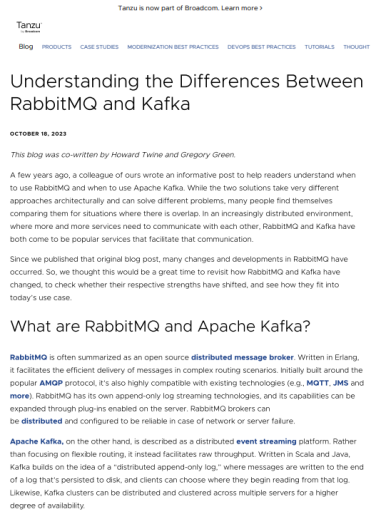
Understanding the Differences Between RabbitMQ and Kafka



Tanzu is now part of Broadcom. Learn more >

Tanzu

Blog PRODUCTS CASE STUDIES MODERNIZATION BEST PRACTICES DEVOPS BEST PRACTICES TUTORIALS THOUGHT

Understanding the Differences Between RabbitMQ and Kafka

OCTOBER 18, 2023

This blog was co-written by Howard Twine and Gregory Green.

A few years ago, a colleague of ours wrote an informative post to help readers understand when to use RabbitMQ and when to use Apache Kafka. While the two solutions take very different approaches architecturally and can solve different problems, many people find themselves comparing them for situations where there is overlap. In an increasingly distributed environment, where more and more services need to communicate with each other, RabbitMQ and Kafka have both come to be popular services that facilitate that communication.

Since we published that original blog post, many changes and developments in RabbitMQ have occurred. So, we thought this would be a great time to revisit how RabbitMQ and Kafka have changed, to check whether their respective strengths have shifted, and see how they fit into today's use case.

What are RabbitMQ and Apache Kafka?

RabbitMQ is often summarized as an open source **distributed message broker**. Written in Erlang, it facilitates the efficient delivery of messages in complex routing scenarios. Initially built around the popular **AMQP** protocol, it's also highly compatible with existing technologies (e.g., **MQTT**, **JMS** and **more**). RabbitMQ has its own append-only log streaming technologies, and its capabilities can be expanded through plug-ins enabled on the server. RabbitMQ brokers can be **distributed** and configured to be reliable in case of network or server failure.

Apache Kafka, on the other hand, is described as a distributed **event streaming** platform. Rather than focusing on flexible routing, it instead facilitates raw throughput. Written in Scala and Java, Kafka builds on the idea of a "distributed append-only log," where messages are written to the end of a log that's persisted to disk, and clients can choose where they begin reading from that log. Likewise, Kafka clusters can be distributed and clustered across multiple servers for a higher degree of availability.

A few years ago, a colleague of ours wrote an informative post to help readers understand when to use RabbitMQ and when to use Apache Kafka. While the two solutions take very different approaches architecturally and can solve different problems, many people find themselves comparing them for situations where there is overlap. In an increasingly distributed environment, where more and more services need to communicate with each other, RabbitMQ and Kafka have both come to be popular services that facilitate that communication.

Since we published that original blog post, many changes and developments in RabbitMQ have occurred. So, we thought this would be a great time to revisit how RabbitMQ and Kafka have changed, to check whether their respective strengths have shifted, and see how they fit into today's use case.