Kafka vs. RabbitMQ: Architecture, Performance & Use Cases

Eran Levy

Cloud Architecture February 1, 2022

Streaming data has many moving parts: event handling, through processing, storage, and transformation. We've partnered with O'Reilly to author a comprehensive new report to help you assess how to best modernize your data pipelines, and maximize the analytics value of your modern data. The report is available for free for a limited time – grab your copy here.

If you're considering whether Kafka or RabbitMQ is best for your use case, read on to learn about the different architectures and approaches behind these tools, how they handle messaging differently, and their performance pros and cons. We'll cover the best use case for each of the tools, and when it might be preferable to rely on a full end-to-end stream processing solution.

Table of Contents

1. The Basics: What are Apache Kafka and RabbitMQ?
2. Kafka vs RabbitMQ – Differences in Architecture
3. Pull vs Push Approach
4. How Do They Handle Messaging?
5. Kafka vs RabbitMQ Performance

6. What are the Best Use Cases?
7. Kafka and RabbitMQ: Summing Up
8. You have streaming data in Kafka. What's next?

The Basics: What are Apache Kafka and RabbitMQ?

Apache Kafka and RabbitMQ are two open-source and commercially-supported pub/sub systems, readily adopted by enterprises. RabbitMQ is an older tool released in 2007 and was a primary component in messaging and SOA systems. Today it is also being used for streaming use cases. Kafka is a newer tool, released in 2011, which from the onset was built for streaming scenarios.

What is RabbitMQ? RabbitMQ is a general purpose message broker that supports protocols including MQTT, AMQP, and STOMP. It can deal with high-throughput use cases, such as online payment processing. It can handle background jobs or act as a message broker between microservices.

What is Apache Kafka? Kafka is a message bus developed for high-ingress data replay and streams. Kafka is a durable message broker that enables applications to process, persist, and re-process streamed data. Kafka has a straightforward routing approach that uses a routing key to send messages to a topic.

Kafka vs RabbitMQ – Differences in Architecture

RabbitMQ Components Architecture

The main components of RabbitMQ can be seen from the diagram below:

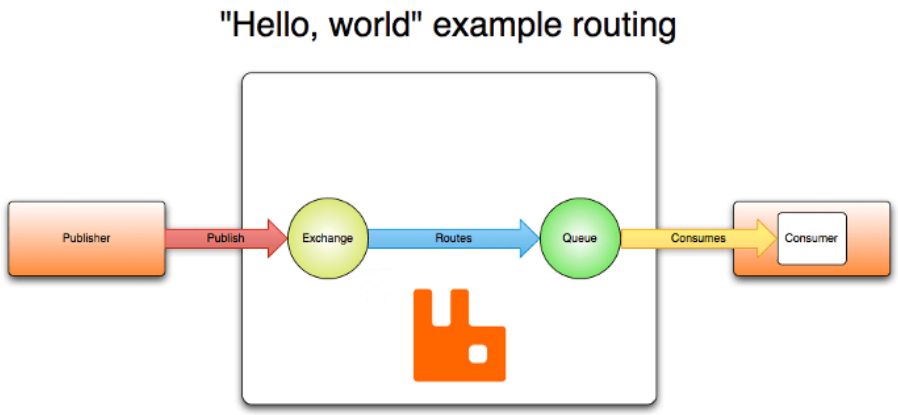


Image source: https://www.rabbitmq.com/tutorials/amqp-concepts.html

As with other message brokers, RabbitMQ receives messages from applications that publish them – known as producers or publishers. Within the system, messages are received at an exchanges – a virtual 'post-office' of sorts, which routes messages onwards to storage buffers known as queues. Applications that read messages, known as consumers, can subscribe to these queues to pick up the latest data that arrives in the 'mailboxes'.

The key features of RabbitMQ are:

General purpose message broker—uses variations of request/reply, point to point, and pub-sub communication patterns.

Smart broker / dumb consumer model—consistent delivery of messages to consumers, at around the same speed as the broker monitors the consumer state.

Mature platform—well supported, available for Java, client libraries, .NET, Ruby, node.js. Offers dozens of plugins.

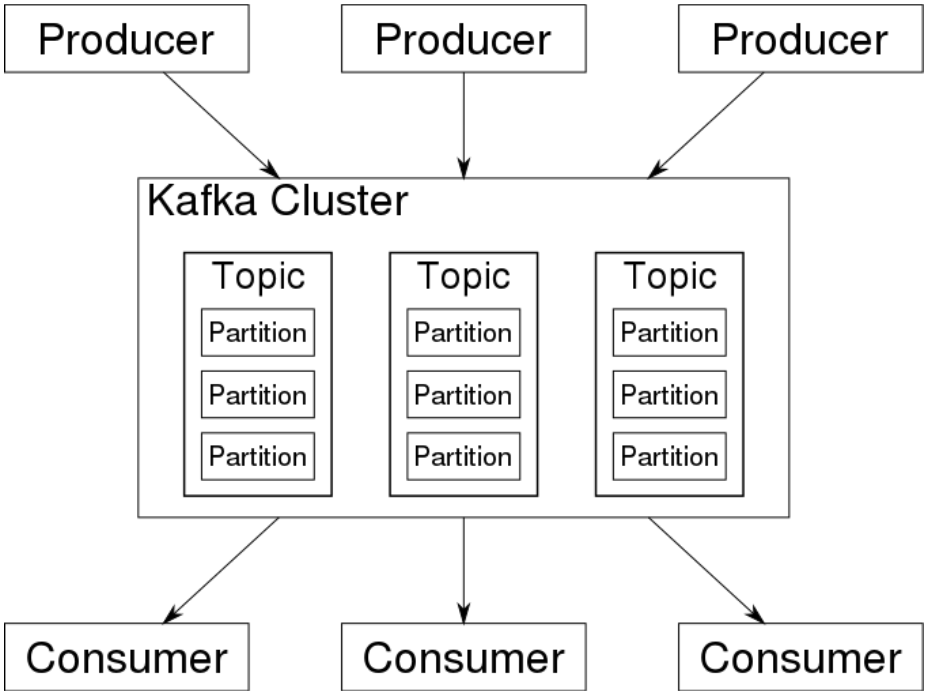
Communication—can be synchronous or asynchronous.

Deployment scenarios—provides distributed deployment scenarios.

Multi-node cluster to cluster federation—does not rely on external services, however, specific cluster formation plugins can use DNS, APIs, Consul, etc.

Apache Kafka Architecture

You can see the main components of a Kafka cluster below:



Producers and consumers are the same here – applications that publish and read event messages, respectively. As we've covered when we discussed Kafka use cases, an event is a message with data describing the event, such as a new user signing up to a mobile application. Events are queued in Kafka topics, and multiple consumers can subscribe to the same topic. Topic are further divided into partitions, which split data across brokers to improve performance.

Important features of Kafka include:

High volume publish-subscribe messages and streams platform—durable, fast, and scalable.

Durable message store—Kafka behaves like a log, run in a server cluster, which keeps streams of records in topics (categories).

Messages are made up of a value, a key, and a timestamp.

Dumb broker / smart consumer model—does not try to track which messages are read by consumers and only keeps unread messages. Kafka keeps all messages for a set period of time.

Managed by external services—in many cases this will be Apache Zookeeper.

Pull vs Push Approach

One important difference between Kafka and RabbitMQ is that the first is pull-based, while the other is push-based. In pull-based systems, the brokers waits for the consumer to ask for data ('pull'); if a consumer is late, it can catch up later. With push-based systems, messages are immediately pushed to any subscribed consumer. This can cause these two tools to behave differently in some circumstances.

Apache Kafka: Pull-based approach

Kafka uses a pull model. Consumers request batches of messages from a specific offset. Kafka permits long-pooling, which prevents tight loops when there is no message past

the offset, and aggressively batches messages to support this

A pull model is logical for Kafka because of partitioned data structure. Kafka provides message order in a partition with no contending consumers. This allows users to leverage the batching of messages for effective message delivery and higher throughput.

RabbitMQ: Push-based approach

RabbitMQ uses a push model and stops overwhelming consumers through a prefetch limit defined on the consumer. This can be used for low latency messaging..

The aim of the push model is to distribute messages individually and quickly, to ensure that work is parallelized evenly and that messages are processed approximately in the order in which they arrived in the queue. However, this can also cause issues in cases where one or more consumers have 'died' and are no longer receiving messages.

How Do They Handle Messaging?

Tool	Apache Kafka	RabbitMQ
Message ordering	provides message ordering thanks to its partitioning. Messages are sent to topics by message key.	Not supported.

Message lifetime	Kafka is a log, which means that it retains messages by default. You can manage this by specifying a retention policy.	RabbitMQ is a que are done away wi and acknowledgm
Delivery Guarantees	Retains order only inside a partition. In a partition, Kafka guarantees that the whole batch of messages either fails or passes.	Doesn't guarantee relation to transac single queue.
Message priorities	N/A	In RabbitMQ, you message priorities message with hig

Kafka vs RabbitMQ Performance

Apache Kafka:

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:

RabbitMQ can also process a million messages per second but requires more resources (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.

What are the Best Use Cases?

Apache Kafka Use Cases

Apache Kafka provides the broker itself and has been designed towards stream processing scenarios. Recently, it has added Kafka Streams, a client library for building applications and microservices. Apache Kafka supports use cases such as metrics, activity tracking, log aggregation, stream processing, commit logs, and event sourcing.

The following messaging scenarios are especially suited for Kafka:

- Streams with complex routing, throughput of 100K/sec events or more, with “at least once” partitioned ordering.
- Applications requiring a stream history, delivered in “at least once” partitioned ordering. Clients can see a “replay” of the event stream.
- Event sourcing, modeling changes to a system as a sequence of events.
- Stream processing data in multi-stage pipelines. The pipelines generate graphs of real-time data flows.

RabbitMQ Use Cases

RabbitMQ can be used when web servers need to quickly respond to requests. This eliminates the need to perform resource-intensive activities while the user waits for a result. RabbitMQ is also used to convey a message to various recipients for consumption or to share loads between workers under high load (20K+ messages/second).

Scenarios that RabbitMQ can be used for:

- Applications that need to support legacy protocols, such as STOMP, MQTT, AMQP, 0-9-1.
- Granular control over consistency/set of guarantees on a per-message basis
- Complex routing to consumers
- Applications that need a variety of publish/subscribe, point-to-point request/reply messaging capabilities.

Kafka and RabbitMQ: Summing Up

This guide has covered the major differences and similarities between Apache Kafka and RabbitMQ. Both can consume several million messages per second, though their architectures differ, and each performs better in certain environments. RabbitMQ controls its messages almost in-memory, using a big cluster (30+ nodes). Comparatively, Kafka leverages sequential disk I/O operations and thus demands less hardware.

To learn more about the differences between Kafka and other message brokers, check out our guide to Kafka vs Kinesis.

You have streaming data in Kafka. What's next?

Getting your data in Kafka is just the first step – to actually drive value from it, you need a way to easily store, manage and analyze your streams. To learn how that's achieved,

check out our popular post on Apache Kafka with or without a Data Lake; or start a free trial of Upsolver to start seeing value from your Kafka streams today.

Published in: Blog , Cloud Architecture

Eran Levy

Eran is a director at Upsolver and has been working in the data industry for the past decade - including senior roles at Sisense, Adaptavist and Webz.io. His writing has been featured on Dzone, Smart Data Collective and the Amazon Web Services big data blog. Connect with Eran on LinkedIn