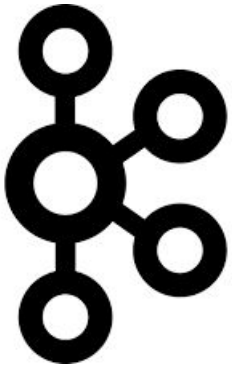


Apache Kafka

Event Streaming Platform

Introduction

Apache Kafka is an open source **publish-subscribe messaging system** based on the concept of a **distributed commit log**.

Messages (aka **records**) in Kafka are distributed, stored durably and in order, and can be read deterministically.

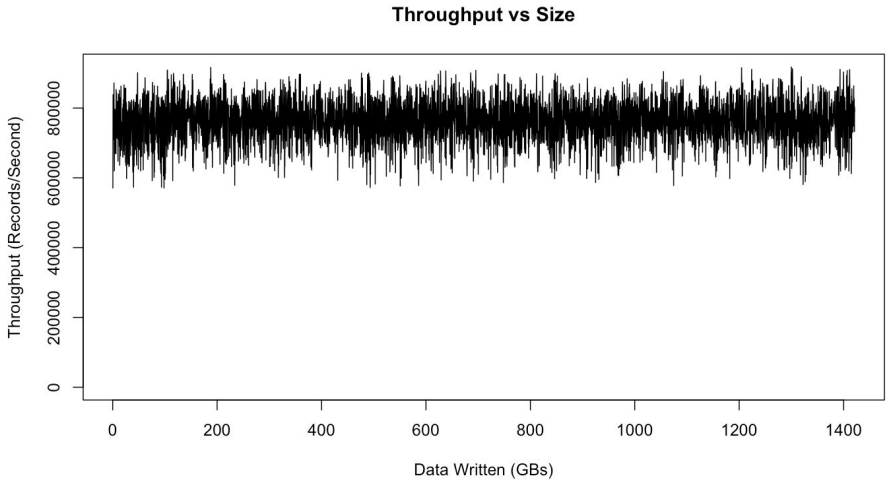
How fast is Kafka?

“Up to 2 million writes/sec on 3 cheap machines”

- Using 3 producers on 3 different machines, 3x async replication

Sustained throughput as stored data grows

- Slightly different test config than 2M writes/sec above.



Why is Kafka so fast?

Fast writes:

While Kafka persists all data to disk, essentially all writes go to the page cache of OS, i.e. RAM.

Fast reads:

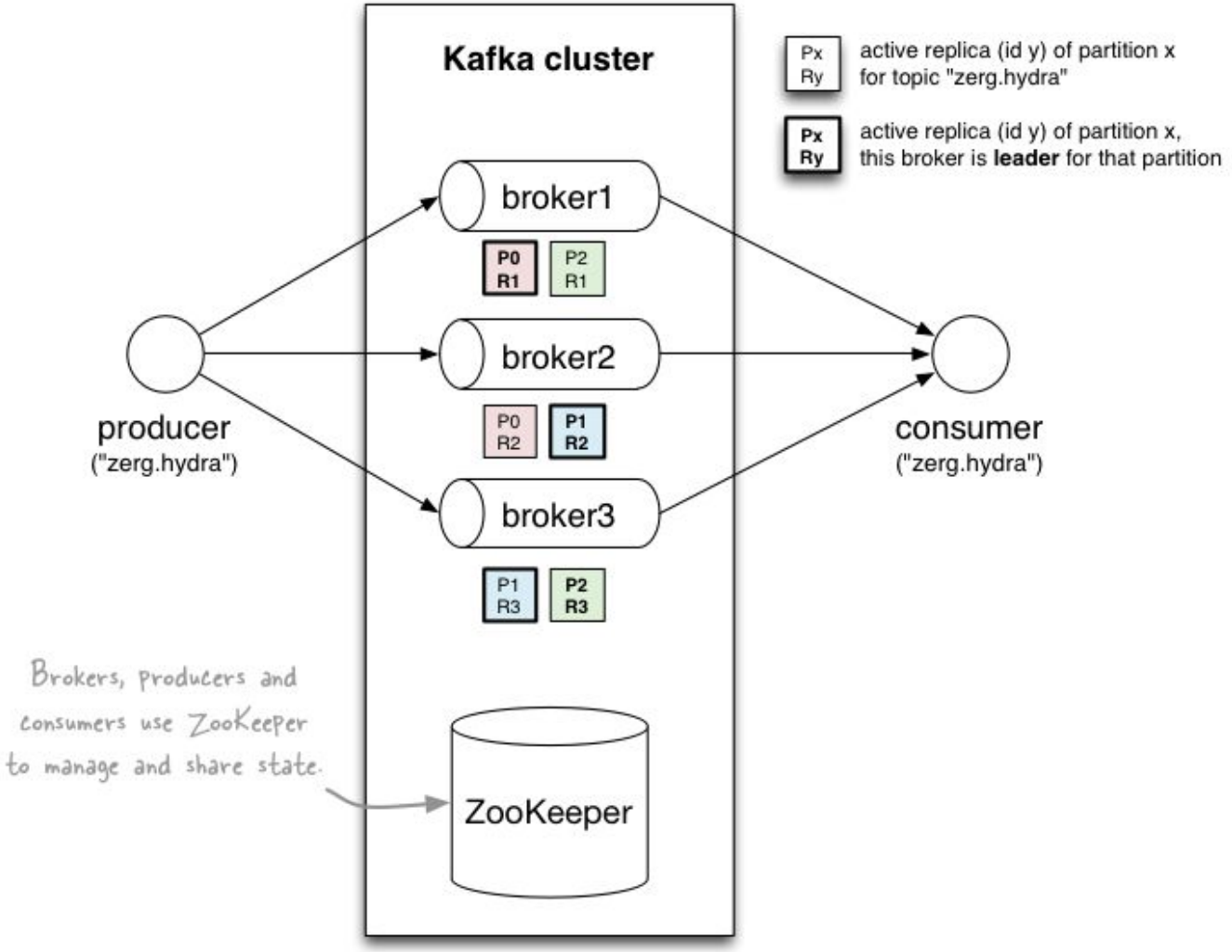
Very efficient to transfer data from page cache to a network socket

Linux: sendfile() system call

Combination of the two = fast Kafka!

Example (Operations): On a Kafka cluster where the consumers are mostly caught up you will see no read activity on the disks as they will be serving data entirely from cache.

First Look


Kafka cluster
Px Ry
active replica (id y) of partition x for topic "zerg.hydra"
Px Ry
active replica (id y) of partition x, this broker is leader for that partition
broker1
P0 R1
P2 R1
broker2
P0 R2
P1 R2
broker3
P1 R3
P2 R3
producer ("zerg.hydra")
consumer ("zerg.hydra")
Brokers, producers and consumers use ZooKeeper to manage and share state.
ZooKeeper

Apache Kafka is an open source "umbrella" project with the following modules:

- Kafka Core
  - Broker
  - Producer and Consumer APIs
- Kafka Streams
- Kafka Connect

From a messaging queue to a distributed streaming platform

Kafka Core

Kafka Core

- Records
- Brokers
- Topics
- Partitions
- Replicas and In-Sync Replicas
- Offsets
- Producers
- Consumers
- Retention

Records

- **Record** (aka **message** or **event**) is the unit of data in Kafka
- Array of bytes (in no particular format)
  - **Apache Avro** as data serialization framework
- Record has a **key** and a **value**
  - Both could be null
- Records are categorized **into topics**

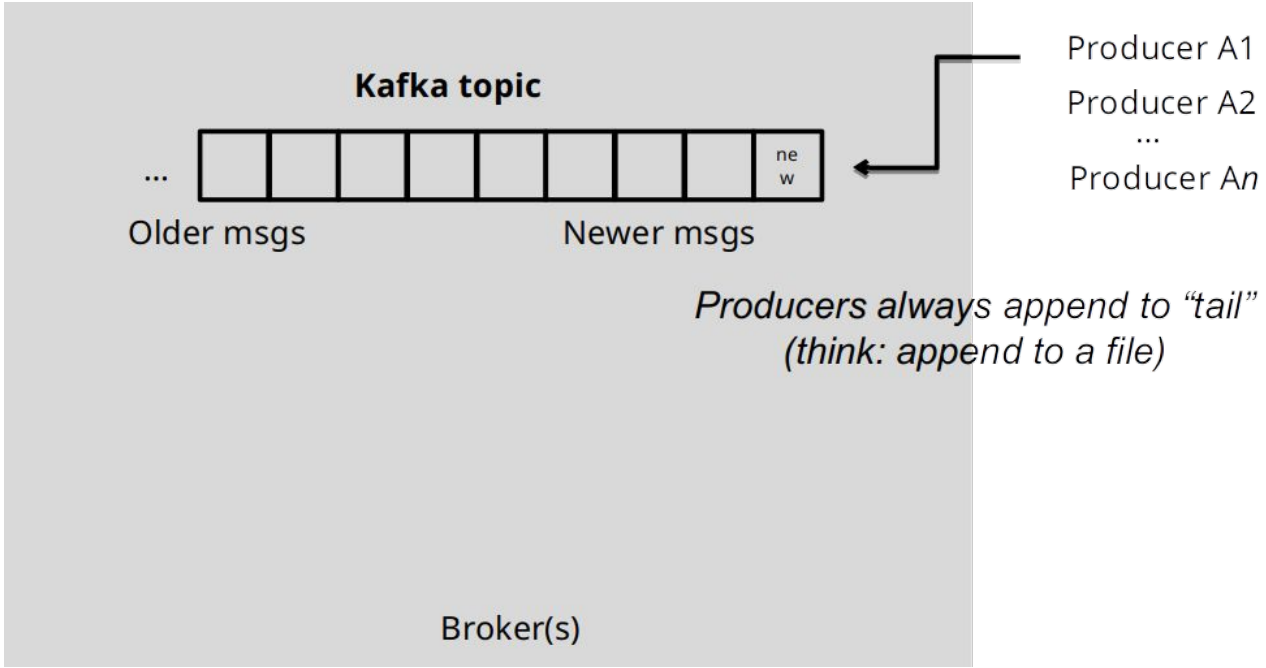
Brokers

- **Kafka Broker** is a Kafka server that manages records
  - Receives messages, assigns offsets, and commits messages to storage on disk
- **Kafka Cluster** consists of one or more brokers
  - Uses Zookeeper as the source of truth

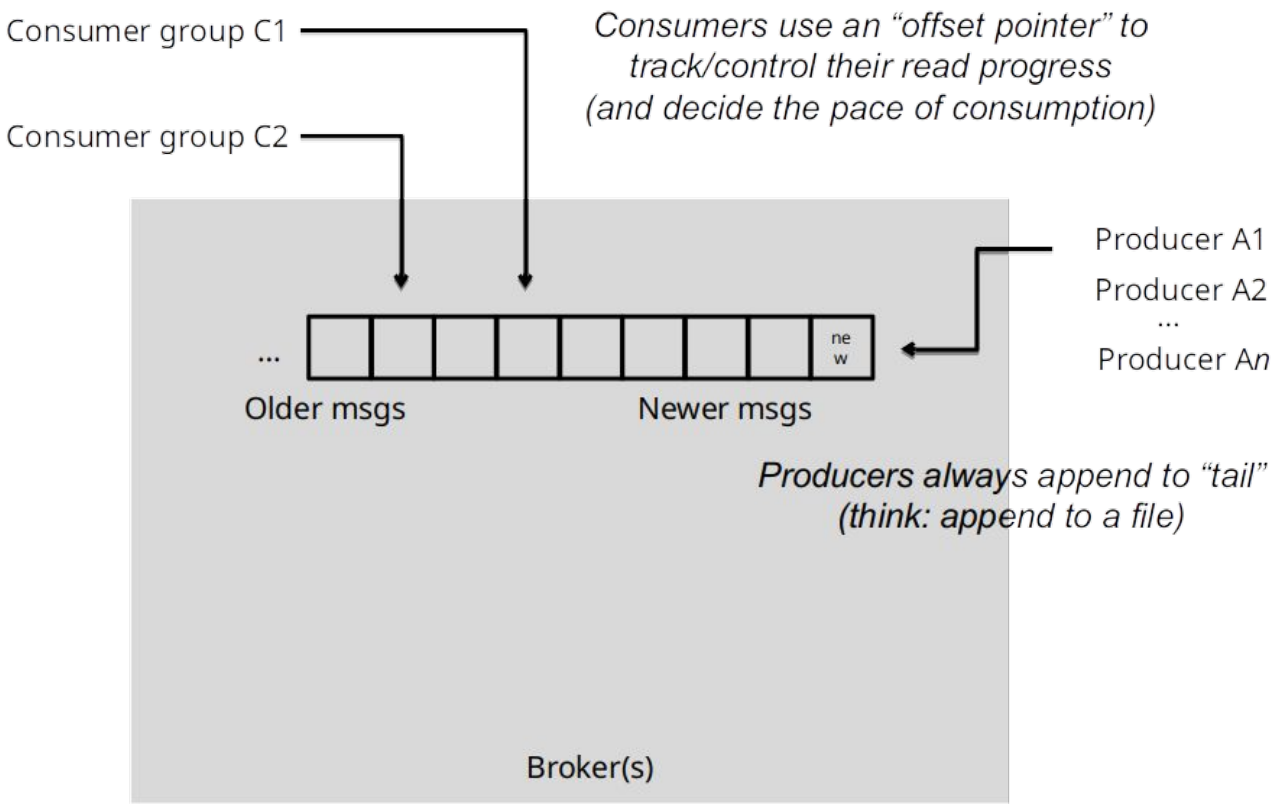
Topics

- Records are categorized into topics
  - Think a table or a directory
- Producers publish messages to topics while consumers consume them
- Topics are partitioned
  - Namespaces of one or many partitions
- kafka-topics shell script manages Kafka topics

Topics


Kafka topic
...
new
Older msgs
Newer msgs
Producer A1
Producer A2
...
Producer An
Producers always append to "tail"
(think: append to a file)
Broker(s)

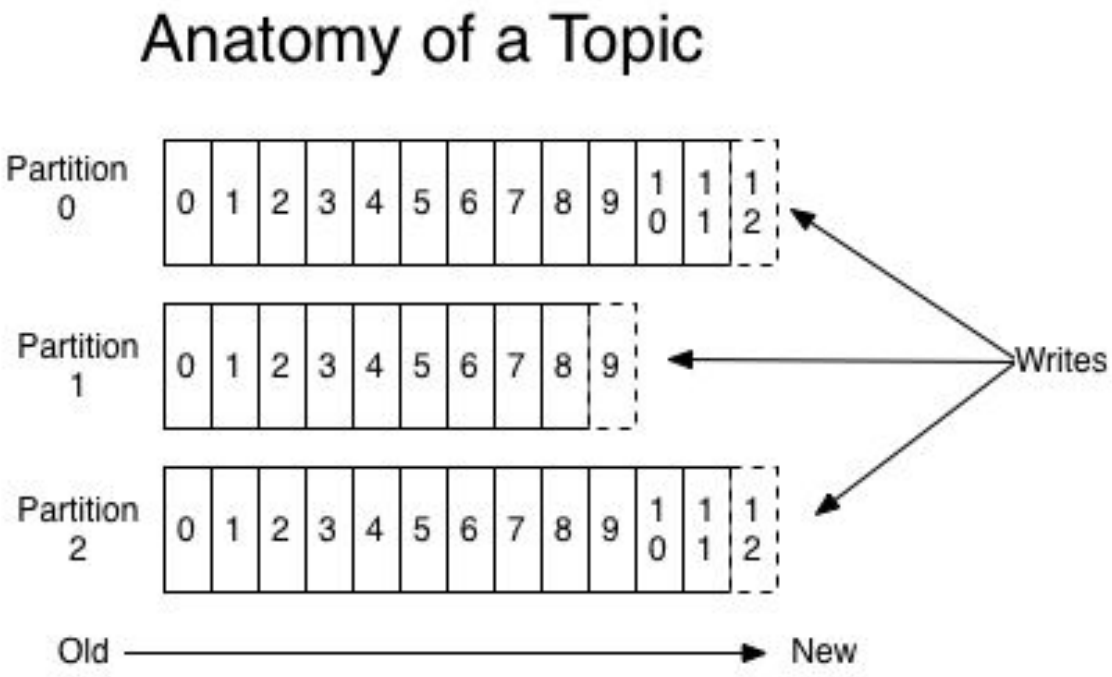
Topics


Consumer group C1
Consumer group C2
Consumers use an "offset pointer" to track/control their read progress (and decide the pace of consumption)
Producer A1
Producer A2
...
Producer An
...
ne w
Older msgs
Newer msgs
Producers always append to "tail" (think: append to a file)
Broker(s)

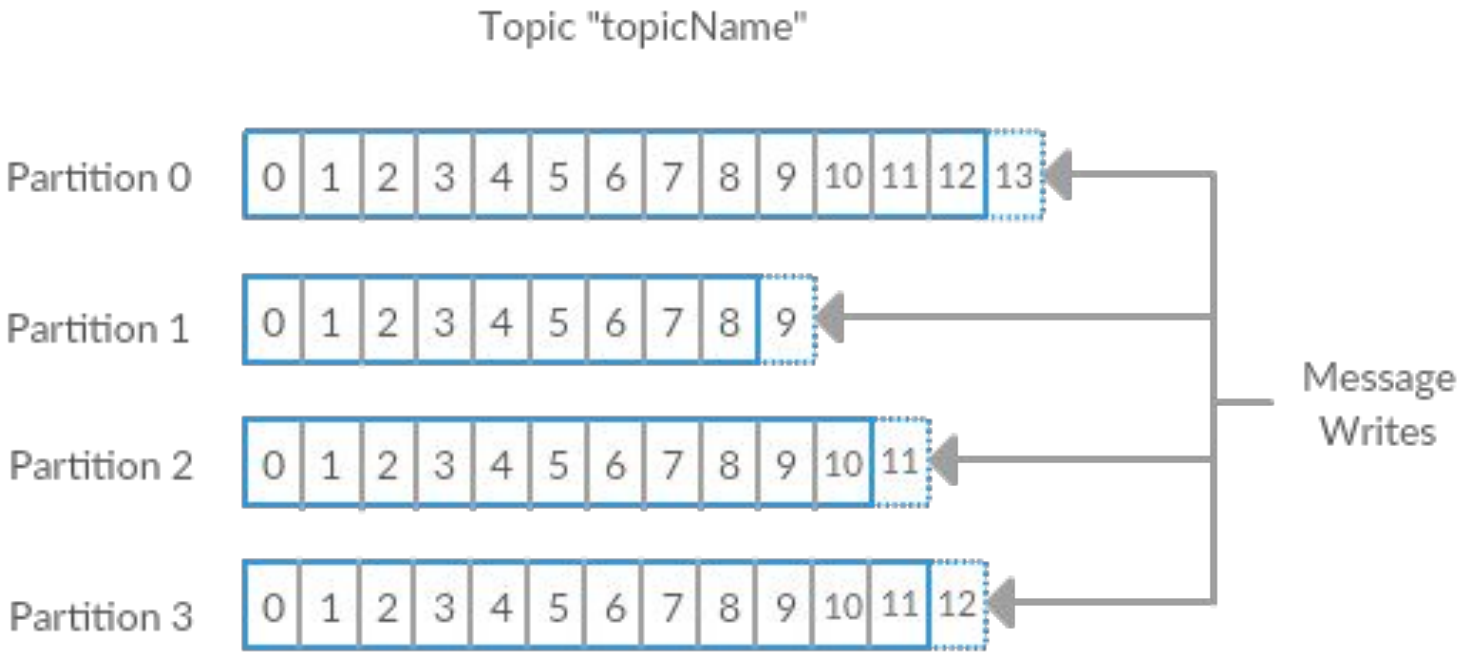
Partitions

- Topics are partitioned into one or more **partitions**
- Partitions hold zero, one or many records
- Ordered (by offsets) immutable sequence of records
- A partition is a single ordered log
- Stored durably on disk
- Partition: **ordered + immutable** sequence of messages
- Records are added to partitions in **append-only** fashion
- Partitions are replicated among brokers as **replicas**
- **In-sync replicas (ISRs)**

Kafka Topics and Partitions

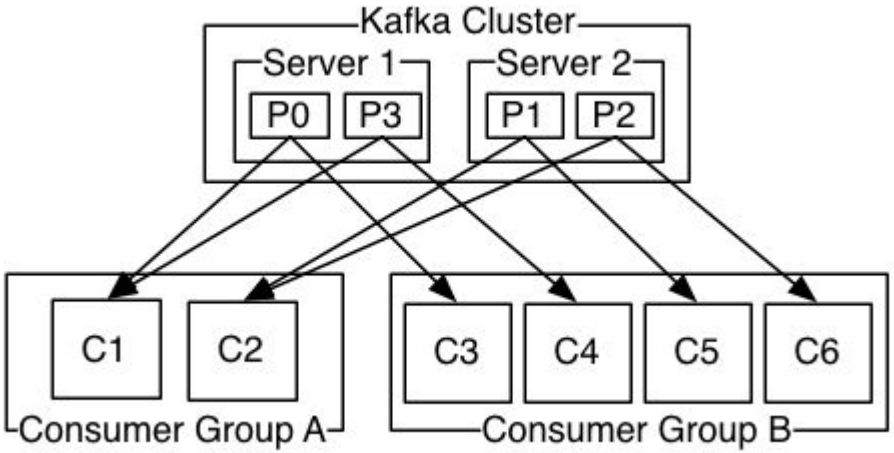


Kafka Topics and Partitions



Partitions

- partitions of a topic is **configurable**
- partitions determines **max consumer (group) parallelism**



- Consumer group A, with 2 consumers, reads from a 4-partition topic
- Consumer group B, with 4 consumers, reads from the same topic

Replicas and In-Sync Replicas

- **Replica** is a copy of a partition
- **Replication factor** is the number of replicas of a topic
  - There can be one or many replicas
  - Allows for automatic failover when a broker fails
  - They exist solely to prevent data loss.
- One replica is the **leader** while others are **followers**
  - Leader handles writes from producers, and the followers merely copy the leader's log
  - Replicas are never read from, never written to.

Replicas and In-Sync Replicas cont...

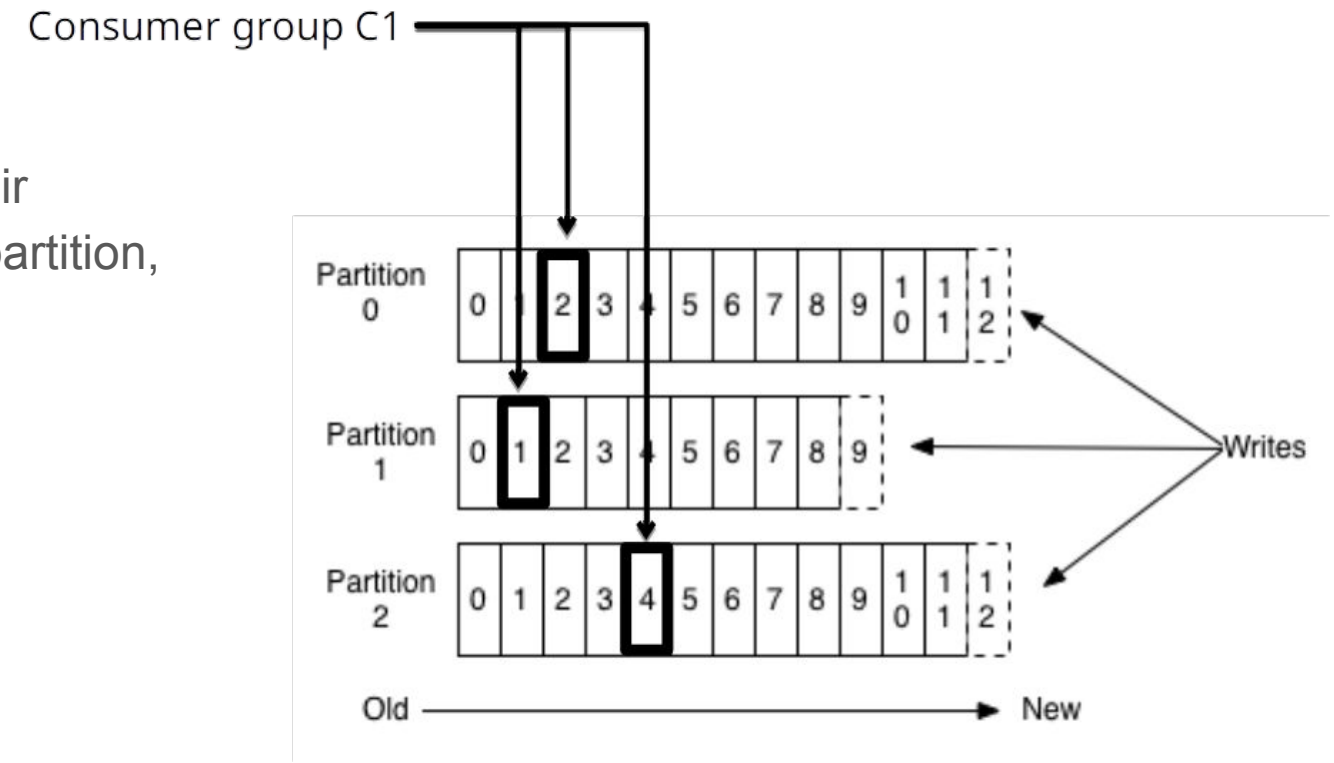
- **In-Sync Replica** is a replica that has enough records to be considered in **partition leader election**
- Kafka tolerates (numReplicas - 1) dead brokers before losing data
  - LinkedIn: numReplicas == 2 1 broker can di

Offsets

- **Offset** is a unique sequential numerical position of a record (in a partition of a topic)
  - A message in a partition has a unique offset
- Offsets start from 0
- Offsets are unique per partition only
  - Not across partitions

Offsets

Consumers track their pointers via (offset, partition, topic) tuples



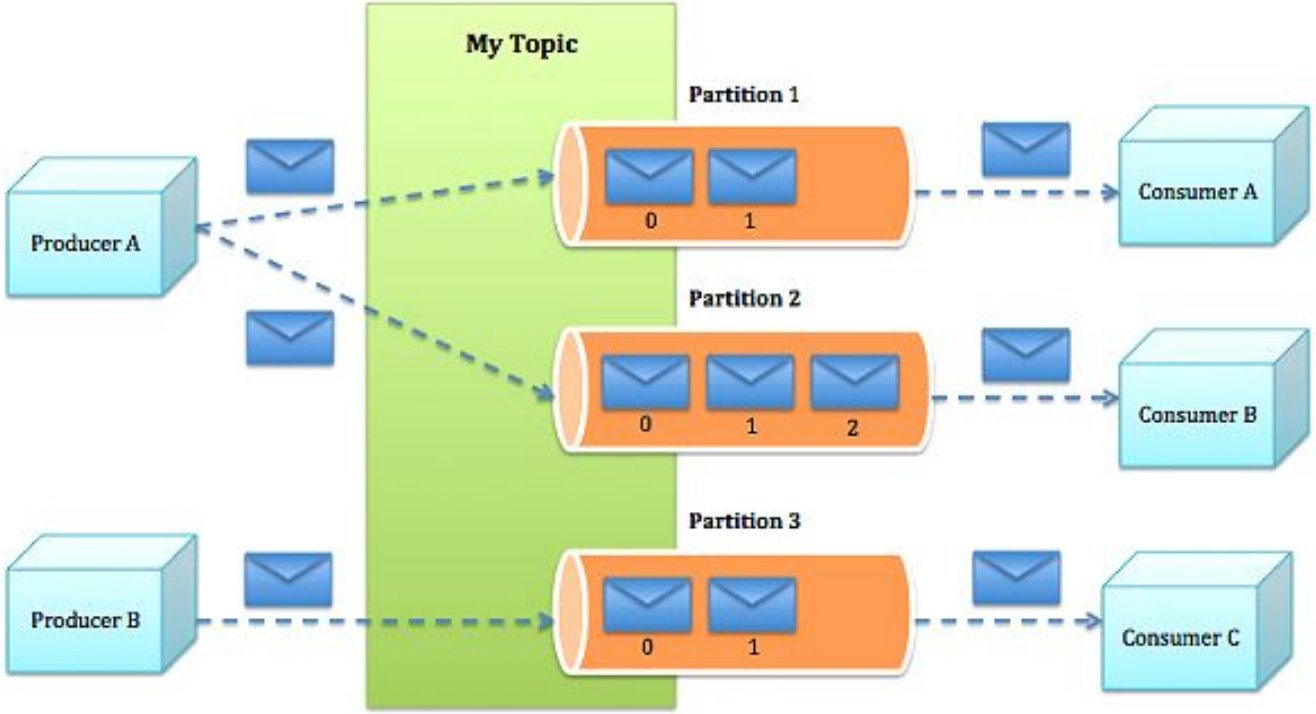
Producers

- Kafka clients that publish records to a Kafka cluster
- Send messages to topics
    - Can optionally specify partitions

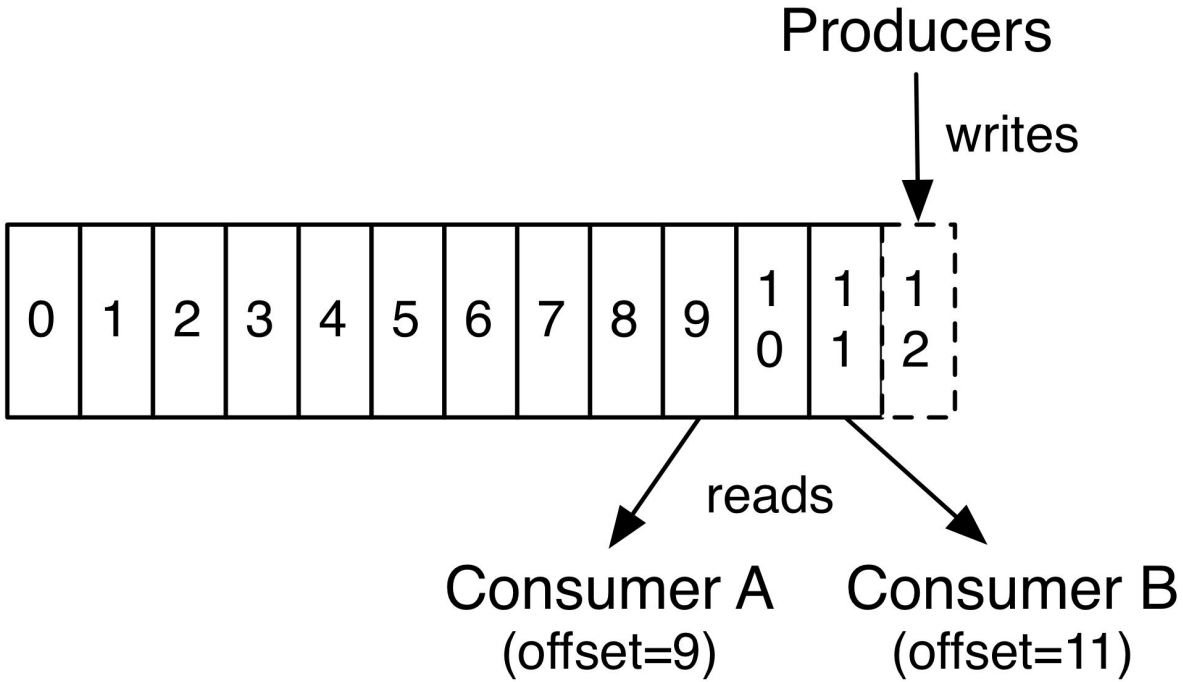
Consumers

- **Kafka clients** that consumes records from a Kafka cluster
- **Subscribes** to one or many topics
- Read messages in the order they were produced
  - Per partition only
- Handles failures of Kafka brokers and adapts as topic partitions migrate within the cluster
- Maintains TCP connections to the necessary brokers to fetch data
- Allows groups of consumers to load balance consumption using consumer groups

Kafka Producers and Consumers



Kafka Producers and Consumers



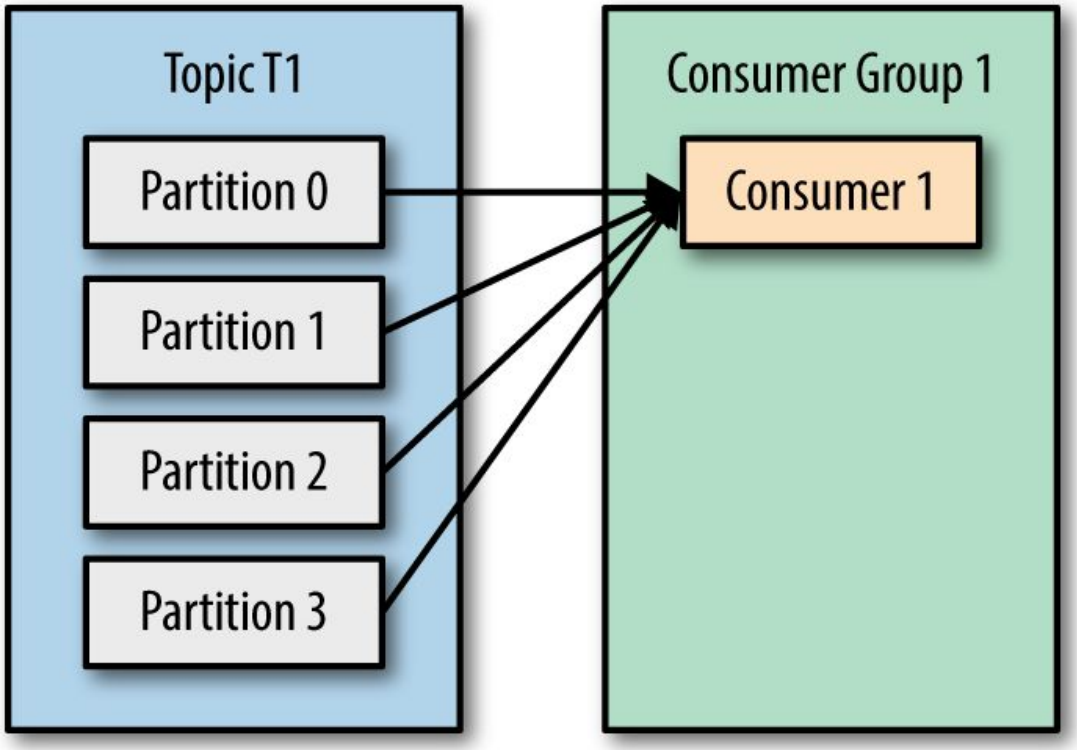
Consumer Groups

- **Consumer Group** is a group of Kafka consumers that has divided the work of consuming and processing records among themselves
- Conceptually, a consumer group is a single logical subscriber that happens to be made up of multiple processes (brokers)
- Kafka consumers with the same **group.id**
- Each partition (of subscribed topics) is assigned to **exactly one consumer** in a consumer group
- Provides scalability and fault tolerance for processing

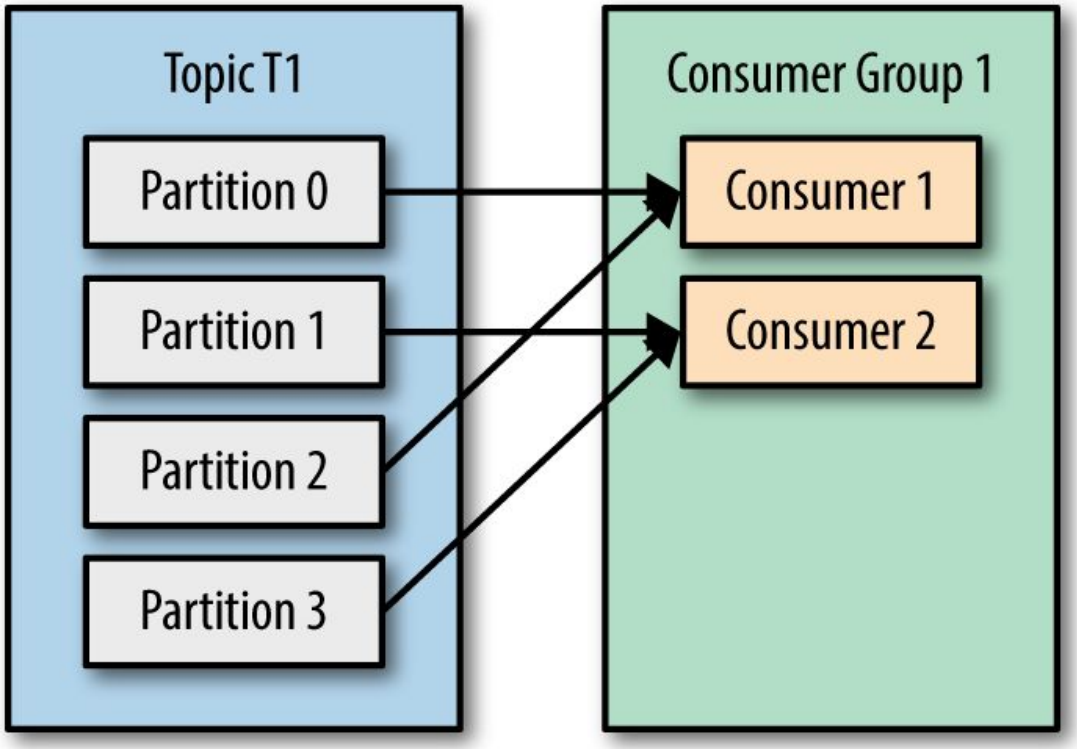
Consumer Groups cont...

- Membership in a consumer group is maintained dynamically
- If a member fails or a new consumer joins the group, the partitions will be reassigned to all group members (at **partition rebalancing** discussed next)
- Members can either be running on the same machine or can be distributed over many machines
- Any number of consumer groups for a given topic is acceptable (without duplicating data)
- Semantics similar to a queue in traditional messaging systems
- Record delivery is balanced among the member of a group

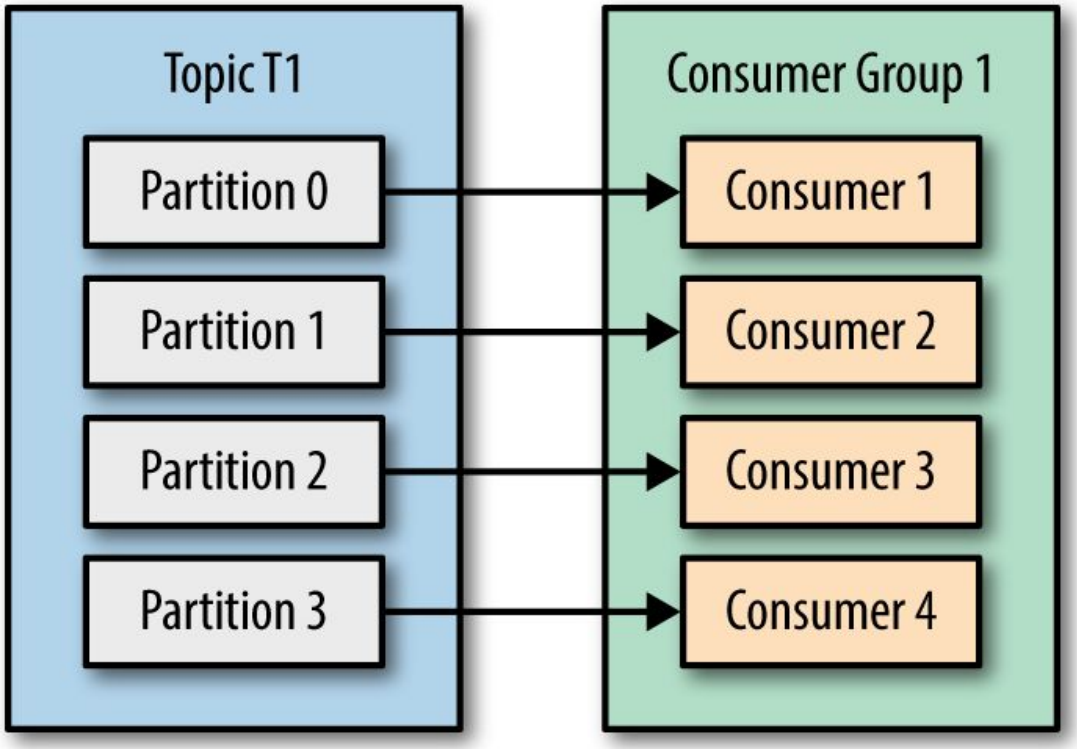
Partitions and Consumer Groups



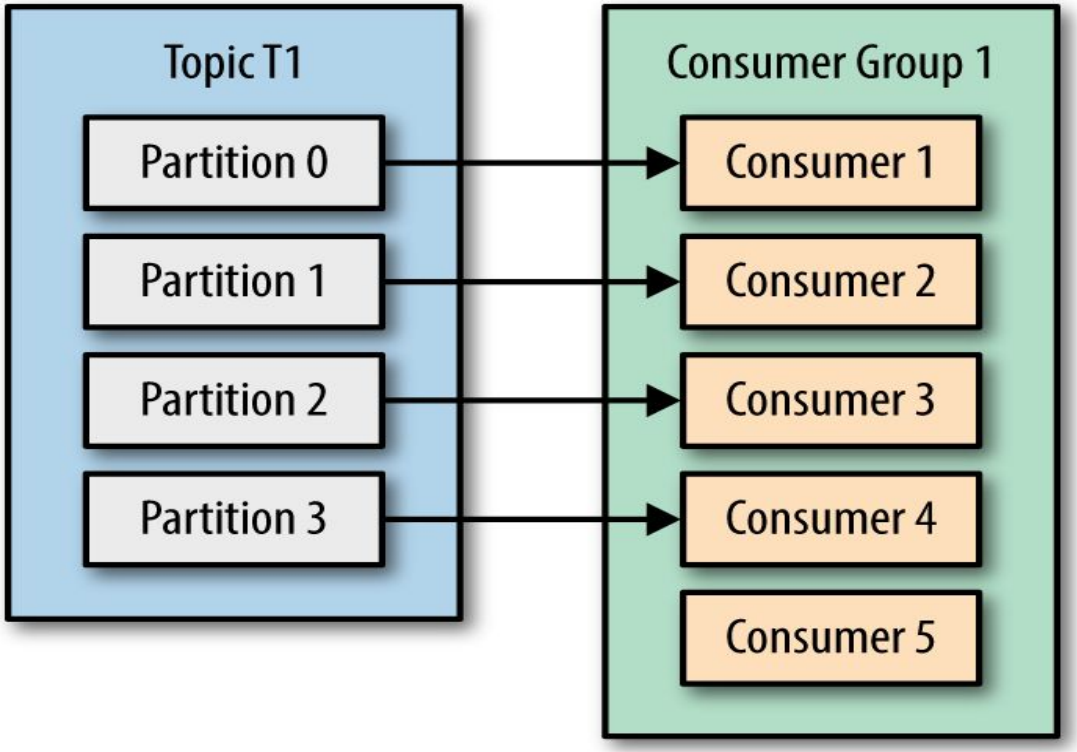
Partitions and Consumer Groups



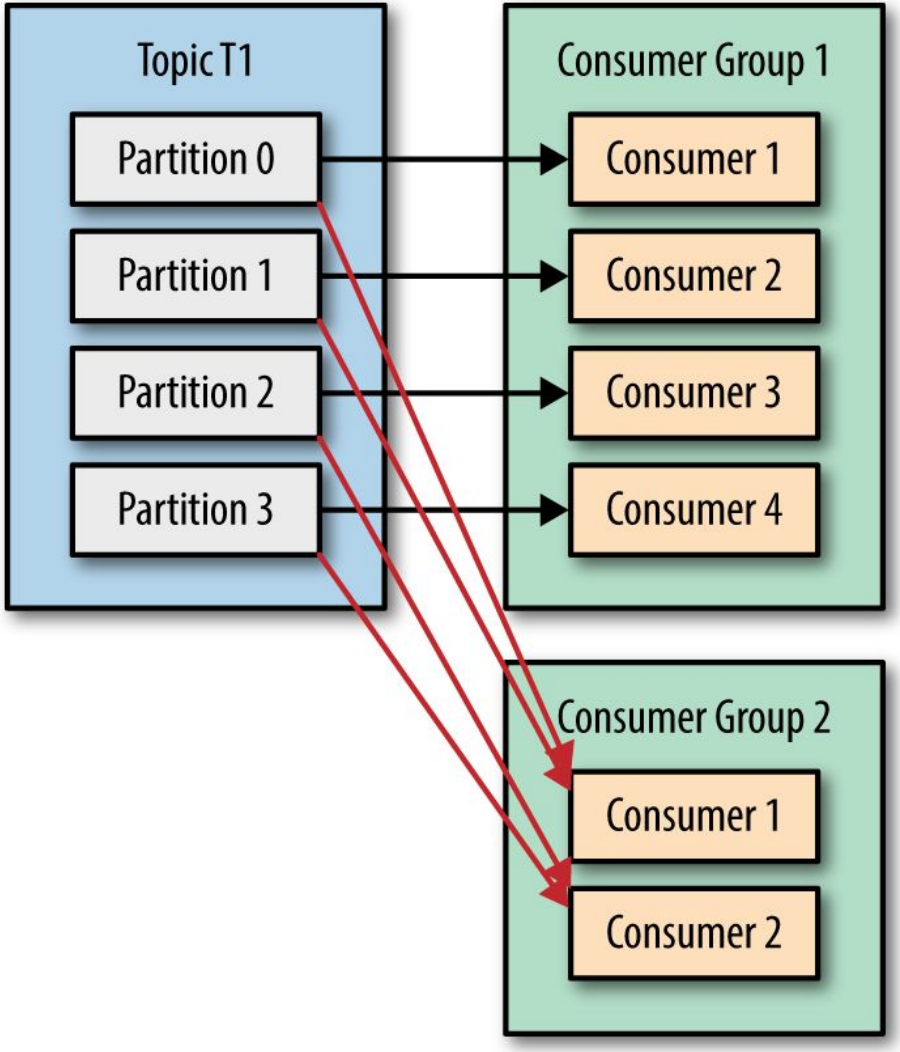
Partitions and Consumer Groups



Partitions and Consumer Groups



Partitions and Consumer Groups



Partition Rebalancing

- **Partition Rebalancing** (aka rebalancing a group) is a process of balancing the partitions (of subscribed topics) between members of a consumer group
- E.g. a topic with 4 partitions and a consumer group with 2 processes will give each consumer would consume from 2 partitions
- Also used when new partitions are added to a subscribed topic or when a new topic matching a subscribed regex is created

Partition Rebalancing cont...

- Changes in group membership or topic subscription will automatically be detected through periodic metadata refreshes
- When group reassignment happens automatically, consumers can be notified through a **ConsumerRebalanceListener** so they can finish necessary application-level logic (e.g. state cleanup, manual offset commits)

Retentions

- **Retention of messages** in topics is how long messages are stored in topics
  - Durable message retention
  - For some period of time, e.g. 7 days
  - Until a topic reaches a certain size in bytes, e.g. 1 gigabyte
- Once these limits are reached, messages are expired and deleted
- Can be selected on a per-topic basis

Features

Features of Kafka

- Thousands of Producers
- Thousand of Consumers
- Client Independence
- High Throughput
- Message Persistence
- Disk-based Retention
- Scalability
- High Performance

Kafka Connect

- Kafka Connect is a framework for a scalable and reliable data streaming between Apache Kafka and other systems
- A framework for Kafka connectors
- Distributed and standalone (single process) modes
- REST interface for connector deployment and management

ksqlDB

- ksqlDB is an open source streaming SQL engine for stream processing on Kafka
- Interactive SQL interface
- "KSQL: Query Your Streams Without Writing Code"
- No need to write code in a programming language like Java or Python
- SQL layer atop Kafka Streams
    - Executing SQL on tables and streams