Introduction to Event Management

Event Management & Kafka

Introduction to Event Management

Event Management is a crucial process in today's fast-paced digital world. It involves capturing, processing, and analyzing events to gain valuable insights and drive business decisions. In this card, we'll explore the fundamentals of Event Management and its importance in optimizing operations and enhancing customer experiences.

Benefits of Event Sourcing

- Scalability
- Auditability
- Flexibility
- Real-time Responsiveness
- Data Integration
- Enhanced Monitoring

Event Management

- **Apache Kafka Apache**
- **Pulsar**
- **RabbitMQ**
- **Amazon Simple Queue Service (SQS)**
- **Google Cloud Pub/Sub**
- **Microsoft Azure Service Bus**
- **IBM MQ**

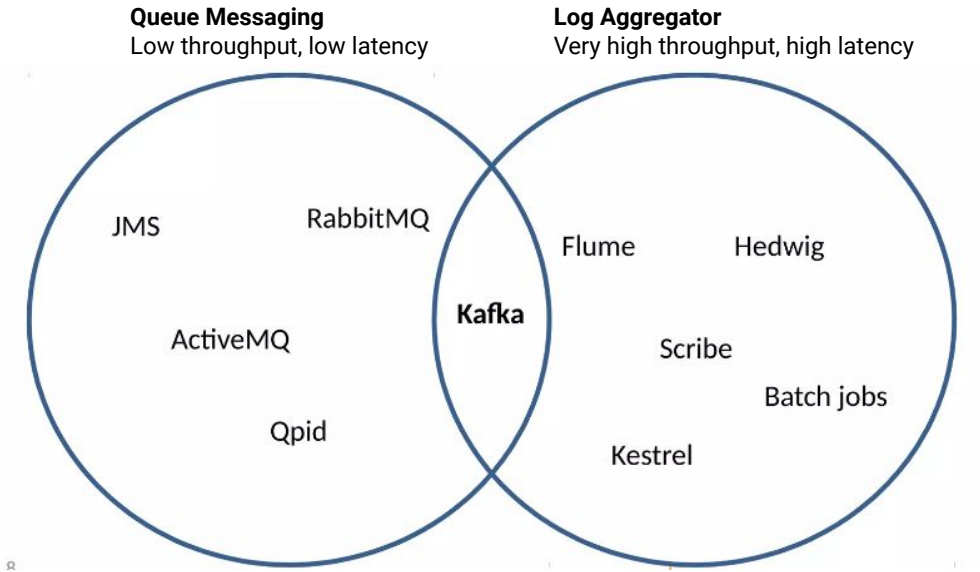
Event Management

Service	Description	Use Cases
Apache Kafka	Distributed streaming platform that focuses on high-throughput, fault-tolerant messaging, and real-time data processing.	Real-time data ingestion and processing, event streaming, log aggregation, messaging systems, and more.
RabbitMQ	Open-source message broker that implements the Advanced Message Queuing Protocol (AMQP) and provides reliable message delivery.	Enterprise messaging, task and workflow management, routing and filtering, and more.
Amazon Simple Queue Service (SQS)	Fully managed message queuing service that enables decoupling and scaling of distributed systems on the AWS platform.	Cloud-based messaging, decoupling of microservices, event-driven compute, and more.
Google Cloud Pub/Sub	Globally distributed messaging service that allows asynchronous communication between independently developed applications.	Real-time messaging, event-driven architectures, decoupling of microservices, and more.

Event Management

- **Apache Kafka**: Distributed streaming platform that focuses on high-throughput, fault-tolerant messaging, and real-time data processing.
- **Apache Pulsar**: Pub-sub messaging system that offers scalability, durability, and low-latency messaging, with support for both streaming and queuing models.
- **RabbitMQ**: Open-source message broker that implements the Advanced Message Queuing Protocol (AMQP) and provides reliable message delivery.
- **Amazon Simple Queue Service (SQS)**: Fully managed message queuing service that enables decoupling and scaling of distributed systems on the AWS platform.
- **Google Cloud Pub/Sub**: Globally distributed messaging service that allows asynchronous communication between independently developed applications.
- **Microsoft Azure Service Bus**: Fully managed enterprise messaging service that provides asynchronous communication and decoupling for cloud-based applications.
- **IBM MQ**: Messaging middleware that facilitates communication and integration between applications, systems, and services across various platforms.

Message Queue Management Services

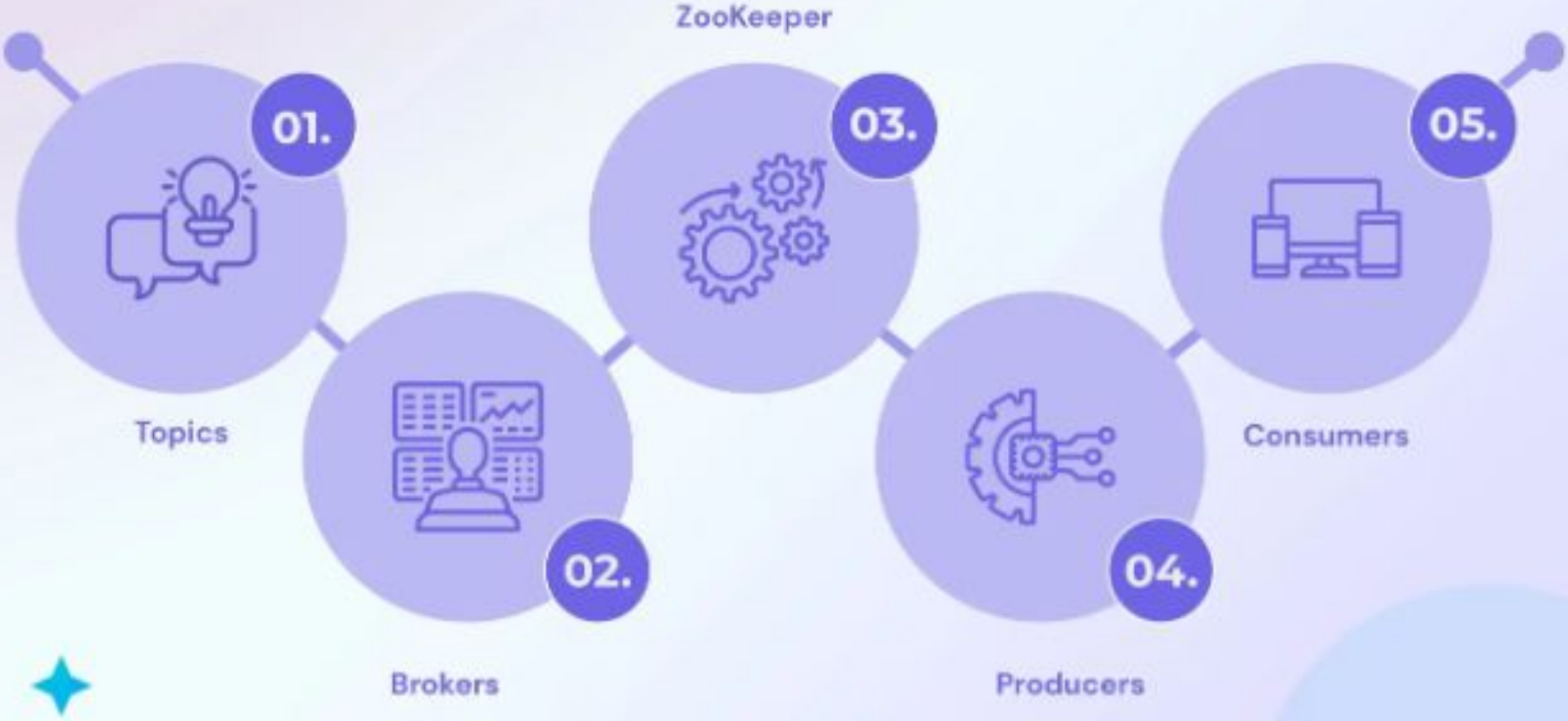


What is Kafka?

- Kafka is an open-source distributed event streaming platform
- It provides a highly scalable, fault-tolerant, and real-time data processing solution
- It allows organizations to efficiently capture, store, and process large volumes of streaming data, enabling seamless integration and real-time analytics

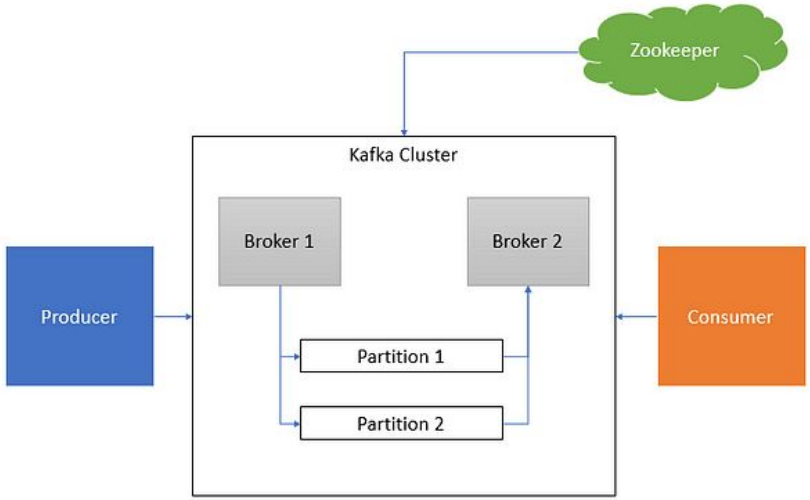
Concept of Kafka and its Specifications

Kafka Clusters Architecture: 5 Major Components



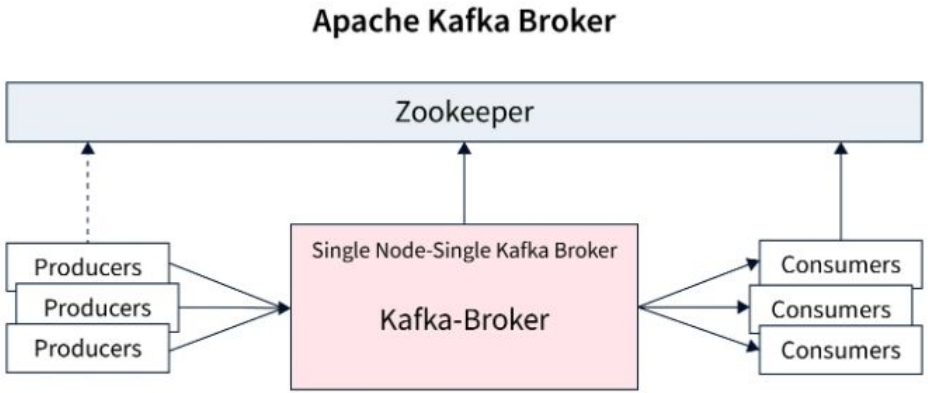
Kafka Architecture

- **Zookeeper**: Coordination service for discovery and registration of Kafka components
- **Kafka Cluster**: Set of servers called brokers that store and distribute messages
- **Producer**: Client that sends messages to Kafka topics
- **Partition**: Division of a Kafka topic to distribute messages across multiple brokers
- **Consumer**: Client that reads messages from one or more Kafka topics



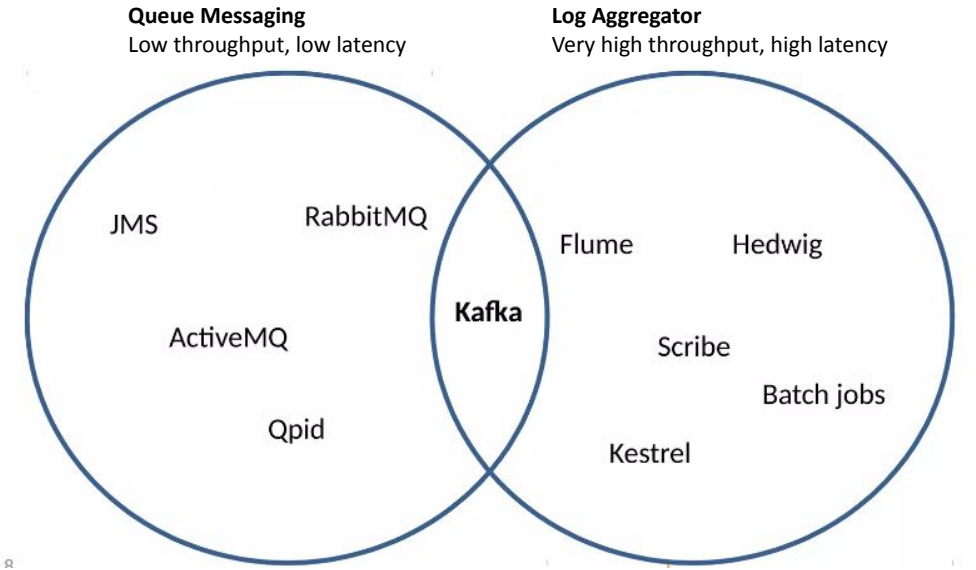
Kafka Architecture

- **Zookeeper**: Coordination service for discovery and registration of Kafka components
- **Kafka Cluster**: Set of servers called brokers that store and distribute messages
- **Producer**: Client that sends messages to Kafka topics
- **Partition**: Division of a Kafka topic to distribute messages across multiple brokers
- **Consumer**: Client that reads messages from one or more Kafka topics
- **Brokers :** Brokers are the servers that make up a Kafka cluster. They are responsible for storing and transmitting the messages within the group.



Comparing Kafka with Other Technologies

Comparison of Kafka with Other Brokers



Comparison of Kafka with Other Brokers

Feature	**Kafka**	**ActiveMQ**	**RabbitMQ**
Scalability	Designed for handling high volumes of data and supports horizontal scaling.	Can scale horizontally but may face performance issues with large data sets.	Can scale horizontally but may require additional configurations for high scalability.
Fault Tolerance	Replicates data across multiple brokers for automatic failover.	Provides fault tolerance but may require manual configuration for replication.	Provides fault tolerance through clustering and replication.
Real-time Processing	Enables low-latency message delivery for real-time data and stream processing.	Supports real-time processing but may have higher latency compared to Kafka.	Supports real-time processing but may have higher latency compared to Kafka.
Storage Efficiency	Stores data in a distributed and durable manner for efficient retention and retrieval.	Stores data but may have limitations on storage capacity and performance.	Stores data but may have limitations on storage capacity and performance.
Integration Capabilities	Integrates seamlessly with various data systems, databases, and frameworks.	Offers good integration capabilities but may require additional configurations.	Offers good integration capabilities but may require additional configurations.

Business Use Cases

LinkedIn

Activity stream, operational metrics tracking, data bus

OVH

Anti-DDOS (OVH)

Netflix

Real-time tracking, real-time processing (Netflix)

Twitter

Component of their real-time architecture

Spotify

Log processing (from 4am to 10am)

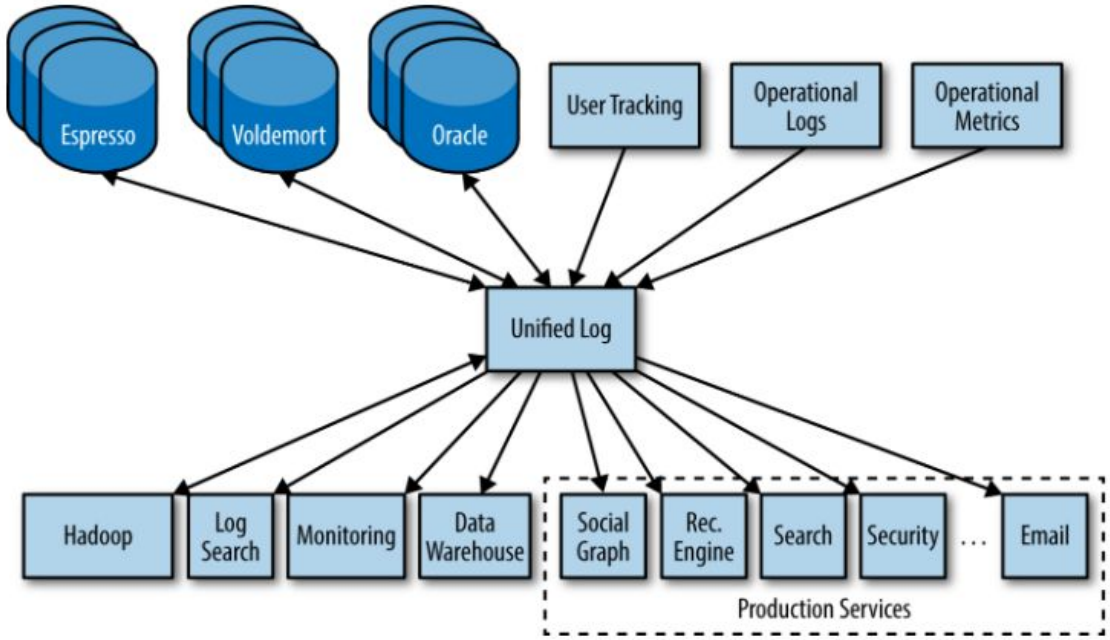
Mozilla

Metric management

Performance - LinkedIn

- 15 Brokers
- 15,500 partitions (replication factor 2)
- Input:
  - 400,000 messages/second
  - 70 MB/second
- Output:
  - 400 MB/second

Performance - LinkedIn



LinkedIn

kafka

Performance - LinkedIn

Data Ingestion:
- Unified Log: Kafka is used to collect and store unified event logs from the system.
- Monitoring Data: Kafka is used to collect and store system monitoring data.

Data Processing:
- Espresso: Kafka is used to stream data to the Espresso data processing platform.
- Voldemort: Kafka is used to stream data to the Voldemort NoSQL data storage system.
- Hadoop: Kafka is used to stream data to the open-source Hadoop data processing platform.

Data Analysis:
- Log Search: Kafka is used to stream logs to the log search and analysis service.
- Recommendation Engine: Kafka is used to stream data to the recommendation engine for job postings and training.

Production Services:
- Production Services: Kafka is used to stream data to LinkedIn's production services.

LinkedIn

kafka

Kafka - Spotify: Real-world Utilization

Spotify

kafka

Kafka - Spotify: Real-world Utilization

Discover how Spotify leverages Kafka to handle millions of music streams and provide real-time recommendations to users, ensuring a seamless music streaming experience.

Example of Data:

Spotify's data includes information about user preferences, song metadata, and streaming activity.

Architecture:

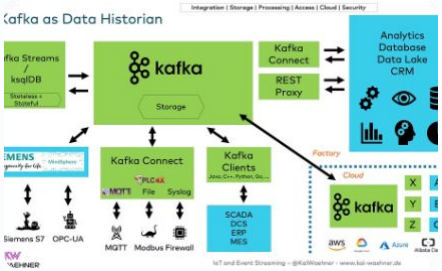
Spotify's architecture involves multiple Kafka clusters for data ingestion, streaming, and processing. The data flows through various components, such as producers, topics, brokers, and consumers, enabling real-time data processing and recommendation generation.

Performance - Spotify

Kafka - Spotify

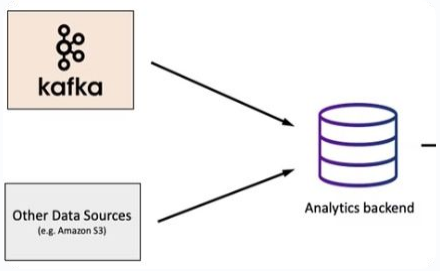
Music Streaming Integration

Kafka is integrated with Spotify for efficient music streaming and real-time data processing.



Real-time Data Processing

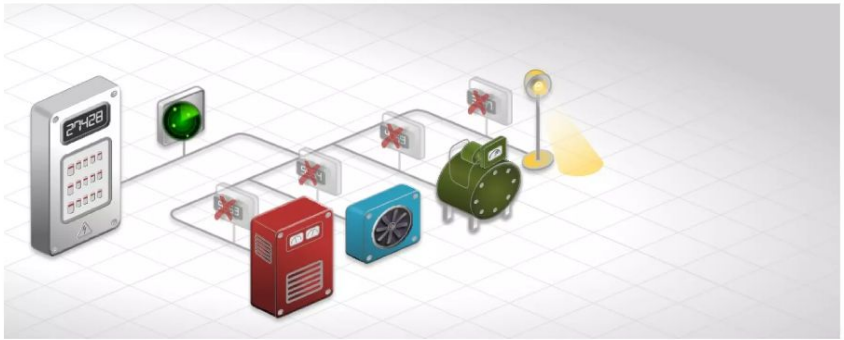
Spotify utilizes Kafka for real-time data processing and low-latency message delivery.



High Volume Event Handling

Kafka provides solutions for efficiently handling high volumes of event data for Spotify's platform.

Addressing Enterprise Challenges with Kafka



Kafka facilitates smooth integration with legacy systems, addressing compatibility challenges faced by enterprises.

Kafka offers solutions for efficiently handling high volumes of event data, a common challenge for enterprises.

Addressing Enterprise Challenges with Kafka

Kafka facilitates smooth integration with legacy systems, addressing compatibility challenges faced by enterprises.

Kafka offers solutions for efficiently handling high volumes of event data, a common challenge for enterprises.

Addressing Enterprise Challenges with Kafka

Kafka facilitates smooth integration with legacy systems, addressing compatibility challenges faced by enterprises.

Kafka offers solutions for efficiently handling high volumes of event data, a common challenge for enterprises.

Addressing Enterprise Challenges with Kafka

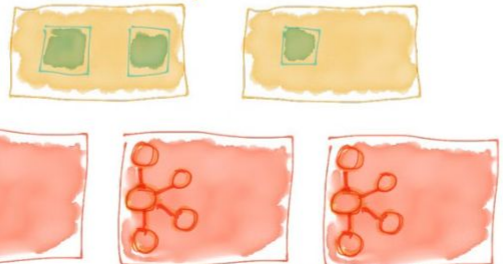
Kafka facilitates smooth integration with legacy systems, addressing compatibility challenges faced by enterprises.

Kafka offers solutions for efficiently handling high volumes of event data, a common challenge for enterprises.

Kafka's concepts

Streaming Integration with Kafka Connect

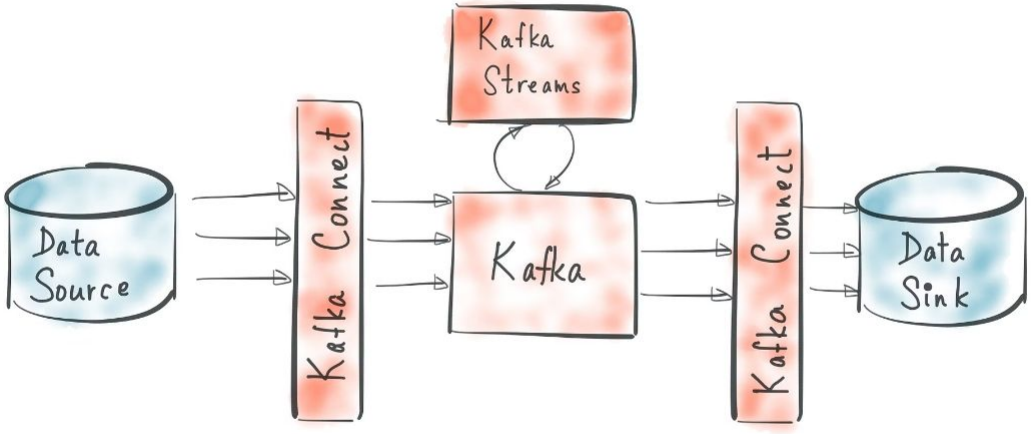
Sources



Kafka Connect

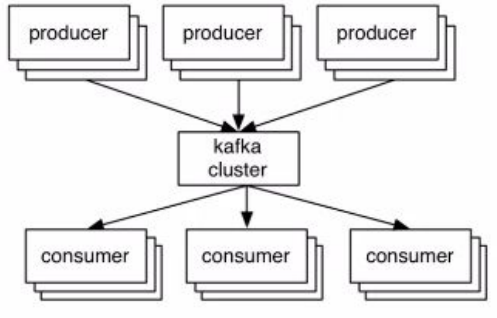
Kafka Brokers

Kafka Connect + Streams



Concepts de Kafka

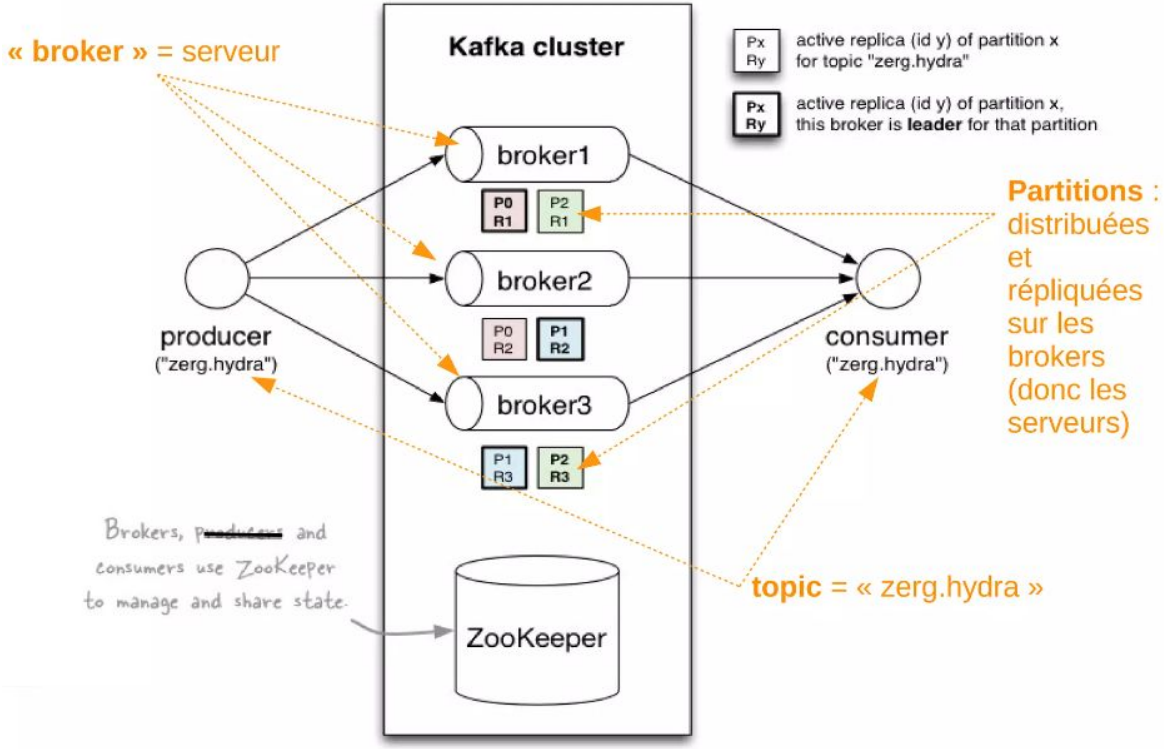
- **Overview**
  - Producers publish data to brokers.
  - Consumers subscribe and retrieve data from brokers.
  - All services are distributed.
  - Data is stored in topics.



- **Details**
  - Topics are divided into partitions and replicated to ensure high availability and scalability.
  - Kafka is not a database system and does not handle SQL queries.

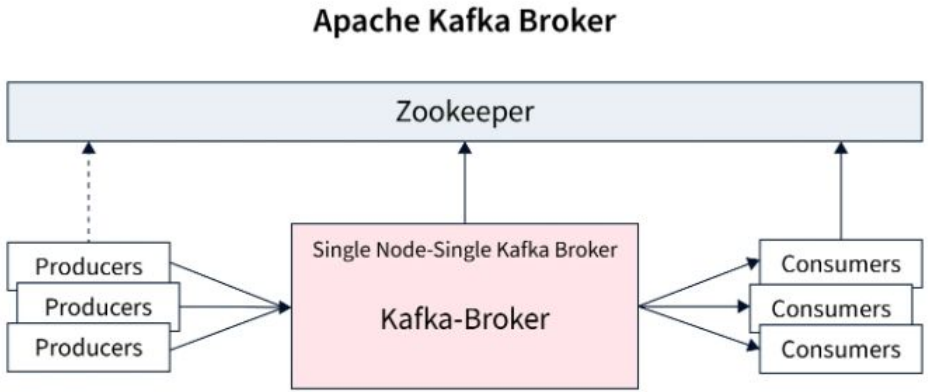
- **Advantages**
  - High availability and scalability
  - Low latency
  - High throughput
  - Asynchronous messaging
  - Real-time streaming

Concepts de Kafka - Overview



How Kafka Works

1. Producers send messages to Kafka topics.
2. Messages are stored on multiple brokers for fault tolerance.
3. Consumers read messages from brokers in order or disorder.
4. Consumers can subscribe to a single partition or a group of partitions.

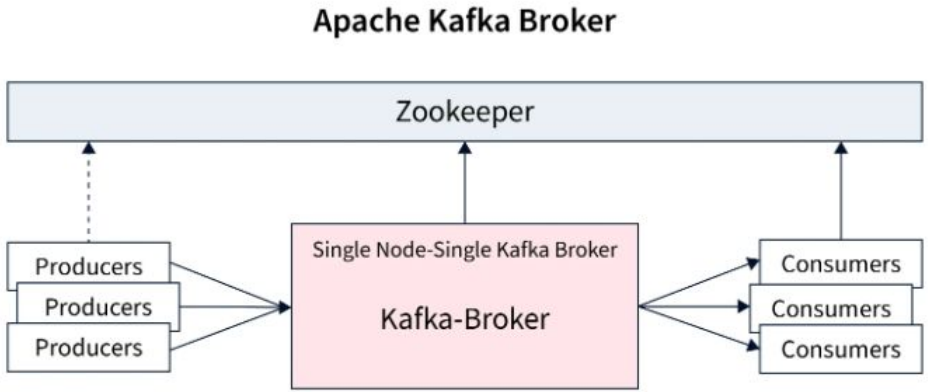


How to Start Kafka Broker?

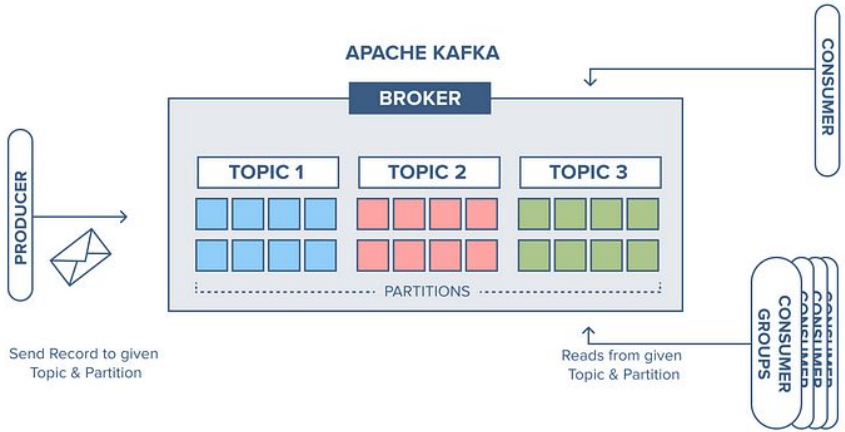
```
./bin/zookeeper-server-start.sh config/zookeeper.properties
```

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

```
$ ./bin/kafka-server-start.sh
USAGE: ./bin/kafka-server-start.sh [-daemon] server.properties [--override property=value]*
```

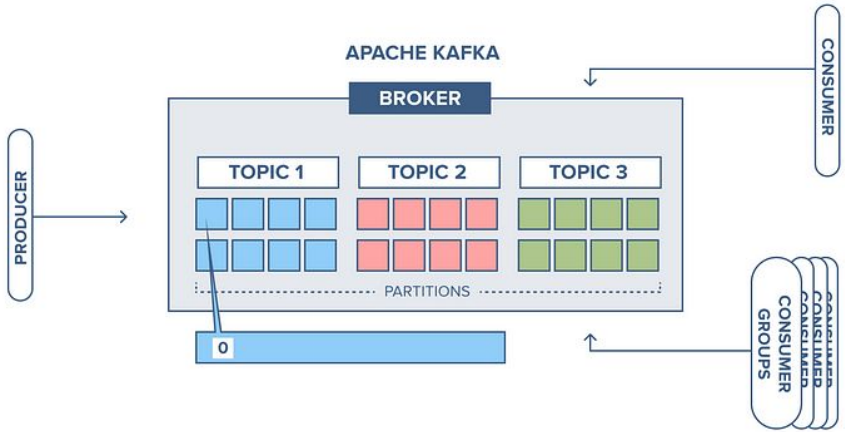


How Kafka Works



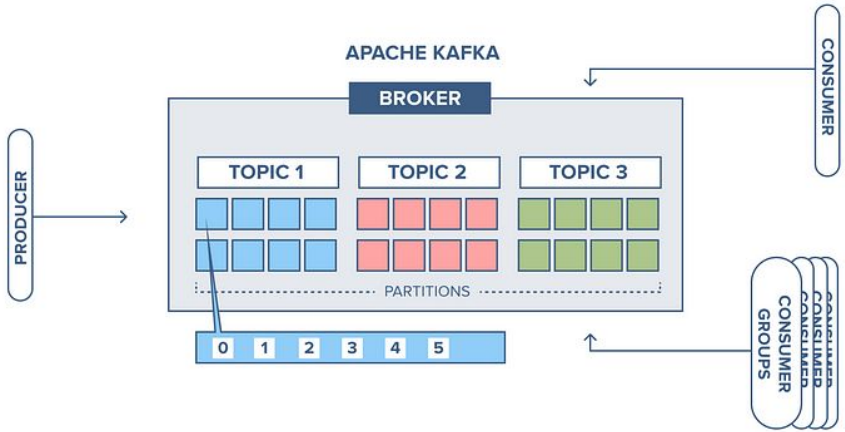
A broker with three topics, where each topic has 8 partitions.

How Kafka Works



The producer sends a record to partition 1 in topic 1 and since the partition is empty the record ends up at offset 0.

How Kafka Works



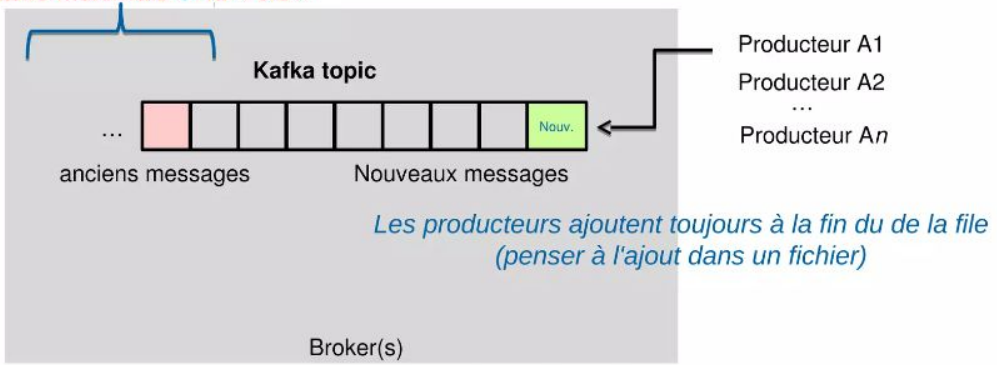
Next record is added to partition 1 will and up at offset 1, and the next record at offset 2 and so on.

Concepts de Kafka - Topic

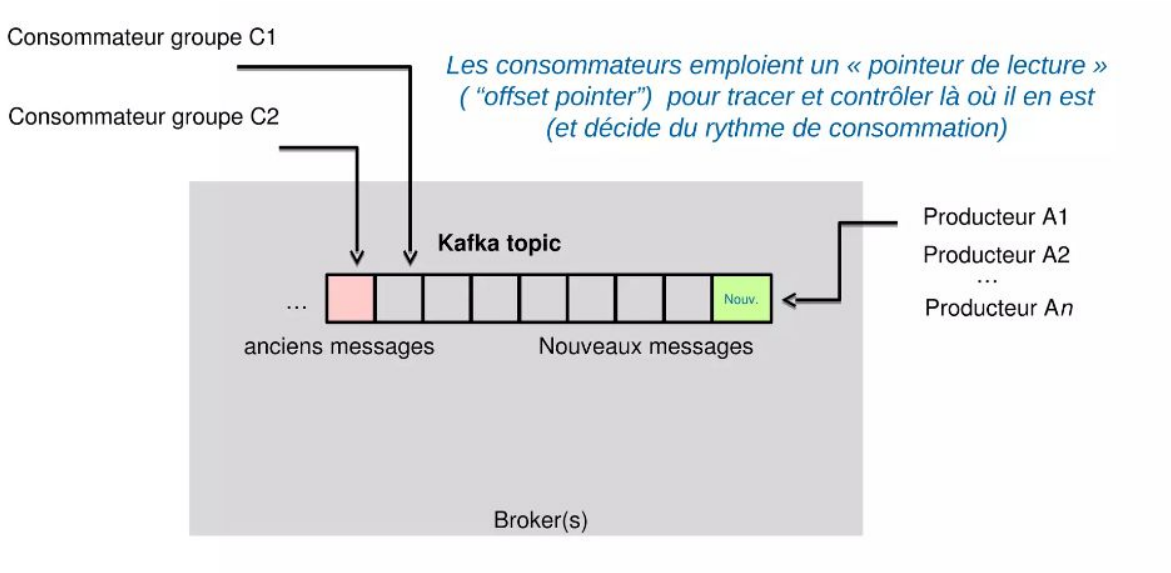
Topic: correspond au nom du flux sur lequel les messages vont être publiés

- Par exemple : “zerg.hydra”

Kafka élague depuis la “tête” en se basant sur l'âge ou la taille maximale or la « clé »

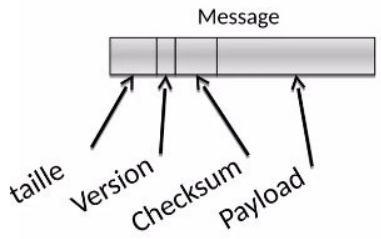


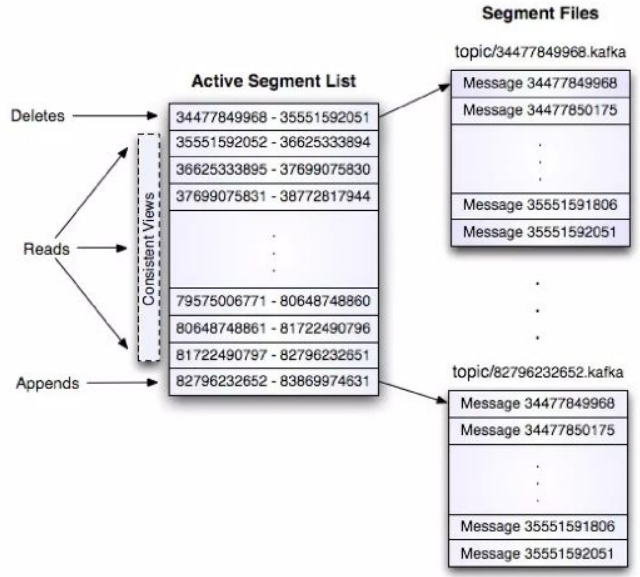
Concepts de Kafka - Topic



Concepts de Kafka - Message

Protocole léger
Traitement des messages par lot (Producteur & Consommateur)
Compression de bout en bout



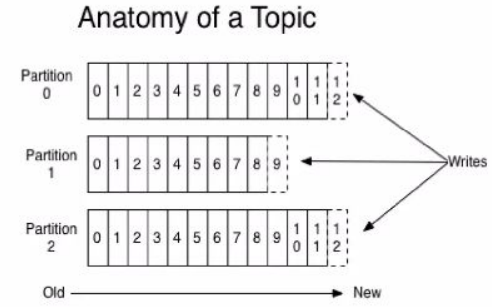


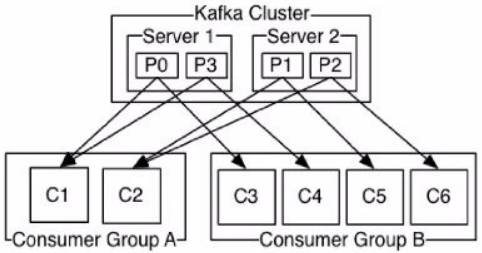
Concepts de Kafka - Partition

Les partitions

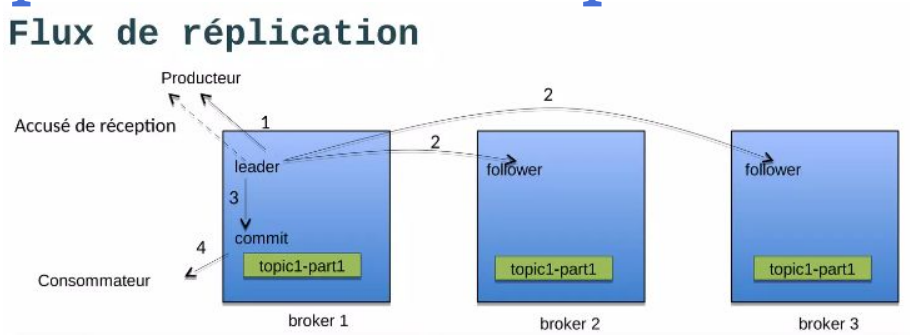
- Ordonnées
- Séquence immuable

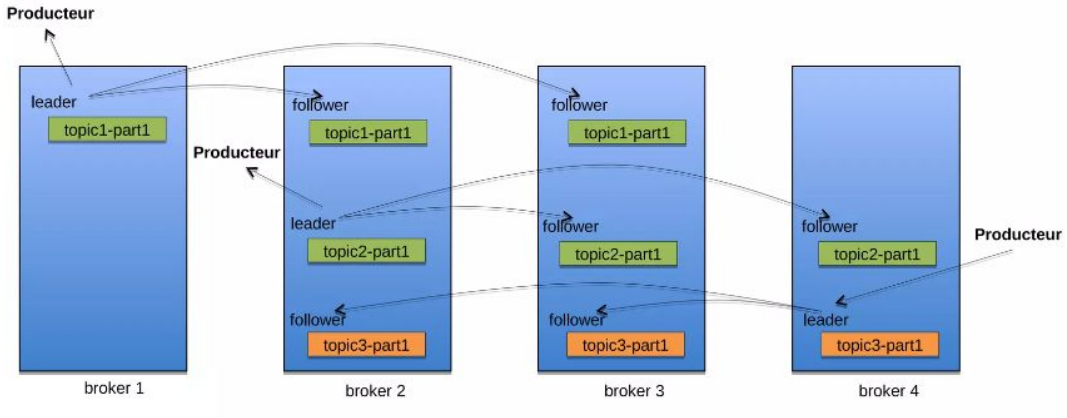
- Le nombre de partitions détermine le nombre maximum de (groupes de) consommateurs





Concepts de Kafka - Replicate





How Kafka Works

Steps to follow when setting up a connection and publishing a message/consuming a message.

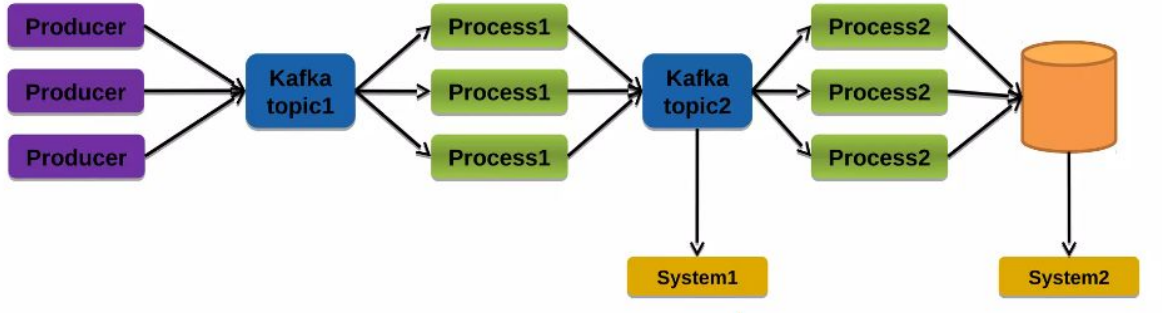
1. First of all, we need to set up a secure connection. A TCP connection will be set up between the application and Apache Kafka.
2. In publisher: Publish a message to a partition on a topic.
3. In subscriber/consumer: Consume a message from a partition in a topic.

```
> import java.util.Properties
  import kafkashaded.org.apache.kafka.clients.producer._
  import org.apache.spark.sql.ForeachWriter


   class  KafkaSink(topic:String, servers:String) extends ForeachWriter[(String, String)] {
        val kafkaProperties = new Properties()
        kafkaProperties.put("bootstrap.servers", servers)
        kafkaProperties.put("key.serializer", "kafkashaded.org.apache.kafka.common.serialization.StringSerializer")
        kafkaProperties.put("value.serializer", "kafkashaded.org.apache.kafka.common.serialization.StringSerializer")
        val results = new scala.collection.mutable.HashMap[String, String]
        var producer: KafkaProducer[String, String] = _

        def open(partitionId: Long,version: Long): Boolean = {
          producer = new KafkaProducer(kafkaProperties)
          true
        }

        def process(value: (String, String)): Unit = {
            producer.send(new ProducerRecord(topic, value._1 + ":" + value._2))
        }

        def close(errorOrNull: Throwable): Unit = {
          producer.close()
        }
    }
```

```
import java.util.Properties
import kafkashaded.org.apache.kafka.clients.producer._
import org.apache.spark.sql.ForeachWriter
defined class KafkaSink
```

Enhancing Event Scheduling with Kafka

Ecosystem


Producer
Producer
Producer
Kafka topic1
Process1
Process1
Process1
Kafka topic2
Process2
Process2
Process2
System1
System2


Apache Kafka adoption spans
companies across industries.
intuit
NETFLIX
ebay
Pinterest
Goldman Sachs
airbnb
cisco
box
Adobe
salesforce
Square
yelp
dish NETWORK
Cerner
WIKIPEDIA
The Free Encyclopedia
verizon
confluent
Spotify
Square
salesforce
fitbit
GoPro
slack
airbnb
coursera
intuit
PayPal
Dropbox
yahoo!
Uber
WIKIPEDIA
The Free Encyclopedia

Suite - Spring Batch Kafka