finance

Digital Partner of Choice

Kafka vs RabbitMQ

Key Differences and Use Cases

finance
Digital Partner of Choice
www.efinance.com.eg

Overview

First, let's understand the need for message brokers like Kafka and RabbitMQ. Big data engineers or developers face challenges with successful data exchange, particularly when they have to make applications interact with each other. Message brokers solve this problem of data exchange by making it reliable and simple using various protocols for messaging that show how a message has to be transmitted and consumed at the receiver.

Apache Kafka and RabbitMQ are messaging systems used in distributed computing to handle big data streams– read, write, processing, etc. They act as the message brokers between applications/services endpoints.

Message brokers are software modules that let applications, services, and systems communicate and exchange information. Message brokers do this by translating messages between formal messaging protocols, enabling interdependent services to directly “talk” with one another, even if they are written in different languages or running on other platforms.

Message brokers validate, route, store, and deliver messages to the designated recipients. The brokers operate as intermediaries between other applications, letting senders issue messages without knowing the consumers’ locations, whether they’re active or not.

What is a messaging system?

A messaging system is a simple exchange of messages between two or more persons, devices, etc. A publish-subscribe messaging system allows a sender to send/write the message and a receiver to read that message. In Apache Kafka, a sender is known as a **producer** who publishes messages, and a receiver is known as a **consumer** who consumes that message by subscribing it.

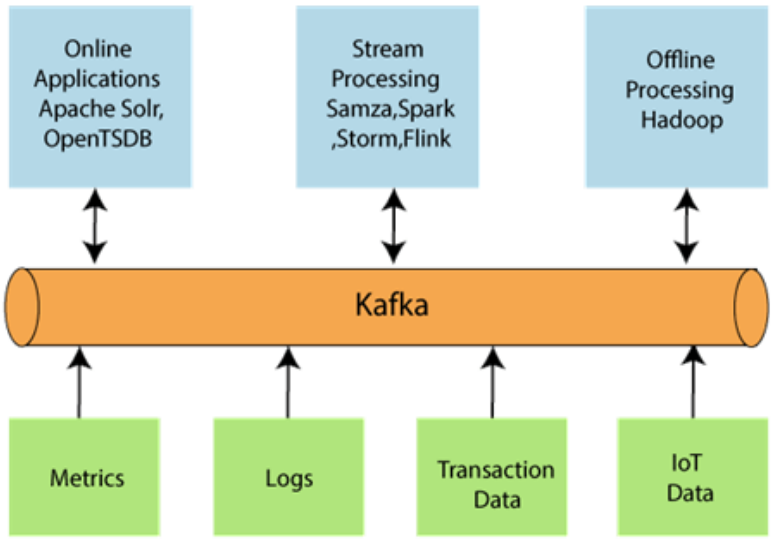
What is Kafka?

Apache Kafka is also a software platform which is based on an open-source distributed a publish-subscribe message system which let exchanging of data between applications, servers, and processors as well. Apache Kafka was originally developed by **LinkedIn**; it was released in the year 2011 which works as middle storage between two applications. The producer writes and stores the message in the Kafka cluster. On the other hand, the consumer consumes messages from the cluster. It also reduces the slow delivery of heavy messages.

What is RabbitMQ?

RabbitMQ is the most widely used, general-purpose, and open-source message broker that supports protocols including MQTT, AMQP, and STOMP. It was released in the year 2007 and was a primary component in messaging systems. Currently, it is used for streaming use cases, such as online payment processing. RabbitMQ was able to handle the background tasks or act as a message broker between microservices. It helped the web applications in reducing the loads. Also, it reduced the delivery time of the servers for those tasks or resources which were time-consuming.

Apache Kafka Architecture

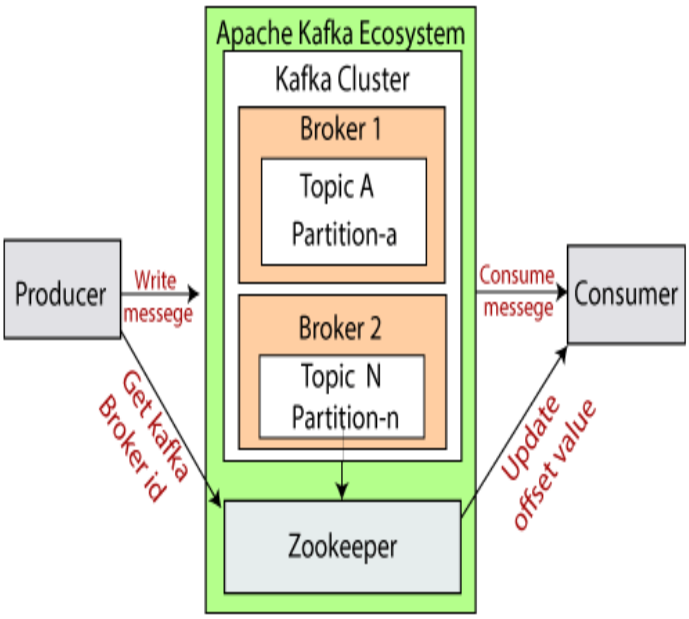


Kafka is a distributed publish-subscribe message delivery and logging system that follows a publisher/subscriber model with message persistence capability. Producers push event streams to the brokers, and consumers pull the data [their type/kind.

Kafka architecture comprises producers, consumers, clusters, brokers, topics, and partitions. Kafka is a publisher/subscriber model where events are stored inside partitions as topics. These partitions reside within the broker. And multiple brokers come together to form a cluster of brokers. A consumer can read the data and process it using the offset number. The offset number itself is written inside the partitions. The data stays in the partition for a specific time, referred to as the retention period. After which, it is deleted. Consumers can also form their clusters, and those are identified by consumer group ID. Consumers know exactly which partition to pool data from. That's the idea behind Kafka's architecture. Recurrent retrieval of data. Copies of the same topics are replicated in multiple brokers to avoid failure.

- **Kafka Cluster**: A Kafka cluster is a system that comprises of different brokers, topics, and their respective partitions. Data is written to the topic within the cluster and read by the cluster itself.

- **Producers**: A producer sends or writes data/messages to the topic within the cluster. In order to store a huge amount of data, different producers within an application send data to the Kafka cluster.



Apache Kafka Architecture

- **Consumers**: A consumer is the one that reads or consumes messages from the Kafka cluster. There can be several consumers consuming different types of data form the cluster. The beauty of Kafka is that each consumer knows from where it needs to consume the data.

- **Brokers**: A Kafka server is known as a broker. A broker is a bridge between producers and consumers. If a producer wishes to write data to the cluster, it is sent to the Kafka server. All brokers lie within a Kafka cluster itself. Also, there can be multiple brokers.

- **Topics**: It is a common name, or a heading given to represent a similar type of data. In Apache Kafka, there can be multiple topics in a cluster. Each topic specifies different types of messages.

- **Partitions**: The data or message is divided into small subparts, known as partitions. Each partition carries data within it having an offset value. The data is always written in a sequential manner. We can have an infinite number of partitions with infinite offset values. However, it is not guaranteed that to which partition the message will be written.

- **ZooKeeper**: A ZooKeeper is used to store information about the Kafka cluster and details of the consumer clients. It manages brokers by maintaining a list of them. Also, a ZooKeeper is responsible for choosing a leader for the partitions. If any changes like a broker die, new topics, etc., occurs, the ZooKeeper sends notifications to Apache Kafka. A ZooKeeper is designed to operate with an odd number of Kafka servers. Zookeeper has a leader server that handles all the writes, and rest of the servers are the followers who handle all the reads. However, a user does not directly interact with the Zookeeper, but via brokers. No Kafka server can run without a zookeeper server. It is mandatory to run the zookeeper server.

Flow Of Messages in Kafka:

Producers send in data in the form of messages to the Kafka Cluster. Messages have a Topic ID data field in them, which is used by Kafka to forward the message to the leader broker for that topic. As the data is written onto the partition in the topic, the Zookeeper saves the Offset number in a unique topic called 'offsets.' Finally, when the consumer pull request arrives, it contains an offset of the last message read a topic. Zookeeper uses this number to send data starting from the offset. It can send messages in a batch that the consumer can set.

How is fail-safe achieved in Kafka?

Here comes the role of ZooKeeper. Zookeeper maintains state logs of Kafka brokers by sending heartbeats (asking, are you okay, are you?) to the brokers. If a leader broker fails or malfunctions accidentally, Zookeeper elects a new

leader among the alive brokers. This fail-safe model comes directly from the world of Big-Data Distributed systems architecture like Hadoop.

Message Replay/Retention in Kafka

Most of the big data use cases deal with messages being consumed as they are produced. When a message is processed successfully, consumers move on to processing new messages. However, under certain circumstances, it might be necessary to replay older messages. An example of this could be a scenario where a bug in the consumer would require it to be deployed on a new version. In such circumstances, replaying of a few or all of the messages would be required. Kafka messages are durable and persistent, meaning they have a retention period before they are removed from the queue, making replaying messages easier.

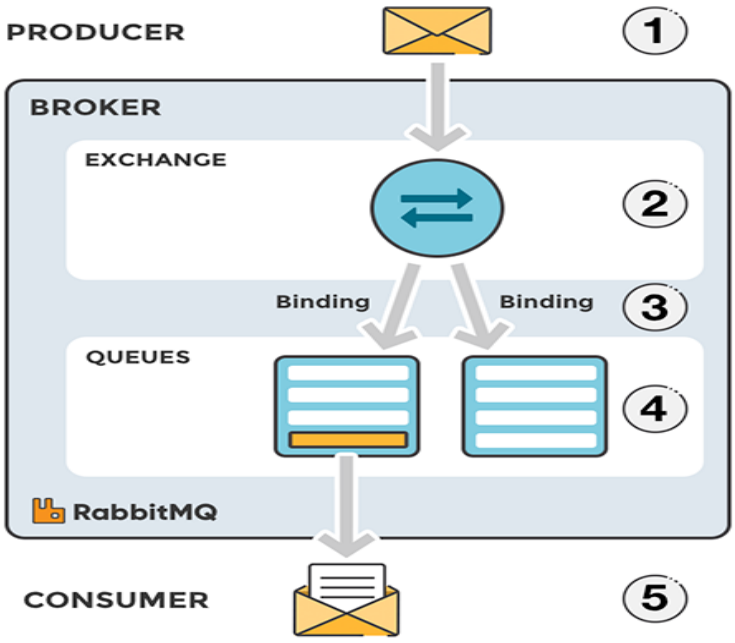
Order and offset in Kafka

Kafka doesn't provide Priority Queues, unlike RabbitMQ. It maintains the order by keeping an offset number for each message, ideally assigned by Zookeeper. The offset is a unique sequential number. The messages of a topic inside a queue are ordered by offset.

Pull Design (Smart Consumer) in Kafka

Kafka follows a pull design where consumers ask the broker for a data batch starting from a particular offset. The consumer has to keep track of the offset and do the logical operations on its end. That is why this design is also called dumb broker, the intelligent consumer.

RabbitMQ Architecture



RabbitMQ is a message broker with producer/consumer design and complex routing rules with a message delivery confirmation feature. RabbitMQ will know precisely when data fails to reach a consumer. It also lends us ways to handle delivery failure scenarios.

The RabbitMQ architecture includes producers, exchanges, queues, and consumers. A producer pushes messages to an exchange, which then routes messages to queues (or other exchanges). A consumer then continues to read messages from the queue, often up to a predetermined limit of messages.

- **Message**: A message is a form of data that is shared from the producer to the consumer. The data can hold requests, information, meta-data, etc.

- **Producer**: A producer is a user program that is responsible to send or produce messages.

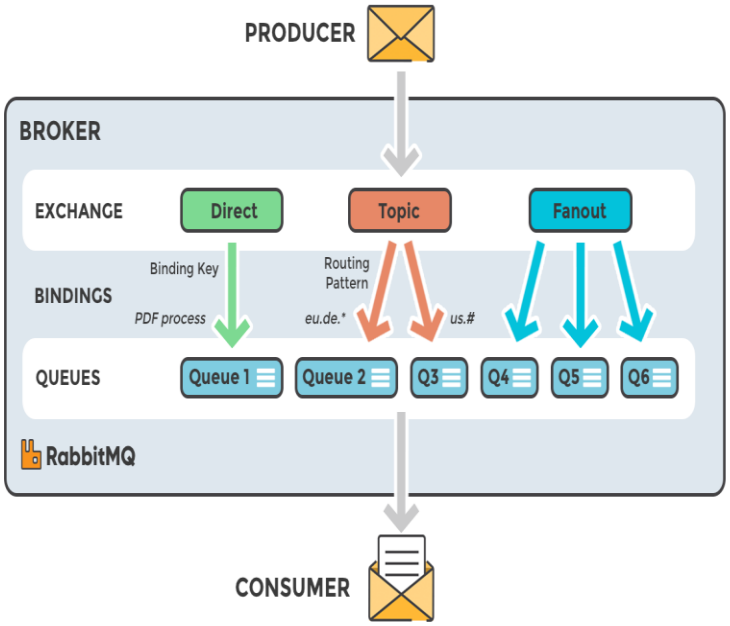
- **Exchange**: Exchanges are message routing agents, it is responsible to route the messages to different queues with the help of header attributes, bindings, and routing keys. Binding is the link between a queue and the

exchange. Whereas the Routing Key is the address that exchange uses to determine which queue the message should be send to.

- **Queue**: A queue is a mailbox or a buffer that stores messages. It has certain limitations like disk space and memory.

- **Consumer**: A consumer is a user program that is responsible to receive messages.

Messages have a header and body: The content is in the body, whereas the header contains routing-related details. Messages from the producer are sent to exchange, which then forwards the message to appropriate queues by checking the binding rules. The way the messages are sent to the correct queues is managed with routing keys and binding.



There are four types of exchanges:

- **Direct**: Messages are sent to every queue which has the same routing key.
- **Fanout**: Messages are delivered to all the queues that the exchange is connected to for broadcasting.
- **Topic**: This uses routing key as well as wildcard character – topic to select the queues that will receive the message.
- **Headers**: Match the header in the message. Either one among all headers should match (ANY), or all the headers in the message should

match (ALL); all is the default state. Headers come in arguments in messages which can contain key-value pairs.

Message deletion from the queue happens via two rules:

- **Automatic deletion**: is when the message is deleted right after the consumer has read/pulled the message.
- **Explicit deletion**: is when a consumer sends back an acknowledgment saying it has received the message. This can happen in three ways; right after the consumer receives the message, the consumer stores the message in persistent storage, after the consumer processes, and then stores the message in storage.

You can set priority for messages, and essentially, RabbitMQ queues can act as a priority queue as well. RabbitMQ follows a push design where data is pushed to the consumer from the broker side automatically. Make sure to set the pre-fetch limit, which tells the broker how many messages or what size it should push to the consumer without overwhelming it.

Kafka vs. RabbitMQ - Architectural Differences

Kafka uses a publisher/subscriber model where events are stored inside partitions as topics. These partitions reside within the broker. And multiple brokers come together to form a cluster of brokers. A consumer can read the data and process it using the offset number. The offset number itself is written inside the partitions. The data stays in the partition for a specific time, referred to as the retention period. After which, it is deleted. Consumers can also form their clusters, and those are identified by consumer group ID. Consumers know exactly which partition to pool data from. That's the idea behind Kafka's architecture. Recurrent retrieval of data. Copies of the same topics are replicated in multiple brokers to avoid failure.

RabbitMQ's architecture, unlike Kafka, has a routing mechanism/design in place. A major difference between Kafka and RabbitMQ architecture is that messages in RabbitMQ aren't supposed to persist for long, though they may be. Finally, queues are an implementation choice for the sequential ordering of messages inside the broker. These three features mainly distinguish RabbitMQ from Kafka's architecture.

Performance

Kafka offers much higher performance than message brokers like RabbitMQ. It uses sequential disk I/O to boost performance, making it a suitable option for implementing queues. It can achieve high throughput (millions of messages per second) with limited resources, a necessity for big data use cases.

RabbitMQ can also process a million messages per second but requires more resources (it needs more number of the hardware around 30 nodes). You can use RabbitMQ for many of the same use cases as Kafka, but you'll need to combine it with other tools like Apache Cassandra.

Message Lifetime

Kafka are stored based on the retention period and are deleted once the retention period is over.

RabbitMQ the consumer has to send a positive ACK (acknowledgment) message to get deleted from the queue.

Message Ordering

Kafka segregates messages using topics. An offset is an integer number maintained for each partition by Zookeeper. So, consumers that want to read from a particular topic will have to use offset to get the messages. Updating offset needs to happen after the Smart Consumer consumes every message.

RabbitMQ by design uses a queue inside the broker in its implementation. So naturally, the order is maintained inside the queue.

Message Protocol

Apache Kafka supports primitives such as int8, int16, etc. and binary messages.

RabbitMQ supports any standard queue protocols such as MQTT, AMQP, STOMP, HTTP protocols.

- AMQP: It is a default implementation in RabbitMQ.
- MQTT: used for light scenarios. Binary exchange.
- STOMP: text-based data exchange.
- HTTP: our very popular internet protocol.

Apache Kafka Use Cases

Some of the best Kafka use cases make use of the platform's high throughput and stream processing capabilities.

- **Message Broker**: Apache Kafka is one of the trending technologies that is capable to handle a large amount of similar type of messages or data. This capability enables Kafka to give high throughput value. Also, Kafka

is a publish-subscribe messaging system that makes users to read and write data more conveniently.

- **Website Activity Tracking**: it is one of the widely used use cases of Kafka. It is because a website activity usually creates a huge amount of data, generating various messages for each particular page view and user's activity. Kafka also ensures that data is successfully sent and received by both parties.

- **Kafka Stream Processing**: We have various popular frameworks that read data from a topic, process it, and write that processed data over a new topic. This new topic containing the processed data becomes available to users and applications such as Spark Streaming, Storm, etc.

- **Log Aggregation**: Several organizations make use of Apache Kafka to collect logs from various services and make them available to their multiple customers in a standard format.

- **Event sourcing:** Kafka can be used to support event sourcing, in which changes to an app state are stored as a sequence of events. So, for example, you might use Kafka with a banking app. If the account balance is somehow corrupted, you can recalculate the balance based on the stored history of transactions.

RabbitMQ Use Cases

Some of the best RabbitMQ use cases make use of its flexibility for routing messages among apps.

Legacy applications: Applications that need to support legacy protocols, such as STOMP, MQTT, AMQP, 0-9-1.

Complex routing: RabbitMQ can be the best fit when you need to route messages among multiple consuming apps, such as in a microservices architecture. RabbitMQ consistent hash exchange can be used to balance load processing across a distributed monitoring service, for example. Alternate exchanges can also be used to route a portion of events to specific services for A/B testing.

Applications that need a variety of publish/subscribe, point-to-point request/reply messaging capabilities.

Source language

Kafka written in Java and Scala, was first released in 2011.

RabbitMQ was built in Erlang in 2007.

Libraries and Language Support

Kafka supports Ruby, Python, Java, and Node.js.

RabbitMQ supports Elixir, Go, Java, JavaScript, Ruby, C, Swift, Spring, .NET, Python and PHP.

Installation of Apache Kafka

- download Apache Kafka on the system, as shown below: Use the link: https://kafka.apache.org/downloads
- Click on the link and download any binary from Binary downloads.

- Go to the respective downloads location and extract the downloaded file using WinRAR.
- Copy the Kafka folder to the root directory(local disk C).
- here are following steps to start the zookeeper
- Go to the Kafka directory and create a new folder as '**data**'.
- Open the newly created data folder and create two more folders under it. Name the folders as '**zookeeper**' and '**kafka**'.
- Now, copy the address of the zookeeper folder. After copying, go back to the Kafka directory.
- Move to the **config** folder under the Kafka directory.
- Open the config folder and go to the **zookeeper.properties** file. Right-click on the file and click on 'Edit with Notepad++' as shown below:
- Edit the value of '**datadir**' by pasting the zookeeper folder address. Save it. This address will be used for clients to get connected.
- The Zookeeper server is ready to start. Open the command prompt and go to the Kafka directory. Then, type the command: '**zookeeper-server-start.bat config\zookeeper.properties**'. Press enter.
- If the below shown output is displayed, it means the zookeeper server is successfully started. In the output screen, a port number **2181** is shown which tells the successful zookeeper server startup.
- Open data>zookeeper. A new folder, '**version-2**' will be created automatically. If not so, something went wrong.
- As the zookeeper server is started, go to the kafka directory>data>kafka. Copy the address of the kafka folder.
- Move back to the Kafka directory>config>server.properties.
- Right-click and click "Edit with Notepad++'. The below shown file will open.
- Again, paste the copied address of the kafka folder as a value of **log.dirs**.
- Open the command prompt and move to the Kafka directory. Type the command: '**kafka-server-start.bat config\server.properties**'.

Installation of RabbitMQ

- we will first need to install Erlang. Got to the Erlang downloads page and download the erlang binary file for windows which is an executable.

- Next run the binary file downloaded and install erlang on your machine.
- Go to RabbitMQ downloads page and download RabbitMQ installation.
- This will be a .exe installation file for windows.
- Run this exe and install RabbitMQ on your machine.
- We will now start Rabbit.
- to start go to the following the next
- Open the command prompt and run the following commands one by one:
- c:\>cd\
- c:\>cd Program Files
- c:\Program Files>cd RabbitMQ Server
- c:\Program Files\RabbitMQ Server>dir
- c:\Program Files\RabbitMQ Server>cd rabbitmq_server-3.8.1
- c:\Program Files\RabbitMQ Server\rabbitmq_server-3.8.1>dir
- c:\Program Files\RabbitMQ Server\rabbitmq_server-3.8.1>cd sbin
- c:\Program Files\RabbitMQ Server\rabbitmq_server-3.8.1\sbin>dir
- c:\Program Files\RabbitMQ Server\rabbitmq_server-3.8.1\sbin>.\rabbitmq-plugins enable rabbitmq_management
- and Run the next command
- .\rabbitmq-plugins.bat enable rabbitmq_management
- Open the browser and type http://localhost:15672. By default, the management plug-in runs on port 15672.
- Provide the Username and Password and click on Login button. The default username and password is guest.

References

https://tanzu.vmware.com/developer/blog/understanding-the-differences-between-rabbitmq-vs-kafka/

https://www.simplilearn.com/kafka-vs-rabbitmq-article?source=frs_left_nav_clicked

https://www.instaclustr.com/blog/rabbitmq-vs-kafka/

https://www.projectpro.io/article/kafka-vs-rabbitmq/451

https://www.projectpro.io/article/apache-kafka-next-generation-distributed-messaging-system/275

https://www.upsolver.com/blog/kafka-versus-rabbitmq-architecture-performance-use-case#:~:text=RabbitMQ%20is%20a%20general%20purpose,MQTT%2C%20AMQP%2C%20and%20STOMP.&text=Kafka%20is%20a%20durable%20message,send%20messages%20to%20a%20topic

https://www.javatpoint.com/apache-kafka-vs-rabbitmq

https://www.javatpoint.com/apache-kafka-use-cases

https://www.codeusingjava.com/boot/rabbithello

https://www.appsdeveloperblog.com/messaging-using-rabbitmq-in-spring-boot-application/

https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html