

Chapter 1 - Introduction to Kafka

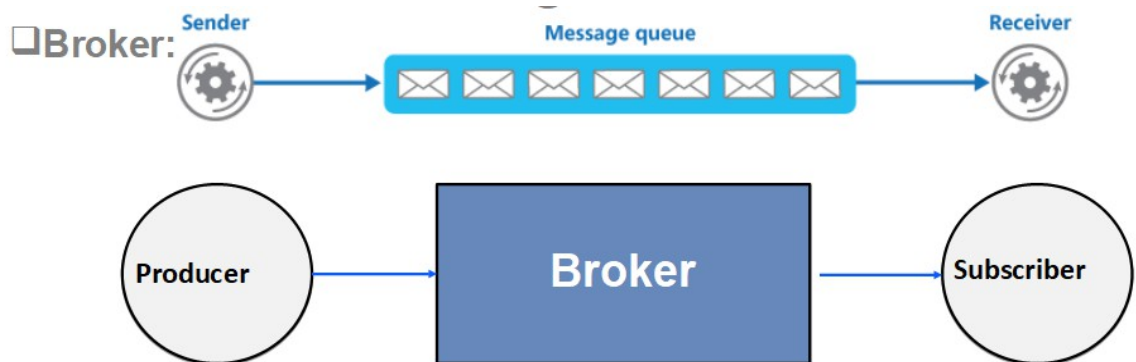
Objectives

Key objectives of this chapter

- Messaging Architectures
- What is Kafka?
- Need for Kafka
- Where is Kafka useful?
- Architecture
- Core concepts in Kafka
- Overview of Kraft
- Cluster, Kafka Brokers, Producer, Consumer, Topic

1.1 Messaging Architectures – What is Messaging?

- Application-to-application communication
- Supports asynchronous operations.
- Message:
 - ◇ A message is a self-contained package of business data and network routing headers.



1.2 Messaging Architectures – Steps to Messaging

- Messaging connects multiple applications in an exchange of data.
- Messaging uses an encapsulated asynchronous approach to exchange data through a network.
- A traditional messaging system has two models of abstraction:

Chapter 1 - Introduction to Kafka

- ◇ Queue – a message channel where a single message is received exactly by one consumer in a point-to-point message-queue pattern. If there are no consumers available, the message is retained until a consumer processes the message.
- ◇ Topic - a message feed that implements the publish-subscribe pattern and broadcasts messages to consumers that subscribe to that topic.
- A single message is transmitted in five steps:
 - ◇ Create
 - ◇ Send
 - ◇ Deliver
 - ◇ Receive
 - ◇ Process

1.3 Messaging Architectures – Messaging Models

- 1. Point to Point
- 2. Publish and Subscribe

Point-to-point (1→1)



Publish-and-subscribe (1→Many)



1.4 What is Kafka?

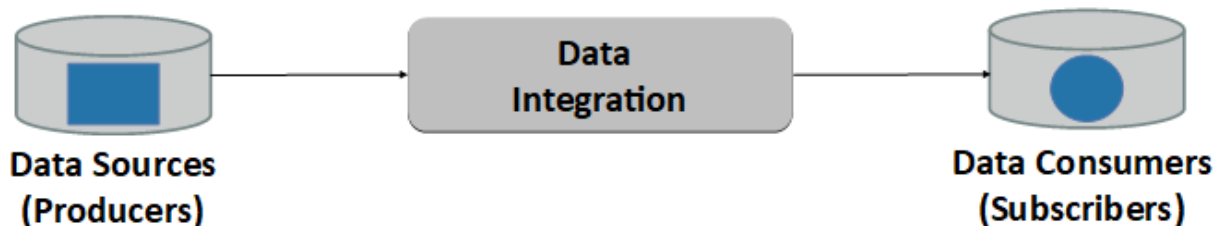
- In modern applications, real-time information is continuously generated by:
 - ◇ Applications (publishers/producers)
 - ◇ Routed to other applications (subscribers/consumers)

Chapter 1 - Introduction to Kafka

- Apache Kafka is an open source, distributed publish-subscribe messaging system.
 - ◇ Written in the Scala language with multi-language support and runs on the Java Virtual Machine (JVM).
- Kafka allows integration of information of producers and consumers to avoid any kind of rewriting of an application at either end.
- Kafka overcomes the challenges of real-time data usage for consumption of data volumes that may grow in order of magnitude, larger than the real data.

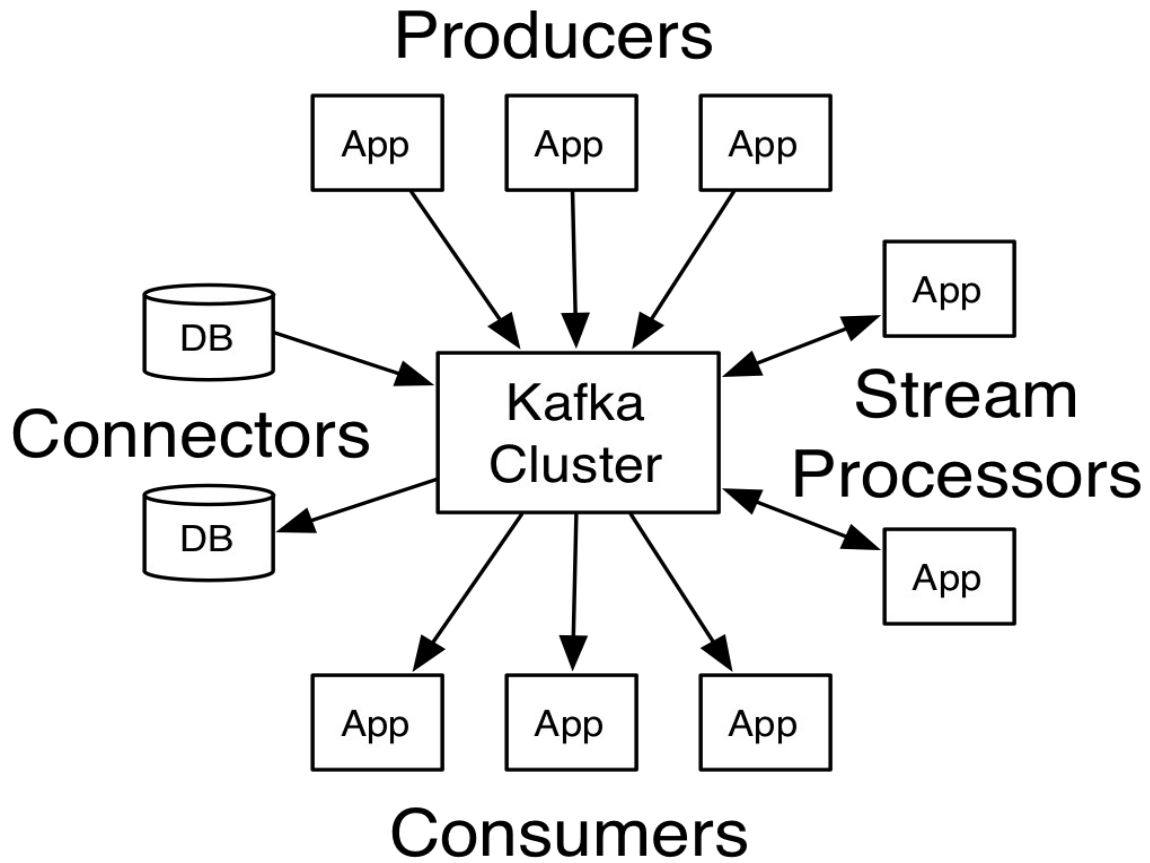
1.5 Kafka Overview

- When used in the right way and for the right use case, Kafka has unique attributes that make it a highly attractive option for data integration.



- Data Integration is the combination of technical and business processes used to combine data from disparate sources into meaningful and valuable information.
 - ◇ A complete data integration solution encompasses discovery, cleansing, monitoring, transforming and delivery of data from a variety of sources
- Messaging is a key data integration strategy employed in many distributed environments such as the cloud.
 - ◇ Messaging supports asynchronous operations, enabling you to decouple a process that consumes a service from the process that implements the service.

1.6 Kafka Overview (Contd.)



Chapter 1 - Introduction to Kafka

1.7 Need for Kafka

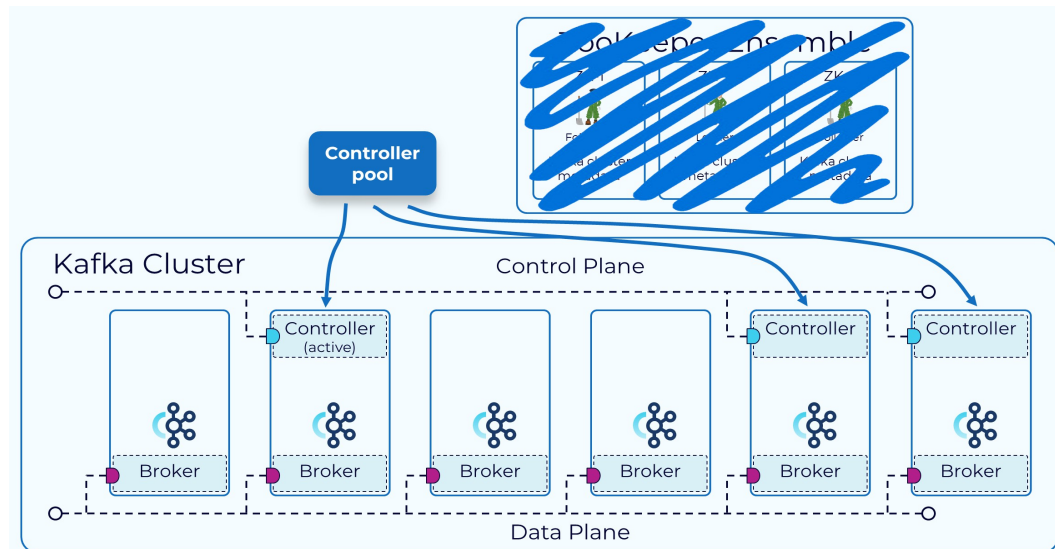
- High Throughput
 - ◇ Provides support for hundreds of thousands of messages with modest hardware
- Scalability
 - ◇ Highly scalable distributed systems with no downtime
- Replication
 - ◇ Messages can be replicated across a cluster, which provides support for multiple subscribers and also in case of failure balances the consumers
- Durability
 - ◇ Provides support for persistence of messages to disk which can be further used for batch consumption
- Stream Processing
 - ◇ Kafka can be used along with real-time streaming applications like Spark, Flink, and Storm
- Data Loss
 - ◇ Kafka with proper configurations can ensure zero data loss - idempotency

1.8 When to Use Kafka?

- Kafka is highly useful when you need the following:
 - ◇ A highly distributed messaging system
 - ◇ A messaging system which can scale out exponentially
 - ◇ high throughput on publishing and subscribing
 - ◇ varied consumers having varied capabilities by which to subscribe these published messages in the topics
 - ◇ A fault tolerance operation
 - ◇ Durability in message delivery
 - ◇ All of the above without tolerating performance degrade

Chapter 1 - Introduction to Kafka

1.9 Kafka Architecture



1.10 Core concepts in Kafka

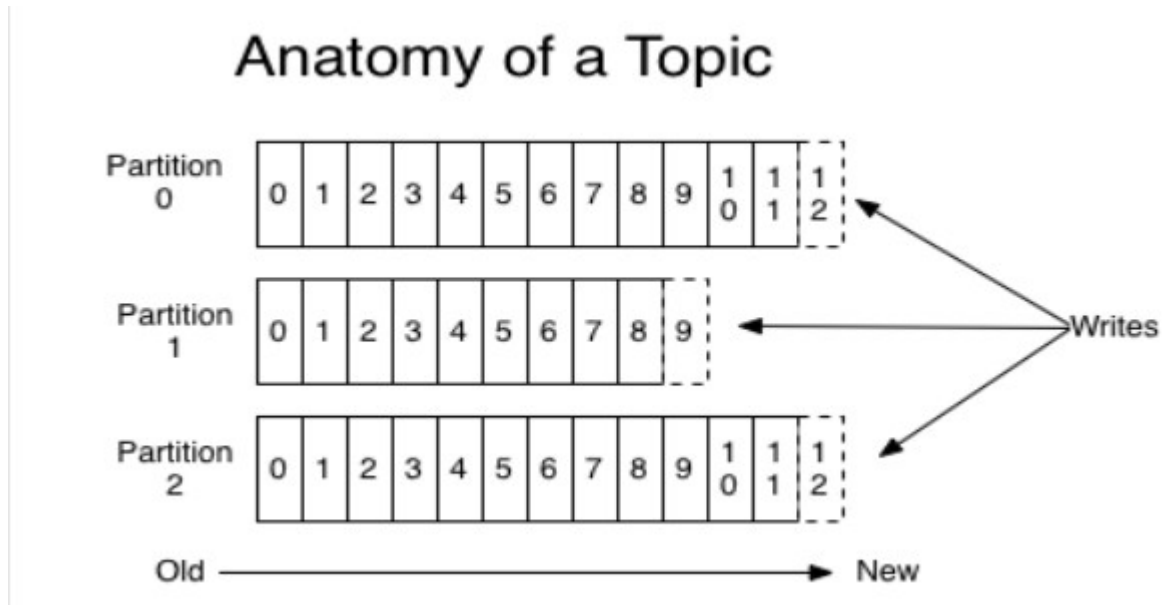
- **Topic**
 - ◇ A category or feed to which messages are published
- **Producer**
 - ◇ Publishes messages to Kafka Topic
- **Consumer**
 - ◇ Subscribes and consumes messages from Kafka Topic
- **Broker**
 - ◇ Handles hundreds of megabytes of reads and writes

1.11 Kafka Topic

- User defined category where the messages are published
- For each topic, a partition log is maintained
- Each partition basically contains an ordered, immutable sequences of messages where each message assigned a sequential ID number called **offset**
- Writes to a partition are generally sequential thereby reducing the number of hard disk seeks

Chapter 1 - Introduction to Kafka

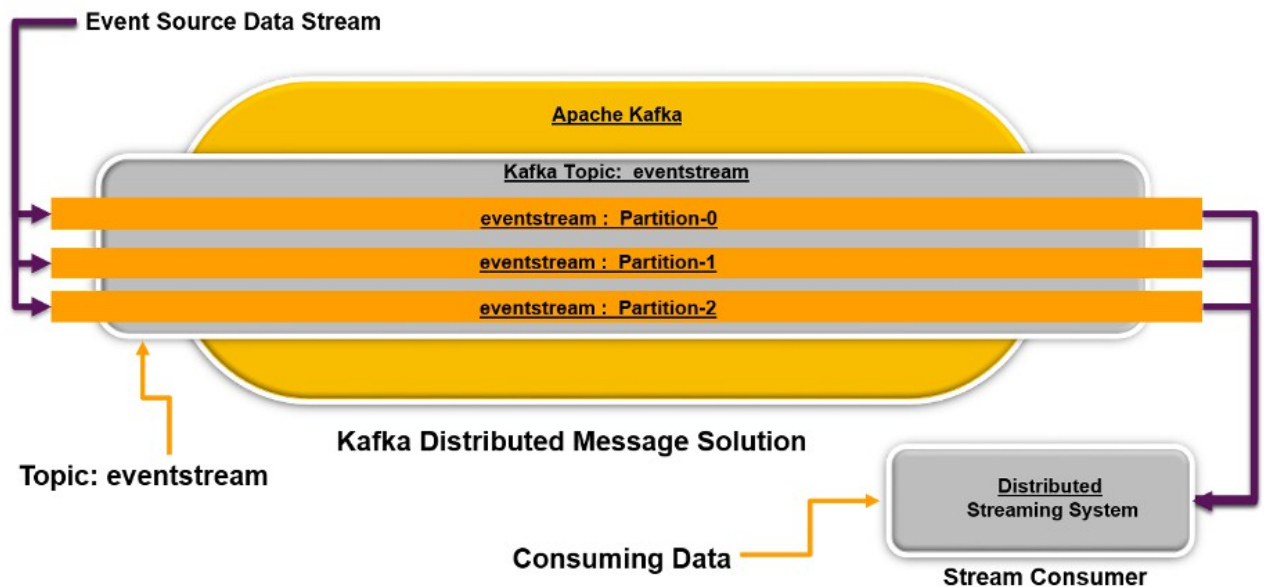
- Reading messages from partition can either be from the beginning and also can rewind or skip to any point in a partition by supplying an offset value



1.12 Kafka Partitions

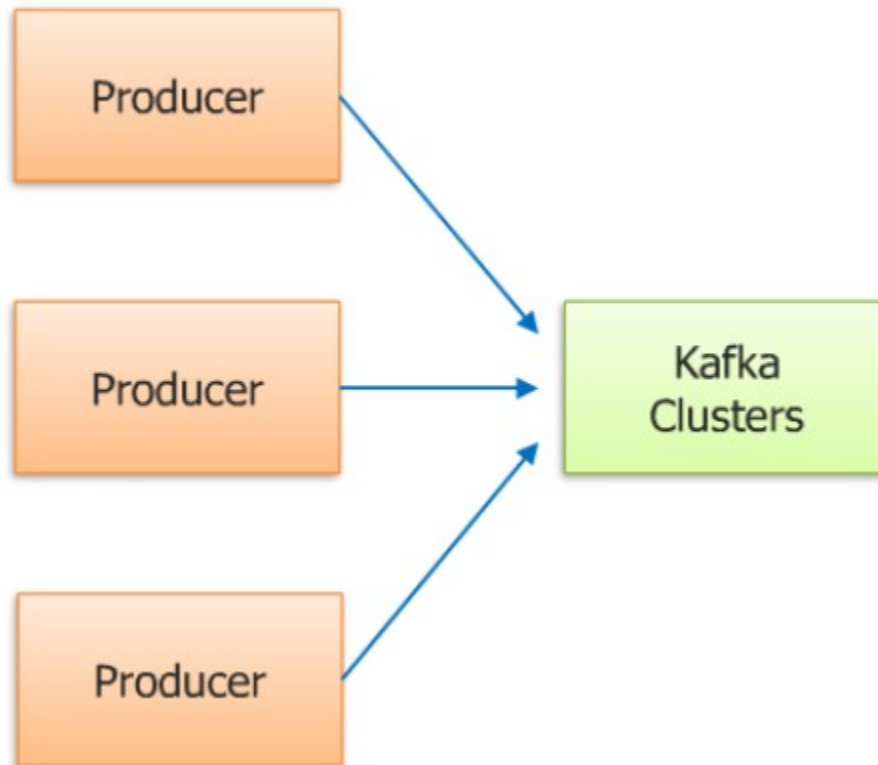
- In Kafka, a topic can be assigned to a group of messages of a similar type that will be consumed by similar consumers.
- Partitions (a unit of parallelism in Kafka), parallelize the consumption of messages in a Kafka topic with the total number in a cluster not less than the number of consumers in a consumer group.

Chapter 1 - Introduction to Kafka



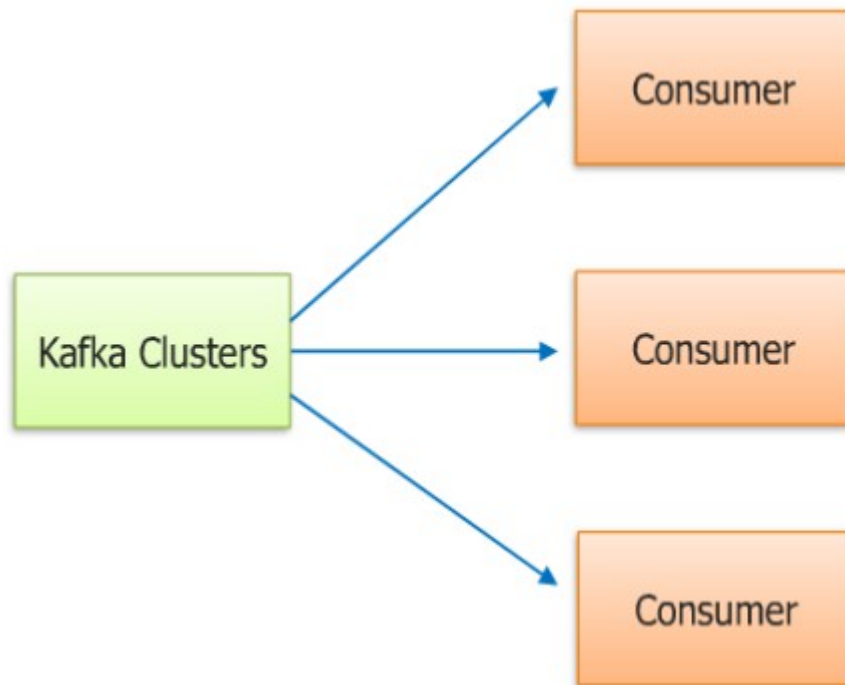
1.13 Kafka Producer

- Application publishes messages to the topic in Kafka Cluster
 - ◇ Can be of any kind like an API Front End, Streaming etc.
- While writing messages, it is also possible to attach a key to the message
 - ◇ By attaching key the producers basically provide a guarantee that all messages with the same key will arrive in the same partition
- Supports both async and sync modes
- Publishes as many messages as fast as the broker in a cluster can handle



1.14 Kafka Consumer

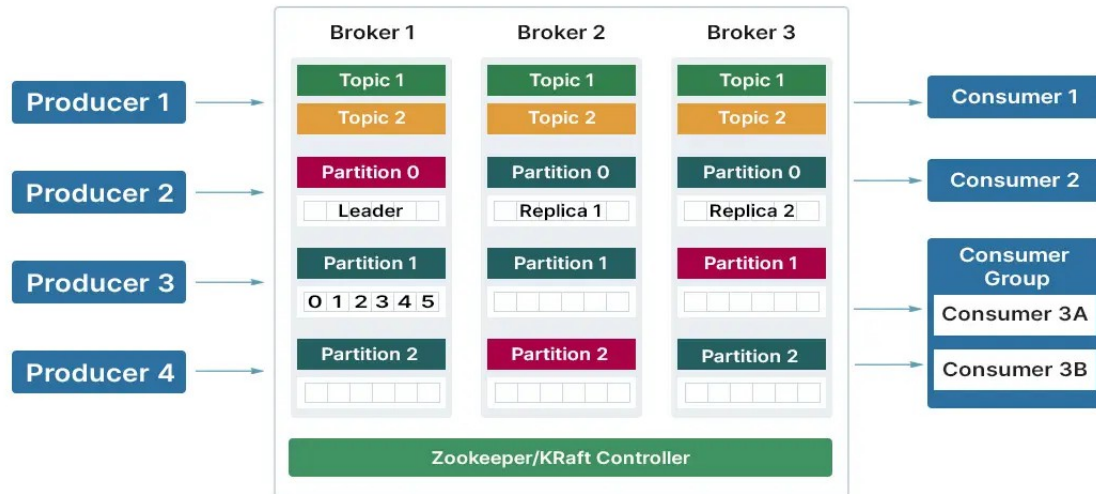
- Applications subscribe and consume messages from brokers in Kafka Cluster
 - ◇ Can be of any kind like real-time consumers, NoSQL consumers etc.
- During consumption of messages from a topic a consumer group can be configured with multiple consumers.
 - ◇ Each consumer of consumer group reads messages from a unique subset of partitions in each topic they subscribe to
 - ◇ Messages with the same key arrive at the same consumer
- Supports both Queuing and Publish-Subscribe
- Consumers have to maintain the number of messages consumed



1.15 Kafka Broker

- Kafka cluster basically is comprised of one or more servers
 - ◇ Each of the servers in the cluster is called a **broker**
- Handles hundreds of megabytes of writes from producers and reads from consumers
- Retains all the published messages irrespective of whether it is consumed or not
- If retention is configured for n days, published message is available for consumption for configured n days and thereafter it is discarded

Kafka Architecture



1.16 Kafka Cluster

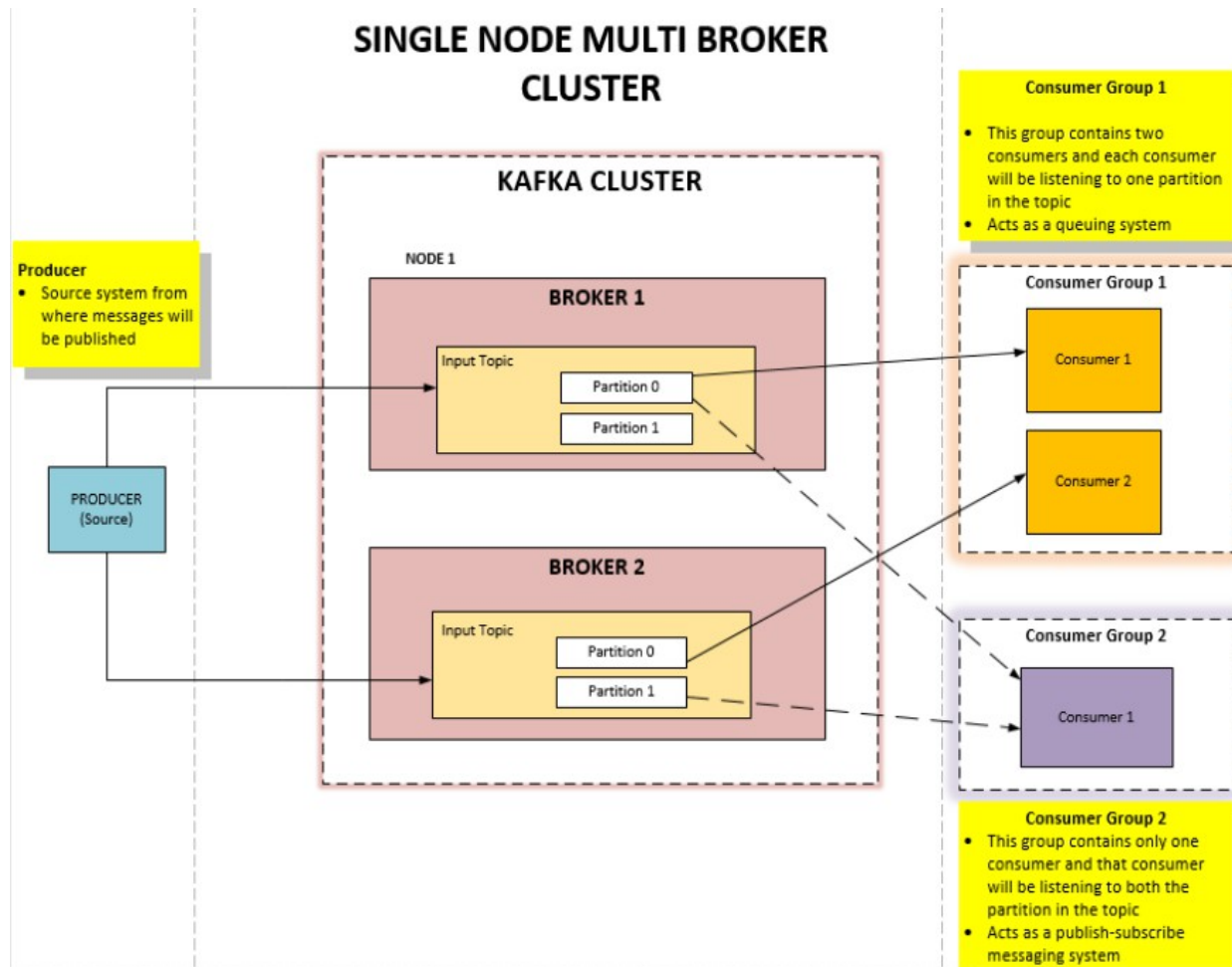
- A Kafka Cluster is generally fast, highly scalable messaging system
- A publish-subscribe messaging system
- Can be used effectively in place of ActiveMQ, RabbitMQ, Java Messaging System (JMS), and Advanced Messaging Queuing Protocol (AMQP)
- Expanding of the cluster can be done with ease
- Effective for applications which involve large-scale message processing

1.17 Why Kafka Cluster?

- Kafka is preferred in place of more traditional brokers like JMS and AMQP
 - ◇ With Kafka, we can easily handle hundreds of thousands of messages in a second, which makes Kafka a high throughput messaging system
 - ◇ The cluster can be expanded with no downtime, making Kafka highly scalable
 - ◇ Messages are replicated, which provides reliability and durability
 - ◇ Fault-tolerant

Chapter 1 - Introduction to Kafka

1.18 Sample Multi-Broker Cluster



1.19 KRaft

- ◇ Apache Kafka® Raft (KRaft) is the consensus protocol that was introduced to remove Kafka's dependency on ZooKeeper for metadata management.
- ◇ This simplifies the architecture and deployment dependencies.

1.20 Schema Registry

- Schema Registry is required for the following reasons:
 - ◇ Ensure data quality and evolvability
 - ◇ Define data standards
 - ◇ Evolve without breaking applications
- Provides centralized schema management and the RESTful interface for storing and retrieving schemas for:
 - ◇ Avro
 - ◇ JSON
 - ◇ Protobuf
- The Schema Registry provides a mapping between topics in Kafka and the schema they use.
- Helps with data governance and architects a loosely coupled and dynamically evolving systems

1.21 Schema Registry (contd.)

- It also enforces compatibility rules before messages are added.
- The Schema Registry will check every message sent to Kafka for compatibility
 - ◇ Ensuring that incompatible messages will fail on publication.
- The Confluent Schema Registry provides both runtime validation of schema compatibility, as well as a caching feature for schemas so they don't need to be included in the message payload.







Documentation site: <https://docs.confluent.io/platform/current/schema-registry/index.html#schemaregistry-intro>

Schema Registry tutorial:

https://docs.confluent.io/platform/current/schema-registry/schema_registry_tutorial.html#schema-registry-tutorial

Chapter 1 - Introduction to Kafka

1.22 Who Uses Kafka?

 <p>Apache Kafka is used at LinkedIn for activity stream data and operational metrics. This powers various products like LinkedIn Newsfeed, LinkedIn Today in addition to our offline analytics systems like Hadoop.</p>	 <p>Real-time monitoring and event-processing pipeline.</p>	 <p>Kafka is used at Spotify as part of their log delivery system.</p>
 <p>As part of their Storm stream processing infrastructure, e.g. this and this.</p>	 <p>Kafka powers online to online messaging, and online to offline messaging at Foursquare. We integrate with monitoring, production systems, and our offline infrastructure, including hadoop.</p>	 <p>Used in our event pipeline, exception tracking & more to come.</p>

1.23 Summary

- Kafka is a unique distributed publish-subscribe messaging system written in the Scala language with multi-language support and runs on the Java Virtual Machine (JVM).
- Kafka relies on another service named Zookeeper – a distributed coordination system – to function.
- Kafka has high-throughput and is built to scale-out in a distributed model on multiple servers.
- Kafka persists messages on disk and can be used for batched consumption as well as real-time applications.

1.24 Review Questions

- ◇ What two messaging architectures does Kafka utilize?
- ◇ What are the four main concepts of Kafka?
- ◇ What factors are important considerations when using Kafka partitions?
- ◇ What is idempotency? How is it achieved with Kafka?