Understanding the Differences Between RabbitMQ vs Kafka

BRIAN MCCLAIN NOVEMBER 16, 2020

Three years ago, a colleague of mine wrote a post to help readers understand when to use RabbitMQ and when to use Apache Kafka, which many found to be very useful. While the two solutions take very different approaches architecturally and can solve very different problems, many find themselves comparing them for overlapping solutions. 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.

It has been three years since that post was published, however, which in technology can be lifetime. We thought this would be a great opportunity to revisit how RabbitMQ and Kafka have changed, check if their 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, while 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.

RabbitMQ vs. Kafka

While they're not the same service, many often narrow down their messaging options to these two, but are left wondering which of them is better. I've long believed that's not the correct question to ask. Instead, you want to focus on what each service excels at, analyze their differences, and then decide which of the two best fits your use case. Even outside of the features of either service, you should also take into consideration the skills needed to operate the services and the developer communities around them.

Requirements and Use Cases

When the initial blog post was written, there was a pretty clear-cut difference in design between RabbitMQ and Kafka, and as such, a difference in use cases. RabbitMQ's message broker design excelled in use cases that had specific routing needs and per-message guarantees, whereas Kafka's append-only log allowed developers access to the stream history and more direct stream processing. While the Venn diagram of use cases these two technologies could fulfill was very tight, there were scenarios in which one was a demonstrably better choice than the other.

Work is currently underway that will alter that balance, however. While RabbitMQ will continue to offer its traditional queue model, it will also introduce a new data structure modeling an append-only log, with non-destructive consuming semantics. This new data structure will work much like Kafka's persistent log, and will be an exciting addition for RabbitMQ users looking to expand their streaming use case. And while this feature will be compatible with the AMQP protocol, it will also introduce a binary-based stream protocol.

Developer Experience

The developer experience of the two services has largely remained the same, with the list of clients and libraries continuing to grow thanks to the work of their respective communities. Both RabbitMQ's and Kafka's client library lists have seen steady growth. As more languages and frameworks have grown in popularity, finding a well-supported and complete library for either service has become easier.

One thing to note is the growth of Kafka Streams, a client library implementation that makes it easier for developers to process streaming data. It's used for the common use case of reading data from Kafka, processing it, and writing it to another Kafka queue. Additionally, ksqlDB is well worth checking out for developers looking to build streaming applications while taking advantage of their familiarity with relational databases.

A similar thing can be accomplished with RabbitMQ with the help of some other pieces, such as Spring Cloud Data Flow. Furthermore, note the upcoming streaming changes coming for RabbitMQ mentioned in the previous section, keeping in mind that this can open new ways of interacting with RabbitMQ for the developer.

Security and Operations

As noted in the initial post, RabbitMQ ships with a useful administration interface to manage users and queues, while Kafka relies on TLS and JAAS. Whether you choose RabbitMQ or Kafka will of course depend on your specific requirements and your use case, but most security scenarios can have a proper conclusion with either technology.

It's important to note the rise of Kubernetes over the last few years and how it affects the operations of services. Substantial work has been done to allow infrastructure operators to run both RabbitMQ and Kafka on Kubernetes. The RabbitMQ operator and Kafka Helm chart both have very fine control over how these services are configured as well as how to run them on Kubernetes specifically. This makes it extremely easy to get up and running with both of them configured and clustered out of the box.

Performance

Performance, as was also noted in the initial post, can be hard to quantify with so many variables coming into play, including how the service is configured, how your code interacts with it, and of course the hardware it's running on. Everything from network to memory and disk speed can dramatically impact the performance of the service. Of course, both RabbitMQ and Kafka optimize for performance, but you should also make sure your use case leverages them to maximize efficiency.

For RabbitMQ, there are some great how-to resources about maximizing performance, such as how to benchmark and size your cluster. These guides detail best practices for how to configure your clusters and how your code should interact with them for the best performance possible. Much of this advice revolves around things like managing queue size and connections, and being careful about how your client consumes messages. The RabbitMQ clustering guide also includes things to keep in mind when building a cluster.

Likewise, Confluent has a great Running Kafka in Production guide that covers many of the same concerns for when you're building the hardware that will run your Kafka cluster, as well as how you configure the cluster itself. There are a couple of things you'll need to keep in mind since Kafka runs on top of the JVM, but it does a great job of pointing those out.

If you're interested in raw numbers, both the RabbitMQ team and the Confluent team have recently put out their respective benchmarks. Both include a lot of details on how the clusters were configured and the workload that was placed on them, so make sure you take that information into consideration when reading the results. Use case and operations should significantly factor into your decision as well.

Making the Call

Deciding whether to use RabbitMQ or Kafka was never easy, and with both technologies improving every day, the margins of advantage have only gotten smaller. The decision you make will depend on your individual scenario. Make use of the knowledge contained both in this post and the original one and apply it to the familiarity you have with your use case along with any proof of concepts.

Learn More

If you would like to take a deeper dive on this topic, check out this video from SpringOne to see when one might be a better fit over the other.

If you're new to messaging services in general, a great place to start learning is with this video on event-driven architectures. If you're a Spring developer, make sure to check out our guides to get started with RabbitMQ, Kafka, and Spring Cloud Stream.

And if you would like to know more about the VMware Tanzu RabbitMQ offering, check out this page or contact us.

Frequently Asked Questions

What is the difference between Kafka and RabbitMQ

When should you use Kafka vs RabbitMQ?

Can Kafka and RabbitMQ be deployed on Kubernetes?

Should you use Kafka or RabbitMQ for microservices?

Is Kafka higher performance than RabbitMQ?

What are the different use cases for Kafka vs. RabbitMQ?

Report an issue