ak系列 – ak01

(Apache Kafka - 基礎)

Introduction to Kafka and the ecosystem

Course Objectives

- What is Kafka
- Learn about the Kafka ecosystem (high level)
- Learn Apache Kafka core API
  - Topics, Partitions
  - Brokers, Replication, Zookeeper
  - Producers, Consumers, Consumer Groups
- Get started with a broker setup on own machine using Docker-Compose

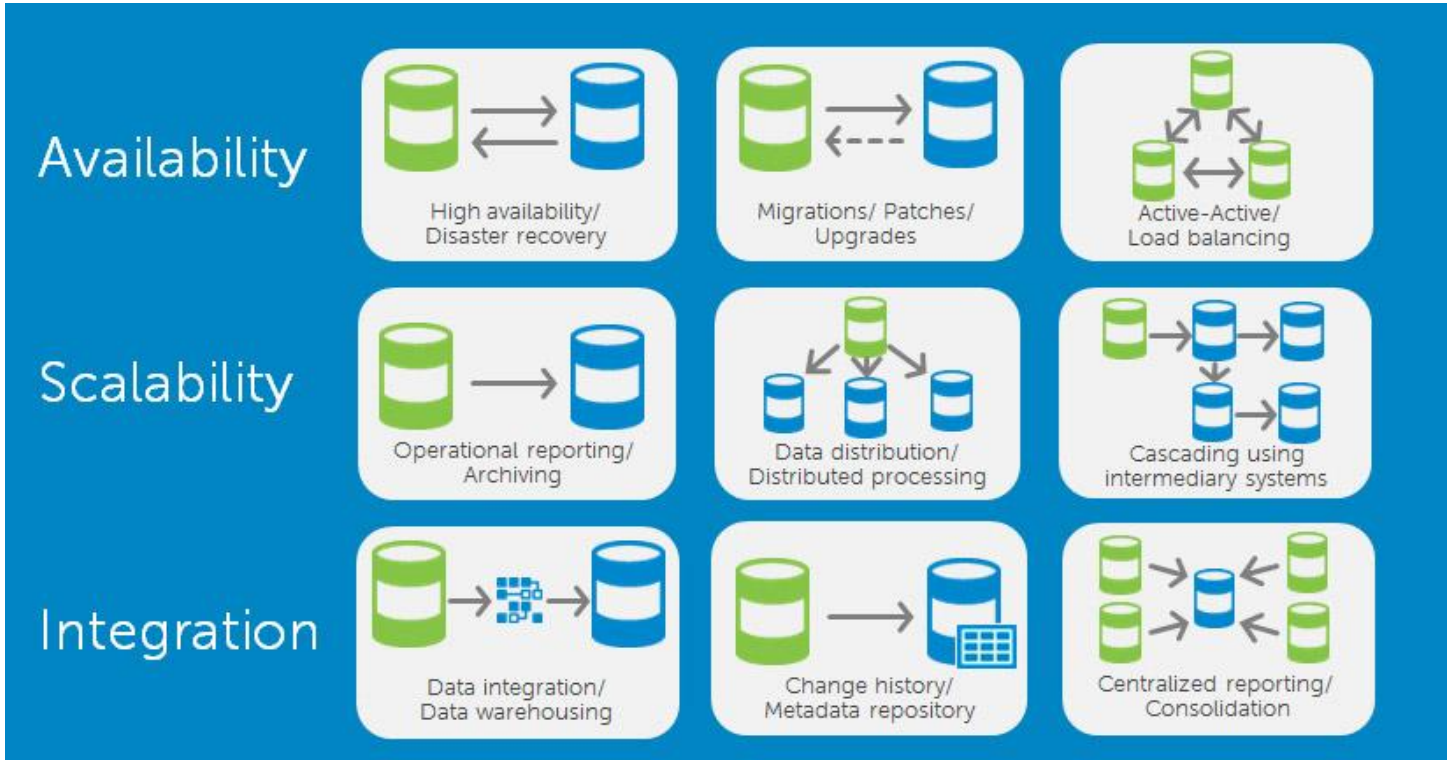
What is Apache Kafka

Kafka was developed at LinkedIn back in 2010

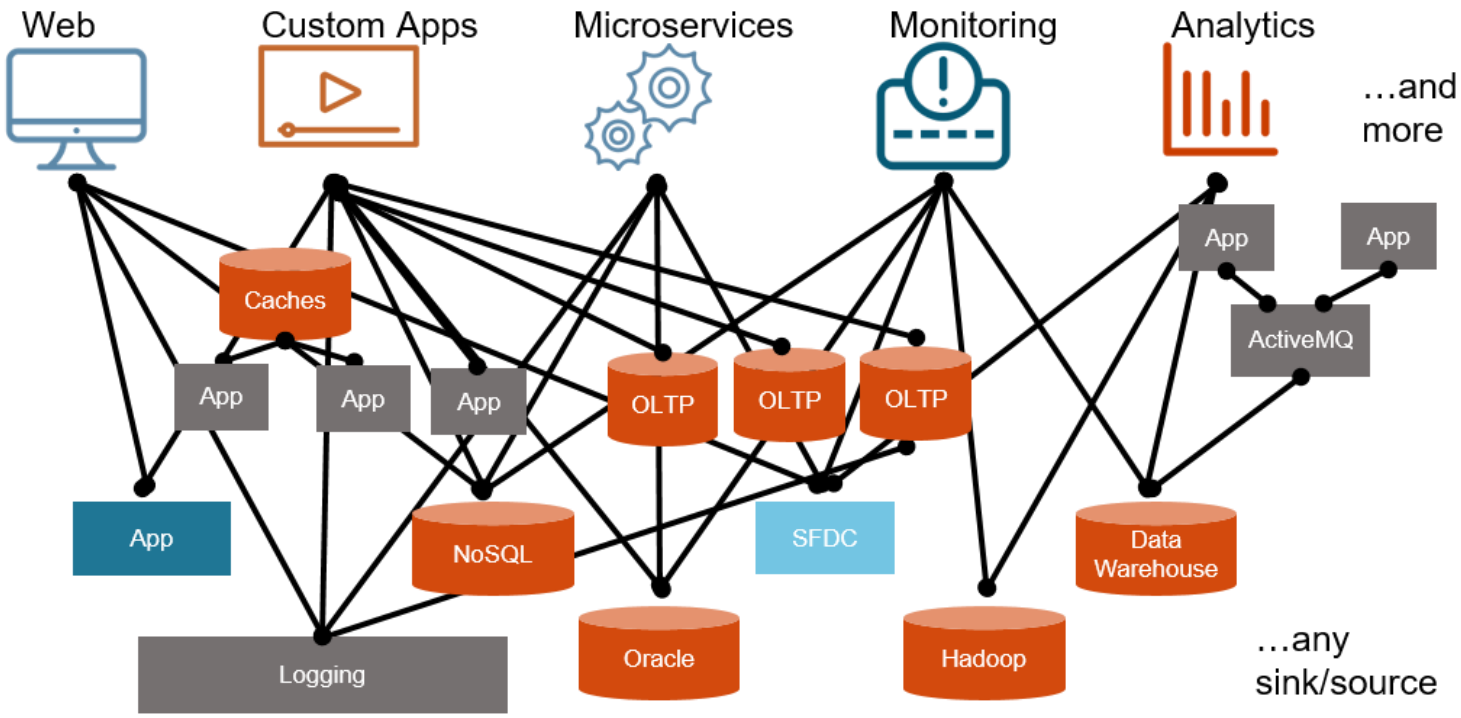
- Apache Kafka was originally developed by LinkedIn, and open sourced in early 2011.
- Written in Scala and Java.
- The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds

Why Apache Kafka

Data is very important

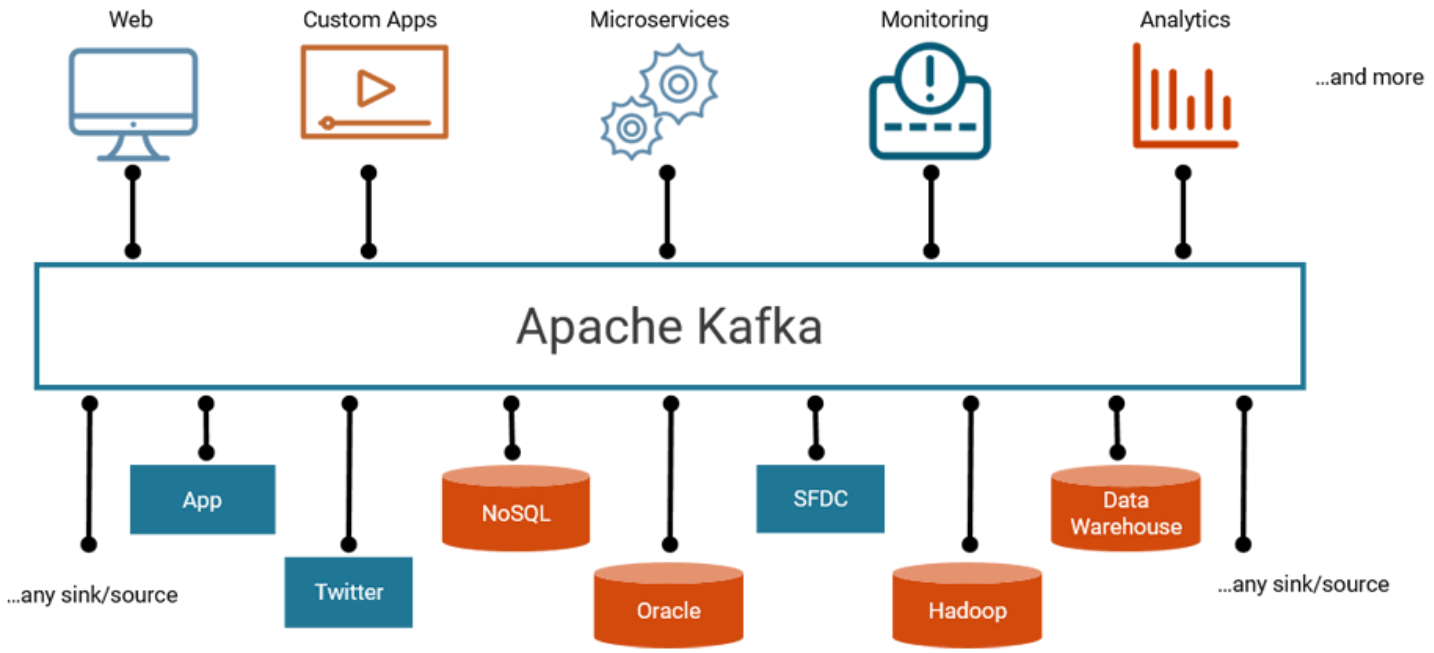


Why Apache Kafka
Integration become very complicate



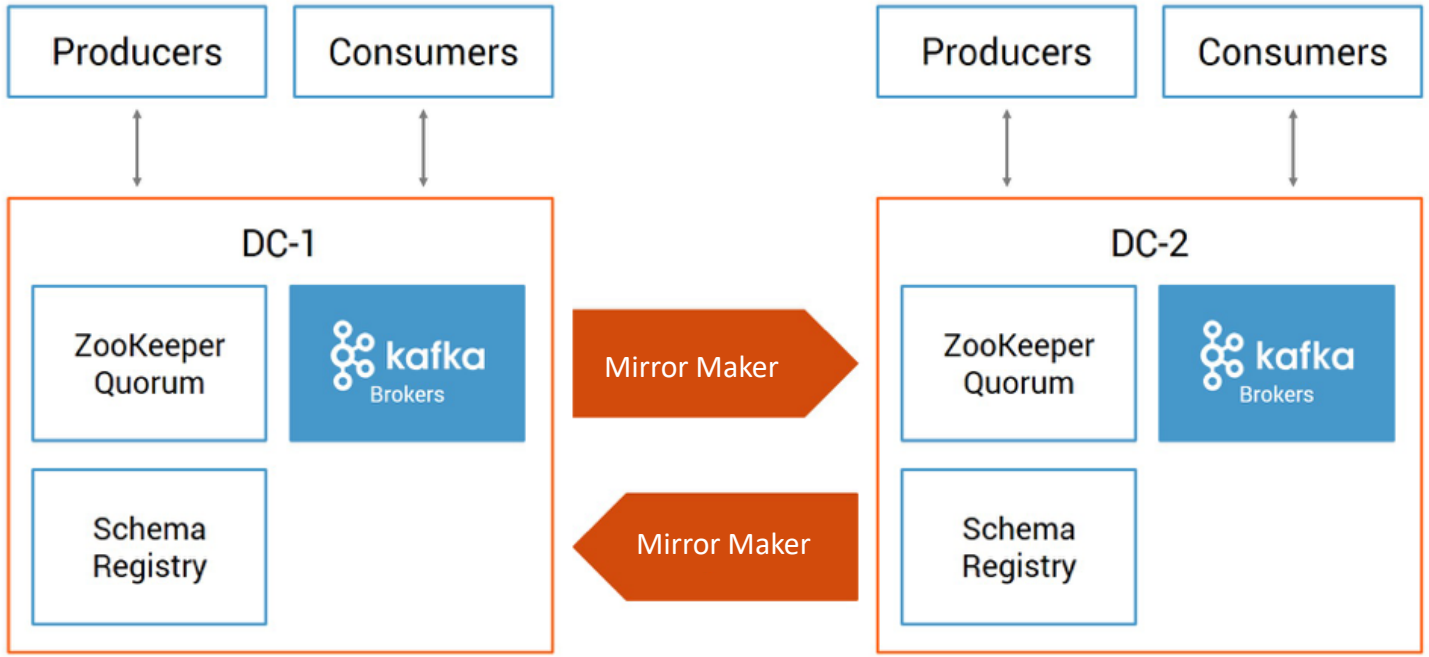
Ref# 4

Why Apache Kafka:
Decoupling of data streams



Ref# 4

Why Apache Kafka:
Decoupling of data location



Ref# **4**

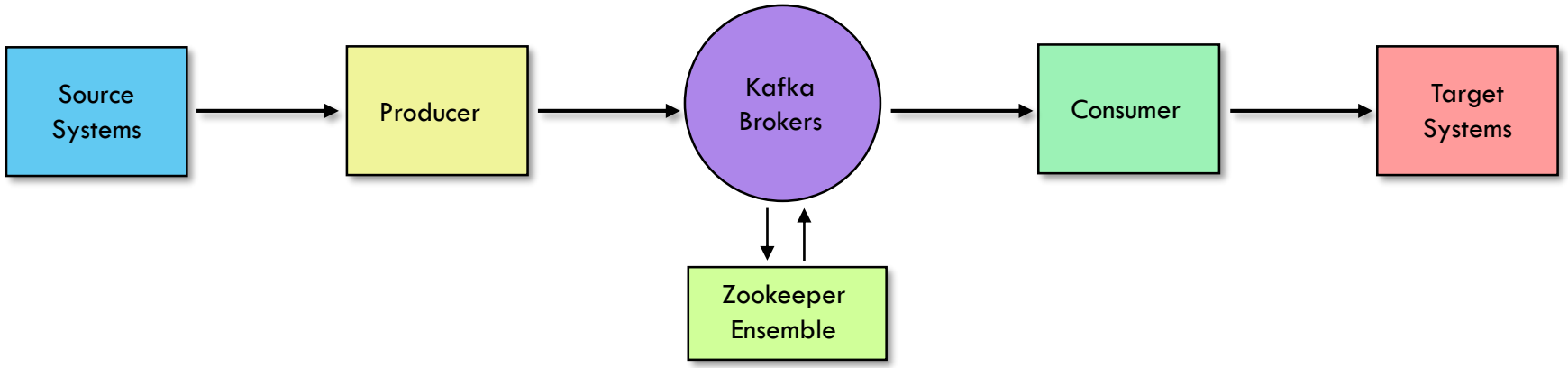
Why Apache Kafka

- Distributed, resilient architecture, fault tolerant
- Horizontal scalability
- High performance (latency of less than 10 ms) – real time
- Used by the 2000+ firms:
  - LinkedIn
  - Netflix
  - AirBnb
  - Yahoo
  - Walmart

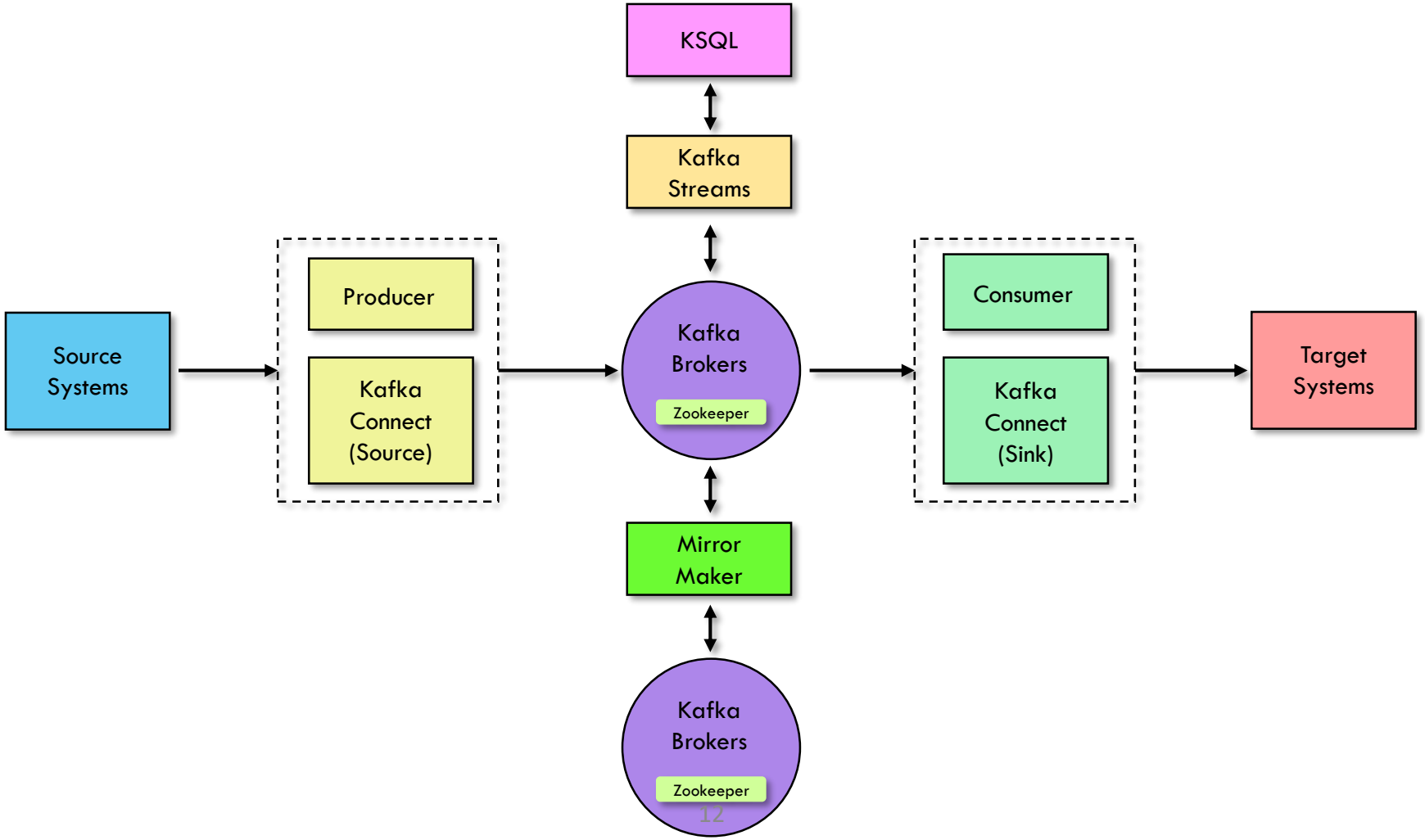
Apache Kafka: Use cases

- Messaging System
- Activity Tracking
- Gather metrics from many different locations (IoT / Industry 4.0)
- Application Logs gathering (Industry 4.0)
- Stream processing (with the Kafka Streams API or Spark for example)
- De-coupling of system dependencies
- Integration with Spark, Flink, Storm, Hadoop, and many other Big Data technologies

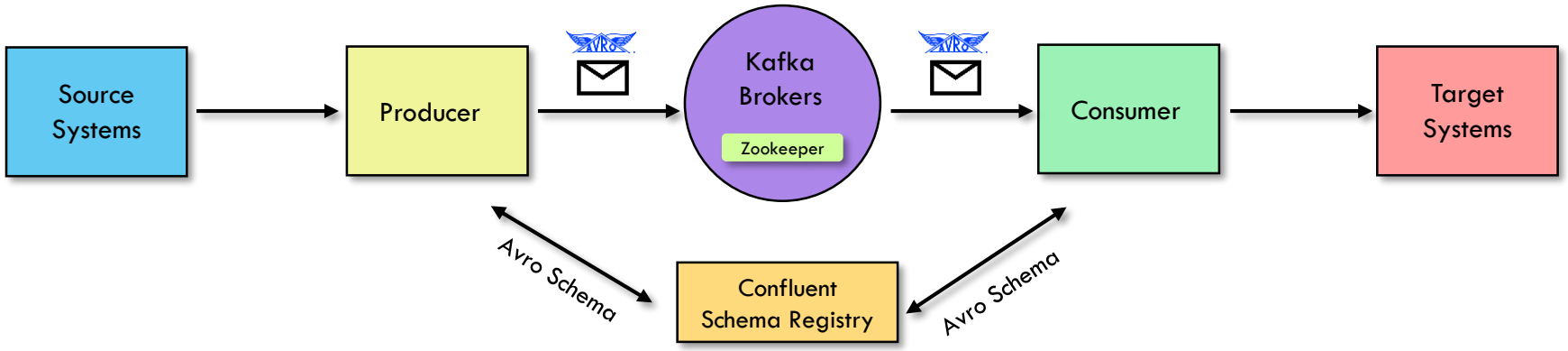
Kafka Ecosystem:
Kafka Core



Kafka Ecosystem:
Kafka Extended API



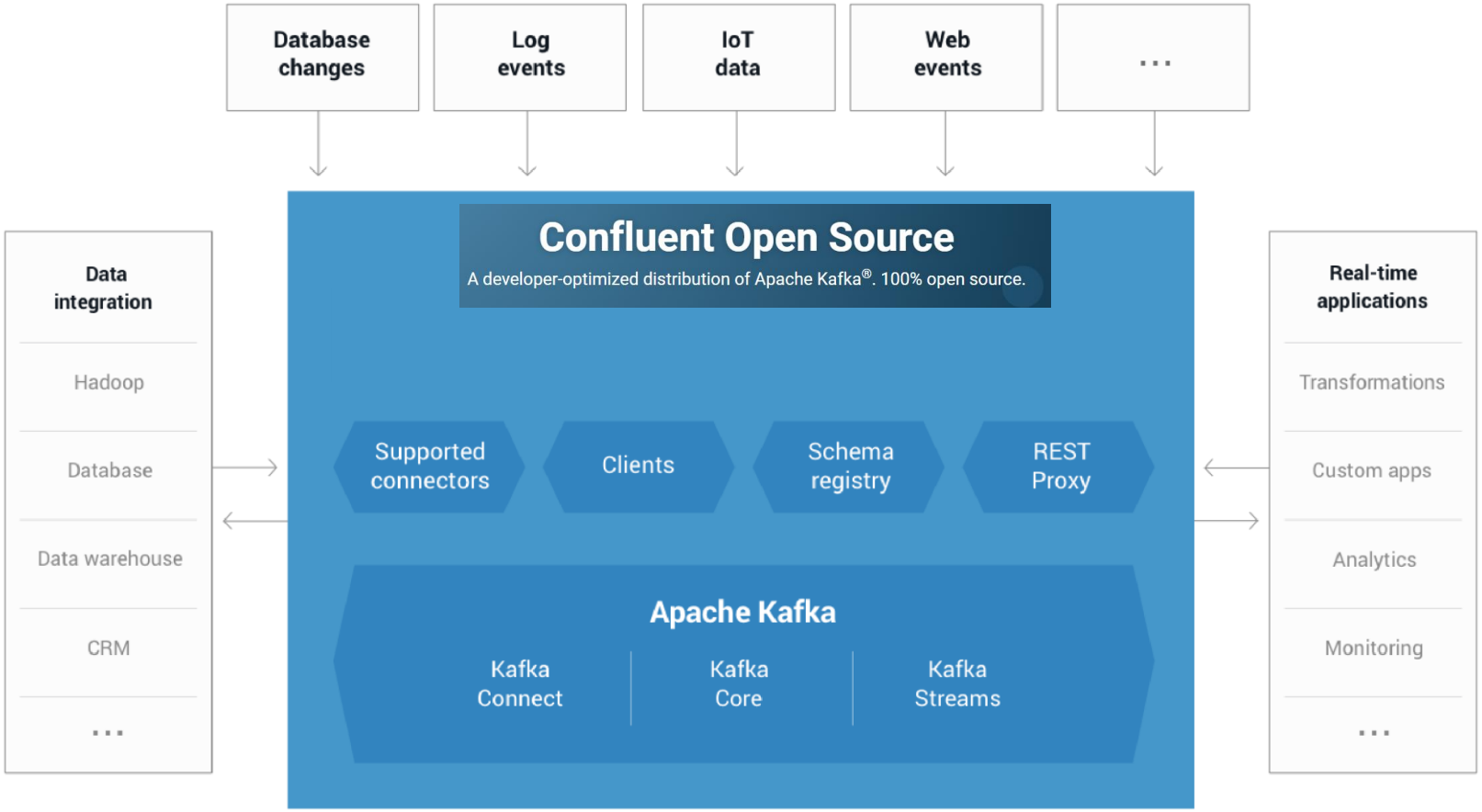
Kafka Ecosystem:
Confluent Components - Schema Registry



Kafka Ecosystem： Administration and Monitoring Tools

- Topics UI (Landoop): View the topics content
- Schema UI (Landoop): Explore the Schema registry
- Connect UI (Landoop): Create and monitor Connect tasks
- Kafka Manager (Yahoo): Overall Kafka Cluster Manager
- Burrow (LinkedIn): Kafka Consumer Lag Checking
- Exhibitor (Netflix): Zookeeper Configuration, Monitoring, Backup
- Kafka Tools (LinkedIn): Broker and topics admin tasks simplified
- Kafkat (Airbnb): More broker and topics admin tasks simplified
- JMX Dump: Dump JMX metrics from Brokers

Kafka in the Enterprise Architecture
Confluent Open Source Version

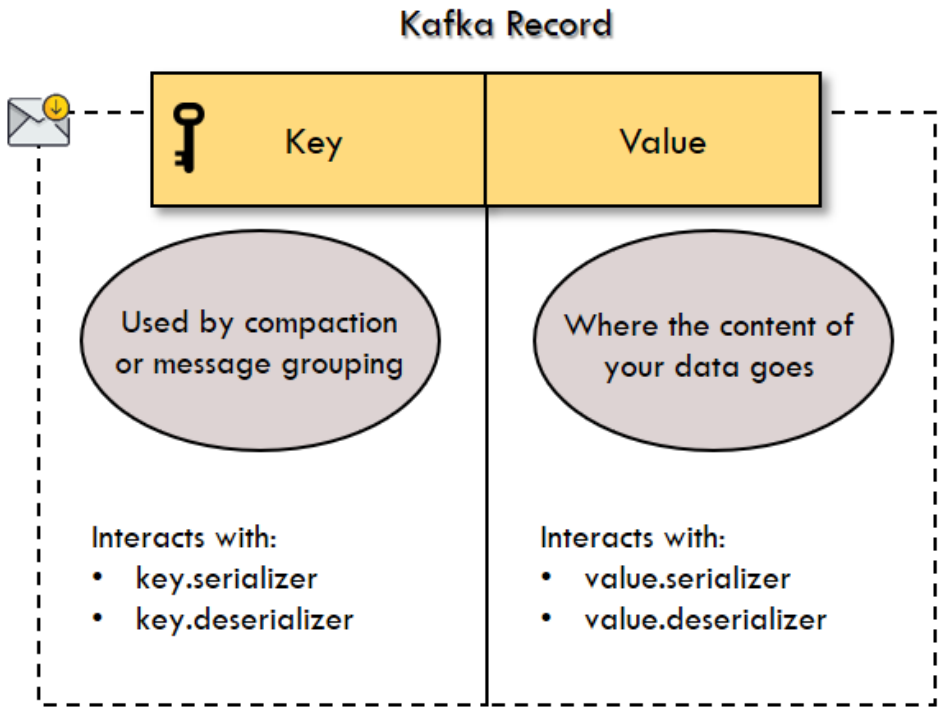


Ref#4

Apache Kafka: Core Concepts

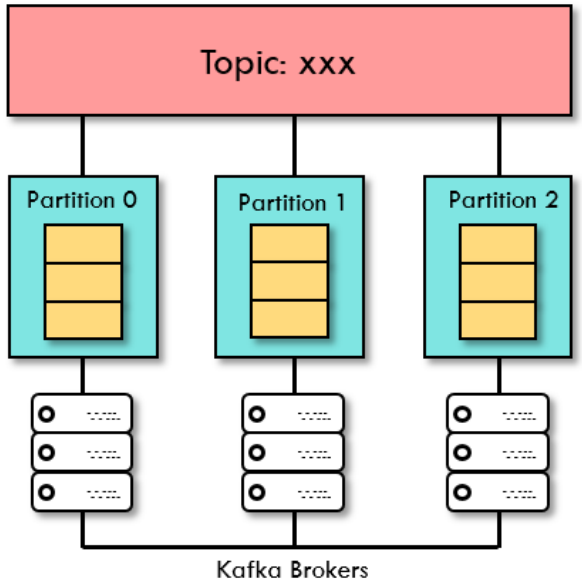
Kafka Record

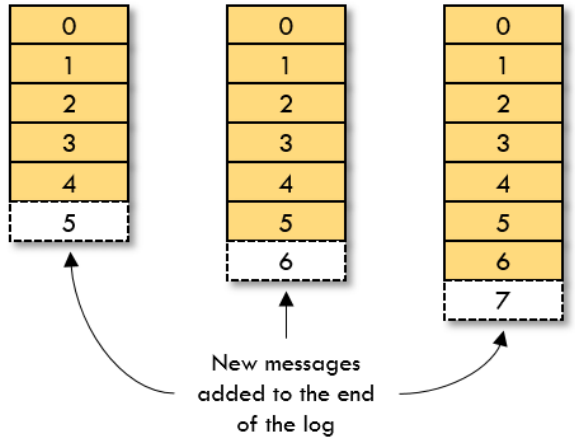
- Every message publish to Kafka called "**Record**"
- **Record** contain two parts:
  - **Key**
    - Used by compaction or for message grouping
  - **Value**
    - The content of data goes



Topics and partitions

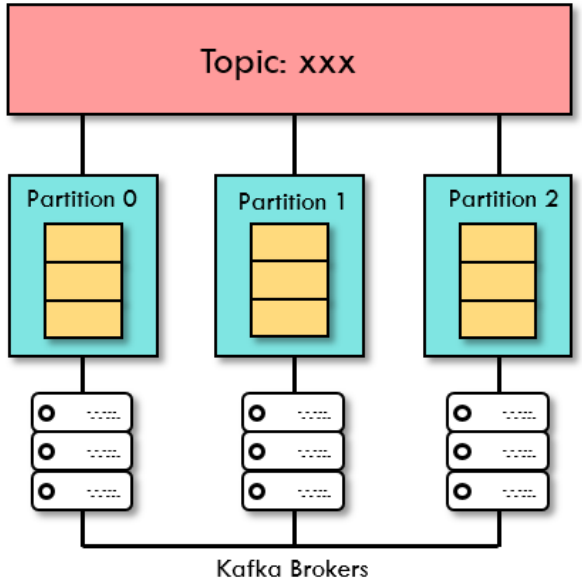
- **Topics**: a particular stream of data
  - Similar to a table in a database (without all the constraints)
  - You can have as many topics as you want
  - A topic is identified by its **name**
- Topics are split in **partitions**
  - Each partition is ordered
  - Each message within a partition gets an incremental id, called **offset**

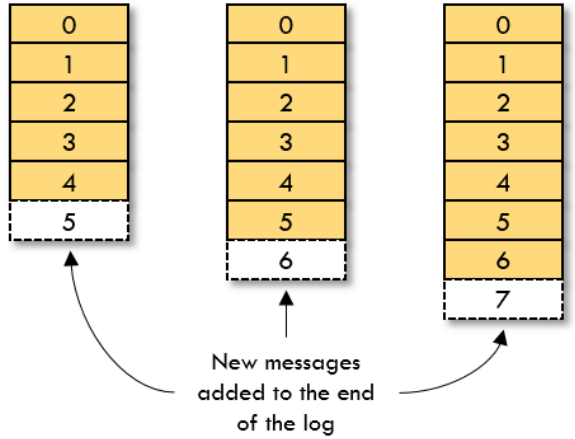




Topics and partitions

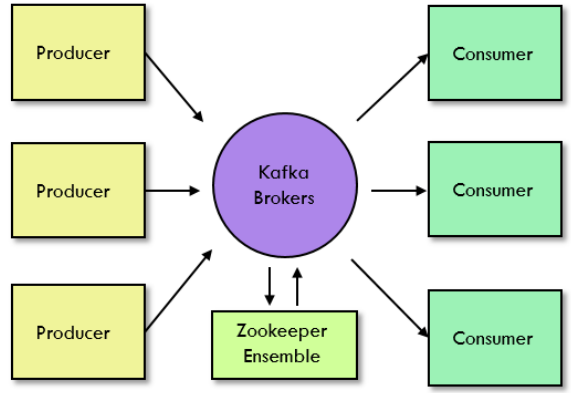
- **Offset** only have a meaning for a specific partition.
  - E.g. offset 3 in partition doesn't represent the same data as offset 3 in another partition
- Order is guaranteed only within a partition (not across partitions)
- Data is kept only for a limited time (default is **one** weeks)
- Once the data is written to a partition, it can't be changed (immutability)
- Data is assigned randomly to a partition unless a **key** is provided (more on this later)
- You can have as many partitions per topics as you want)

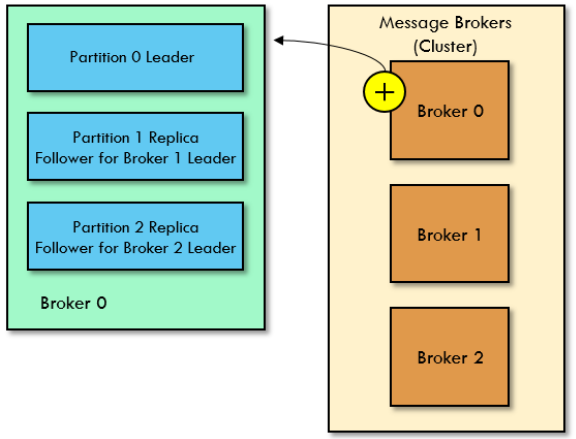




Brokers

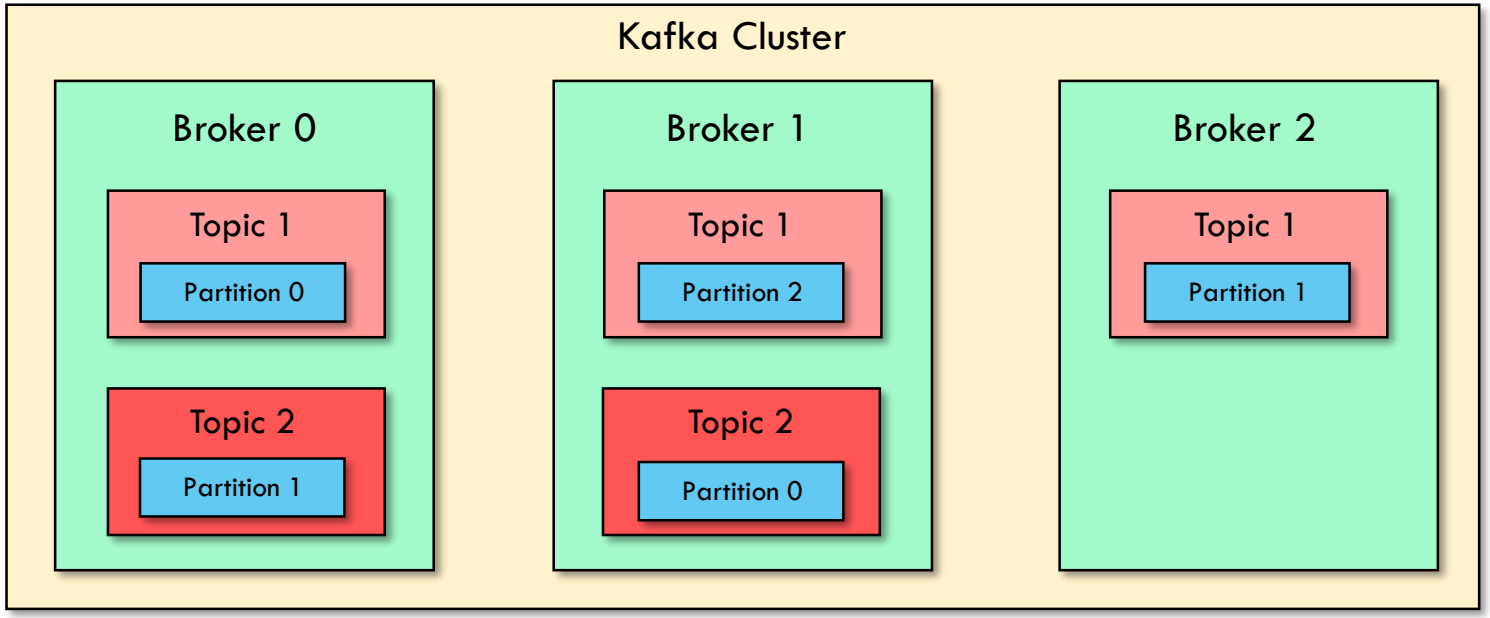
- A Kafka cluster is composed of multiple **brokers** (servers)
- Each broker is identified with its ID (integer)
- Each broker contains certain topic partitions
- After connecting to any broker (called a bootstrap broker), you will be connected to the entire cluster
- A good number to get started is **3** brokers, but some big clusters have over 100 brokers





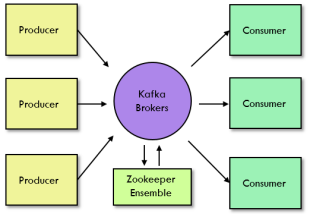
Brokers and topics

- Example of 2 topics (3 partitions and 2 partitions)

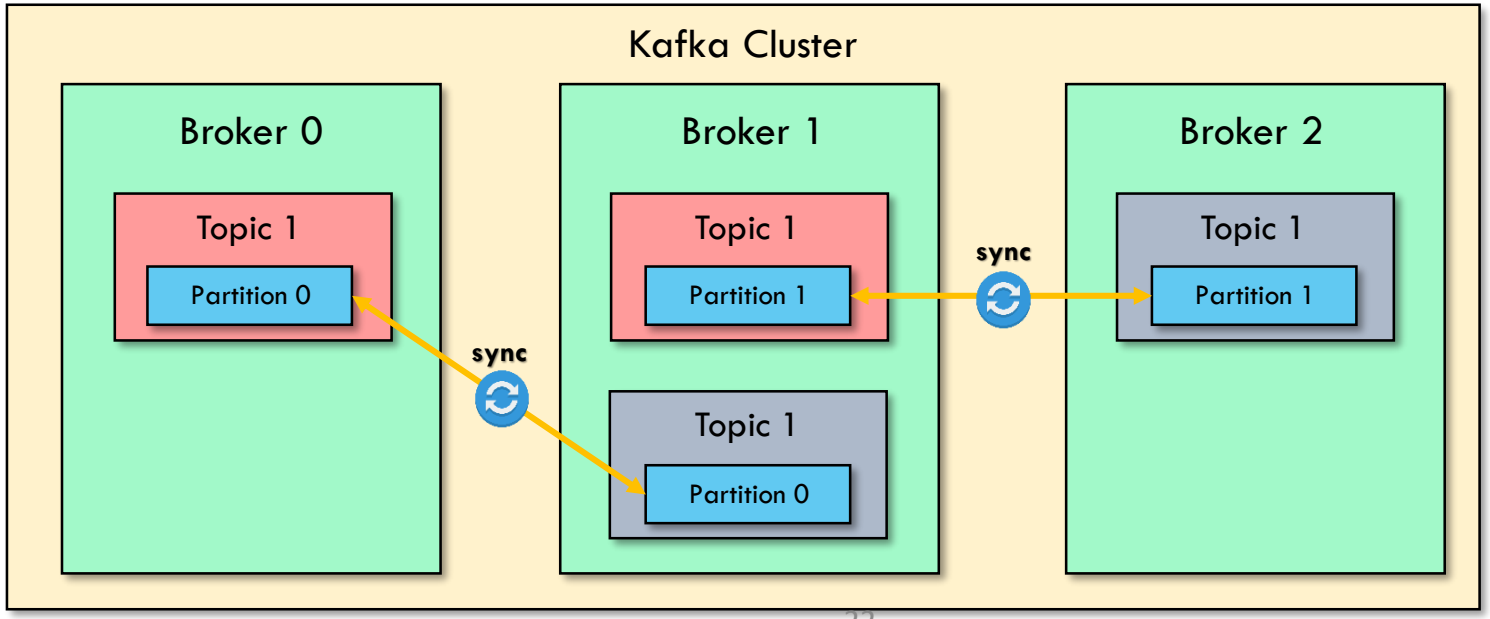


- Data is distributed and Broker 2 doesn’t have any Topic 2 data

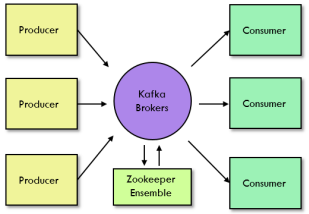
Topic replication factor



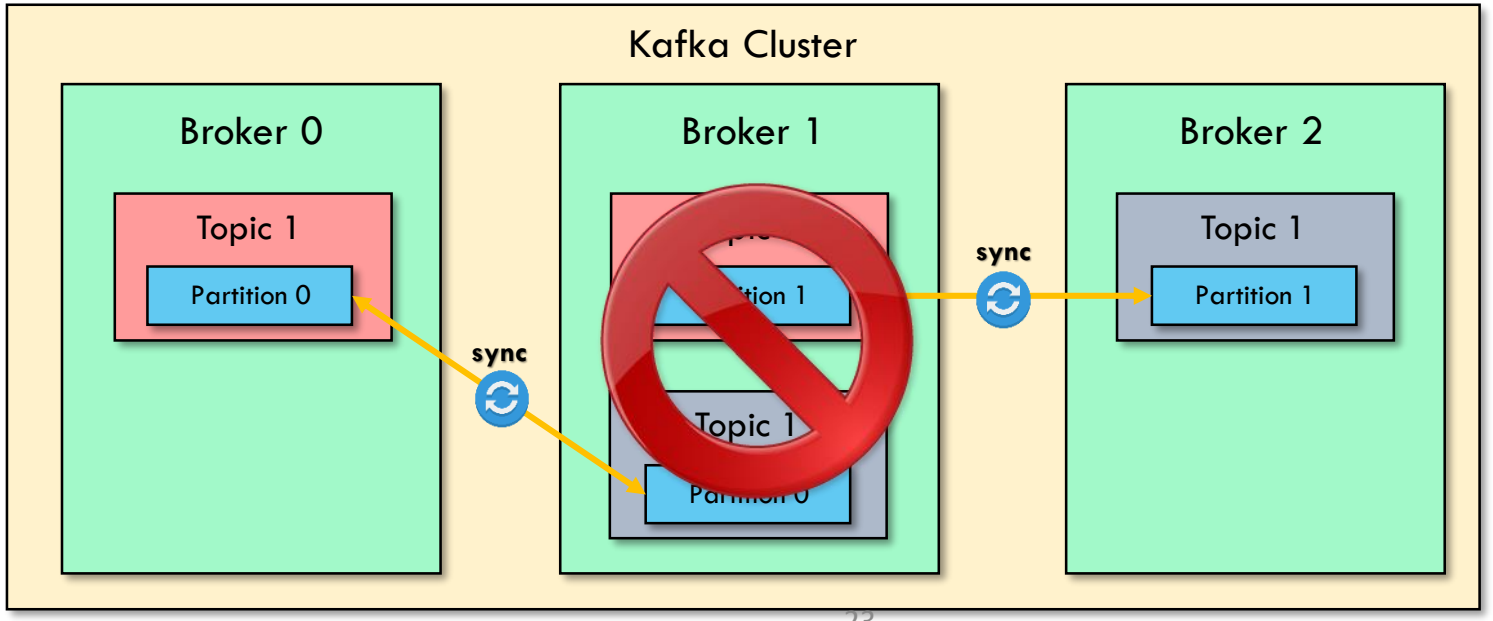
- Topics should have a replication factor > 1 (usually between 2 and 3)
- This way if a broker is down, another broker can serve the data
- Example: 1 **topic** with 2 **partitions** and **replication factor** of 2



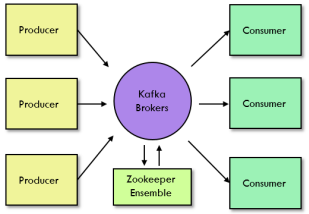
Topic replication factor



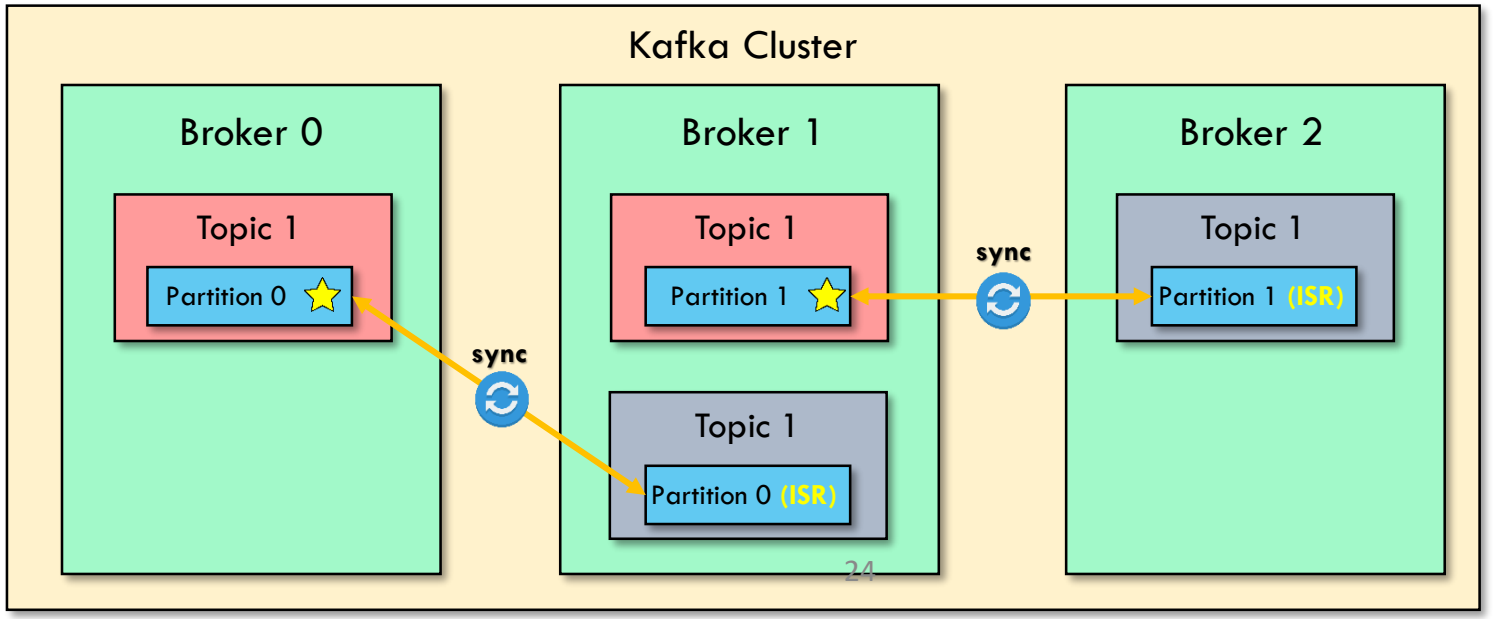
- Example: we lost Broker 1
- Result: Broker 0 and 2 can still serve the data



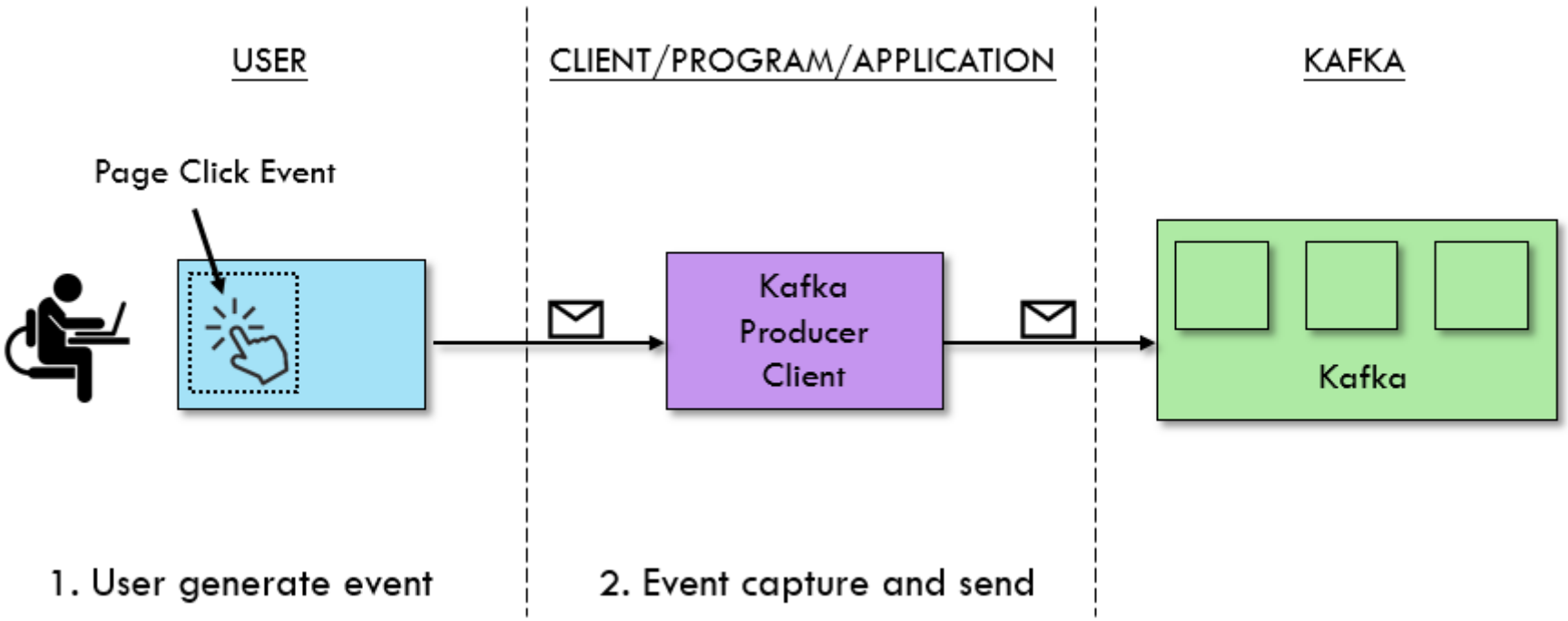
Concept of Leader for a partition



- At any time only 1 broker can be a leader for a given partition
- Only that leader can retrieve and serve data for a partition
- The other brokers will synchronize the data
- There each partition has: one **leader**, and multiple **ISR** (in-sync replica)

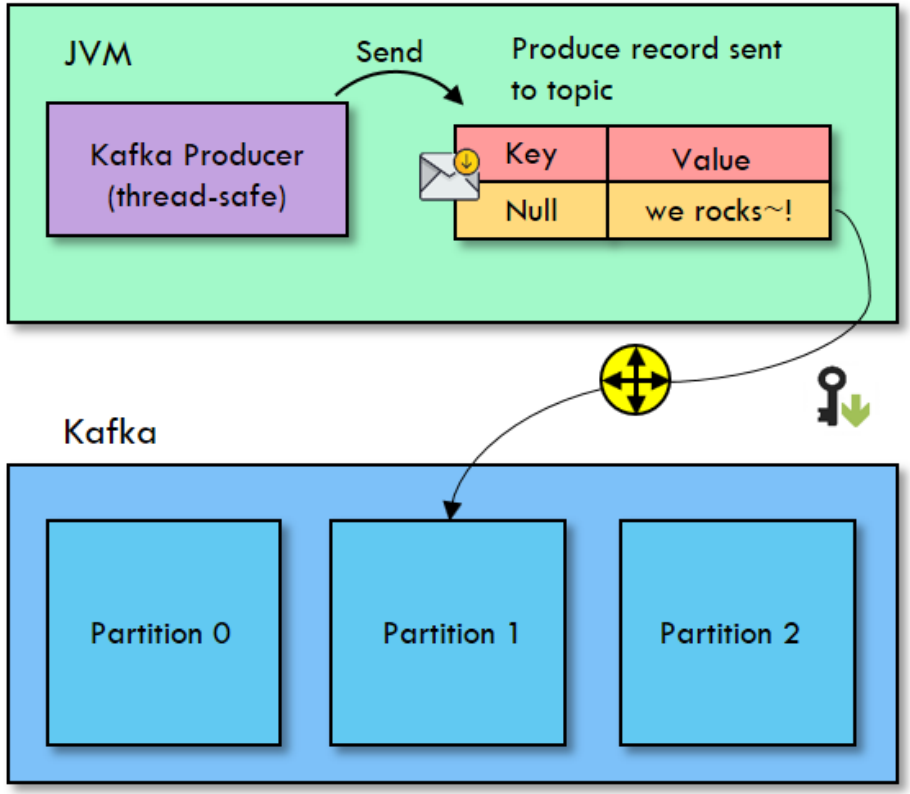


Kafka Producer



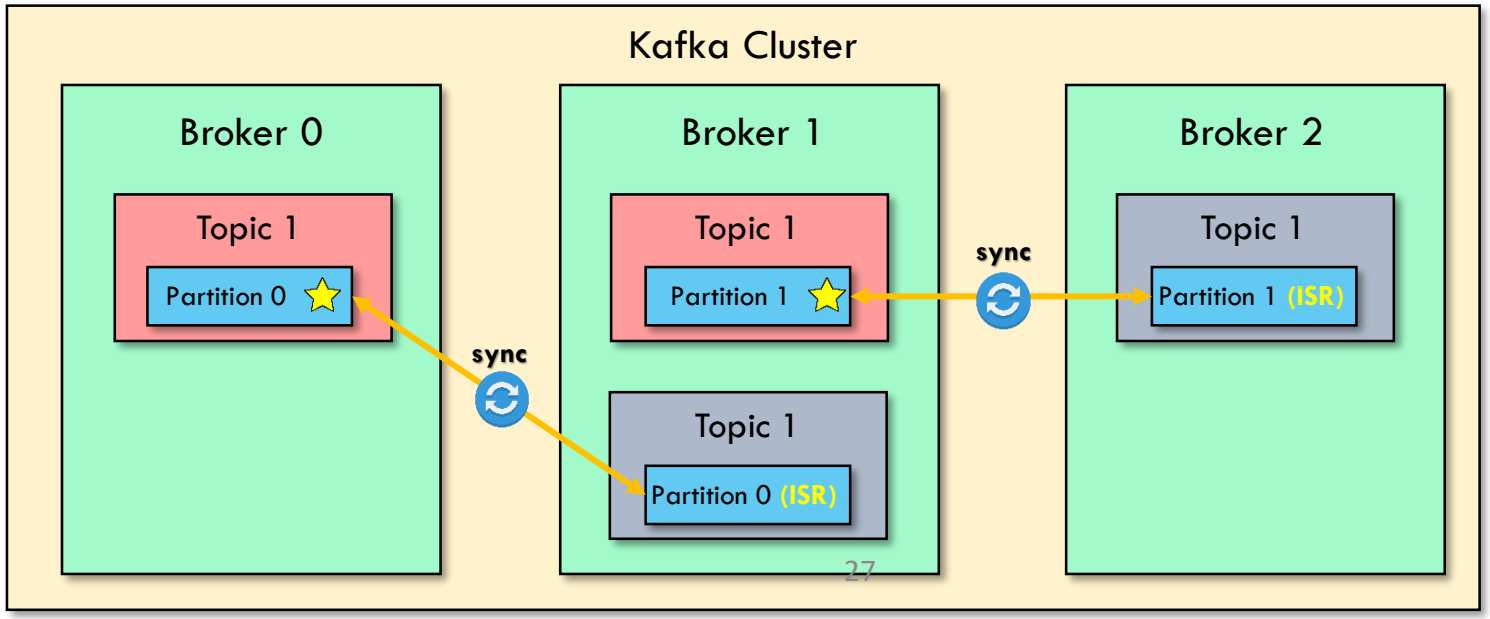
Producers

- **Producers** write data to **topics**.
- They only have to specify the topic name and one broker to connect to, and Kafka will automatically take care of routing the data to the right brokers.



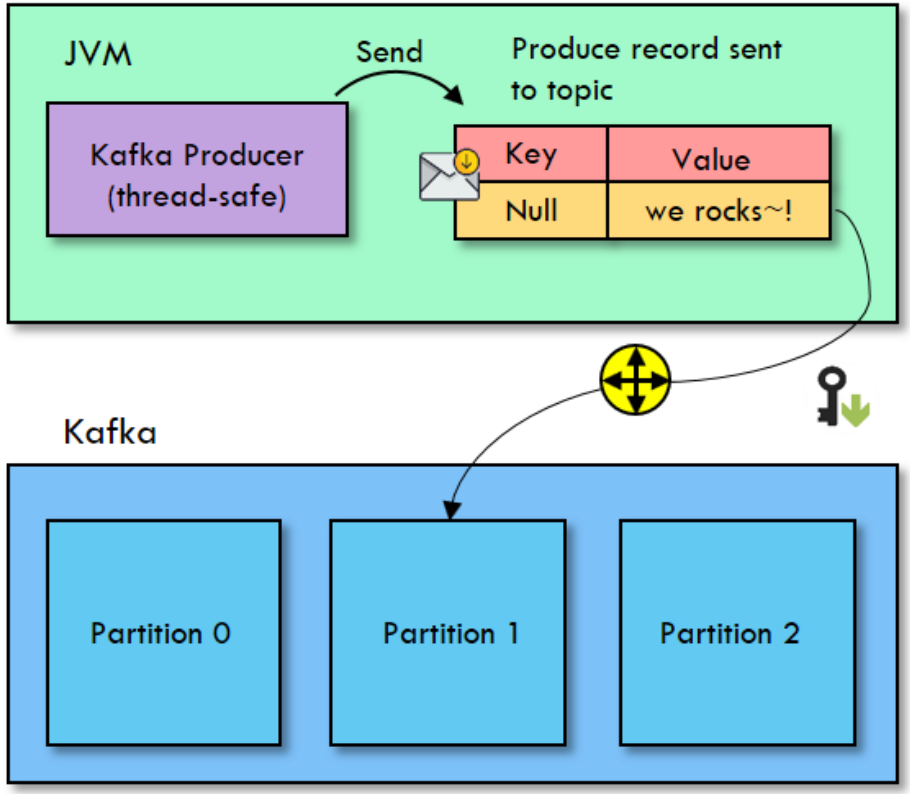
Producers: Broker acknowledgement

- Producer can choose to receive acknowledgement of data writes:
  - **acks** = 0 : Producer won't wait for acknowledgement (possible data loss)
  - **acks** = 1 : Producer will wait for leader acknowledgement (limited data loss)
  - **acks** = all (-1) : Leader + replicas acknowledgment (no data loss)

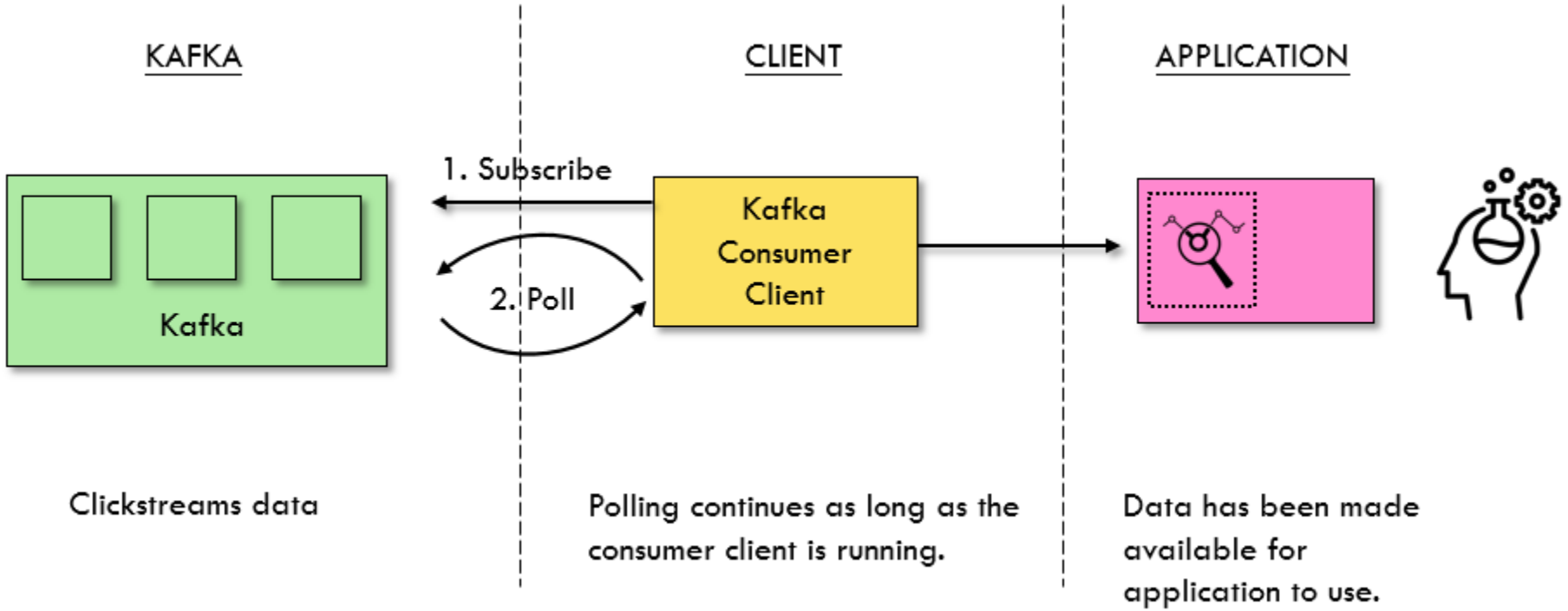


Producers: Message keys

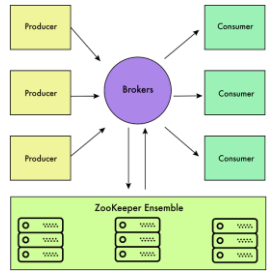
- Producers can choose to send a key with the message
- If a **key** is sent, then the producer has the guarantee that all messages for that key will always go to the same partition
- This enables to guarantee ordering for a specific **key**



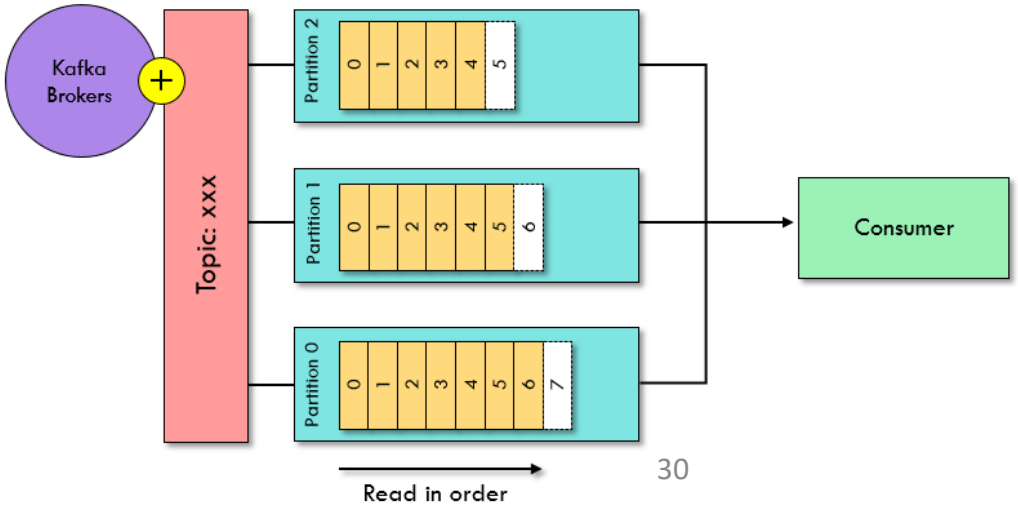
Kafka Consumer



Consumers

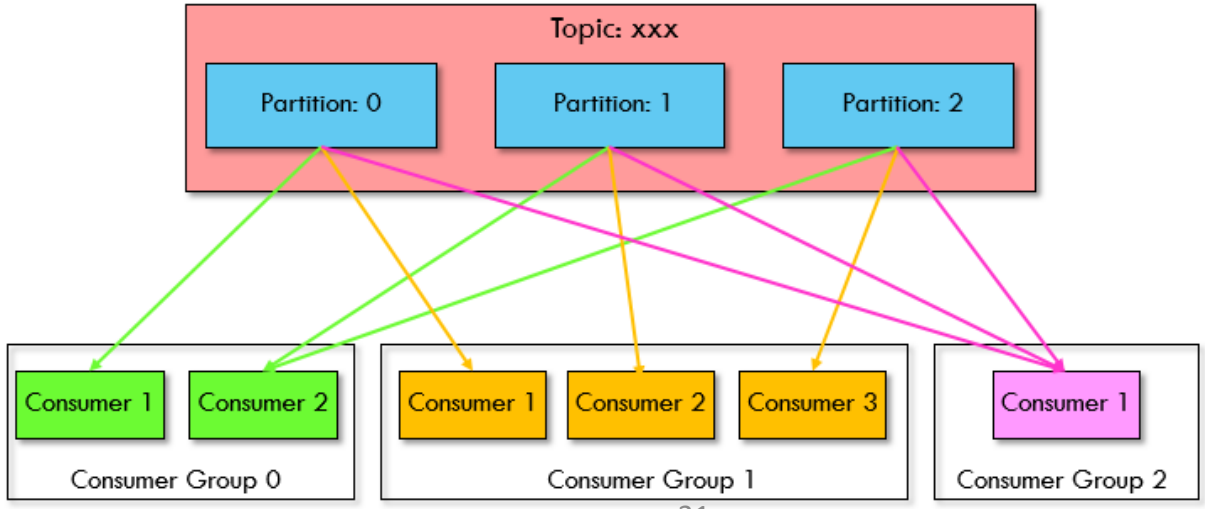


- **Consumers** read data from a topic
- They only have to specify the topic name and one broker to connect to, and Kafka will automatically take care of pulling the data from the right brokers
- Data is read in order **for each partitions**



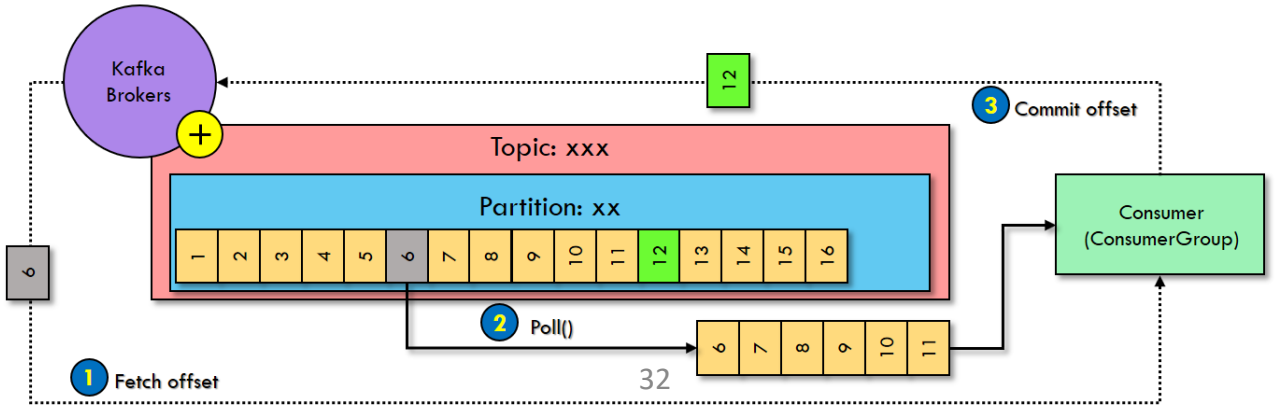
Consumer Groups

- Consumers read data in **consumer groups**
- Each consumer within a group reads from exclusive partitions
- You should control consumers instances less or equal than partitions (otherwise some will be inactive)



Consumer Offsets

- Kafka stores the offsets at which a **consumer group** has been reading
- The **offsets** commit live in a Kafka topic named "**__consumer_offsets**"
  - Key = [group, topic, partition] , Value=offset
- When a consumer has processed data received from Kafka, it should be **committing** the **offsets**
- If a consumer process dies, it will be able to read back from where it left off



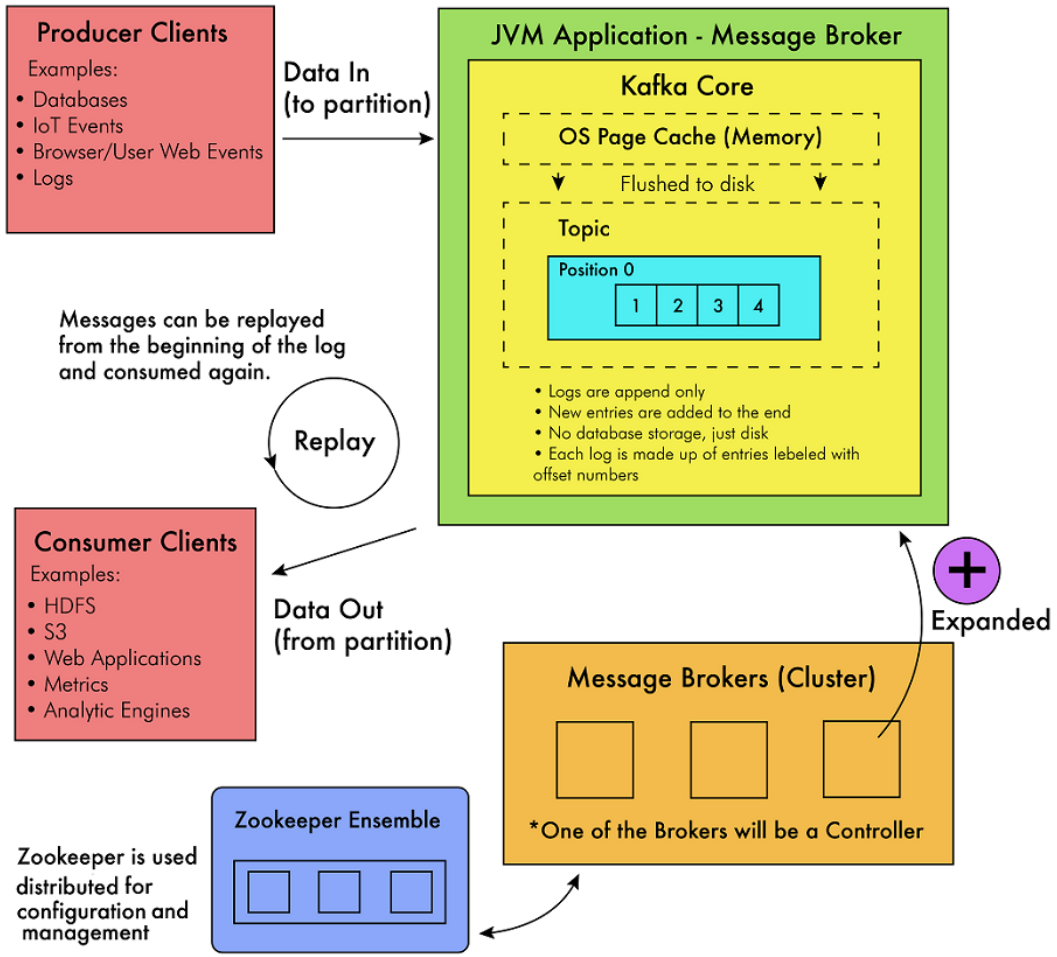
Kafka Guarantees

- Messages are appended to a **topic-partition** in the order they are sent
- Consumers read messages in the order stored in a **topic-partition**
- With a replication factor of N, producers and consumers can tolerate up to N-1 brokers being down
- This is why a replication factor of 3 is a good idea:
  - Allows for one broker to be taken down for maintenance
  - Allows for another broker to be taken down unexpectedly
- As long as the number of partitions remains constant for a topic (no new partitions), the same key will always go to the same partition

Delivery semantics for consumers

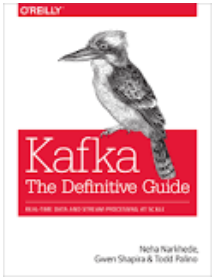
- As learned before, consumers choose when to commit offsets.
- **At most once**: offsets are committed as soon as the message is received. If the processing goes wrong, the message will be lost (it won't be read again).
- **At least once**: offsets are committed after the message is processed. If the processing goes wrong, the message will be read again. This can result in duplicate processing of messages. Make sure your processing is idempotent (i.e. processing again the message won't impact your systems)
- **Exactly once**: Very difficult to achieve / need strong engineering. Since version 0.11, Kafka start to support Exactly-once.

Terminology of Kafka

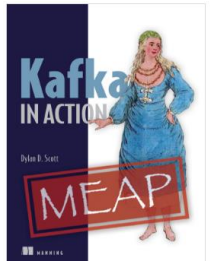


Reference

1. Kafka: The Definitive Guide – O'REILLY



2. Kafka In Action - MANNING



3. Learn Apache Kafka for Beginners – Udemy (Stephane Maarek)

4. Confluent Document of Kafka – confluent.io

confluent

36