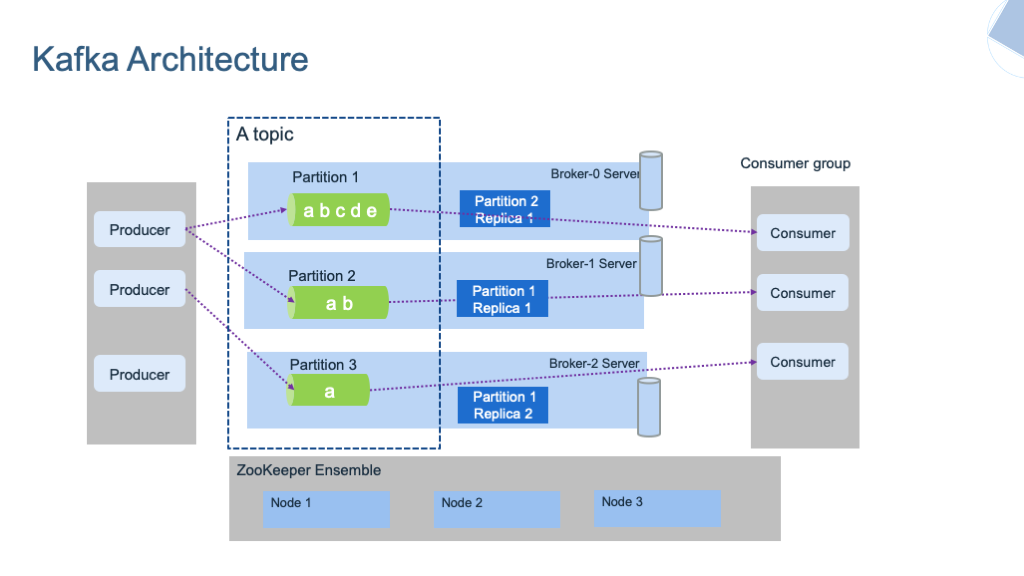
Apache Kafka



* What is Kafka?
  + Apache Kafka is an open-source distributed event streaming platform.
  + It is designed for high-throughput, fault-tolerant, and real-time data streaming.
  + Kafka is used for building real-time streaming data pipelines and applications.
* Key Concepts:
  + Producer: Publishers of messages to Kafka topics.
  + Consumer: Subscribers to Kafka topics, processing the messages.
  + Broker: Kafka server that stores and manages the streams of records.
  + Topic: A category or feed name to which records are published.
  + Partition: Each topic is split into partitions, allowing parallelism and scalability.
  + Offset: A unique identifier of each record within a partition.
  + ZooKeeper: Coordinates distributed brokers and consumers.
* Features:
  + Scalability: Kafka scales horizontally by adding more brokers.
  + Fault Tolerance: Replication of data across brokers ensures fault tolerance.
  + Durability: Messages are persisted on disk, enabling data replay and recovery.
  + High Throughput: Supports millions of messages per second.
  + Real-time: Low-latency message delivery.
  + Integration: Integrates with various systems like Hadoop, Spark, and Storm.
* Use Cases:
  + Log Aggregation: Collecting and centralizing logs from multiple services.
  + Stream Processing: Real-time analytics, monitoring, and ETL pipelines.
  + Event Sourcing: Storing state changes as a sequence of events.
  + Messaging: Building real-time messaging systems.
  + Metrics Monitoring: Collecting and analyzing real-time metrics data.
* Architecture:
  + Kafka cluster consists of multiple brokers.
  + Each broker can have multiple partitions, and each partition can be replicated across brokers.
  + Producers publish messages to topics, and consumers subscribe to topics to receive messages.
  + ZooKeeper manages cluster state, leader election, and metadata.
* APIs:
  + Kafka provides Producer API, Consumer API, Streams API, and Connect API.
  + Producer API allows applications to publish streams of records.
  + Consumer API allows applications to subscribe to topics and process streams of records.
  + Streams API enables building stream processing applications.
  + Connect API facilitates building and running connectors for importing/exporting data.
* Community and Ecosystem:
  + Active Apache community and extensive ecosystem of tools and integrations.
  + Support for various programming languages and frameworks.
  + Kafka Connectors for integrating with databases, messaging systems, and other data sources/destinations.
* Best Practices:
  + Proper topic partitioning and replication for scalability and fault tolerance.
  + Monitoring Kafka cluster health, performance, and lag.
  + Capacity planning and tuning for optimal performance.
  + Implementing security measures like encryption, authentication, and authorization.
* Limitations:
  + Complexity in setup and configuration.
  + Operational overhead, especially in managing ZooKeeper.
  + Storage costs can be significant for retaining large amounts of data.
  + Need for careful planning and monitoring to avoid data loss or corruption

Difference between Kafka and RabbitMQ

<https://www.linkedin.com/pulse/kafka-vs-rabbitmq-biggest-differences-which-should-you-mostafa-taheri>

Configuration with Spring:

spring.kafka.producer.key-serializer

spring.kafka.producer.value-serializer

spring.kafka.consumer.key-deserializer

Spring.kafka.consumer.value-deserializer

Producer configuration:



Consumer Configuration:



port=9092

advertised.host.name=localhost