
kafka

Module 1: Kafka Basics

Objectives

After completing of this module, you should be able to:

- ✓ Understand Messaging Systems
- ✓ Why Kafka?
- ✓ Understand Kafka Components
- ✓ Understand the role of Zookeeper
- ✓ Zookeeper and Kafka Installation
- ✓ Know about Types of Kafka Clusters
- ✓ Work with Single Node Single Broker Cluster

Let’s have a look at Messaging Systems

Real-time data collection & analysis

In the present Big-Data world there are two challenges associated with real-time data.

1. Collecting data efficiently as the data grows big.

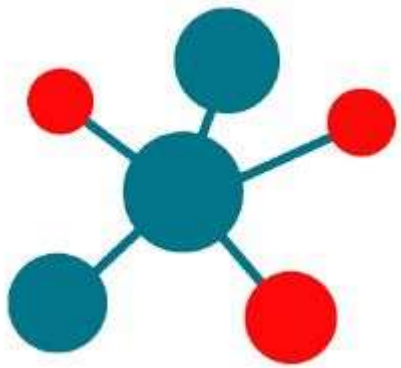
2. Analyzing data in real time : Data analysis typically include extracting many types of information from the data such as:

    - Usage Metrics
    - User Behavior Analysis
    - User Activity Log Analysis
    - Application Performance Metrics
    - Event Messages and many more…

Messaging Systems provide a solution

- Modern day Messaging Systems provide a means to solve the problems associated with efficient data collection and processing.

- Messaging Systems provide seamless integration among distributed applications with the help of messages that share these messages among them using a publisher/subscriber model



Publish / Subscribe (also know as pub/sub) messaging is a pattern that is characterized by the sender (publisher) of a piece of data (message) not specifically directing it to a receiver.

Data Pipelines

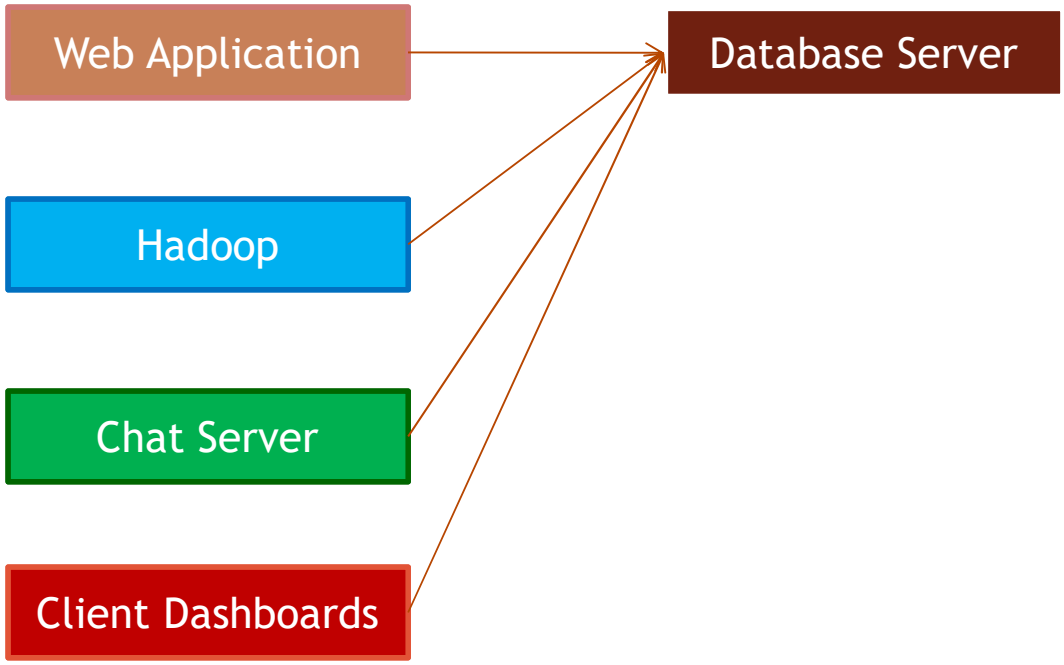
- Data Pipelines provide a communication channels that are required between different applications in an enterprise.

- Real-time data pipelines power applications and insights, providing the digital infrastructure for active data. This helps data-driven companies understand how their customer base is behaving in the moment, especially when it involves live operations.

A chat server needs to communicate with database server to store messages in real time.

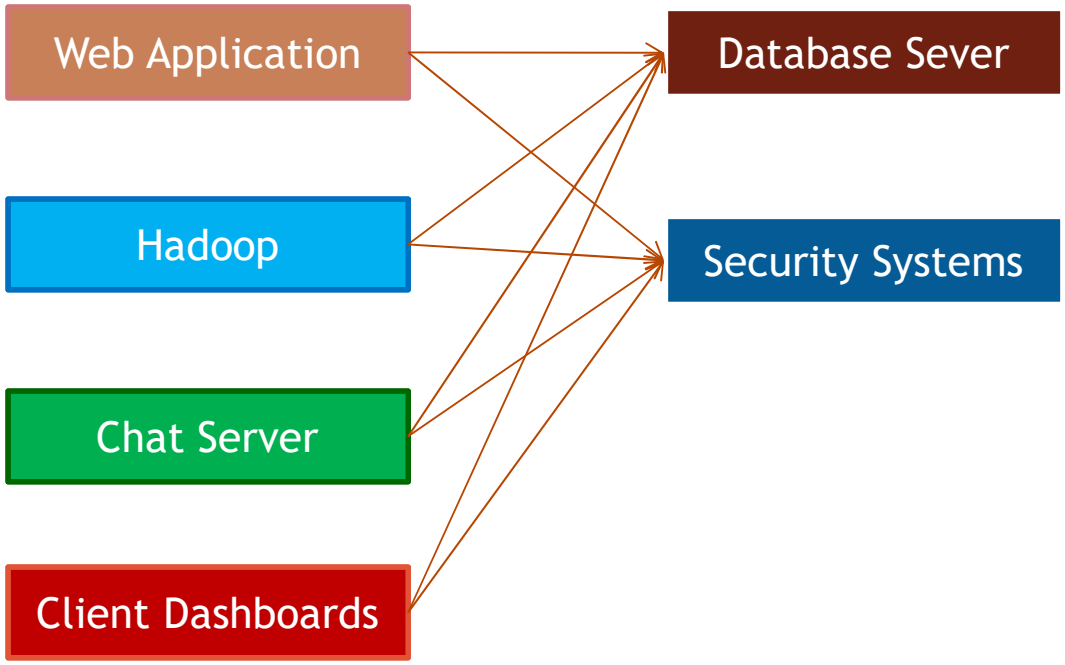
Data Pipelines with more nodes

There could be many different applications in an enterprise that may want to communicate with a central service such as a Database server



Data Pipelines can become complex

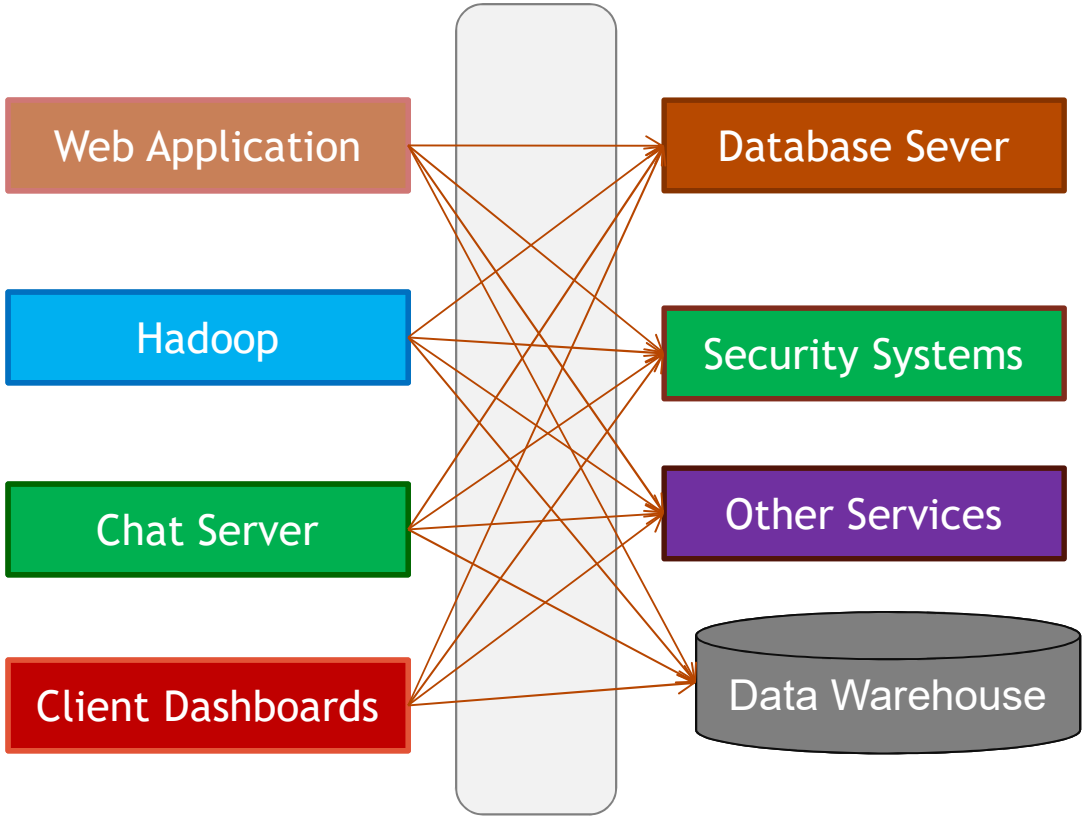
The same applications may also be communicating with security systems in addition to database server



- As we can see, as the communicating nodes increase, the data pipeline can become very complex.

Data Pipelines can become complex

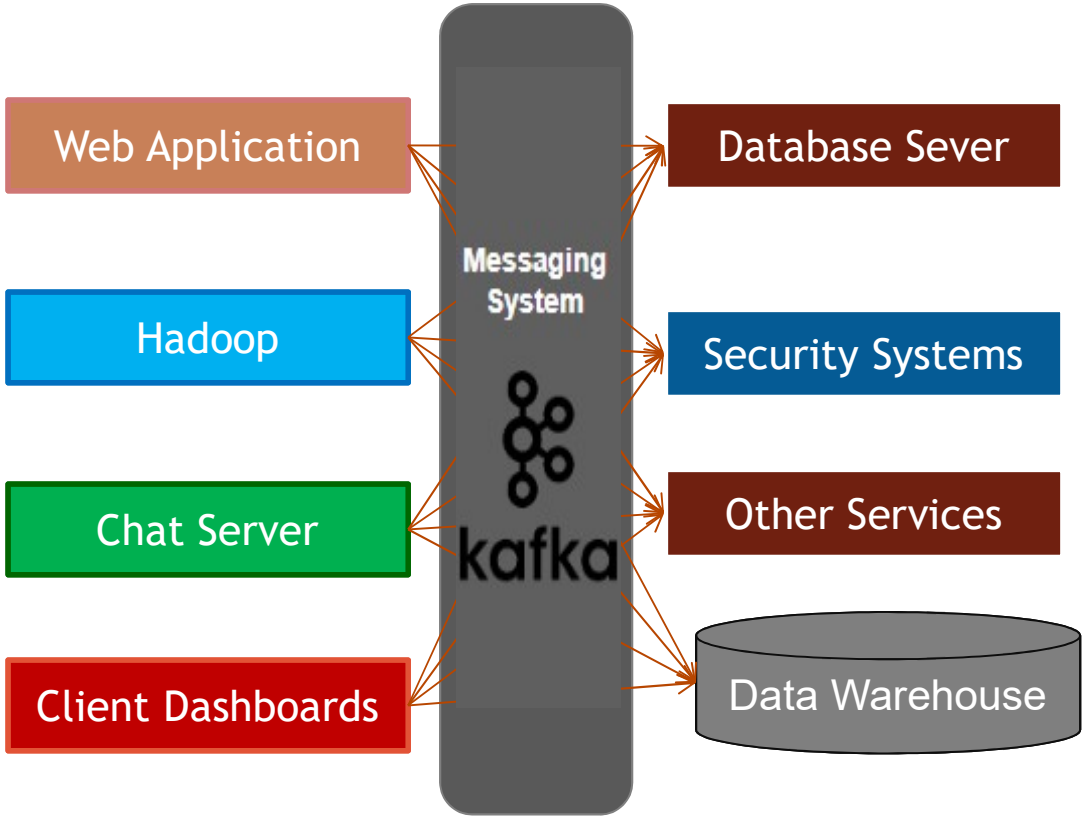
The same applications may also be communicating with security systems in addition to database server



- As the data pipeline becomes complicated, we need a better and efficient messaging system to efficiently handle data communication.

Kafka – solution to complex data pipelines

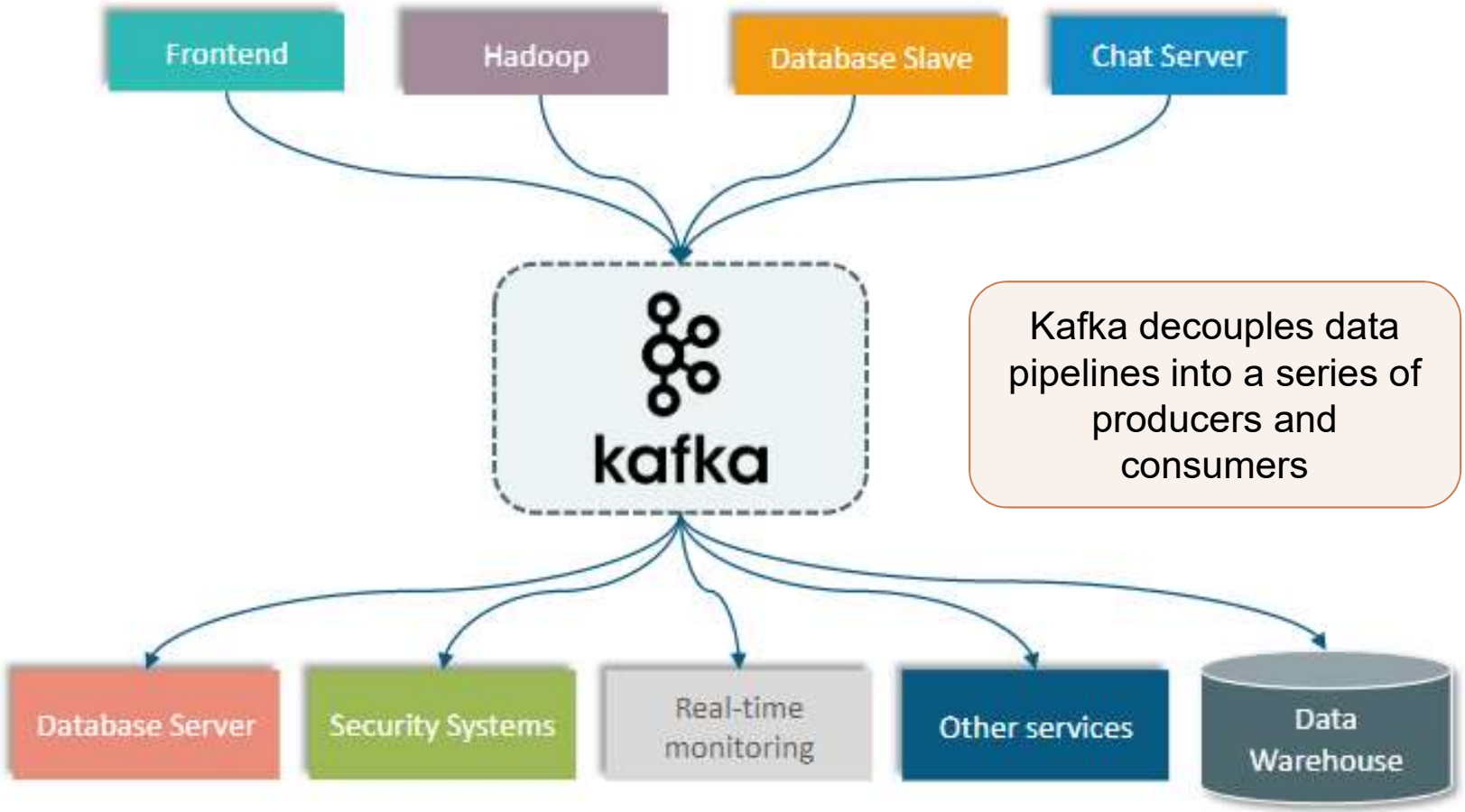
Publish/Subscribe systems like Kafka can help efficiently manage the complexity of the data pipelines



- Lets look at one such Pub/Sub messaging system called Apache Kafka.

Let’s have a look at Kafka’s solution

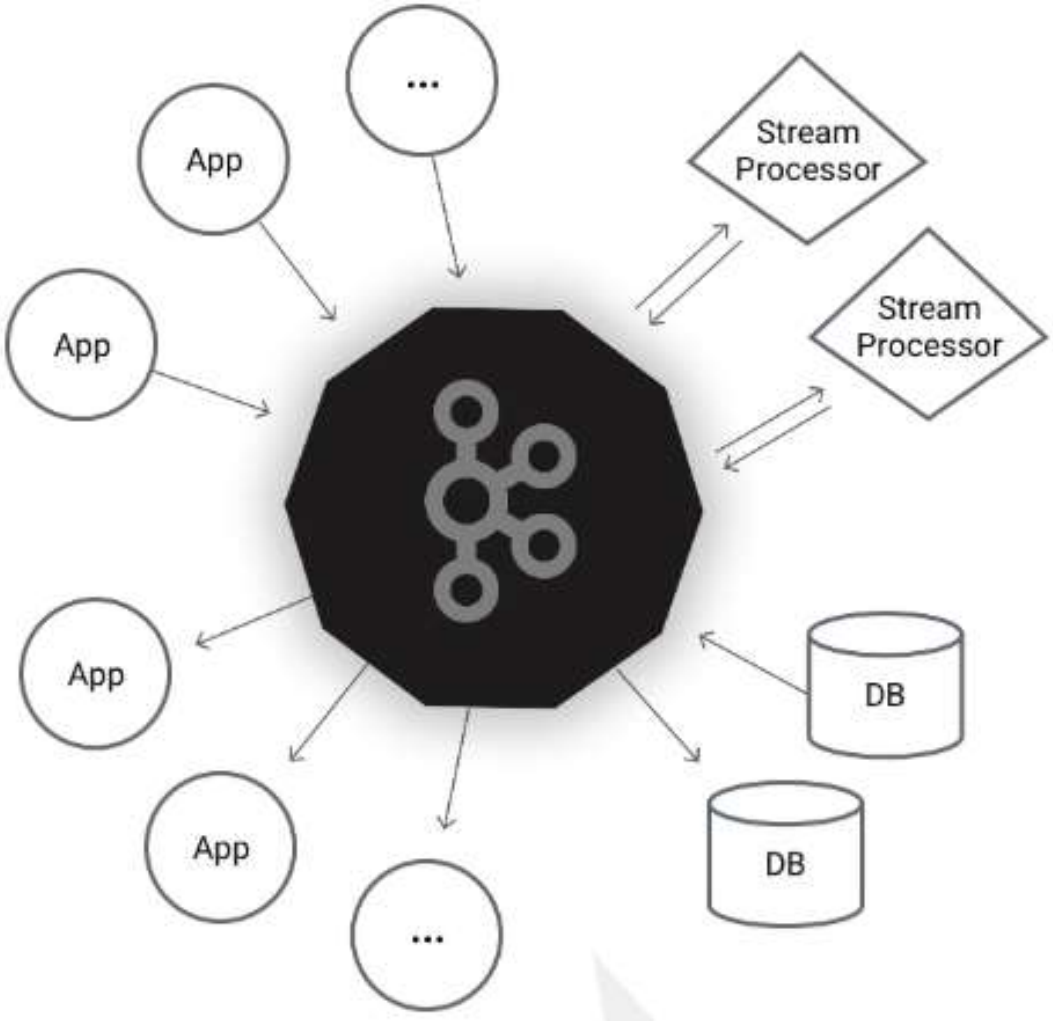
Kafka Decouples Data Pipelines



What is Kafka?

- Apache Kafka is a publish/subscribe messaging system designed to solve the problem of data pipeline complexity.
- Used for building real-time data pipelines and streaming apps.
- Kafka is described as “distributed commit log” or a “distributed streaming platform.”
- Kafka is designed as a scalable, durable, fast, fault-tolerant and high-throughput distributed messaging system and streaming platform.

What is Kafka? Contd..



Used to build streaming applications and real-time data pipelines

Horizontally scalable

Fault-tolerant and Durable

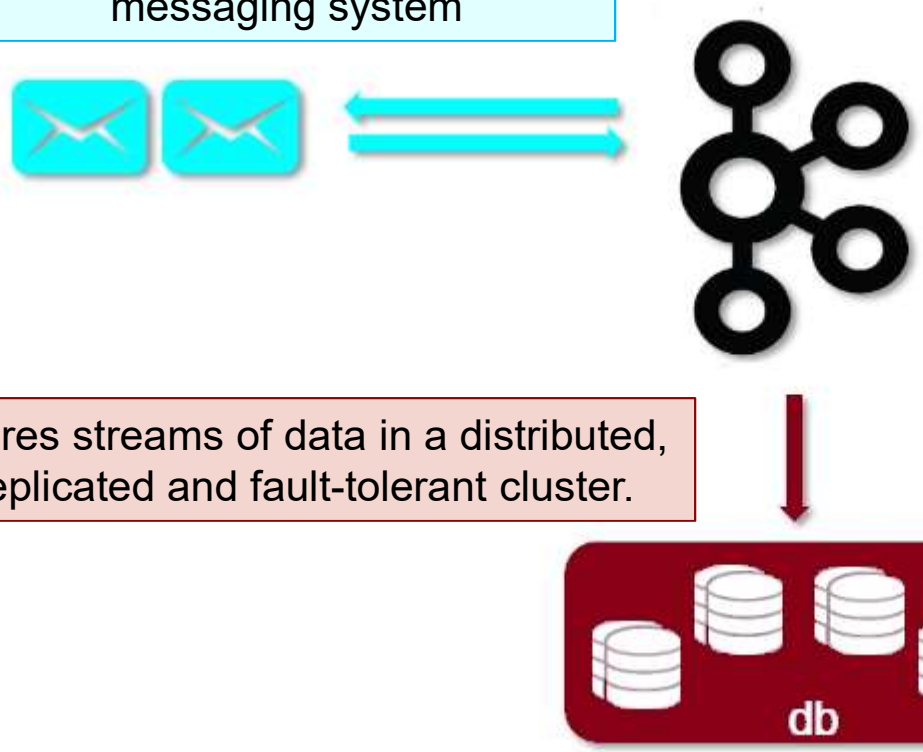
Wicked Fast

Runs in production in thousands of companies

Originated at LinkedIn and later on open-sourced as an Apache Project.

Apache Kafka Overview

Reads and writes data like a messaging system



Building real-time streaming applications that transform or react to the streams of data

Stores streams of data in a distributed, replicated and fault-tolerant cluster.

Apache Kafka - Core APIs

The **Producer API** allows an application to publish a stream of records to one or more Kafka topics.

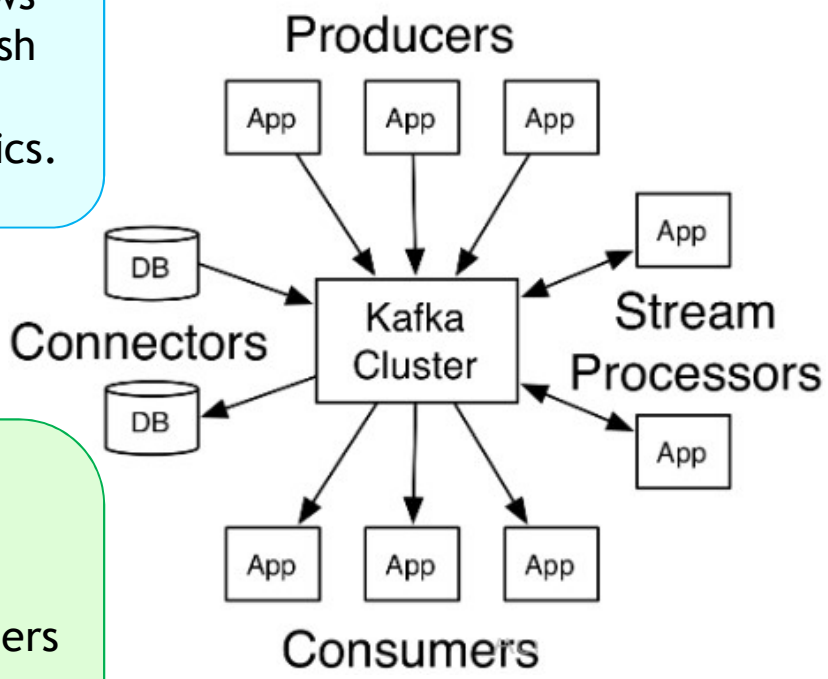
The **Consumer API** allows an application to subscribe to one or more topics and process the stream of records produced to them.

The **Streams API** allows an application to act as a stream processor, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.

The **Connector API** allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

Apache Kafka - Core APIs

The **Producer API** allows an application to publish a stream of records to one or more Kafka topics.

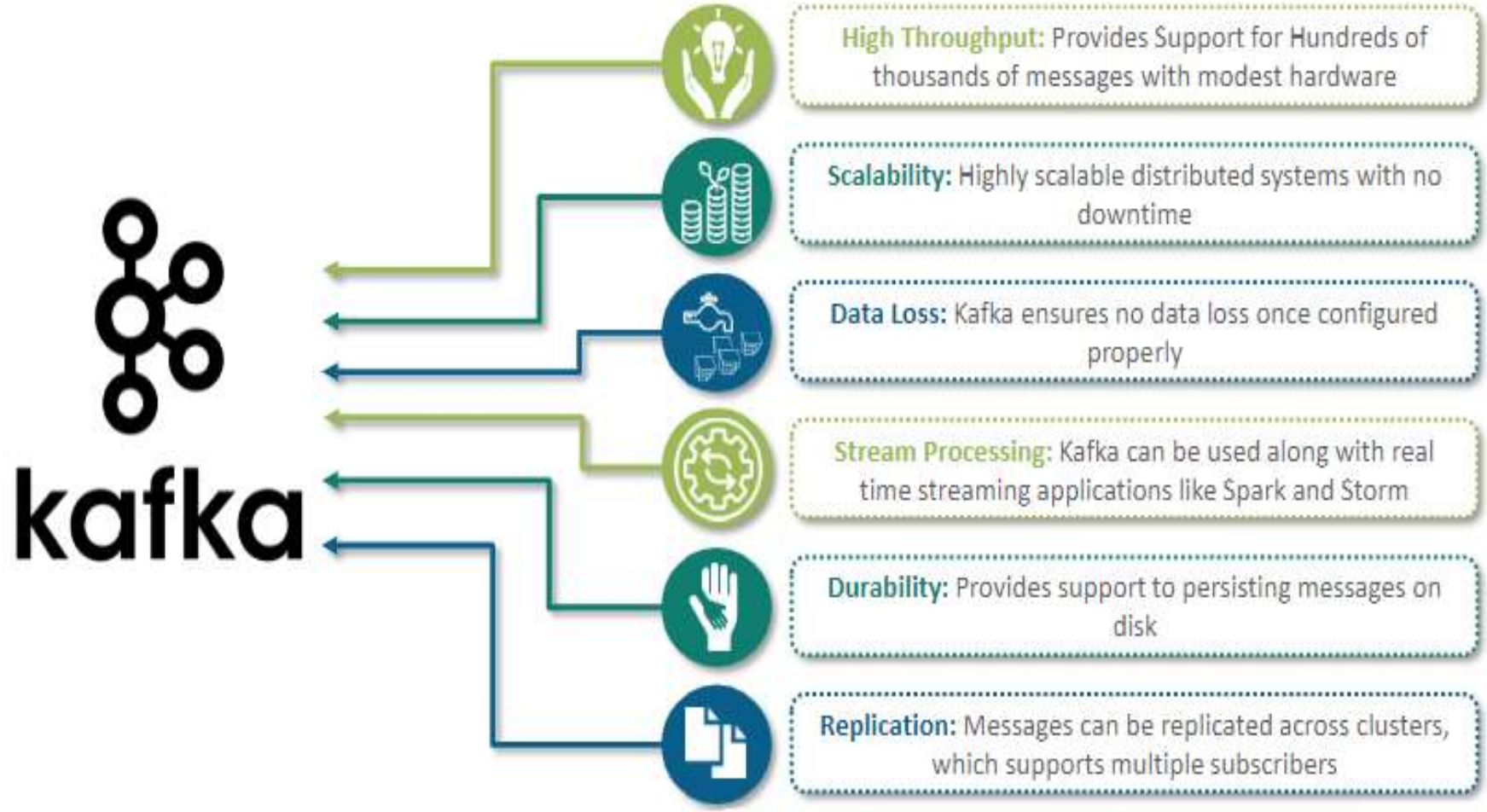


The **Streams API** allows an app to act as a stream processor, consuming an input stream from topics and producing an output stream to output topics, transforming the input streams to output streams.

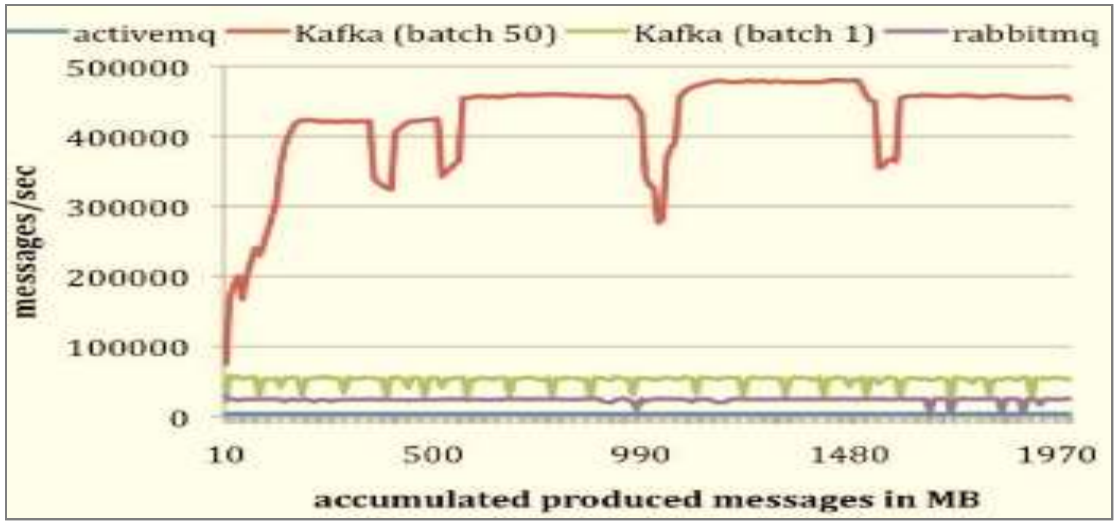
The **Connector API** allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems.

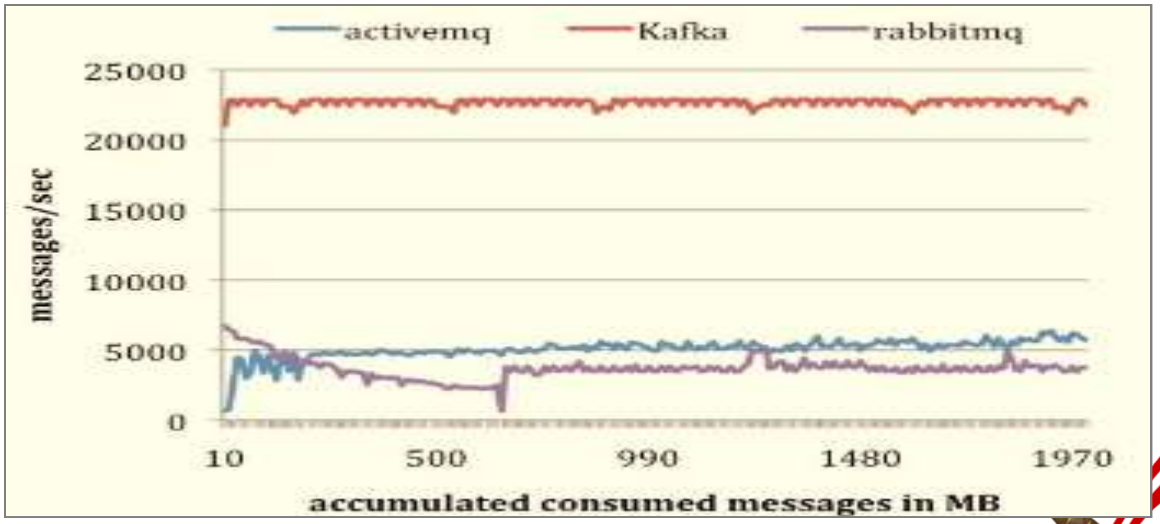
The **Consumer API** allows an application to subscribe to one or more topics and process the stream of records produced to them.

Apache Kafka - Features



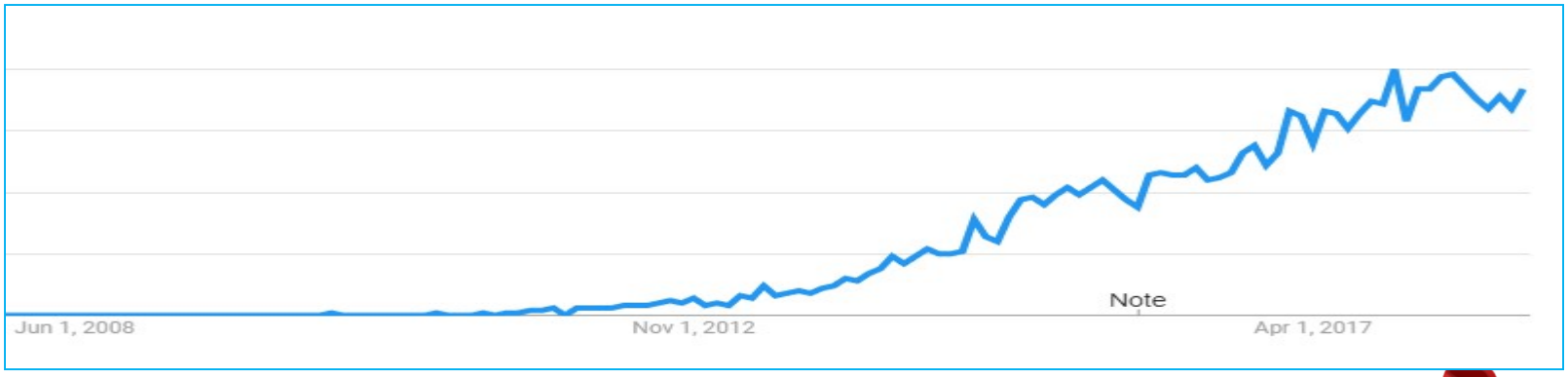
Kafka Vs Other Messaging Systems





Kafka Trends

- More than 35% of Fortune 500 companies use Kafka
- LinkedIn, Netflix, Microsoft process billions of messages per day with Kafka
- Kafka is used to collect real-time streaming data for big-data analysis and stream processing.



Kafka Basic Concepts

Terminology used in Kafka

Producer

A **producer** is an application that publishes messages to a topic.

Consumer

A **consumer** is an application that subscribes to a topic and consumes messages

Topic

A **topic** is a category or feed name to which records are published.

Partition

Topics are broken into ordered commit logs called **partitions**.

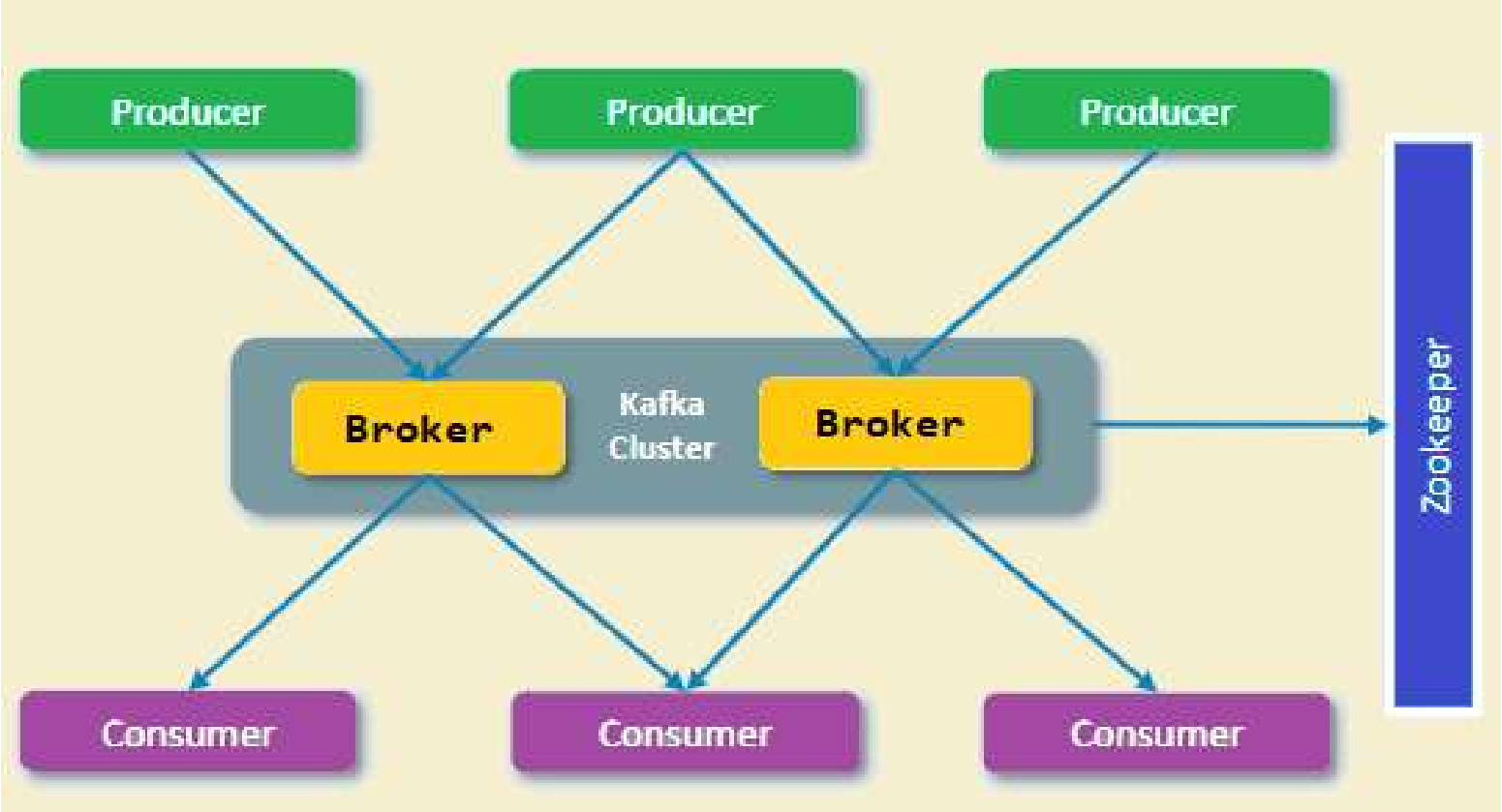
Broker

Each of the servers in a Kafka cluster is called as a **broker**.

Zookeeper

Kafka brokers depend on **Zookeeper** broker management and coordination services.

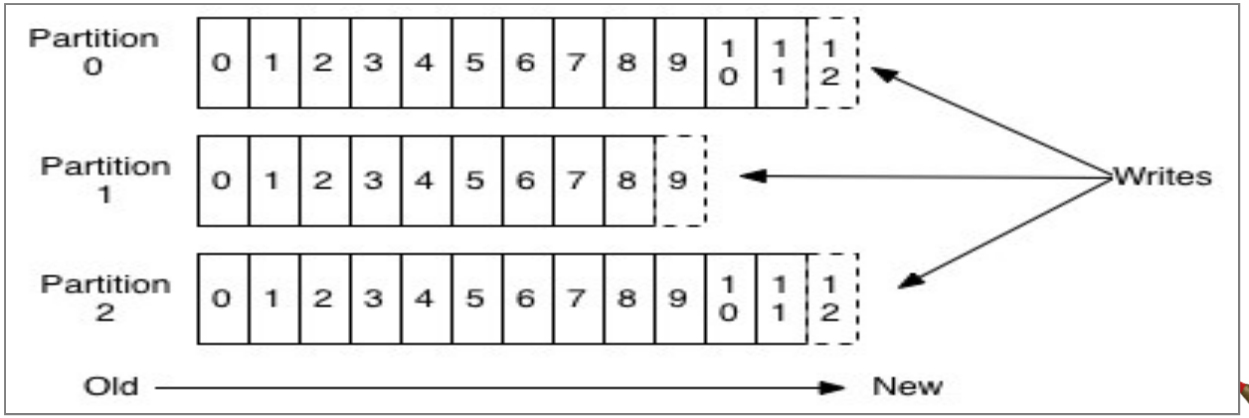
Kafka Cluster



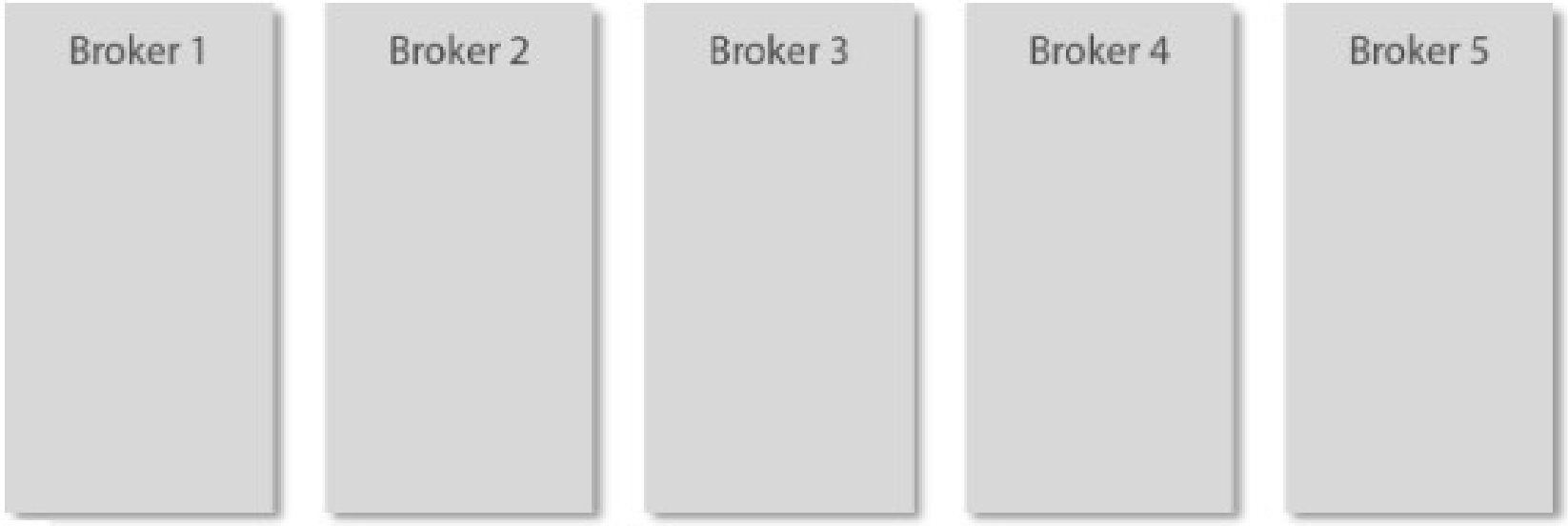
Kafka Topics & Partitions

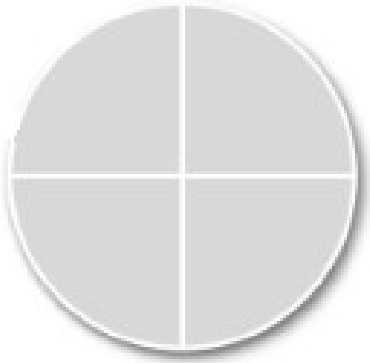
- A **topic** is a category or feed name to which records are published.
- Topics are broken up into ordered commit-logs called **partitions**.
- Data in a topic is **retained** for a configured period of time or size.
- Each message in a partition is assigned an **offset** (sequential-id)
- Writes to a partition are **sequential**.
- Reading messages can be either from the beginning or can rewind or skip to any point by specifying and offset value.

Anatomy of a Topic



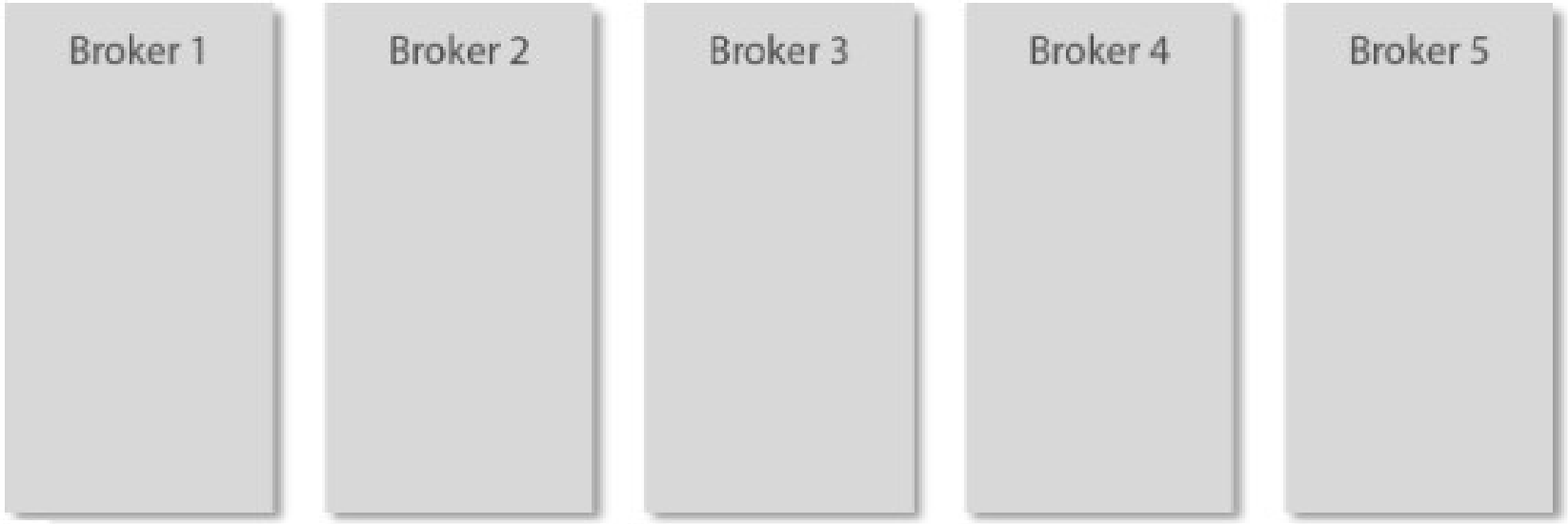
Kafka Topics, Partitions & Replicas

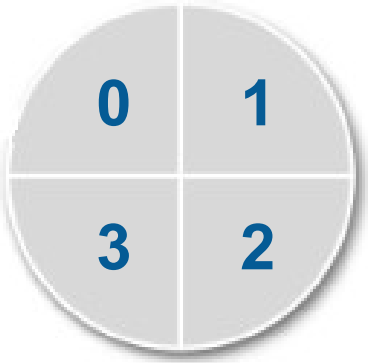




- **A topic is configured to use 4 partitions**

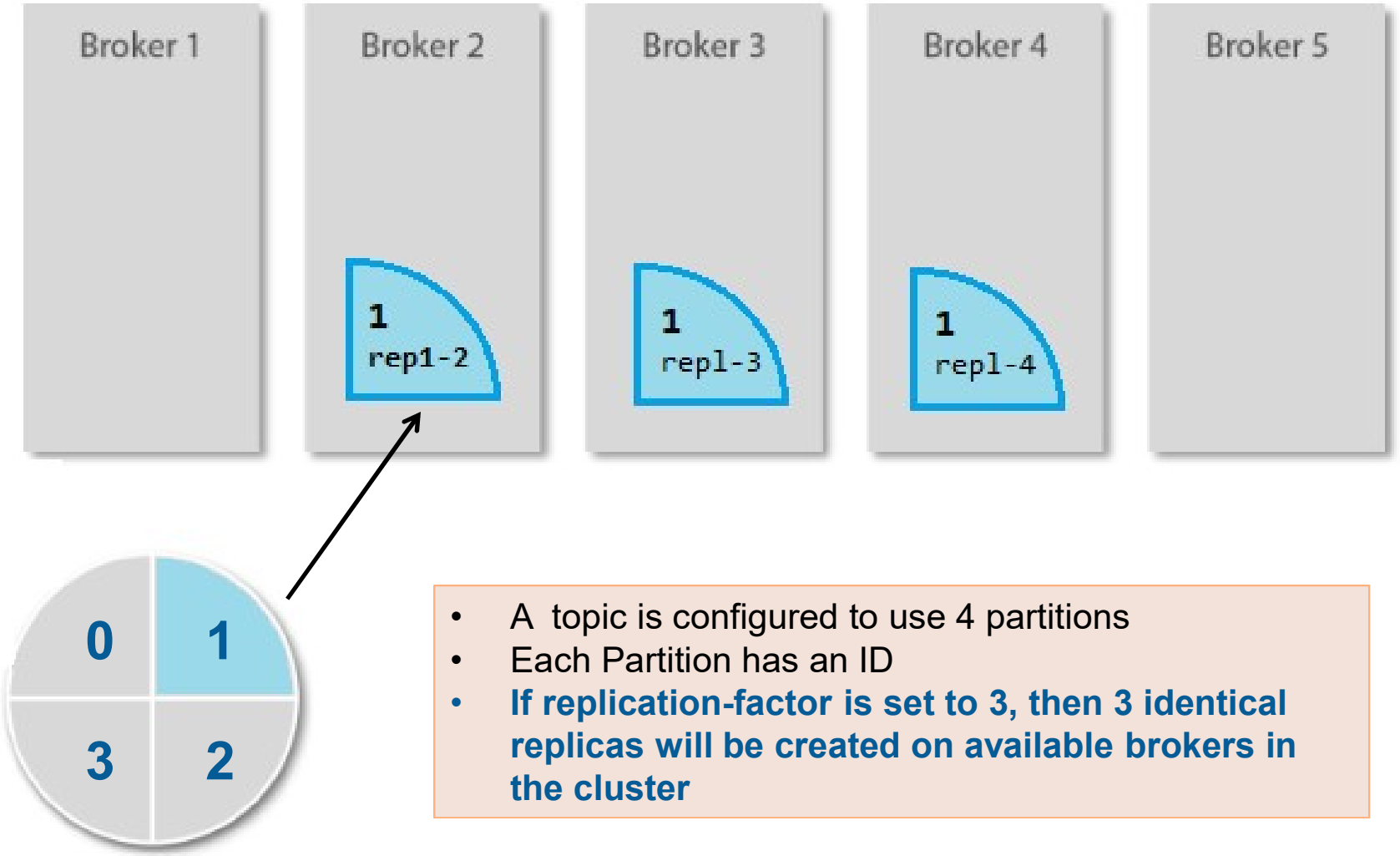
Kafka Topics, Partitions & Replicas





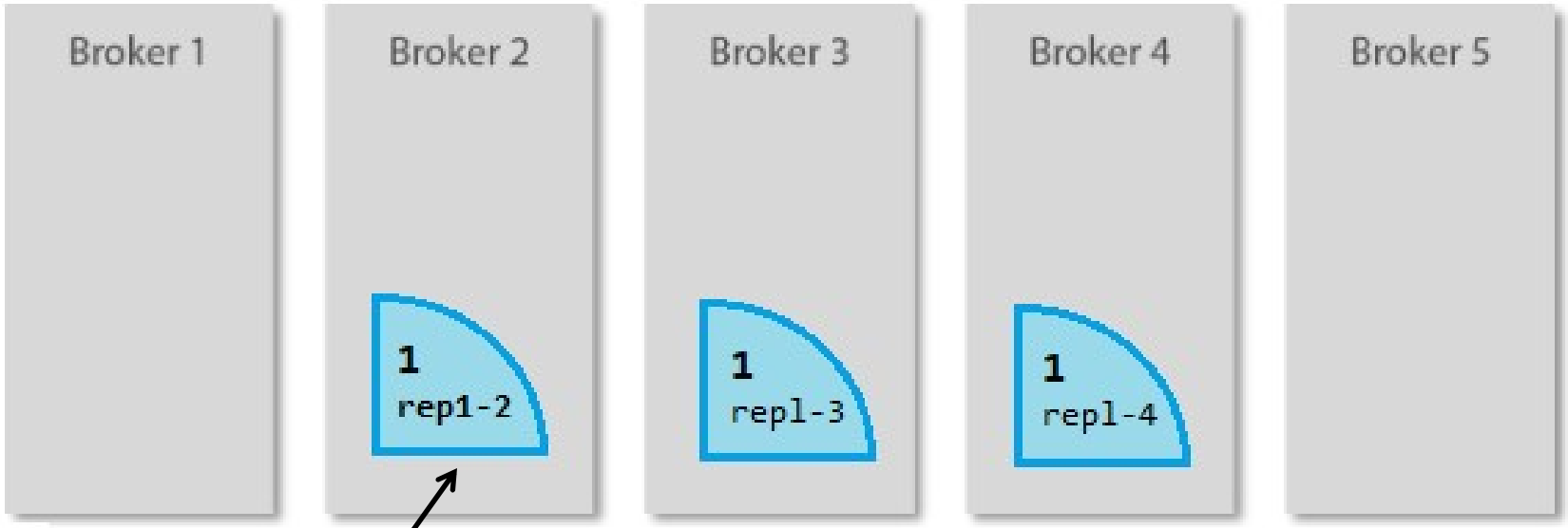
- A topic is configured to use 4 partitions
- **Each Partition has an ID**

Kafka Topics, Partitions & Replicas

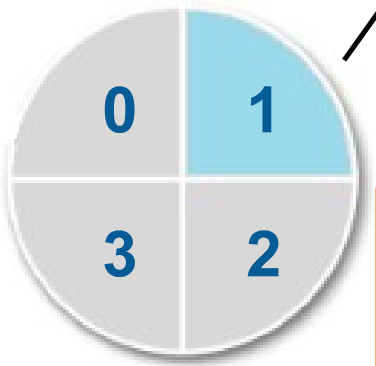


- A topic is configured to use 4 partitions
- Each Partition has an ID
- **If replication-factor is set to 3, then 3 identical replicas will be created on available brokers in the cluster**

Kafka Topics, Partitions & Replicas

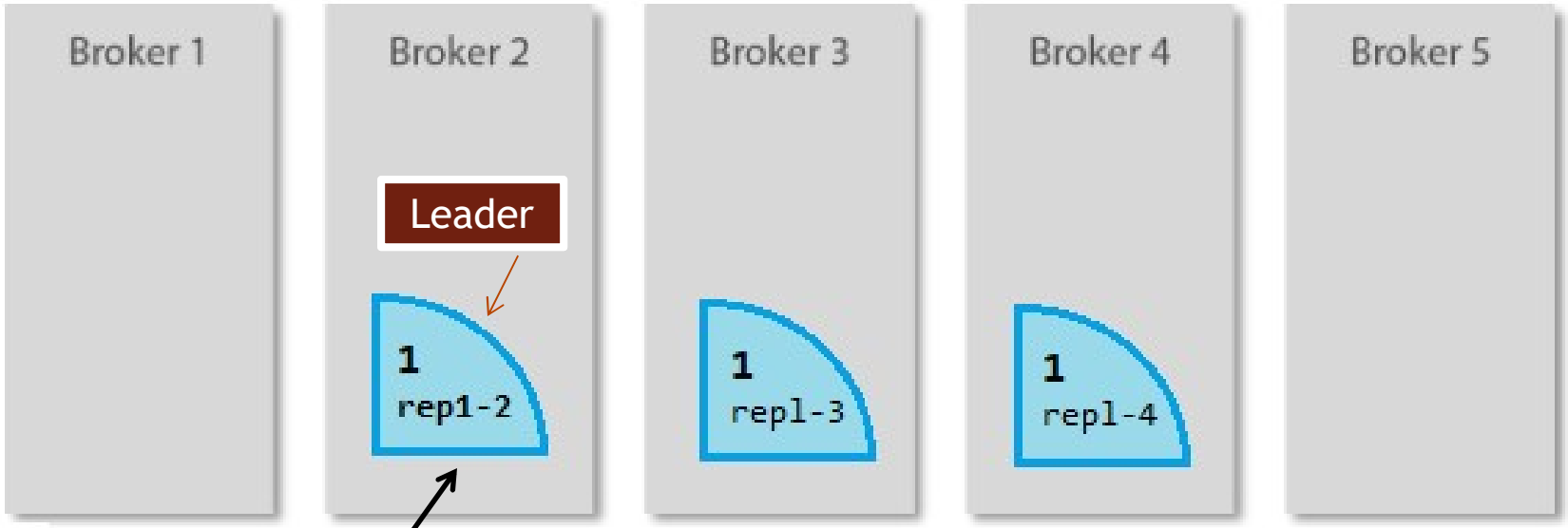


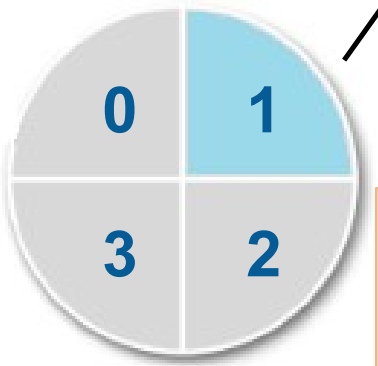
- **ID of the replica is same as broker-id**



- A topic is configured to use 4 partitions
- Each Partition has an ID
- If replication-factor is set to 3, then 3 identical replicas will be created on available brokers in the cluster

Kafka Topics, Partitions & Replicas





- ID of the replica is same as broker-id
- **For each partition Kafka broker elects one replica as the leader of the partition.**

- A topic is configured to use 4 partitions
- Each Partition has an ID
- If replication-factor is set to 3, then 3 identical replicas will be created on available brokers in the cluster

Kafka Components - Messages

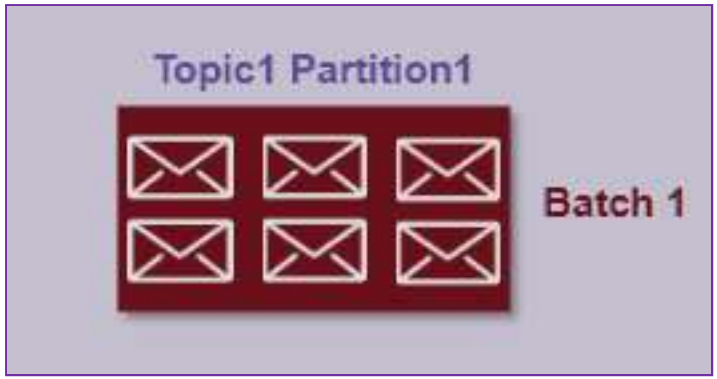
- A unit of data in Kafka is called as a *Message (or Record)*
- A message can be thought of as a database *record*
- To control messages that are to be written to a partition, a *key* is used.
- Message with the *same key* are written to the *same partition*.
- Each message typically consists of a key, a value, and a timestamp.

Kafka Components - Batches

- A batch is a collection of messages that are produced to the same topic and partition.

- Batches can be compressed to increase the data transfer rates and storage. We can use *snappy*, *gzip*, or *lz4* compression types

- Larger batches have more messages that are handled in per unit of time where as individual messages take longer in terms of propagation time.



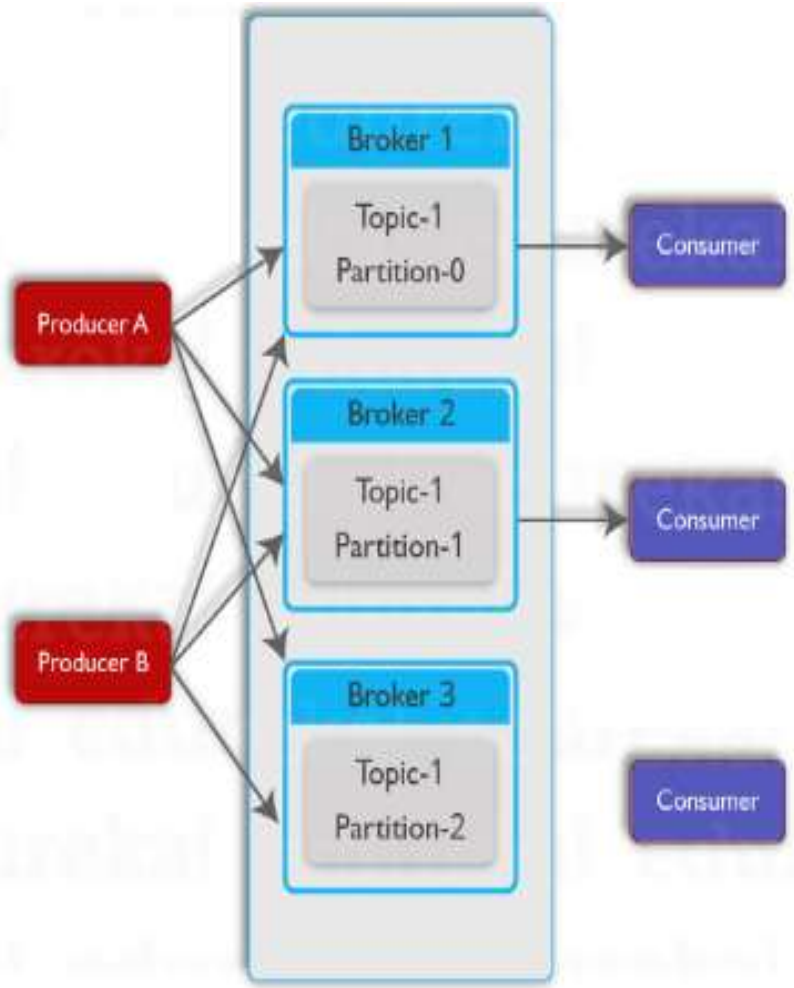
Kafka Components - Broker

Kafka cluster comprises of one or more servers. Each server in a cluster is called as a broker.

A broker typically handles 100s of MBs of writes from producers and reads from consumers.

Retains the published messages for a preconfigured time-period irrespective of whether the messages are consumed or not.

After the expiry of the message retention period (default 168 hrs) the messages are discarded.



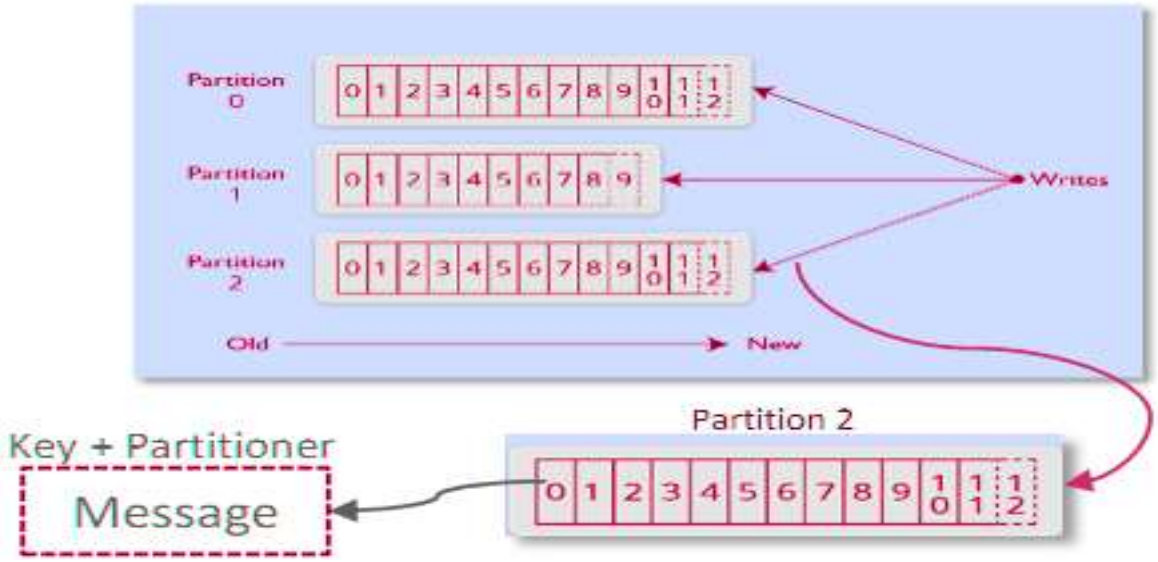
Kafka Components - Producer

1. Kafka **Producer** writes new messages to a **specified topic**.

2. Producer does not care what partition a specified message is written to. Messages are balanced over partitions of a topic based on the value of the key and the partitioner used.

3. Directing messages to a topic is done using the **message key** and a **partitioner**. The partitioner generates the hash of the key and maps it to a partition.

4. Messages are published in the form of **key:value** pairs.



Kafka Components - Consumer

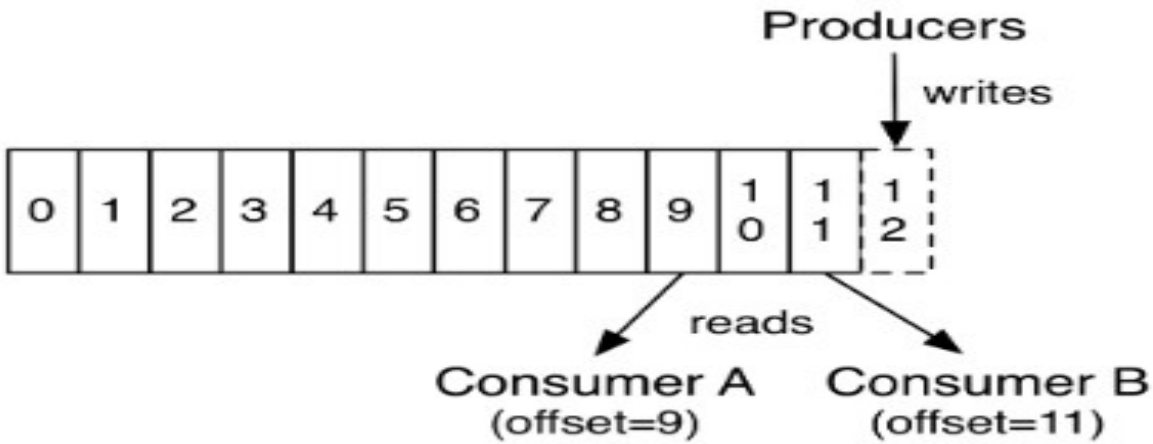
1. Consumers (subscribers or readers) consume/read messages

2. Consumer subscribes to topics and reads the messages sequentially

3. Consumer keeps track of the messages by keeping track of the message offset

4. Kafka adds an offset (which is a sequential integer-id) to each message produced.

5. Using the **offset** of the last consumed message, a consumer can **stop** and **restart** with out losing the current state.



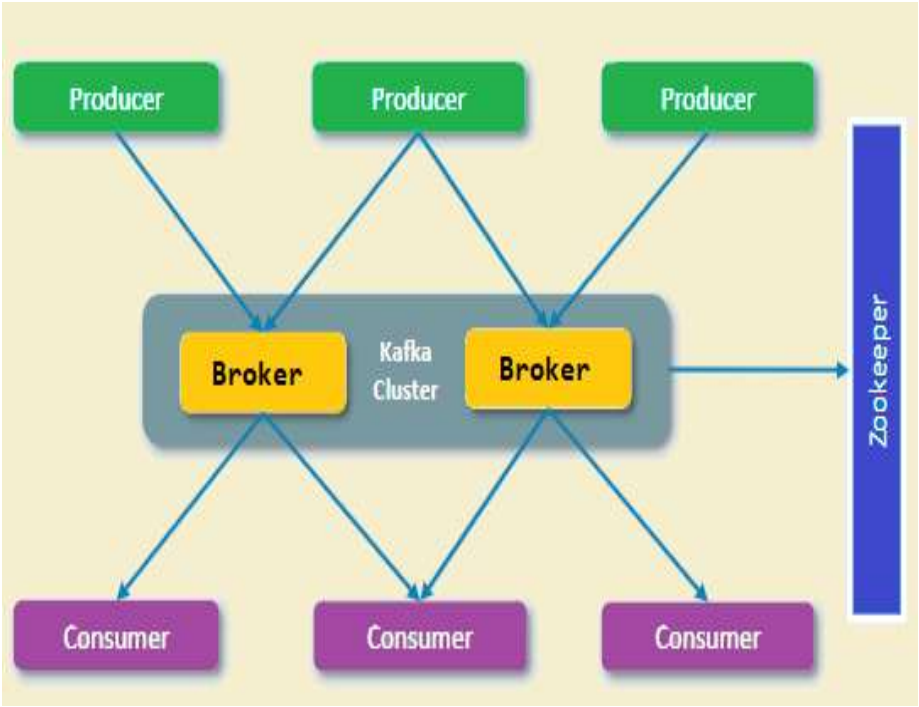
Kafka Components - Zookeeper

Zookeeper is used for managing and coordinating Kafka broker

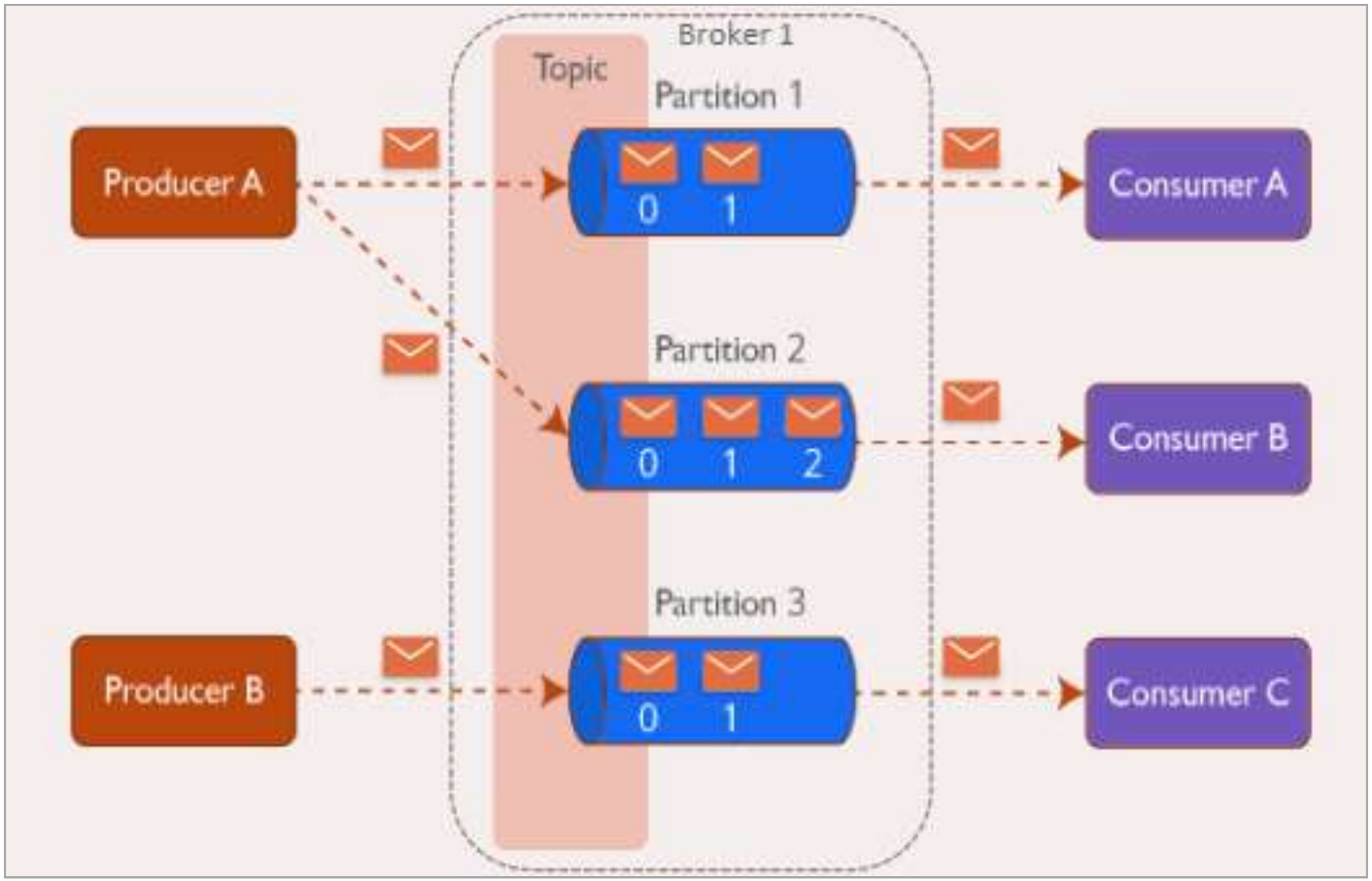
Kafka utilizes Zookeeper for storing metadata information about the brokers, topics and partitions.

Zookeeper service is mainly used for coordinating between brokers in the Kafka cluster.

Kafka cluster is connected to Zookeeper to get information about any node failures.



Kafka Architecture



Kafka Use Cases

Kafka Use Cases - Activity Tracking

Activity Tracking | Messaging | Metrics & Logging | Commit Log | Stream Processing

The original use case for Kafka, as it was designed at **LinkedIn**, is that of user activity tracking.

When users interact online with frontend applications, which generate messages regarding actions the user is taking. This can be passive information, such as page views and click tracking, or it can be more complex actions, such as information that a user adds to their profile.

The messages are published to one or more topics, which are then consumed by applications on the backend. These applications may be generating reports, feeding machine learning systems, updating search results, or performing other operations that are necessary to provide a rich user experience.

Kafka Use Cases - Messaging

Activity Tracking | **Messaging** | Metrics & Logging | Commit Log | Stream Processing

Kafka is also used for **messaging**, where applications need to send notifications (such as emails) to users. Those applications can produce messages without needing to be
concerned about formatting or how the messages will actually be sent. A single application can then read all the messages to be sent and handle them consistently,
including:

- Formatting the messages using a common look and feel
- Collecting multiple messages into a single notification to be sent
- Applying user's preferences for how they want to receive messages

Kafka Use Cases - Metrics and Logging

Activity Tracking | Messaging | **Metrics & Logging** | Commit Log | Stream Processing

- Kafka is also ideal for collecting application metrics and logs.
- Applications publish metrics to a topic on a regular basis and those metrics are consumed by systems for monitoring and alerting.
- Log messages can be published in the same way and routed to dedicated log search systems like ElasticSearch or security analysis applications.

Kafka Use Cases - Metrics and Logging

Activity Tracking | Messaging | Metrics & Logging | **Commit Log** | Stream Processing

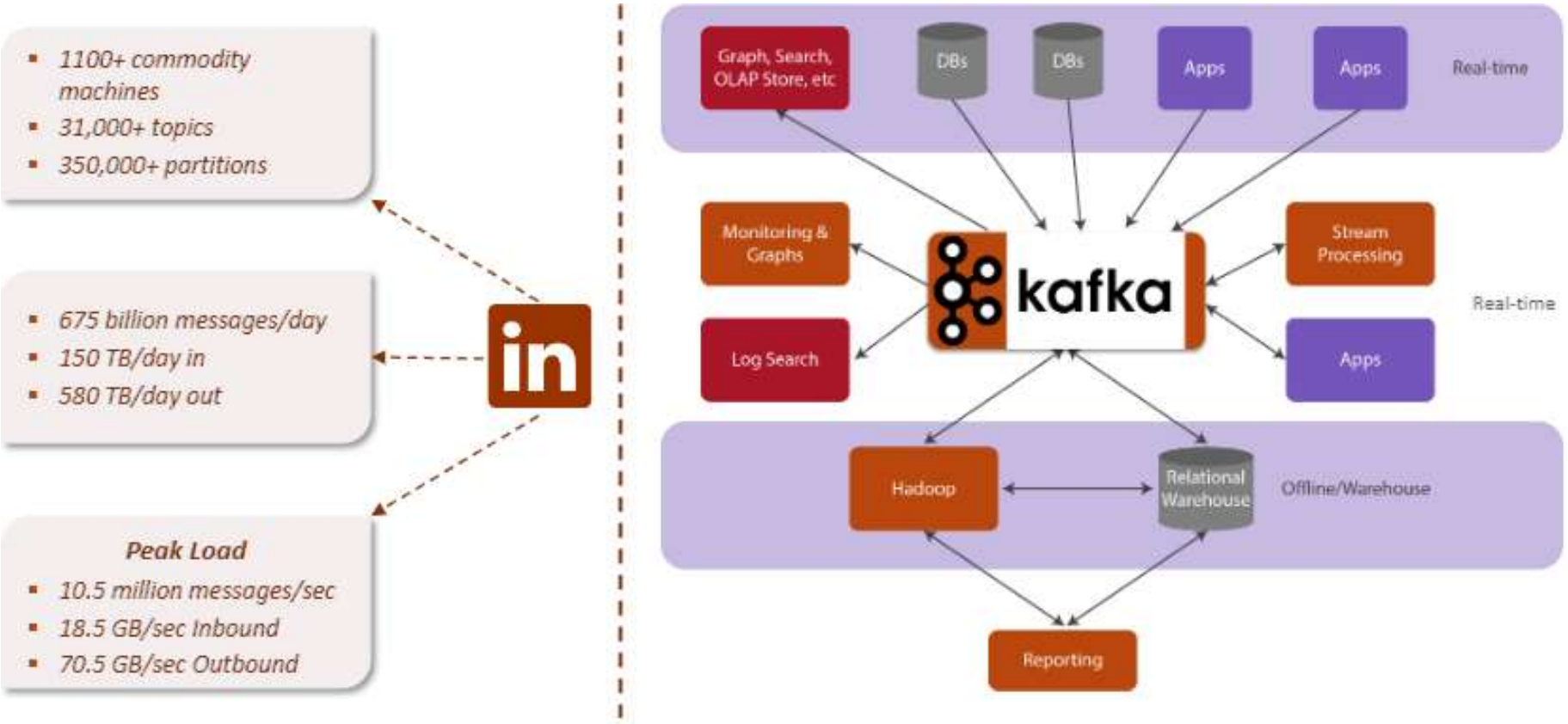
- Database changes can be published to Kafka and applications can easily monitor this stream to receive live updates as they happen.
- This change-log stream can also be used for replicating database updates to a remote system, or for consolidating changes from multiple applications into a single database view.
- Durable retention is useful here for providing a buffer for the change-log, meaning it can be replayed in the event of a failure of the consuming applications.
- Log-compacted topics can be used to provide longer retention by only retaining a single change per key.

Kafka Use Cases - Stream Processing

Activity Tracking | Messaging | Metrics & Logging | Commit Log | **Stream Processing**

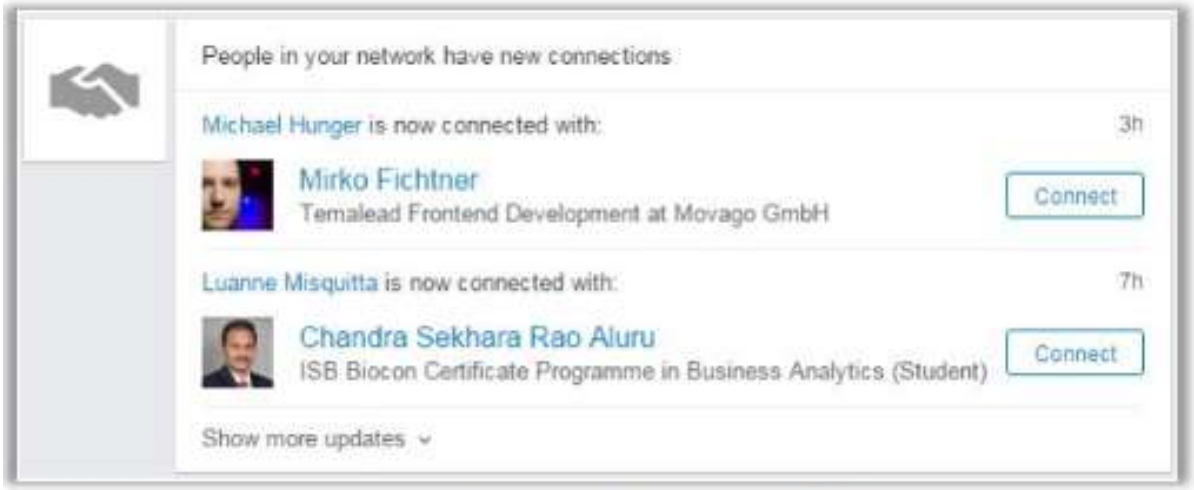
- Stream Processing term is typically used to refer to applications that provide similar functionality to map/reduce processing in Hadoop.
- Stream-processing Stream processing operates on data in real time, as quickly as messages are produced.
  - write small applications to operate on Kafka messages
  - performing tasks such as counting metrics
  - partitioning messages for efficient processing by other applications, or transforming messages using data from multiple sources.

How Kafka is used @ LinkedIn

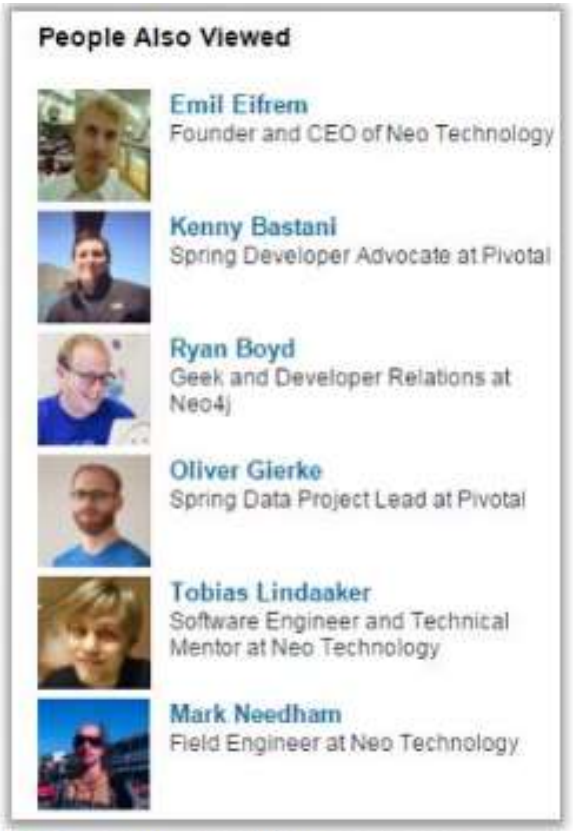


Stream centric data architecture built around Kafka

How Kafka is used @ LinkedIn

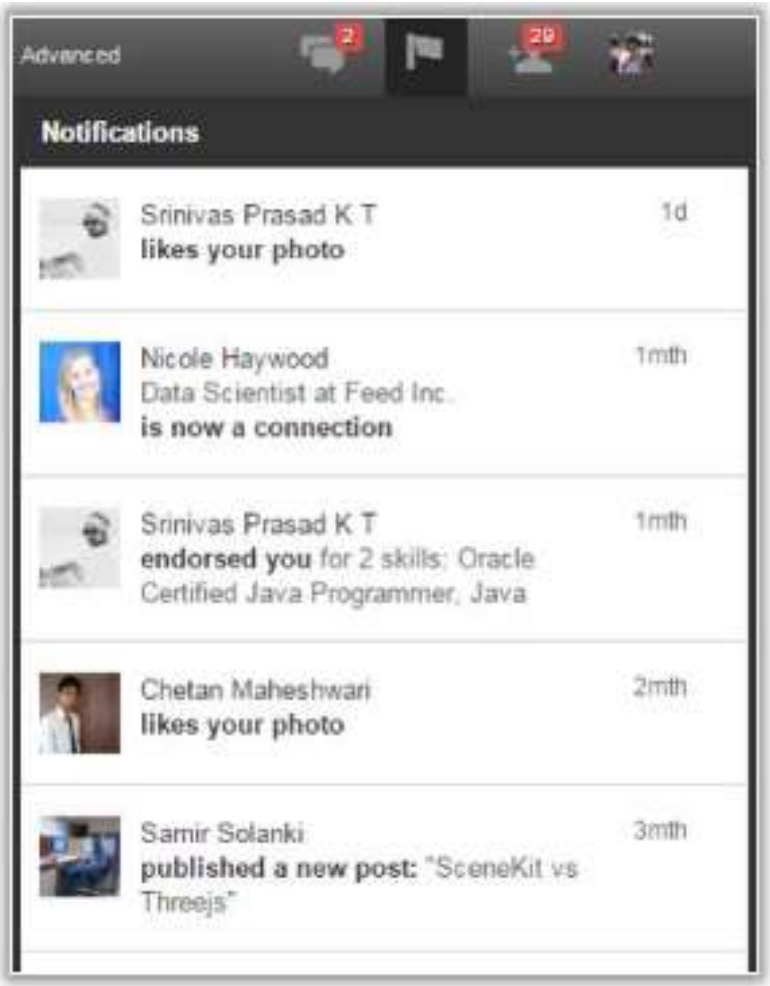


LinkedIn Newsfeed is powered by Kafka



LinkedIn recommendations are powered by Kafka

How Kafka is used @ LinkedIn



LinkedIn notifications are powered by Kafka

In addition to this, Kafka is used for many more applications at LinkedIn and elsewhere, for purposes like Streaming Data Analysis, Log Monitoring, Improving Search Efficiency, Performance Metrics etc.

Getting started with Kafka

Getting started with Kafka

Prior to installing either Zookeeper or Kafka, you will need a Java environment set up and functioning. This should be a Java 8 version, and can be the version provided by your OS or one directly downloaded from java.com. Though Zookeeper and Kafka will work with a runtime edition of Java, it may be more convenient when developing tools and applications to have the full JDK

Components:

Kafka

Installing Zookeeper

Zookeeper Installation:

Step 1 : Download ZooKeeper .tar file from : http://www-eu.apache.org/dist/zookeeper/stable

Installing Zookeeper

Step 2

- Extract the .tar file and move the extracted directory to `/usr/lib` directory

Step 3:

- Go into the `conf` directory present inside the `zookeeper` directory.
  - `cd /usr/lib/zookeeper-3.4.10/conf`
- Duplicate the contents of zoo_sample.cfg to zoo.cfg
  - `sudo cp zoo_sample.cfg zoo.cfg`

Step 4:

- Open the `zoo.cfg` file in a text editor
  - `sudo gedit zoo.cfg`

Installing Zookeeper

Step 5

- Make the following changes to the zoo.cfg file

a. clientPort=2181 b. tickTime=2000

c. initLimit=5 d. syncLimit=2

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=5
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=2
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=/tmp/zookeeper
# the port at which the clients will connect
clientPort=2181
# the maximum number of client connections.
# increase this if you need to handle more clients
#maxClientCnxns=60
#
# Be sure to read the maintenance section of the
# administrator guide before turning on autopurge.
#
# http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_maintenance
#
# The number of snapshots to retain in dataDir
#autopurge.snapRetainCount=3
# Purge task interval in hours
```

Installing Zookeeper

Step 6

- Start ZooKeeper server
  - `$bin/zkServer.sh start`

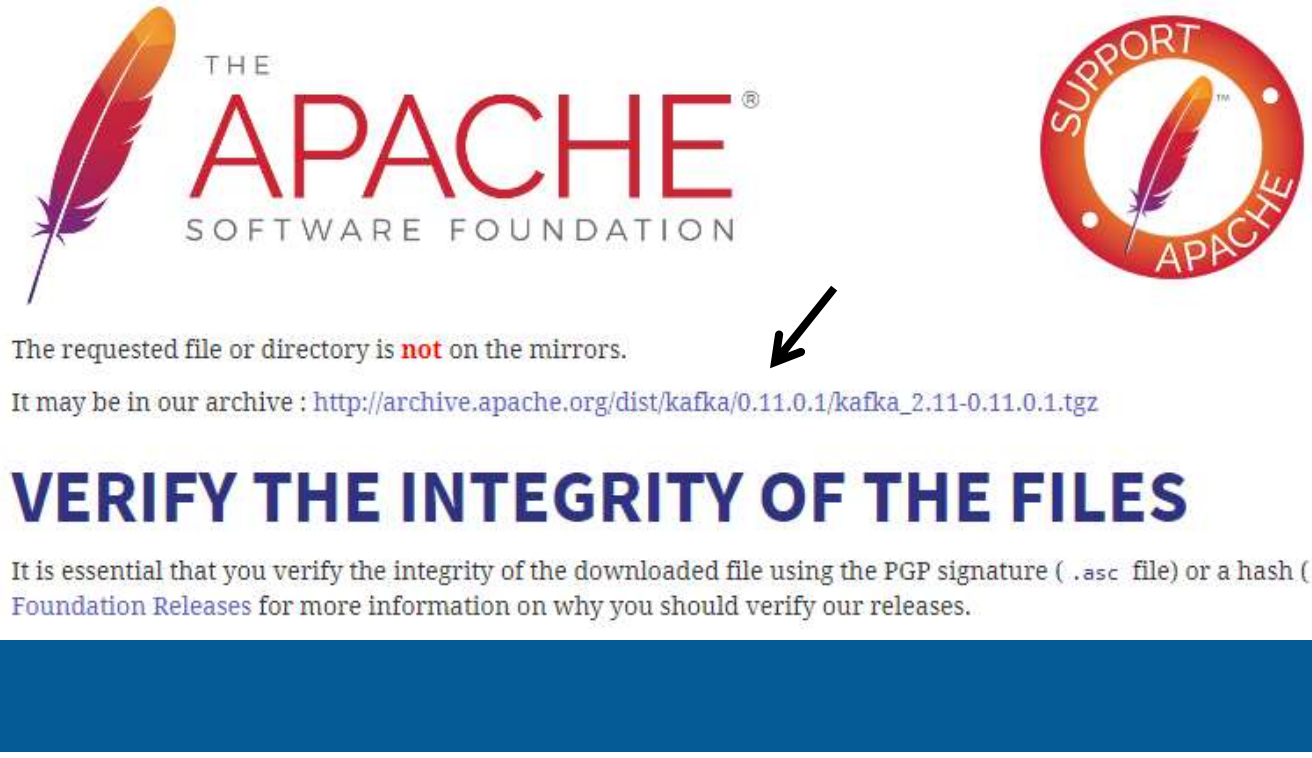
```
JMX enabled by default
Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

Installing Kafka

Kafka Installation:

Step 1

- Download Kafka from the following link:
  http://archive.apache.org/dist/kafka/0.11.0.1/kafka_2.11-0.11.0.1.tgz



Installing Kafka

Step 2

- Extract the .tar file
- Move extracted directory to `/usr/lib` directory
  - `$sudo mv Downloads/kafka_2.11-0.11.0.1 /usr/lib`
- Go inside the Kafka folder
  - `$cd kafka_2.11-0.11.0.1`
- Start the Kafka server
  - `bin/kafka-server-start.sh config/server.properties`

Different types of Kafka Clusters

Understanding Kafka Cluster

Kafka brokers are designed to operate as part of a **cluster**.

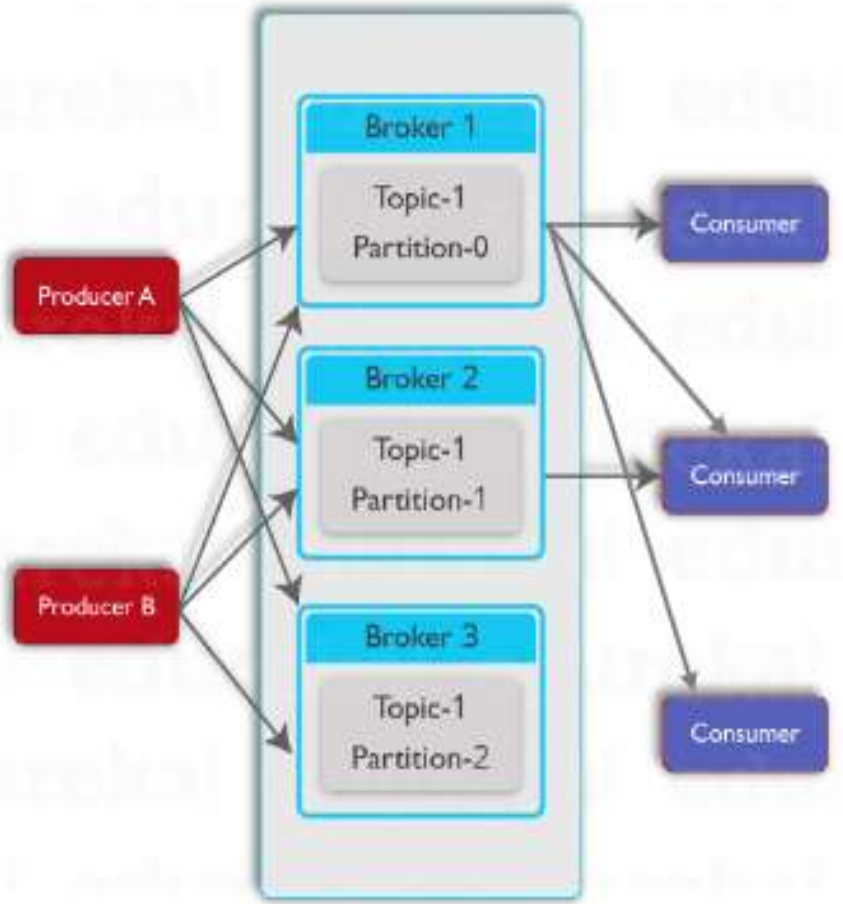
One broker in the cluster operates as **a cluster controller.**

Controller is responsible for **administrative operations**.

- Assign partitions to brokers
- Monitoring for broker failures

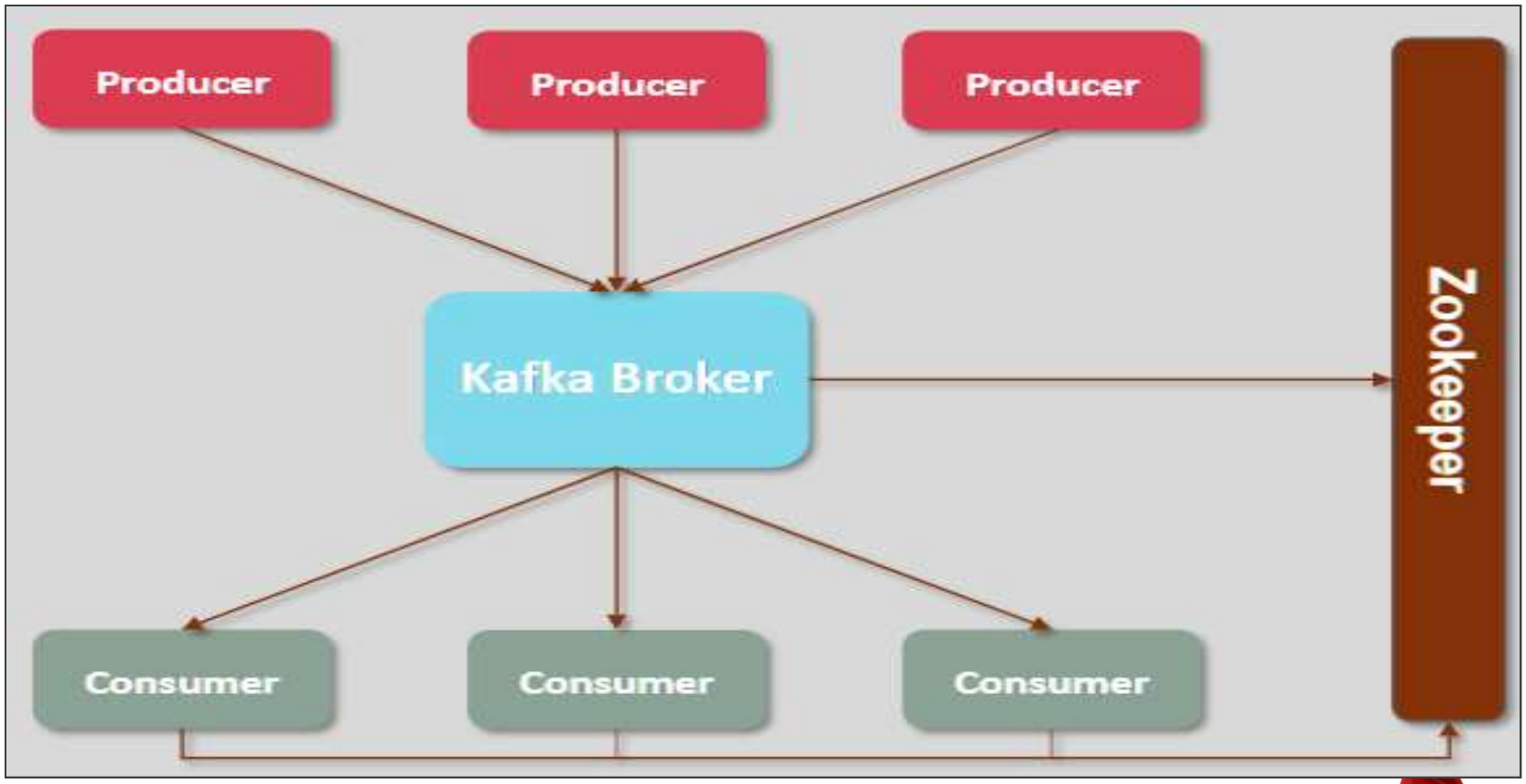
A particular partition is owned by a broker, and that broker is called the **leader of the partition**.

All producers and consumers operating on that partition **must connect to the leader**.



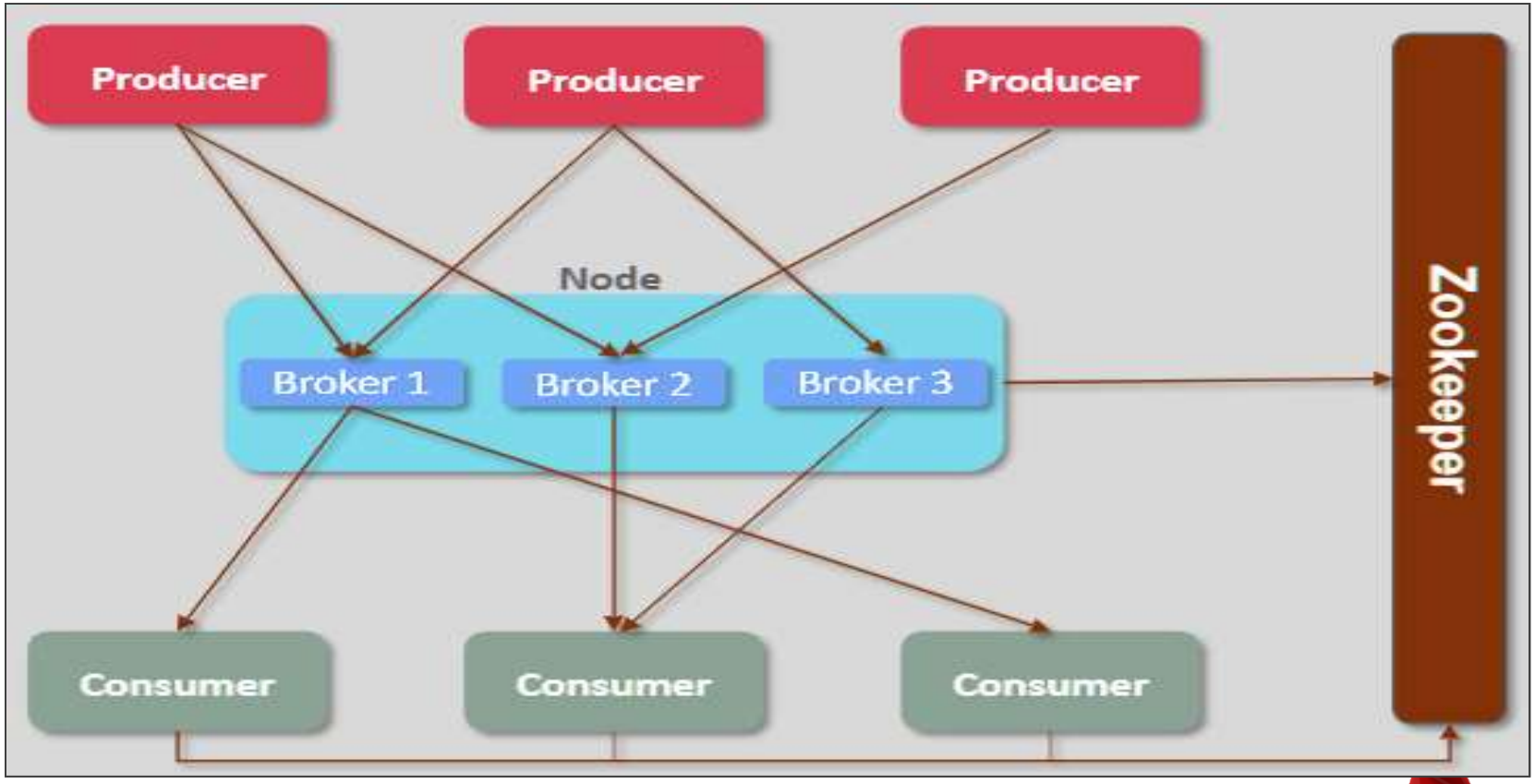
Types of Kafka Clusters

Single Node Single Broker Cluster



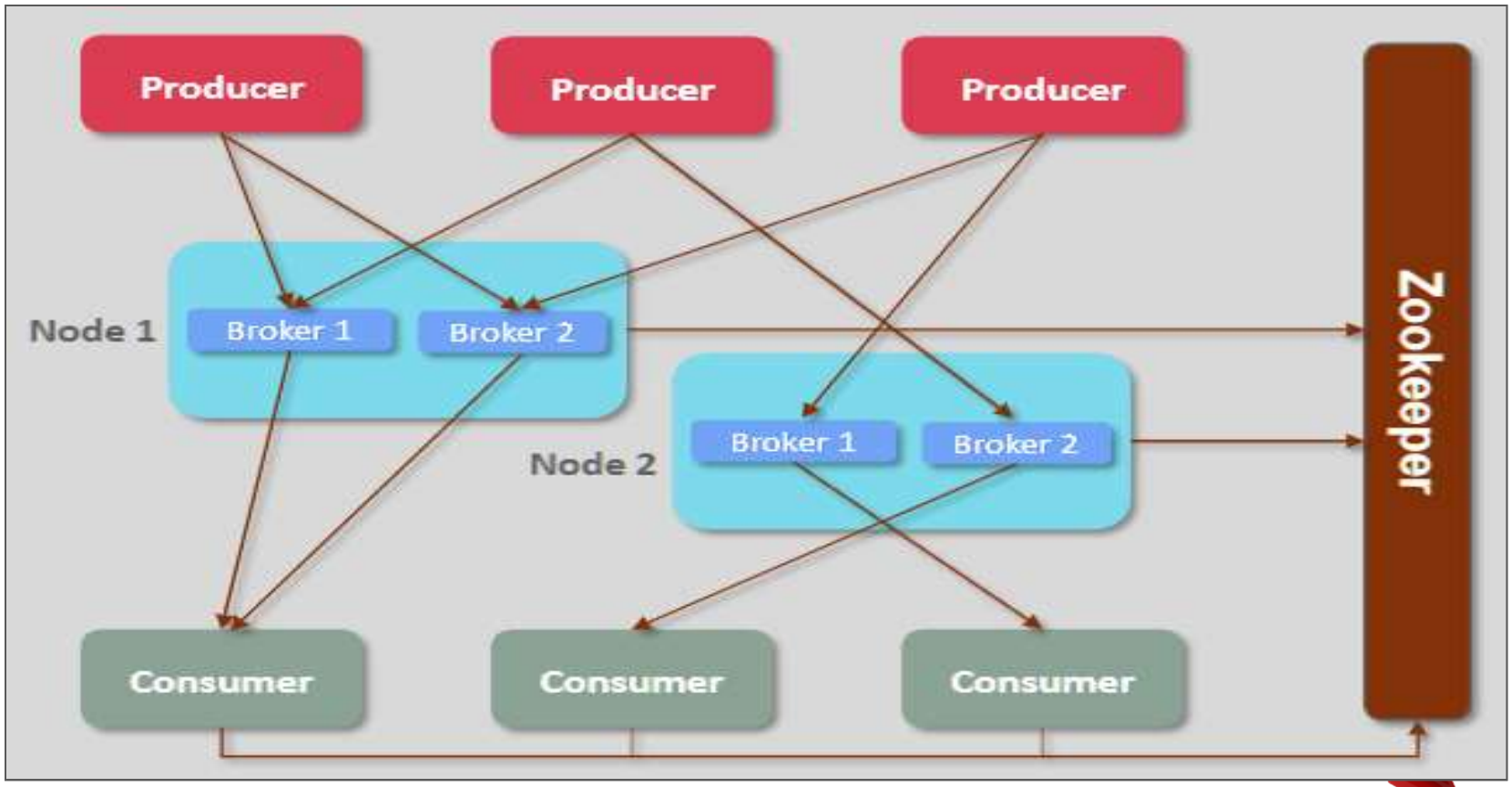
Types of Kafka Clusters

Single Node Multiple Broker Cluster



Types of Kafka Clusters

Multiple Node Multiple Broker Cluster



Single-Node Single-Broker Cluster Configuration

Single-Node Single-Broker Cluster

- Step 1: Start the Zookeeper Server

```
$ bin/zkServer.sh start
```

```
JMX enabled by default
Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

- Step 2: Start the Kafka Server from Kafka installation location.

```
$ bin/kafka-server-start.sh config/server.properties
```

Single-Node Single-Broker Cluster

- Step 3: Create a topic with the name "`mytopic`"

```
$ bin/kafka-topics.sh –create –zookeeper localhost:2181
  --partitions 1 --replication-factor 1 –topic mytopic
```

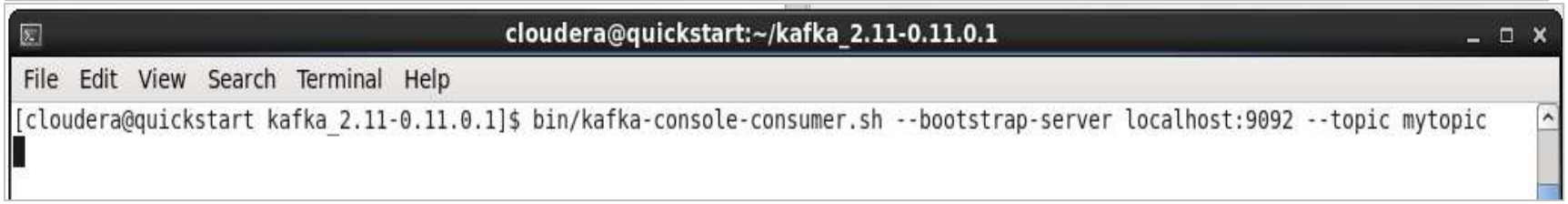
- Step 4: Start a Console Producer

```
$ bin/kafka-console-producer.sh –broker-list
      localhost:9092 –topic mytopic
```

Single-Node Single-Broker Cluster

- Step 5: Start a Console Consumer

```
$ bin/kafka-console-consumer.sh –bootstrap-server
          localhost:9092 –topic mytopic
```



- Step 6: As we keep publishing messages from the producer into the topic, the consumer that subscribed to that topic receives those messages


Thank you!