Use Cases

May 7, 2019 2:42:07 PM / by Eran Levy

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.

In this page:

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.

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.

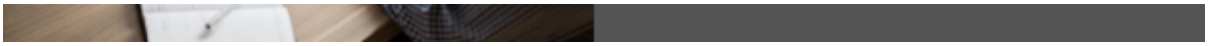
Kafka vs RabbitMQ - Differences in Architecture

RabbitMQ Architecture

- **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

- **High volume publish-subscribe messages and streams platform**—durable, fast and scalable.
- **Durable message store**—like a log, run in a server cluster, which keeps streams of records in topics (categories).
- **Messages**—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.
- **Requires external services to run**—in some cases Apache Zookeeper.



Pull vs Push Approach

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.

A pull model is logical for Kafka because of its partitions. 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.

How Do They Handle Messaging?

	always there. You can manage this by specifying a message retention policy.	with once consumed, and acknowledgment is provided.
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 atomicity, even in relation to transactions involving a single queue.
Message Priorities	N/A	In RabbitMQ, you can specify message priorities, and consumed message with high priority at the onset.

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:

- 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 millions of messages per second, though their architectures differ, and each performs better in certain environments. RabbitMQ controls its messages almost in-memory, using big cluster (30+ nodes). Comparatively, Kafka leverages sequential disk I/O operations and thus demands less hardware.

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

Written by Eran Levy

Director of Marketing at Upsolver

Upsolver

About Us

Why Upsolver

Customers

Leadership

Careers

Support

Contact

Product

Product Overview

Data Ingestion

Integrations

Deployment

Pricing

Upsolver vs Alternatives

Schedule a Demo

Resources