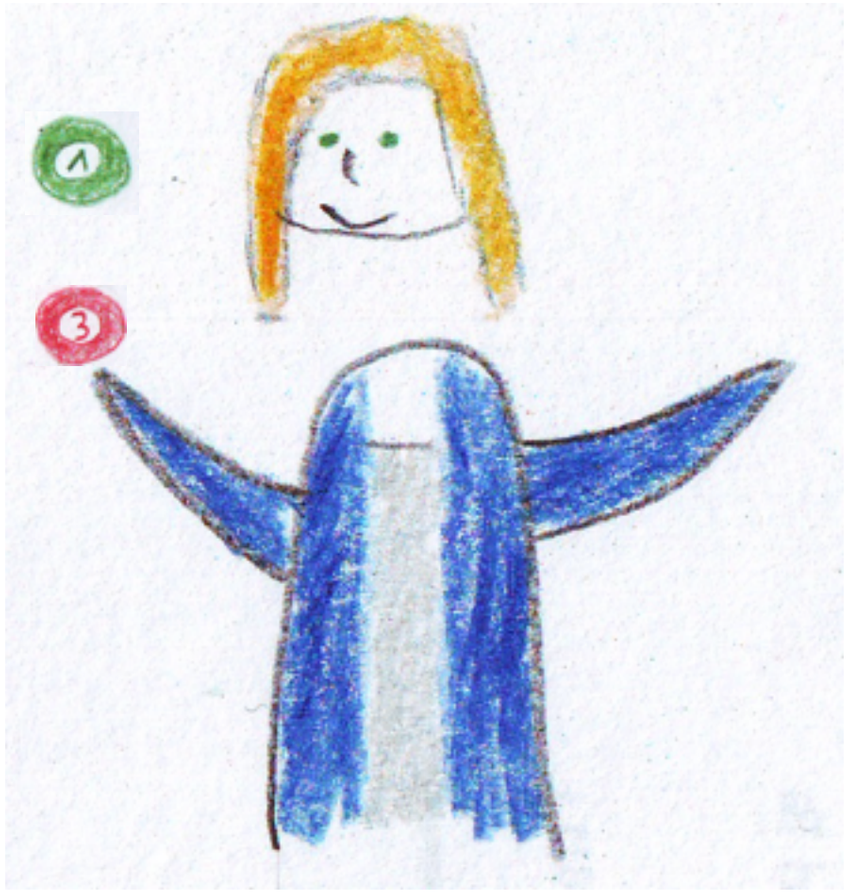




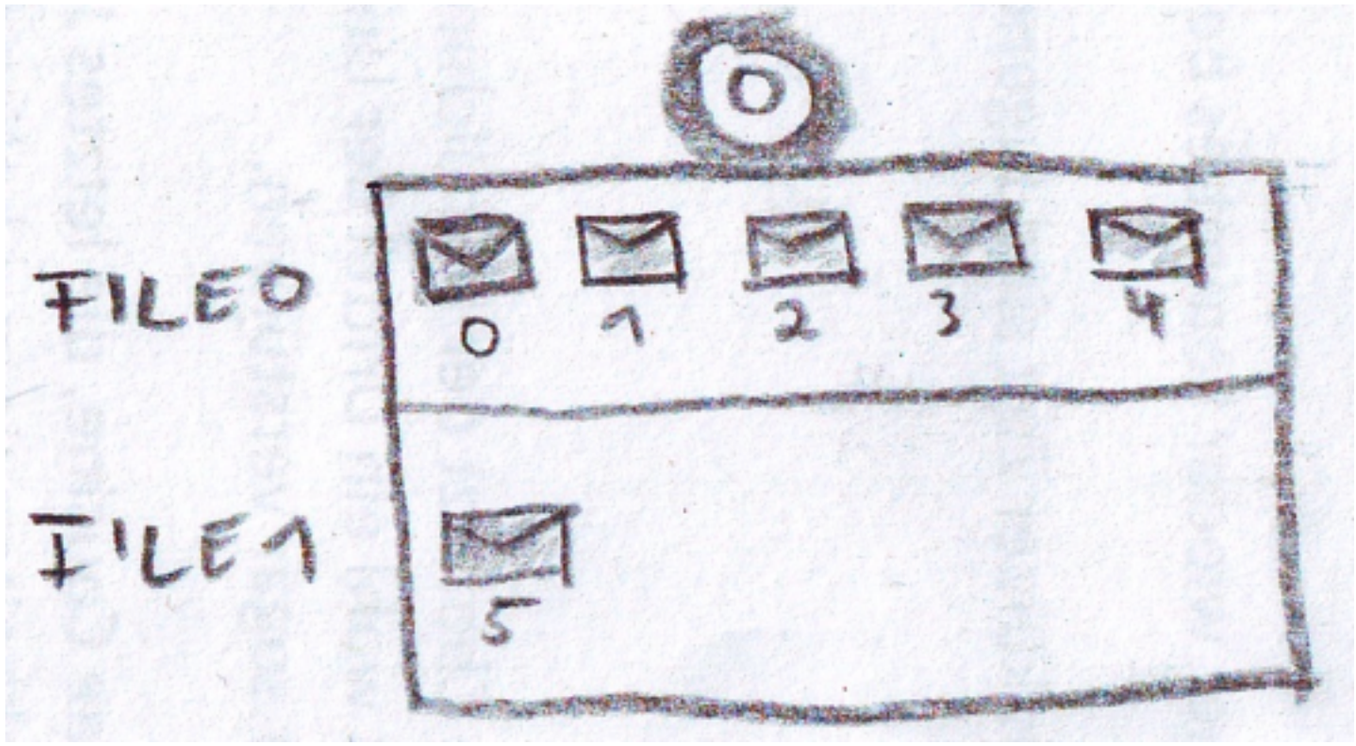




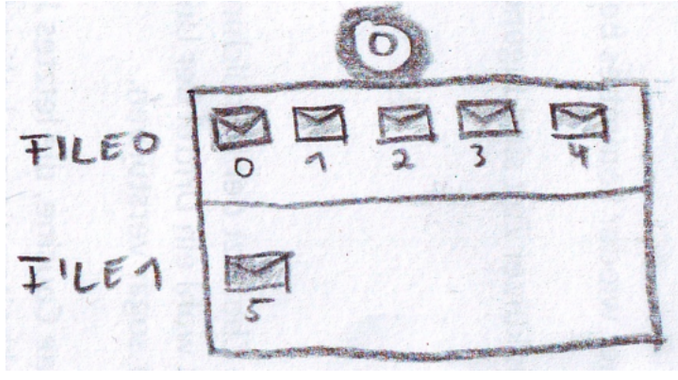
Partitions are assigned to Brokers



Partitions are stored as a commit log

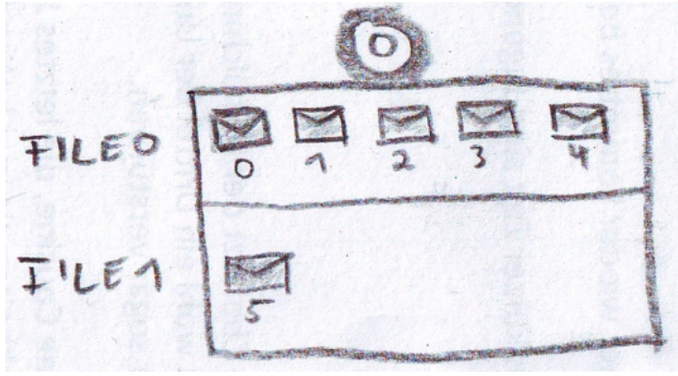


Messages are appended to partitions



- n partitions per Broker (even from single topic!)
- Each message in a partition gets a unique offset
  - uniqueness per partition!

Don't fear the file system!



- Append-only files (1..n per partitions)
- Configurable retention time
- Optimised for linear reads and writes
- In most use cases no disk access due to page cache
- Zero-Copy
  - Data goes directly from page cache to socket!
  - allows Kafka to have (comparatively) low memory footprint
  - Not for SSL :(

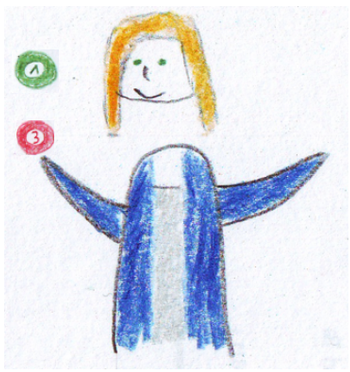
Producing messages

Meet the Producers

Producers send Messages to Brokers

- They decide which partition a message belongs to
- Messages always get sent directly to responsible Broker
- Producers specify expected consistency levels

Producing Messages_



Consuming messages

Hello Consumers

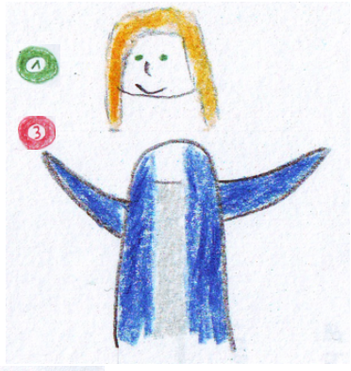
Consumers

- Subscribe to a topic
- Get partitions assigned
- Specify their “Consumer Groups”
  - Partitions are automatically divided among all consumers in a Consumer Group
- Only one concurrent consumer per partition and Consumer Group

Offsets

- Consumers commit their read offsets
  - automatically
  - manually
- Can restart from last committed offset after failure or restart
- Can rewind to historical offsets

Consumers_



Carl+Christine

Charlotte

Failure tolerance

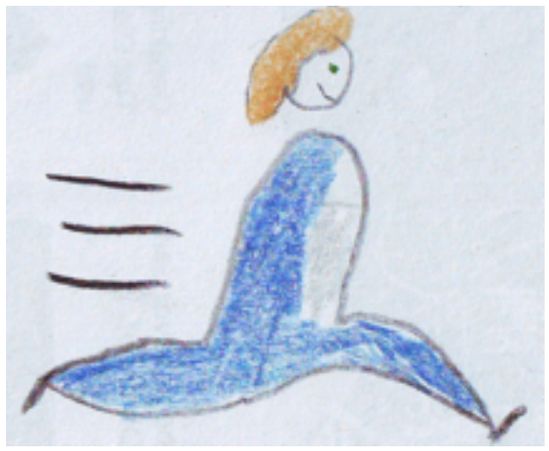
Each topic gets a replication factor

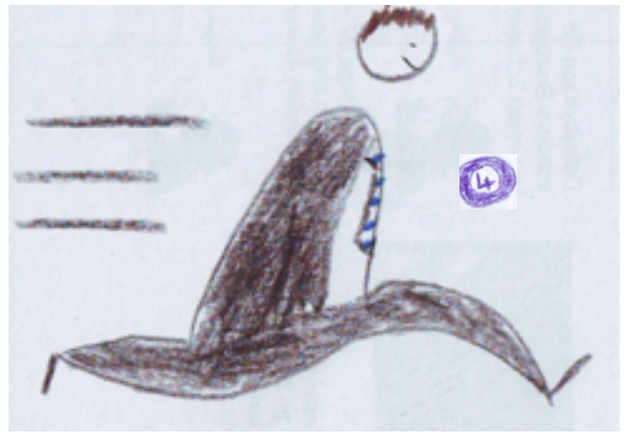
- One broker acts as leader for a partition
- n brokers serve as followers
  - Just serving as backup, never delivering messages for that partition!
- On leader failure, there is a leadership election
  - External dependency on Apache Zookeeper!
  - Leader gets reelected from pool of In-Sync-Replicas
  - Clients get notified via Kafka Protocol

Replication factor 2

Replication as pull by followers

Replicas that are caught up with the leader (per partition)



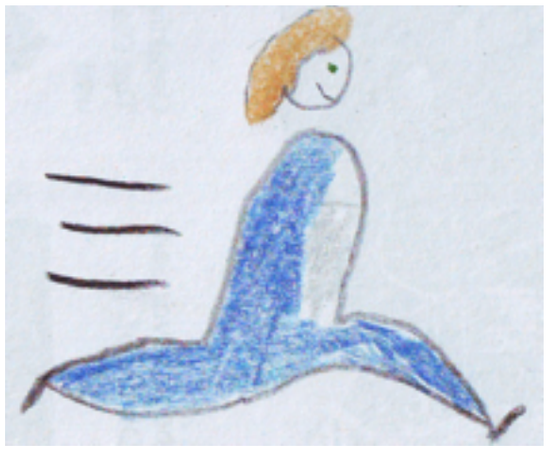


In-Sync-Replicas

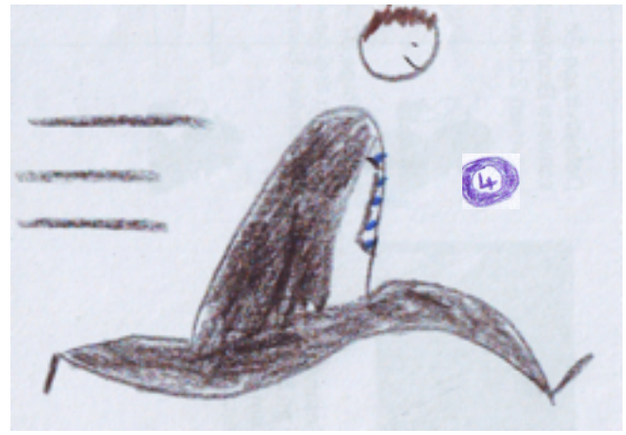
Leader

Replicas that are no longer caught up with the leader

Out-of-Sync-Replica



In-Sync-Replica



Leader

Producers set desired Consistency Level

- Number of acknowledgements from In-Sync-Replicas
- Acknowledgement to Producer after CL serviced
  - *0*
    - *no acknowledgement required*
  - *1*
    - *as soon as leader writes message*
  - *all*
    - *when all In-Sync-Replicas answered*
    - *if there is no In-Sync-Replica, this will be satisfied when Leader is done!*