RabbitMQ vs Kafka

Group-Pi, February 8th

What are Message Brokers?

Message Brokers are programs that allow services to communicate with each other.

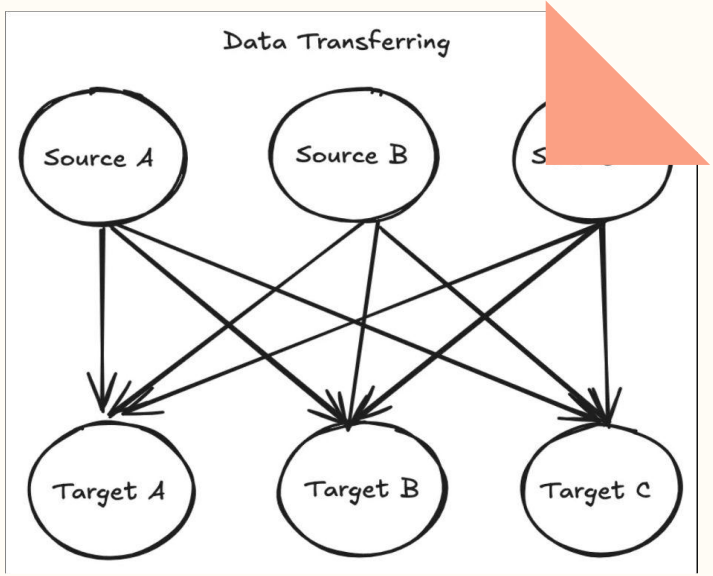
Might sound similar to API?

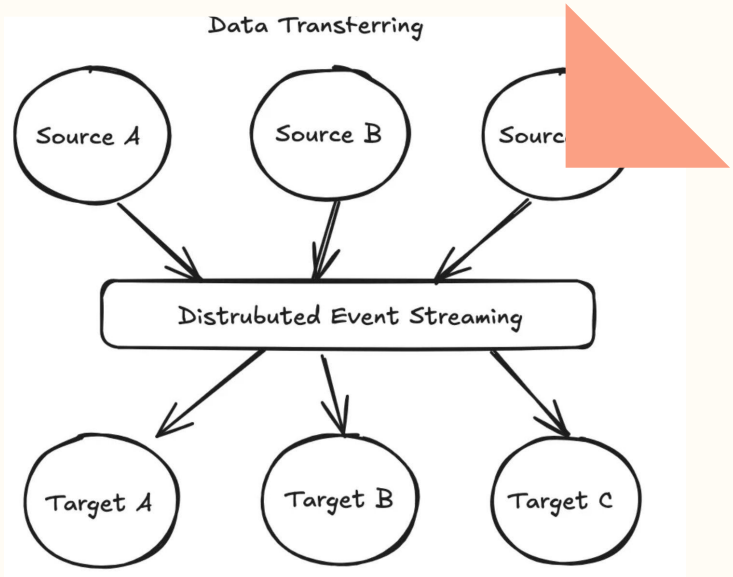
Not really, Message Broker acts as a queue and allows services to communicate asynchronously, even when they are written in different languages.

What is Kafka?

What is Kafka?

Kafka is a medium that helps services talk to each other by sending and receiving messages. Kafka acts as a mid-components and stores are messages for the consumers to consume.

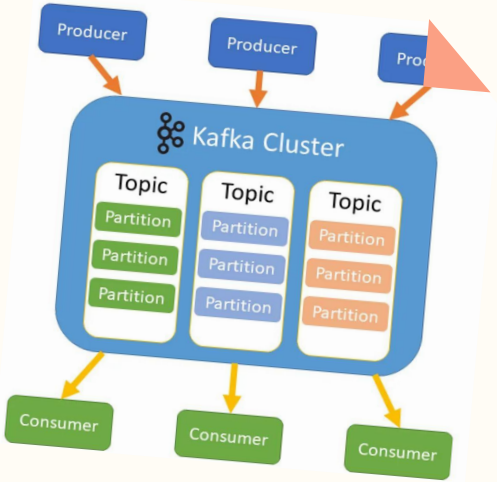




Kafka Architecture

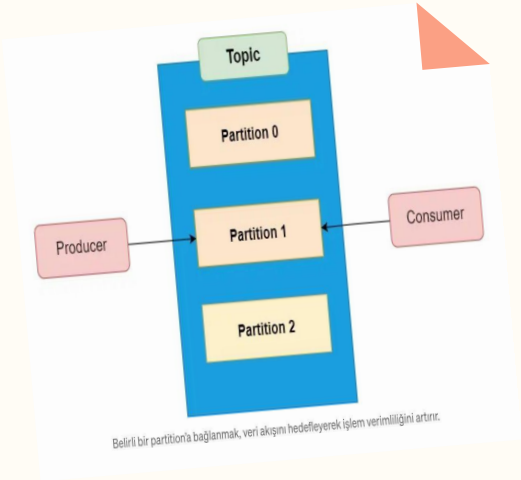
Producer and Consumer

Producers are the applications that send data to the kafka topics. Consumers on the other hand, read the data from these topics.



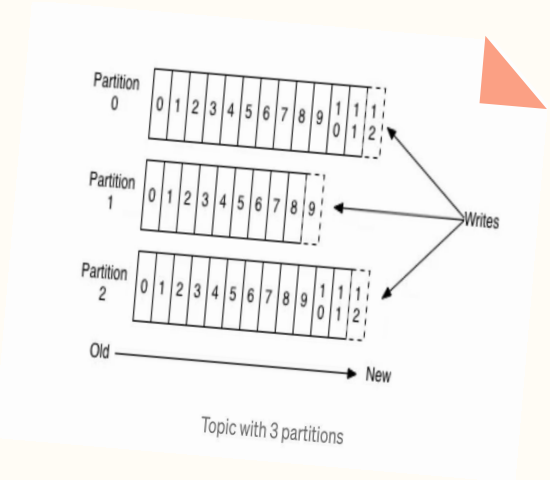
Kafka Topic and Partition

A topic is a logical channel to which events are published and consumed from. A topic is further divided horizontally into partitions for parallel processing.



Offset

An offset is an unique identifier that is assigned to each message within a partition. It helps track the position of consumer in partition log.



Pros

- Replayability of Messages
- Support for Multiple Consumers
- High-Throughput, Low-Latency Design
- Real-time stream processing

Cons

- Message Ordering Complexity
- No routing features
- Not Suitable for Low-Latency Individual Message Processing
- No built in DLQ

When would you use it ?

Event Stream Replays:

- Example: A stock trading platform can replay trade data for regulatory compliance and fraud analysis.
- The message replayability feature of Kafka retains messages for a period that can be set by the developer

Multiple Consumers:

- Example: A news website like BBC can use Kafka to stream live updates to mobile apps, web browsers, and social media simultaneously
- Kafka allows multiple consumers to pull data from a topic and makes sure their actions do no affect each other and allow them to work as independent entities.

When would you not use it ?

Low-Latency Individual Message Processing:

- Example: A real-time chat application like WhatsApp may experience delays if implemented with Kafka instead of a dedicated messaging system

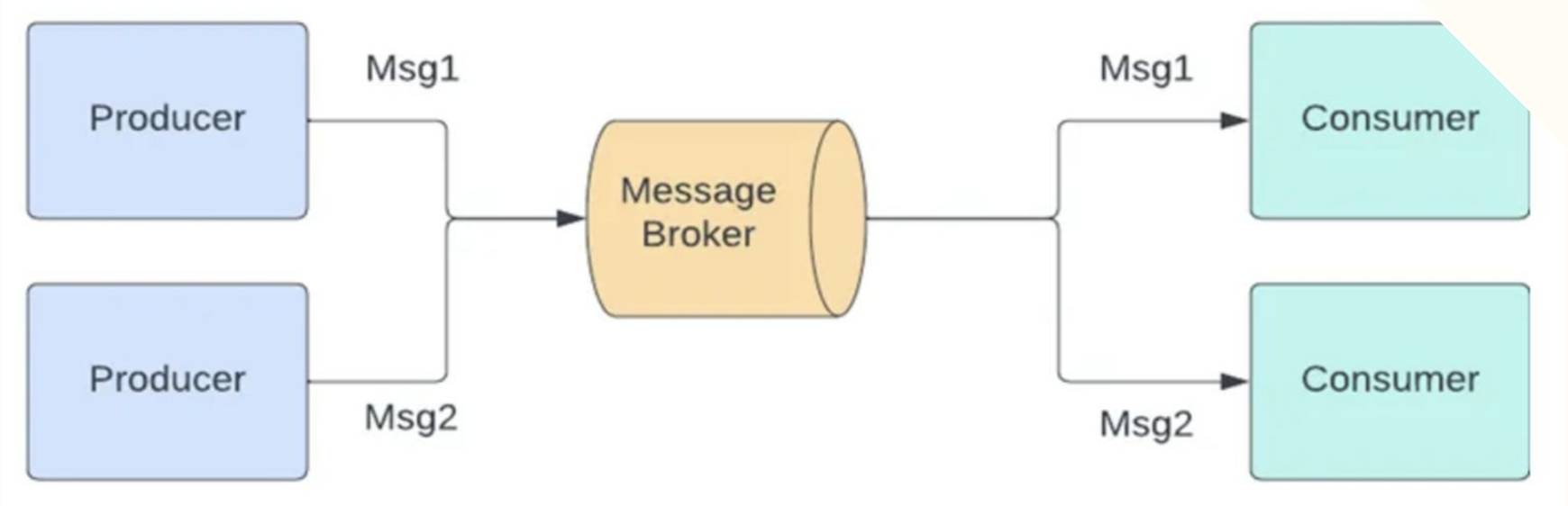
Message Ordering:

- Example: In a banking system, you need to ensure that transactions between accounts are processed in the correct sequence to avoid issues like double-spending or inconsistent account balances

What is RabbitMQ?

What is The Architecture of RabbitMQ?

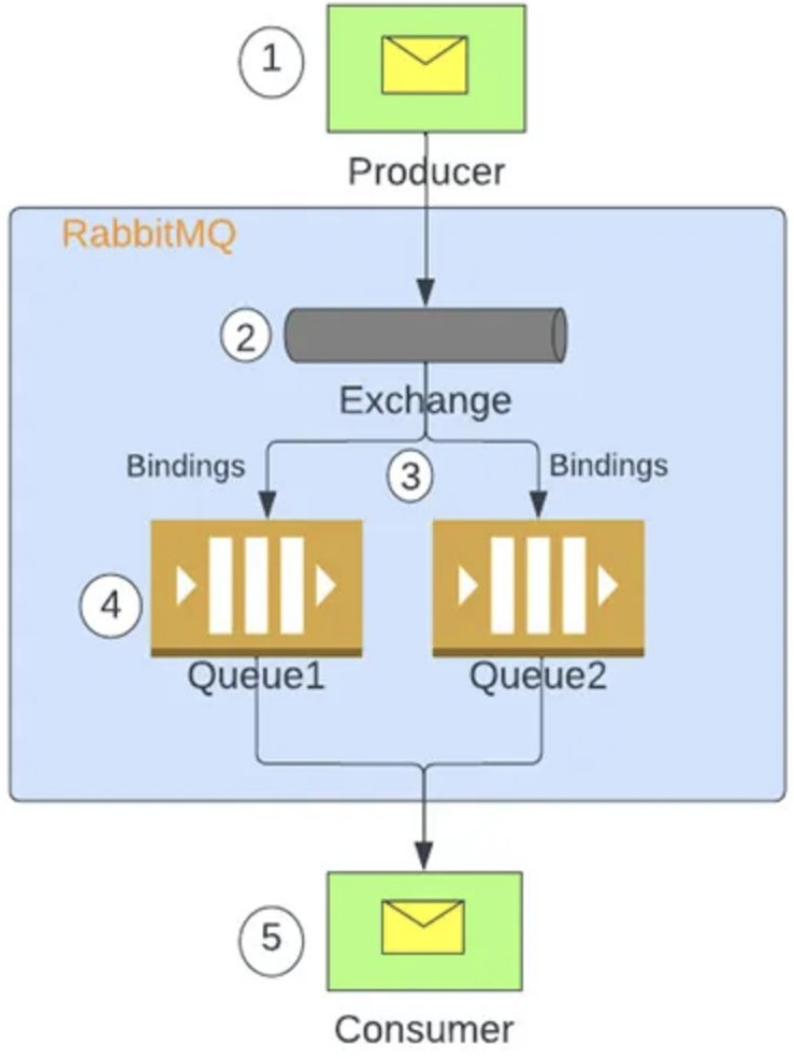
A RabbitMQ broker allows for low latency and complex message distributions using an exchange, queue and bindings



RabbitMQ Architecture

Components of a broker

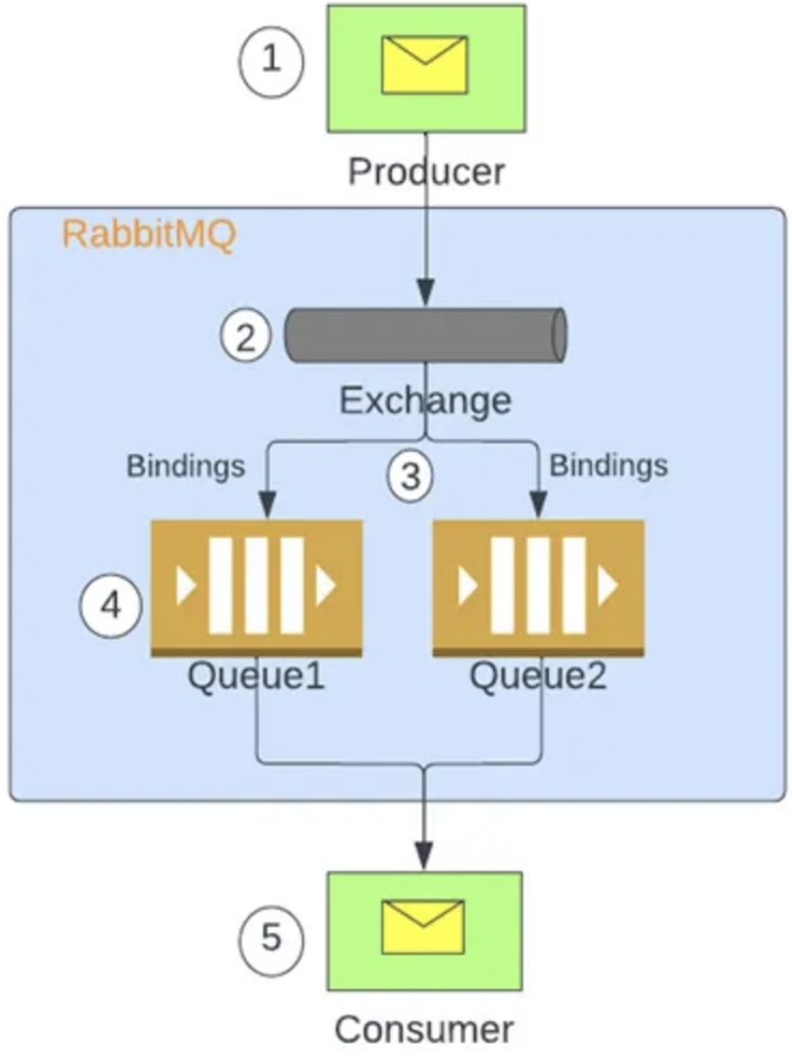
1. An exchange receives messages from the producer and determines where they should be routed to
2. A queue is storage that receives messages from an exchange and sends them to consumers
3. A binding is a path that connects an exchange and a broker



RabbitMQ Architecture

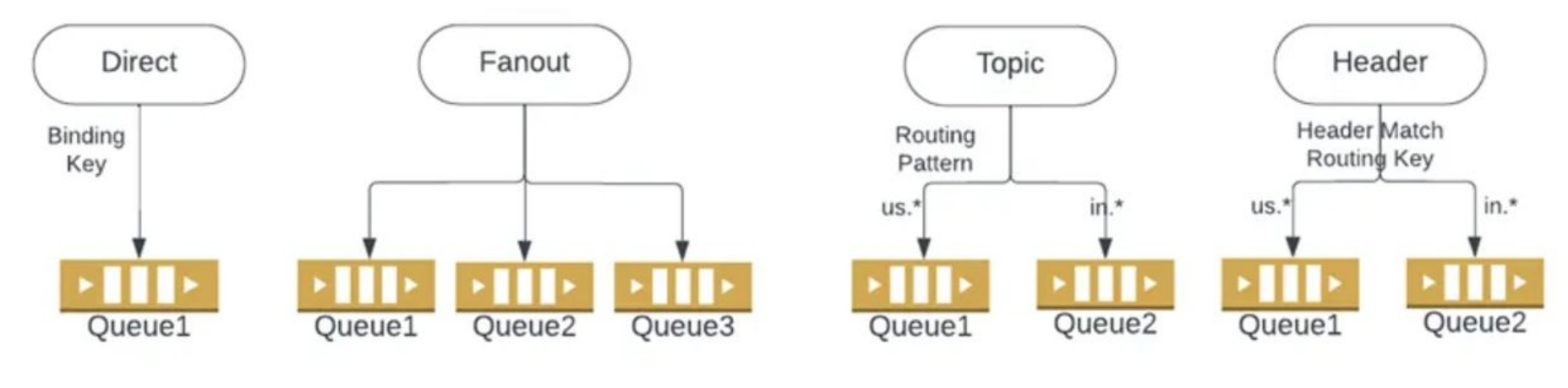
Message flow in RabbitMQ

1. Producer publishes messages to exchange via a channel established between them at the time of application startup.
2. Exchange receives the message and finds appropriate bindings based on message attributes and exchange types.
3. Selected binding is then used to route messages to intended queues.
4. The message stays in the queue until handled by the consumer.
5. Consumers receive the messages using channels established usually at application startup.



RabbitMQ Architecture

Different Exchange Types



Pros

- Protocol Flexibility: Supports multiple messaging protocols (AMQP, MQTT, STOMP), making it adaptable for diverse applications.
- Low Latency: Ideal for real-time, low-latency message delivery scenarios.
- Complex Routing: Advanced exchange types allow dynamic message routing, suitable for microservices architecture.

Cons

- No Message Replay: Does not support re-reading of consumed messages; messages are deleted after acknowledgment.
- Conditional Message Ordering: FIFO requires a specific architecture (one exchange, one queue, one consumer).
- Lower Throughput: Slower publishing to queues compared to Kafka, making it less suitable for high-volume, big-data scenarios.

When would you use?

Complex Routing Architecture:

- Example: A travel booking system routing messages to different microservices for flight, hotel, and car rental bookings.
- RabbitMQ's dynamic exchange and binding configurations efficiently handle this scenario.

Effective Message Delivery:

- Example: An inventory management system where message delivery must follow a specific sequence to update stock levels accurately.
- RabbitMQ's push-based model guarantees message delivery.

When would you not use?

Event Stream Replays:

- Example: A social media platform analyzing user activity data to generate retrospective insights.
- Kafka's ability to retain messages and support replayability is more suitable here.

High-Throughput Data Streaming:

- Example: A video analytics platform that processes continuous streams of large surveillance video data.
- Kafka's partitioning and scalability are better suited for handling such massive data streams.

How are messages handles differently in Kafka and RabbitMQ?

Message Consumption

- RabbitMQ: Consumers passively wait for the broker to push messages (push-based).
- Kafka: Consumers actively pull messages and track their offsets (pull-based).

Message Ordering

- RabbitMQ: Sends messages in the order they are received unless interrupted by higher priority messages
- Kafka: Organizes messages by topics and partitions; consumers pull messages in potentially different orders

How are messages handles differently in Kafka and RabbitMQ?

Message Deletion

- RabbitMQ: Messages are deleted after acknowledgment (ACK) from consumers
- Kafka: Messages persist in log files until retention period expires, allowing reprocessing

Message Priority

- RabbitMQ: Supports priority queues for urgent messages (e.g., backup messages prioritized over sales transactions)
- Kafka: No priority queue; all messages are treated equally