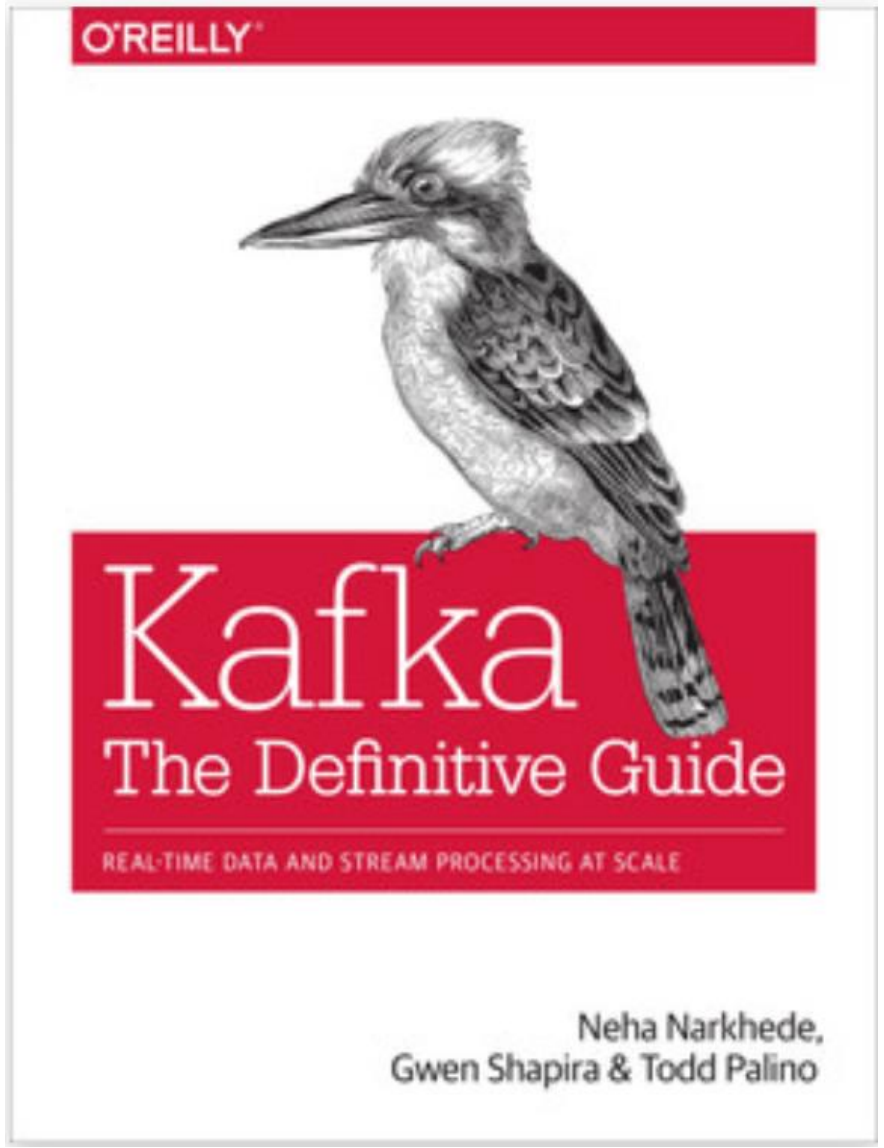
Kafka

The Definitive Guide

Amin FARVARDIN
Nov 2023



Agenda

- Introduction to Apache Kafka
- Kafka Architecture Components
- Key Concepts in Kafka
- Best Practices and Considerations
- Limitations and Challenges
- Practical Examples and Exercises

Introduction to Apache Kafka

What is Kafka?

- Apache Kafka is a **distributed data store** optimized for ingesting and processing streaming data in real time.

- It supports streaming analytics and mission-critical use cases with guaranteed **ordering**, **no message loss**, and **exactly-once processing**.

- It is massively scalable
  - because it allows data to be distributed across multiple servers
- It's extremely fast
  - because it decouples data streams, which results in low latency.
- It is fault tolerance
  - Because it distributes and replicates partitions across many servers

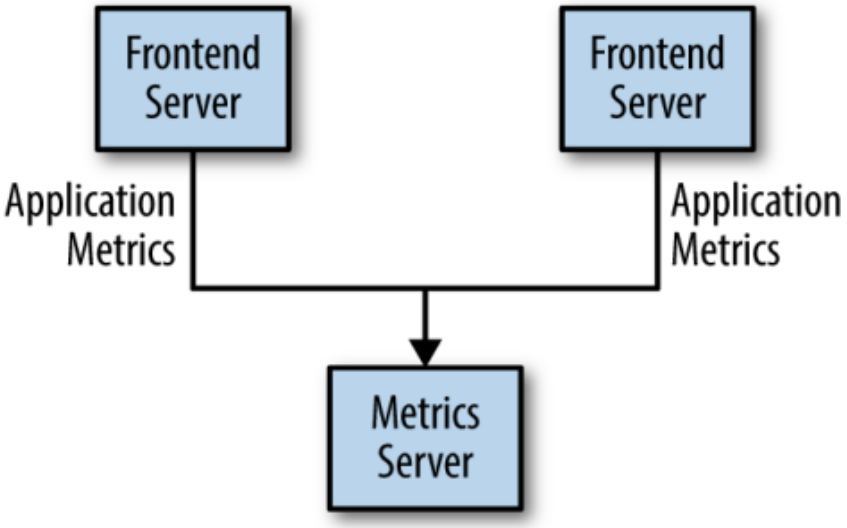
Introduction to Apache Kafka

Kafka’s Origin

- Kafka was developed around 2010 at LinkedIn
- To solve the low-latency ingestion of large amounts of event data
- The key was the "real-time" processing

Introduction to Apache Kafka

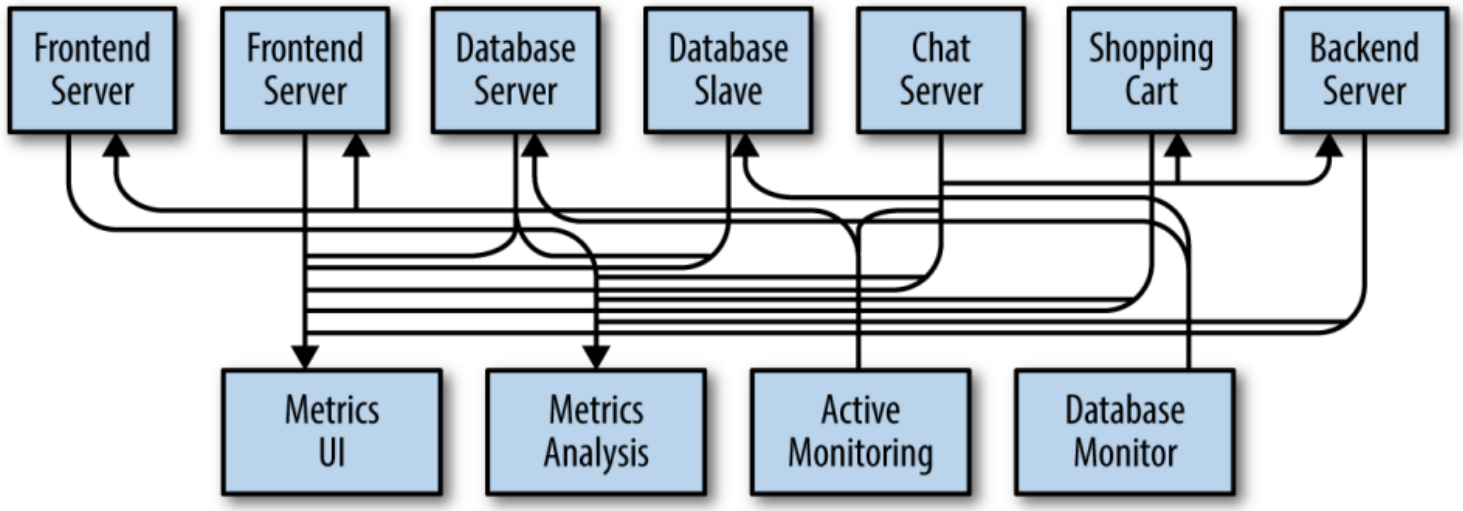
Publish/Subscribe Messages



A single, direct metrics publisher

Introduction to Apache Kafka

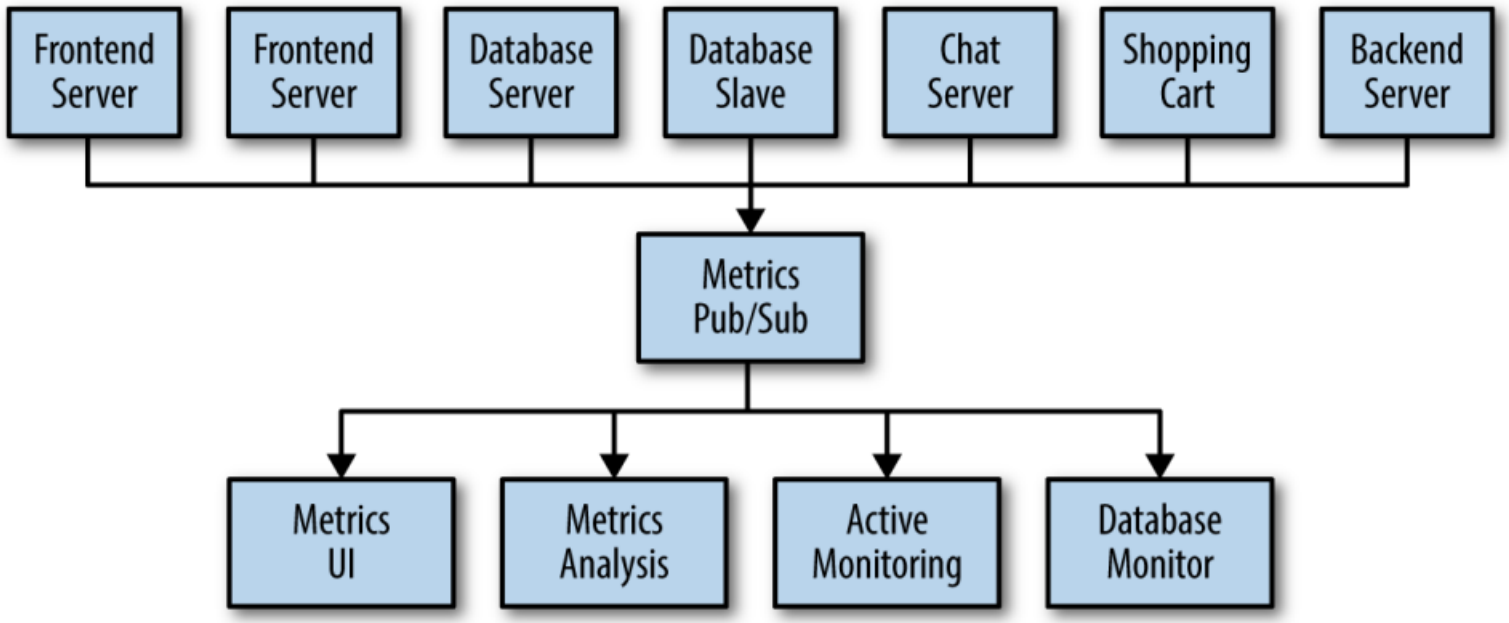
Publish/Subscribe Messages



Many metrics publishers, using direct connections

Introduction to Apache Kafka

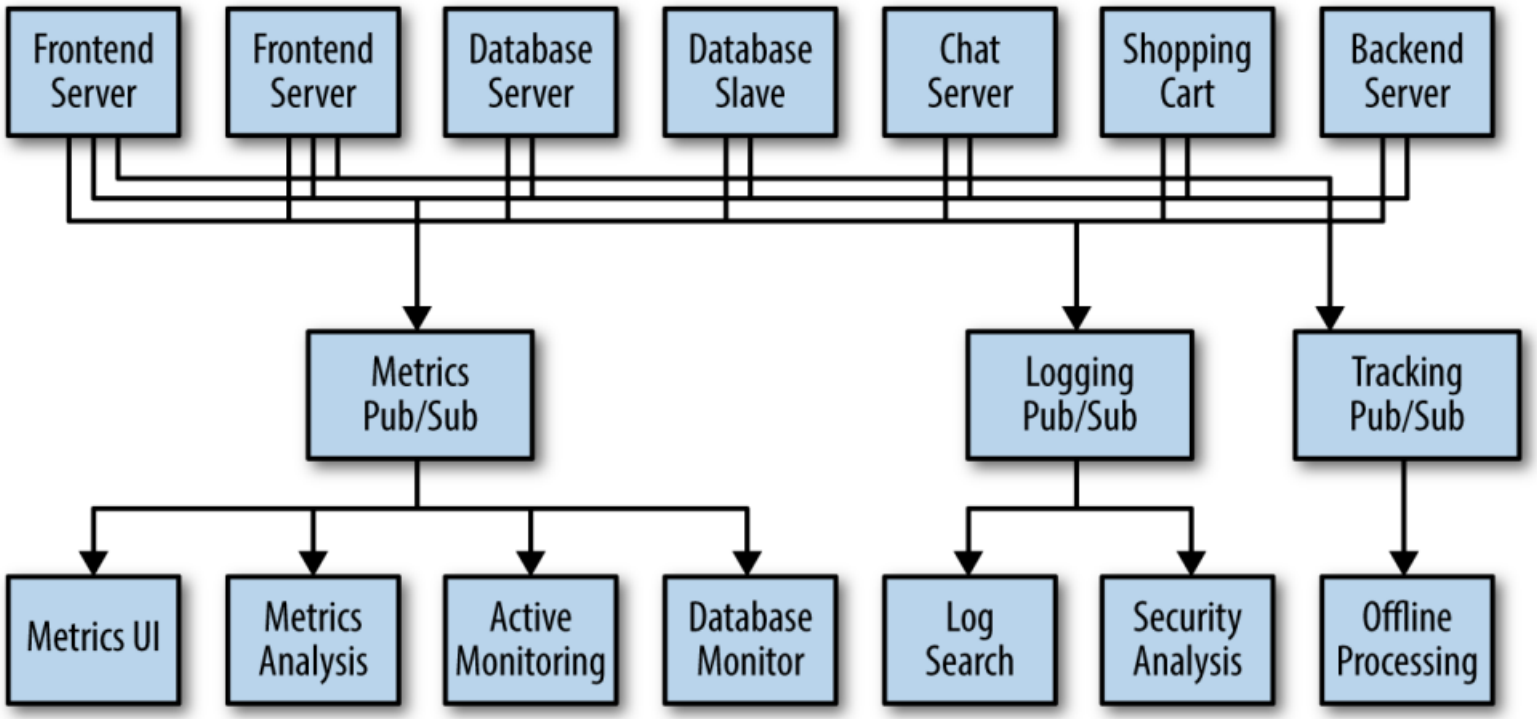
Publish/Subscribe Messages



A metrics publish/subscribe system

Introduction to Apache Kafka

Publish/Subscribe Messages (Individual Queue Systems)



Multiple publish/subscribe systems

Introduction to Apache Kafka

Publish/Subscribe Messages

- Publish/subscribe messaging is a pattern characterized by the sender (publisher) of a piece of data (message) not directing it to a receiver
- Publisher classifies the message somehow
- Receiver (subscriber) subscribes to receive certain classes of messages
- They often have a broker (a central point where messages are published)

Introduction to Apache Kafka

Some characteristics

- In the beginning Kafka was often described as:
  - Distributed commit log:
    - An ordered, immutable sequence of messages that are continually appended to
  - Distributing streaming platform:
    - A software system that manages the constant flow of information in real-time or near real-time
- Data within Kafka is stored
  - durable,
  - in order,
  - can be read deterministically
- In Kafka, data can be distributed within the system to provide
  - additional protections against failures
  - significant opportunities for scaling performance

Introduction to Apache Kafka

Messages & Batches

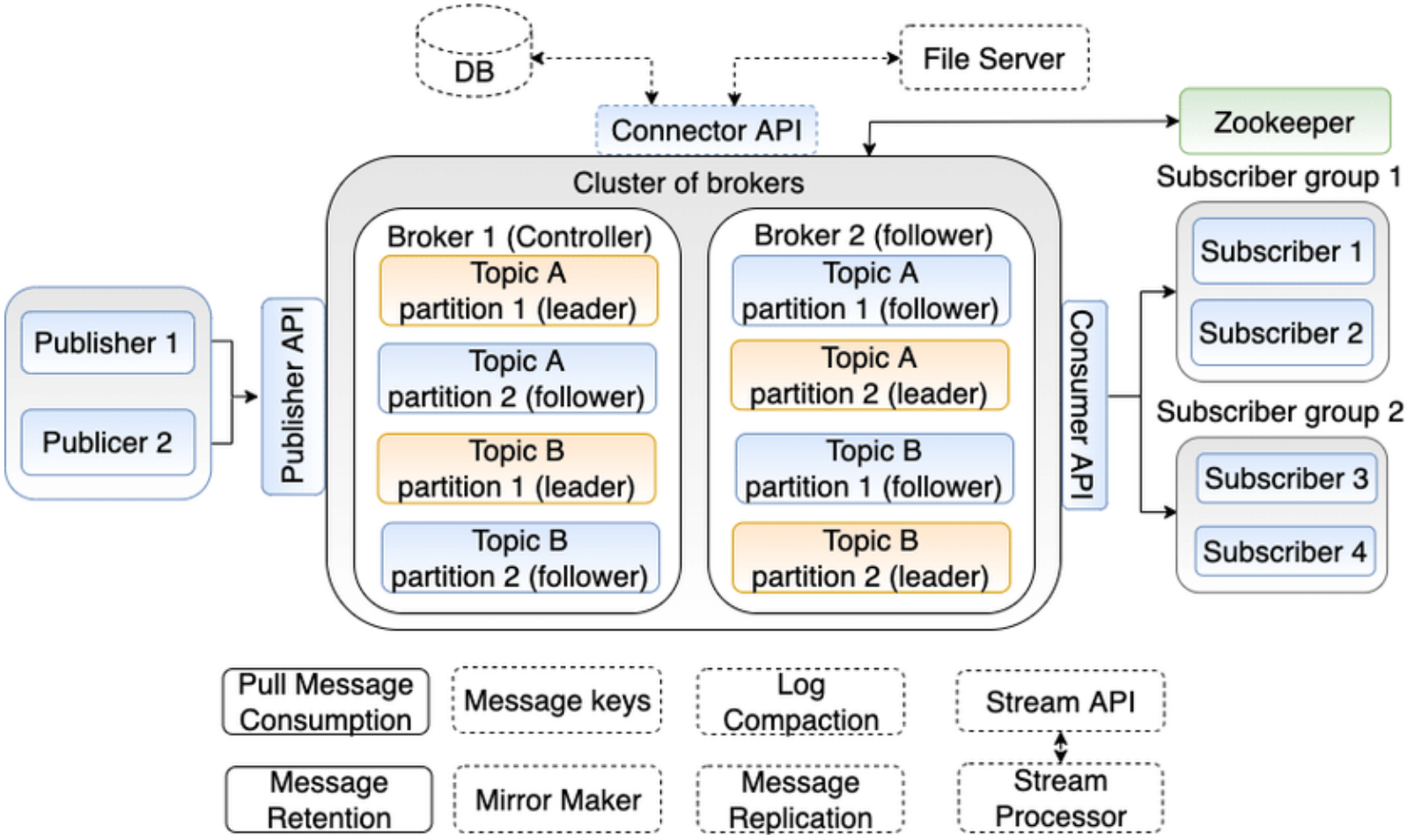
- The unit of data called a **message**
- For database background: think of a **row or a record**
- A message is simply an array of bytes
- A message can have an optional bit of metadata, **referred to as a key**
  - A message or a key is a byte array. Have no specific meaning to Kafka
- Keys are used for a controlled message when writing to partitions
- For efficiency, messages are written into Kafka in batches
  - A batch is just a collection of messages,
  - They are being produced on the same topic and partition

Agenda

- Introduction to Apache Kafka
- Kafka Architecture Components
- Key Concepts in Kafka
- Best Practices and Considerations
- Limitations and Challenges
- Practical Examples and Exercises

Kafka Architecture Components

Overview



Apostolos Lazidis, Konstantinos Tsakos, Euripides G. M. Petrakis :
Publish-Subscribe approaches for the IoT and the cloud: Functional and performance evaluation of open-source systems. Internet Things 19: 100538 (2022)

Agenda

- Introduction to Apache Kafka
- Kafka Architecture Components
- Key Concepts in Kafka
- Best Practices and Considerations
- Limitations and Challenges
- Practical Examples and Exercises

Key Concepts in Kafka

Topics and Partitions

- Kafka topics are the categories used to organize messages
- Any data format, such as CSV, JSON, or AVRO, is supported by the Kafka topic
- The sequence of topics is called "data streaming."
- Each topic has a unique name in the entire Kafka cluster
- Messages are sent to and read from specific topics
- In other words, producers write data to topics, and consumers read data from topics

Key Concepts in Kafka

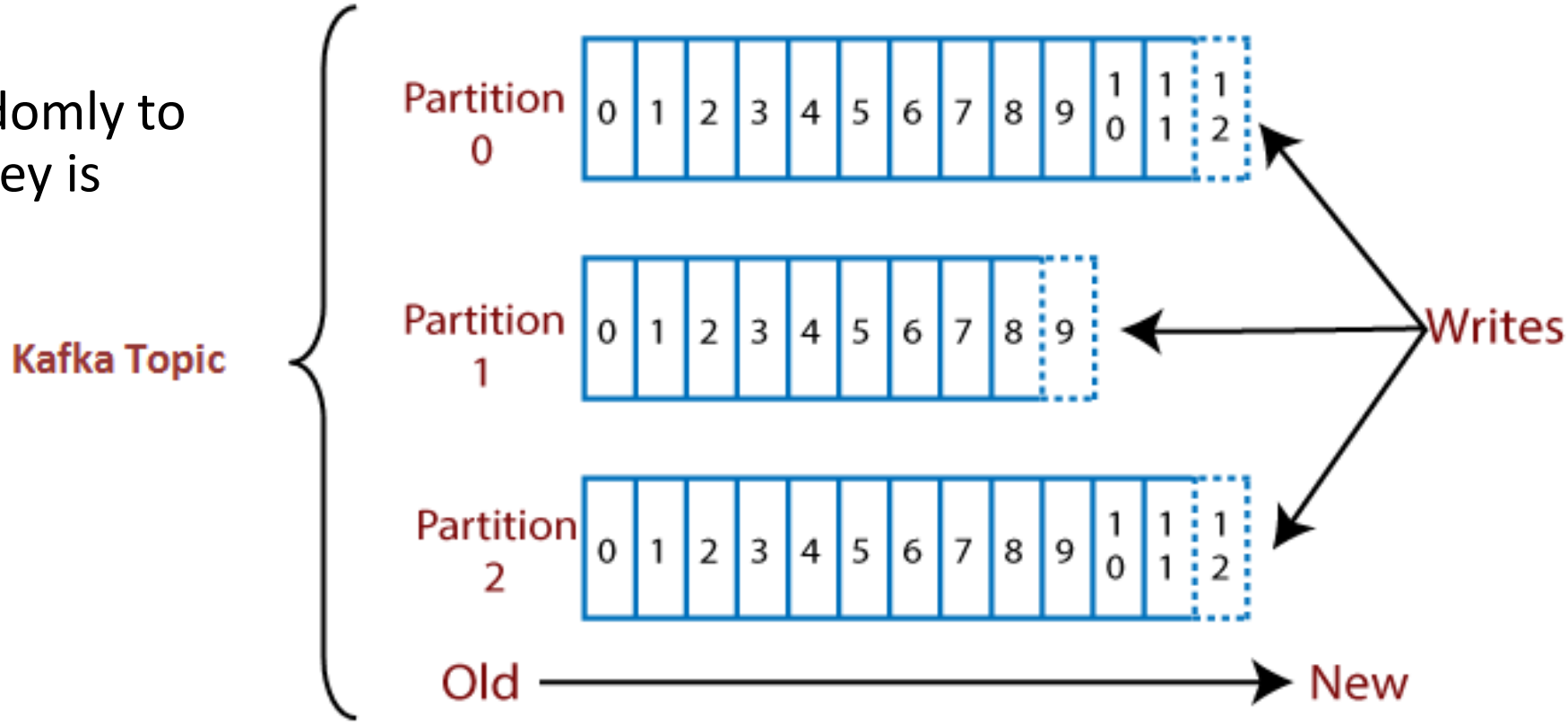
Topics and Partitions

- Kafka topics are similar to database tables
- Topics can have multiple partitions
- Partitions are ordered and have their own incremental IDs called offsets
- Offsets have meaning only within a specific partition
- Order is guaranteed only within a partition
- Data in Kafka is kept for a limited time, by default 168 hours or one week, and offsets keep incrementing
- New messages are appended to the end of a partition
- Understanding topics, partitions, and offsets is important for working with Kafka

Key Concepts in Kafka

Topics and Partitions

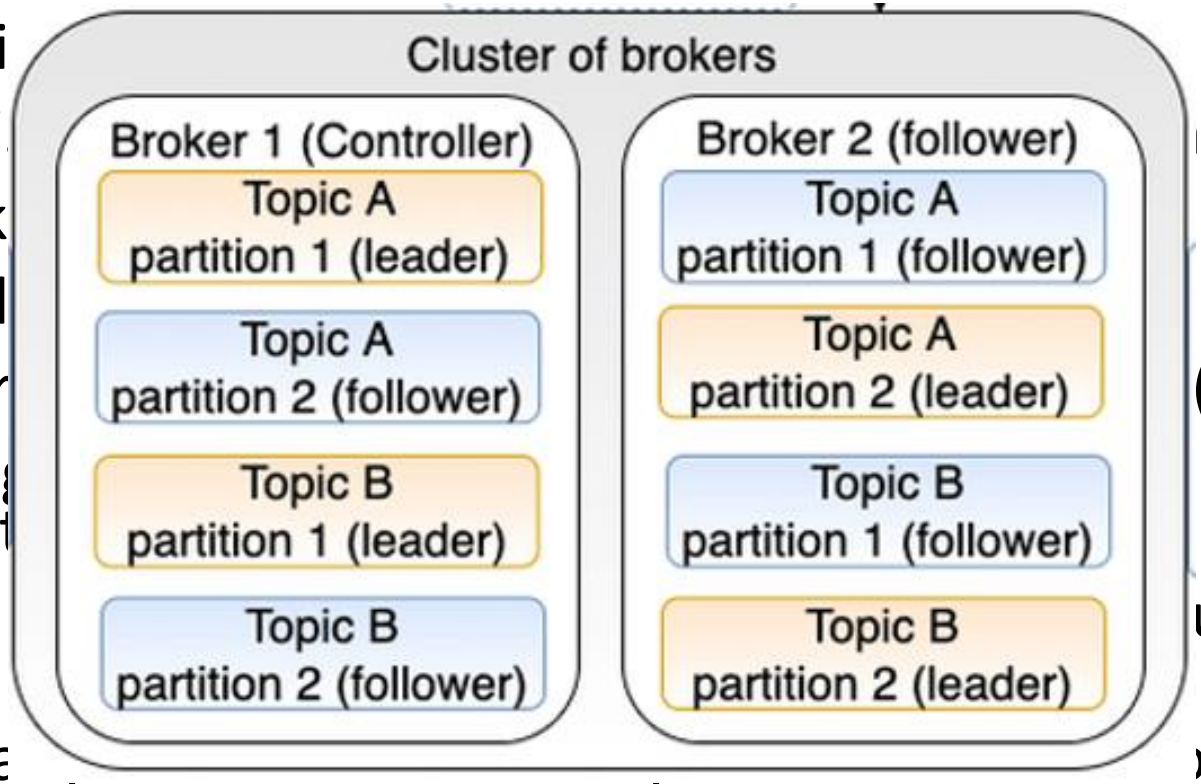
- Once the data is written to a partition, it can't be changed (Immutability)
- Data is assigned randomly to a partition unless a key is provided



Key Concepts in Kafka

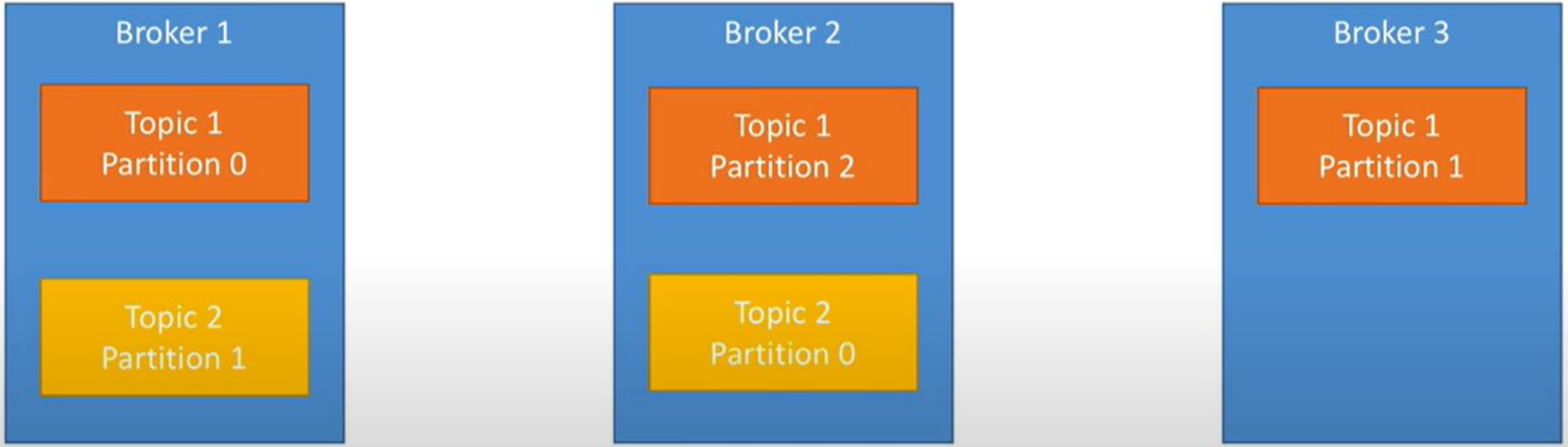
Brokers

- A Kafka cluster i
- A Kafka broker i d send the data
- Each Kafka brok
- Each broker will
- All the topic par (load balanced)
- After connecting have connectivity to t
- A good number usters have 100+ brokers
- You can create a that (Horizontal Scaling)



Key Concepts in Kafka

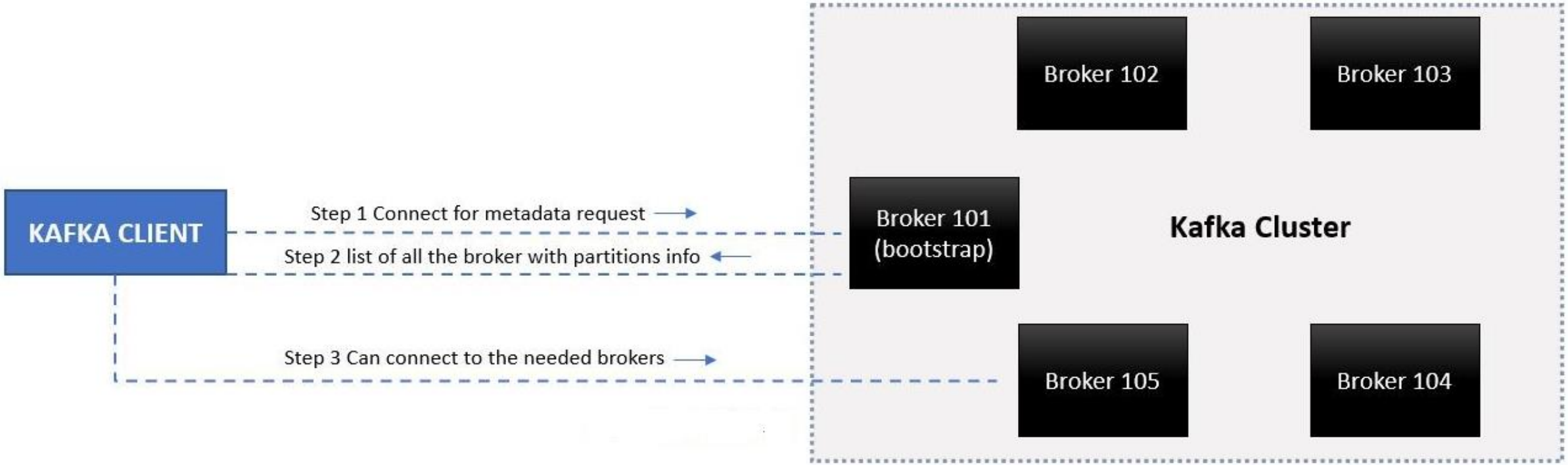
Brokers



Key Concepts in Kafka

Brokers

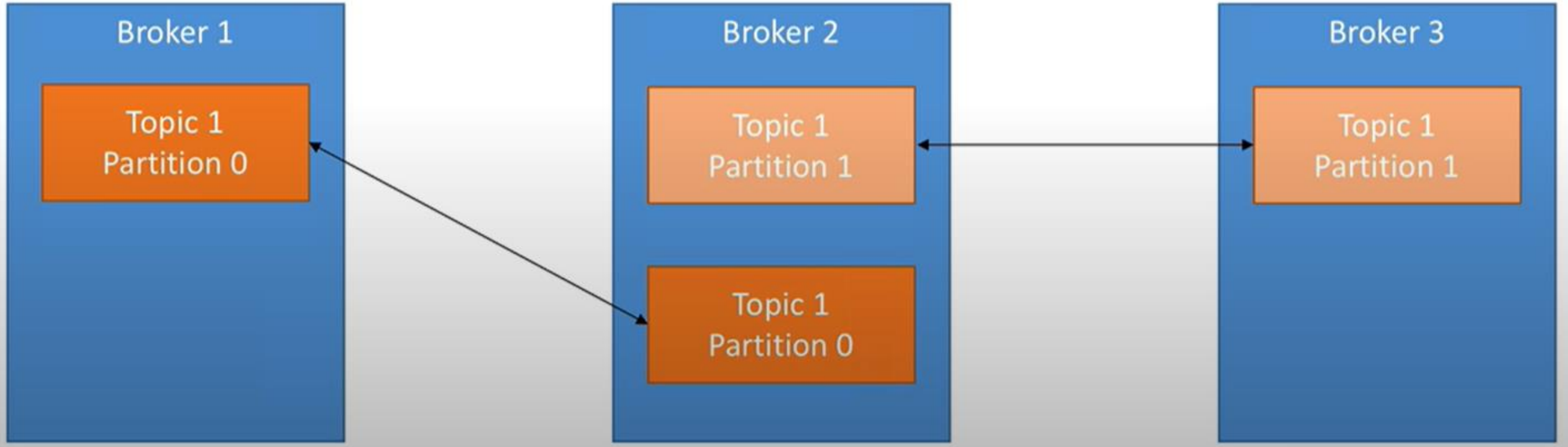
- Every Kafka broker is called a “bootstrap server”
- By connecting to one broker, the Kafka clients will know how to be connected to the entire cluster.
- One bootstrap broker connectivity helps to understand the other clusters’ partitions
- Each broker knows about all brokers, topics, and partitions.



Key Concepts in Kafka

Topic Replication Factor

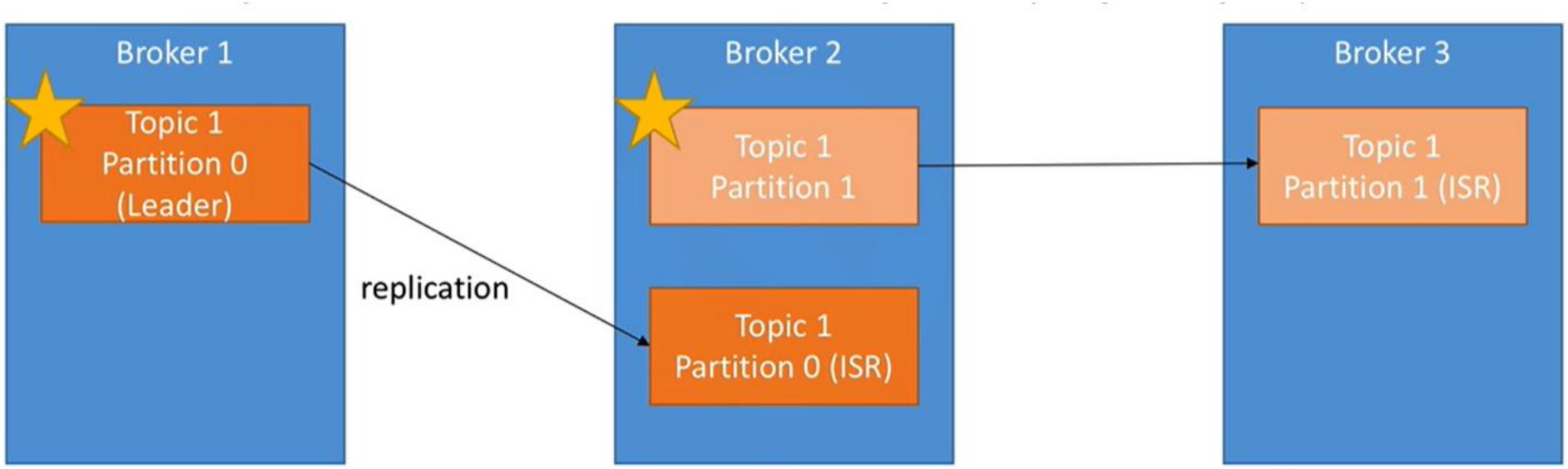
- Topics should have replication factor > 1 (min 2 and optimal is 3 replicas)
- If a broker is down, another broker serves the data
- Ex: Topics with 2 partitions and replication factor of 2



Key Concepts in Kafka

Leader vs. follower partitions

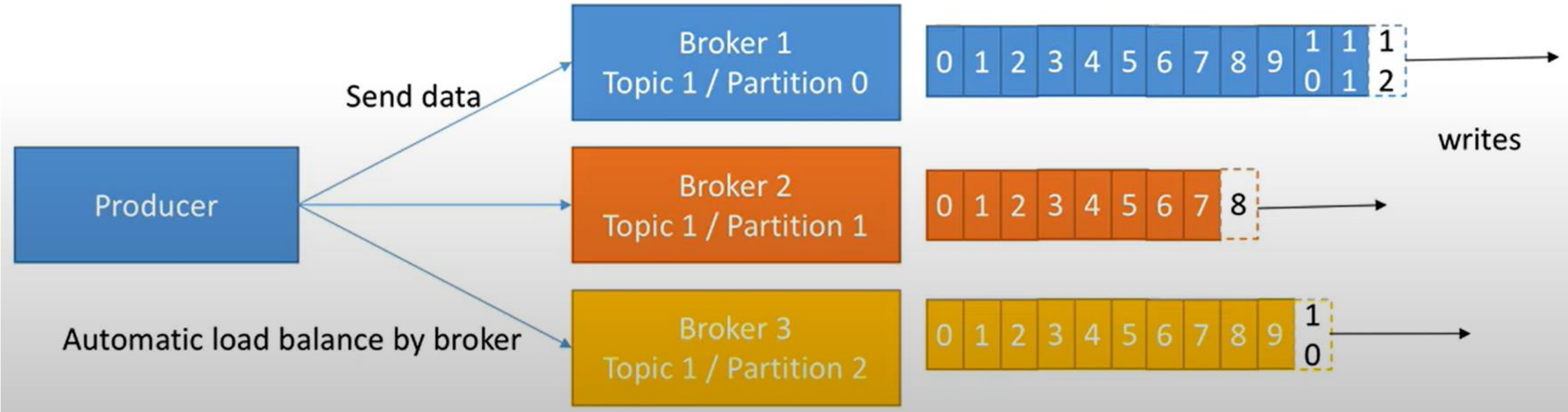
- Leader:
  - It is the first broker that receives the copy of your desired partition and is not only responsible for receiving data but also sending data to other available brokers.
- Followers:
  - Each follower represents a copy of the leader's data partitions and is also known as an in-sync replica.
- At any time only ONE broker can be a leader for a given partition



Key Concepts in Kafka

Producers

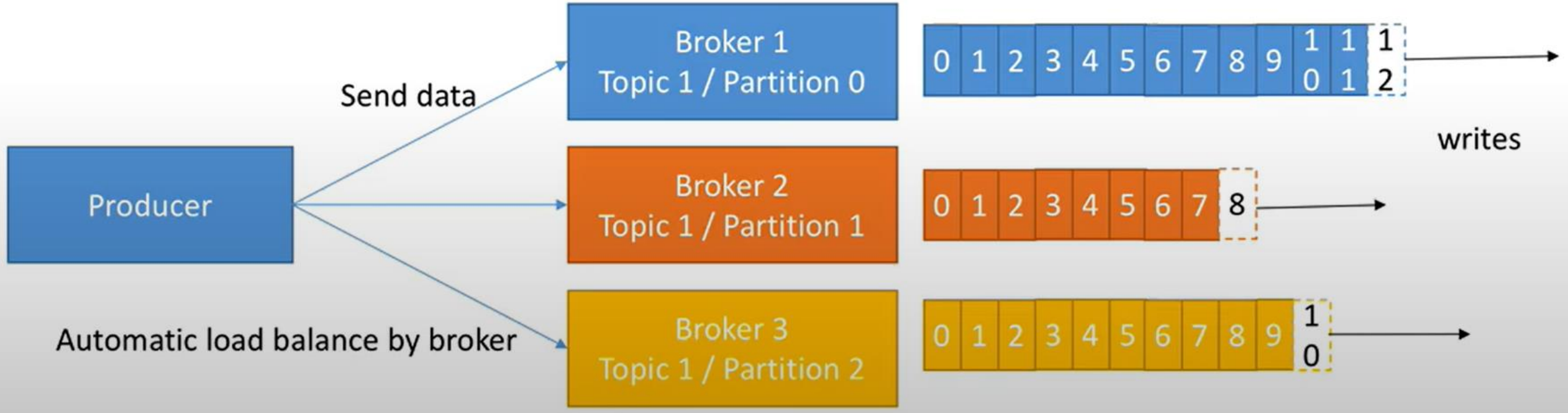
- Producers write data on topics.
- Topic name and one broker is enough to write data on Kafka.
- Kafka will take care of writing data into right brokers



Key Concepts in Kafka

Producers - Acknowledgments

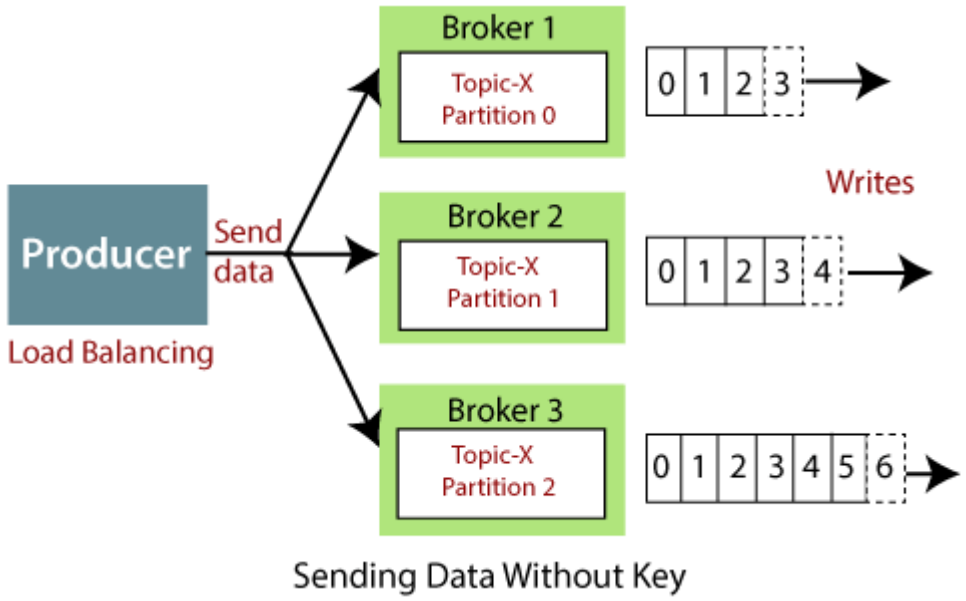
- **Ack=0** – Do not wait for any acknowledgment (possible data loss)
- **Ack=1** – Producer will wait for leader acknowledgment (limited data loss)
- **Ack=all** – Leader + replicas acknowledgment (No data loss)



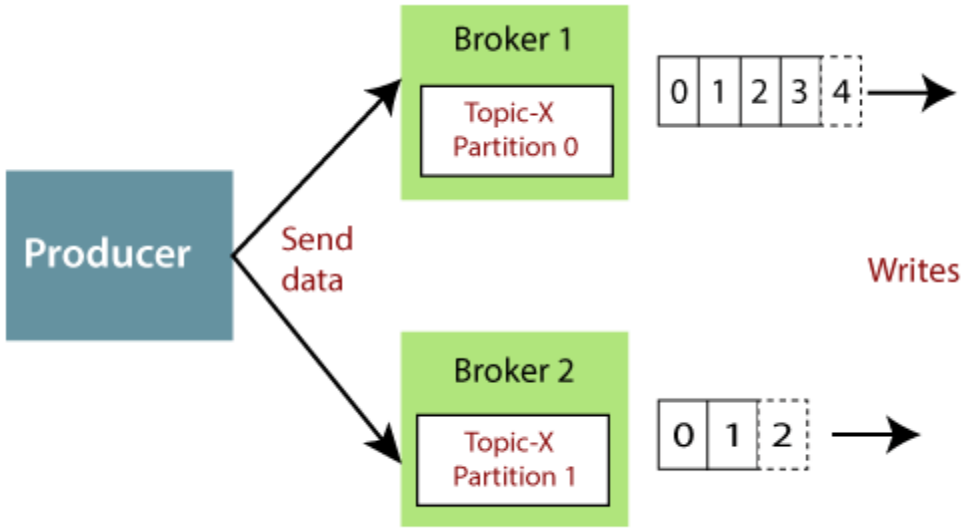
Key Concepts in Kafka

Producers – Message Keys

- Besides the Topic name and Broker, writing could be:
  - Randomly: without a key
  - Or into a specific partition and specific broker: with a key



Sending Data Without Key

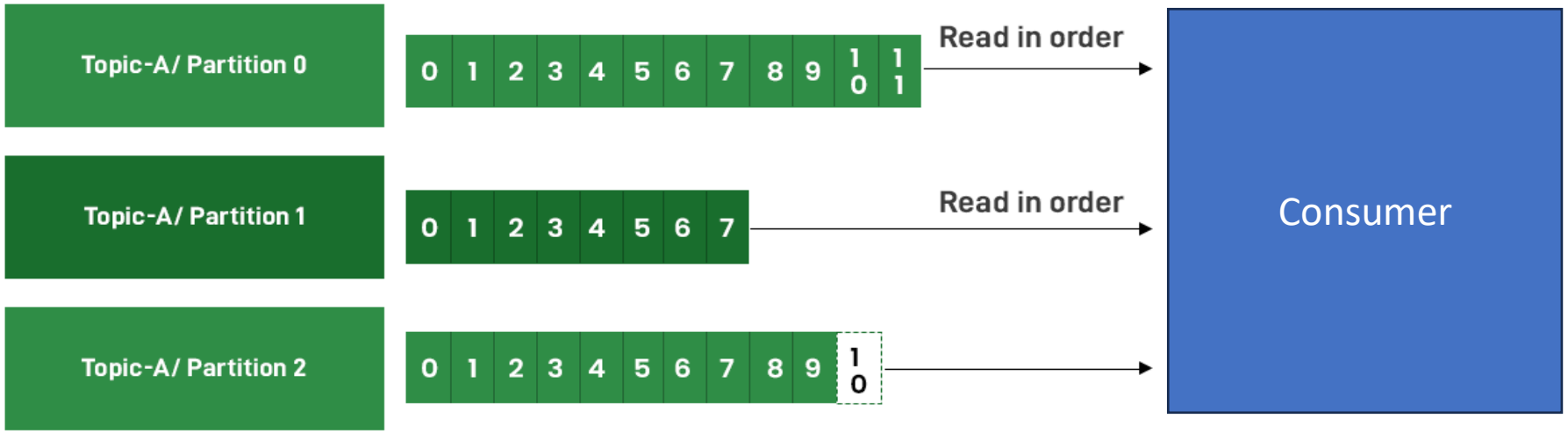


Sending data with key = Prod_id

Key Concepts in Kafka

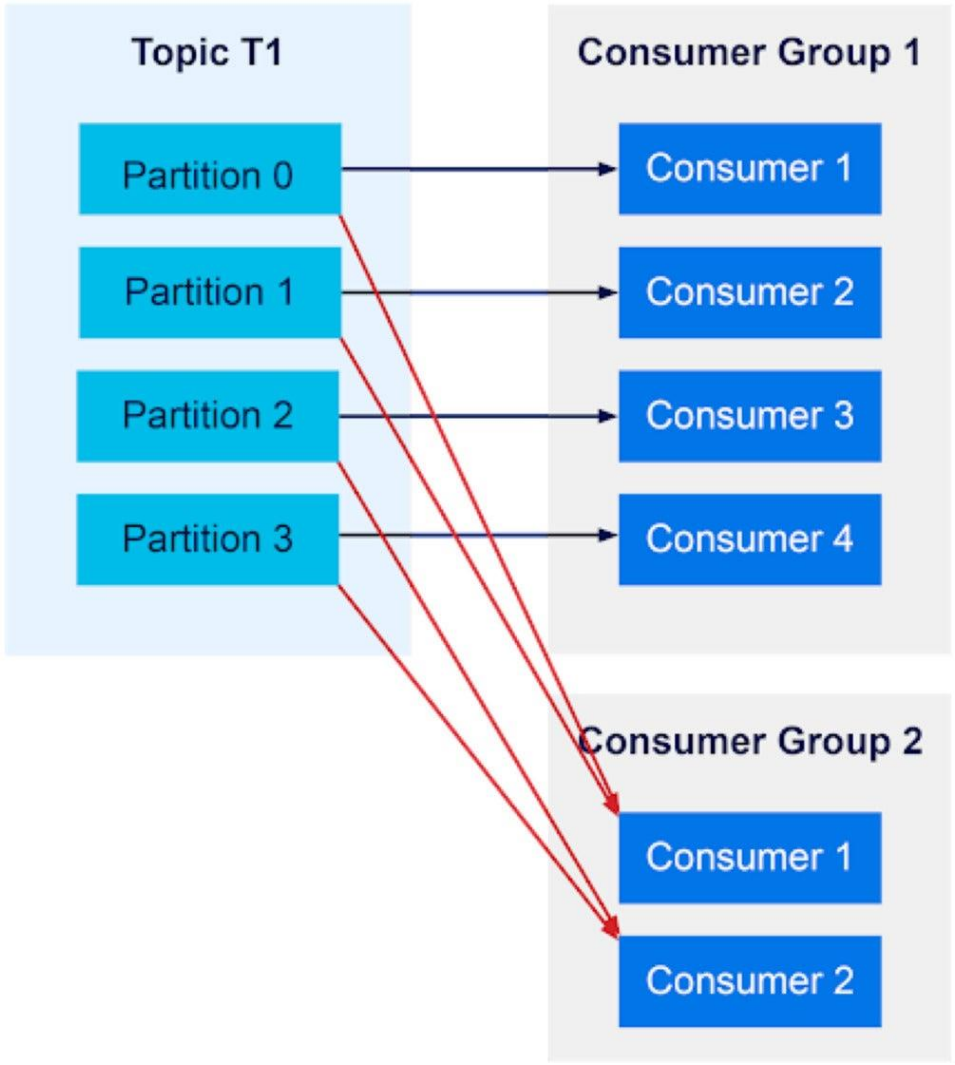
Consumers

- Consumers read data from a topic
- Consumers need to mention the topic name and a broker to connect
- Kafka will take care of pulling data from the right brokers
- Data reads **in order** for each partition
- Messages read in order within a partition and in parallel across partitions



Key Concepts in Kafka
Consumers Groups

- Consumers read data in Consumer groups
- Each consumer within a group reads data from a partition
- Having consumers more than partitions, keeping some consumers ideal



Key Concepts in Kafka

Consumers Offset

- Kafka stores the offsets at which a consumer group has been reading
- The offsets commit live in a Kafka topic named *__consumer_offsets*
- Consumer commits the offsets where the data has been read
- If the consumer process dies, it can read back from where it left off, thanks to the consumer offset

Key Concepts in Kafka

- **Log Compaction:** Kafka supports log compaction, which ensures that the latest message for each key is retained, making it useful for maintaining the latest state for particular records.
- **Event Timestamp:** Kafka records include an event timestamp that represents when the message was produced, allowing consumers to work with events in chronological order.
- **Connectors:** Kafka connectors are used to integrate Kafka with other systems and data sources. They facilitate the transfer of data in and out of Kafka topics, making it a central hub for real-time data processing

Key Concepts in Kafka

Role of Zookeeper in Kafka

- **Zookeeper (Optional):** In older Kafka versions, Apache ZooKeeper was used for distributed coordination and managing broker metadata. However, more recent Kafka versions have reduced the dependency on ZooKeeper, and it is no longer a strict requirement.

Agenda

- Introduction to Apache Kafka
- Kafka Architecture Components
- Key Concepts in Kafka
- Best Practices and Considerations
- Limitations and Challenges
- Practical Examples and Exercises

Best Practices and Considerations

- Install Kafka & Produce and consume the first message

https://github.com/afarvardin/dauphine_tunis/blob/main/Kafka/Installation

Best Practices and Considerations

Kafka Use Cases

Activity tracking: This was the original use case for Kafka. LinkedIn needed to rebuild its user activity tracking pipeline as a set of real-time publish-subscribe feeds. Activity tracking is often very high volume, as each user page view generates many activity messages (events).

- User clicks, registrations, likes, time spent on certain pages, orders, environmental changes, and so on.

Example scenario:

- An online e-commerce platform can manage users' tracking activities in real-time.

Best Practices and Considerations

Kafka Use Cases

Real-time data processing

Many systems require data to be processed as soon as it becomes available. Kafka transmits data from producers to consumers with very low latency (5 milliseconds, for instance). This is useful for:

- Financial organizations, Predictive maintenance (IoT)
- Autonomous mobile devices, which require real-time data processing to navigate a physical environment.
- Logistical and supply chain businesses, to monitor and update tracking applications

Example scenario: A bank could use Kafka for processing transactions in real-time.

Best Practices and Considerations

Kafka Use Cases

Messaging

- Good replacement for traditional message brokers
  - throughput,
  - built-in partitioning,
  - replication, and
  - fault-tolerance, and
  - better scaling attributes.
- **Example scenario**:
  - A microservices-based ride-hailing app could use Kafka for sending messages between different services.

Best Practices and Considerations

WHEN NOT to use Kafka

“Little” Data

- As Kafka is designed to handle high volumes of data, it’s overkill if you need to process only a small amount of messages per day (up to several thousand). Use traditional message queues such as RabbitMQ for relatively smaller data sets or as a dedicated task queue.

Streaming ETL

- Despite the fact that Kafka has a streaming API, it’s painful to perform data transformations on the fly.

Agenda

- Introduction to Apache Kafka
- Kafka Architecture Components
- Key Concepts in Kafka
- Best Practices and Considerations
- Limitations and Challenges
- Practical Examples and Exercises

Limitations and Challenges

- **Steep learning curve** and technological complexity
- **Kafka is complex** in terms of cluster setup and configuration, maintenance, and data pipeline design.
- **Deployment may take days to weeks to even months**
  - Depending on the type of deployment (cloud or on-premise), cluster size, and the number of integrations. Many companies end up asking for help from Kafka vendors (mainly, **Confluent**) or other third-party service providers.
- **Need for extra tech experts**
- **You can't just install Kafka and let it go by itself**. A couple of in-house experts are necessary to continuously monitor what's going on and tune up the system.
- **No managing and monitoring tools**
- Apache Kafka **doesn't provide UI** for monitoring, though it's critical for running jobs correctly.
- ZooKeeper issue (to coordinate brokers inside the cluster)
  - doesn't allow a Kafka cluster to support more than **200,000 partitions** and significantly slows the system's work

Agenda

- Introduction to Apache Kafka
- Kafka Architecture Components
- Key Concepts in Kafka
- Message Handling in Kafka
- Kafka Cluster
- Kafka API’s
- Scalability, Durability, and Fault Tolerance
- Best Practices and Considerations
- Limitations and Challenges
- Practical Examples and Exercises

Practical Examples and Exercises

http://bit.ly/3u3RrMb