A LOOK AT ENTERPRISE MESSAGING

RTInsights.com
Accelerate Your Business with Real-Time Insights

IBM

TABLE OF CONTENTS

Chapter 1:
An Introduction to Enterprise Messaging

Enterprise messaging is essential for building fully connected and scalable modern applications. It uses standard methodologies to send messages between computer systems. That enables applications built on loosely coupled architectures to use information (such as a generated event) to be consumed by different elements of an application when that information is needed.

A common example cited of when and how enterprise messaging is used centers on airline travel. If a passenger changes a flight, that event has an impact on many other systems and applications. For example, it is likely that the traveler arranged transportation to the departing airport, a rental car and hotel room at his destination, and perhaps made dinner reservations for the night of arrival.

In the past, each of those things would need to be changed manually. But with modern travel systems, that one event, the flight change, could be shared with multiple systems. When each system gets the new information, it can automatically make adjustments to accommodate a traveler's new plans. How can that be accomplished?

One approach would be to set up point-to-point connections between the airline reservation system and each of the other systems that relate to that person's travels. As one would imagine, that approach is not practical. Every one-to-one connection would need to be independently managed. Such an approach would not scale well at all.

Enterprise messaging can help address this problem. An enterprise messaging system could use structured messages (such as using XML or JSON) and middleware that handles the message exchange. Such systems must offer a number of features, including:

- **Security:** Messages must be protected if they travel over public networks. Additionally, messages must be authenticated or digitally signed for the receiver to have confidence that the messages have not been tampered with in transit.
- **Routing:** Messages need to be routed efficiently from the sender to the receiver.
- **Subscription capabilities:** Systems should be able to subscribe to all messages that match a specific pattern.

What's changed?

For years, enterprises have had many options for the underlying technology that powers enterprise messaging. Systems have been built based on service-oriented architecture (SOA), open-source enterprise service bus (ESB), message brokers, Web service and REST, and more. Each was appropriate for specific scenarios.

But two relatively recent trends have changed the requirements for enterprise messaging. And that has narrowed the choice of technologies that can help businesses today.

The first change is the widespread adoption of cloud-native development for applications. A cloud-native approach lets businesses build and run scalable applications in modern, dynamic environments, including on private, public, and hybrid clouds. The approach uses containers, microservices, service meshes, immutable infrastructures, and declarative APIs. They enable businesses to build loosely coupled systems that are resilient and manageable.

The other change is that real time is the name of the game. Rather than dealing with discrete messages or events, many modern applications make use of message and event streams.

Combined, these two changes (the embracement of cloud-native technologies and a move to real-time operations) place stringent requirements on the intermediary technology.

Depending on the data being produced and the way it is consumed, different applications need different approaches. In one case, a traditional message broker, which deletes messages once they are received by a consumer, might be a good fit. In other cases, a solution based on event streams, which retains records of events allowing them to be replayed, would be best. Compared to systems that work with one-off messages, systems that support streaming handle a continuous flow of information.

Specifically, due to the way modern applications are created and dependent on a variety of data types from multiple sources, integration becomes a critical issue. What is needed is technology that can meld and make use of the various data streams. That essential enterprise messaging technology is messaging middleware.

Chapter 2:
The Role and Benefits of MQ

A modern application or digital platform may have billions of messages flowing through it each day, with real-time updates considered the standard by customers and enterprises. Ensuring that messages are not duplicated or lost in the process is an arduous task and one that is the focus of IBM MQ, an enterprise-grade messaging solution that has been on the market for decades.

It offers enterprise-class features that other MQ offerings do not. With IBM MQ, messages are delivered once-and-once-only, which is key to avoiding duplication or loss. If the server is not ready to receive a message, MQ will wait until it is through its Message Queue Interface. Other messaging-orientated middleware products may send regardless of server activity, which results in loss, while others may send a follow-up message if the original isn't received, which can sometimes cause duplicate results.

Another value of the queueing system is for programs that run independently of each other and may not be online all the time, such as Internet of Things (IoT) devices and smart sensors. This reduces the amount of time sensors need to be online, potentially improving hardware life cycles.

IBM MQ can be deployed for multiple modes of operation, such as:

- **Point-to-point:** Sender sends to queue and receiver receives in a queue.
- **Publish/subscribe:** The application publishes a message which is then sent out to all subscribers of the application.
- **File transfer:** Managed File Transfer manages the transfer of files into messages.
- **Telemetry:** IBM has its own MQ Telemetry solution that extends the MQ environment to IoT and other devices that sit at the edge of a network.

Security is applied to each message sent using Transport Layer Security (TLS). IBM also embeds Advanced Message Security (AMS) in its MQ Advanced for those that need it. AMS encrypts messages both at rest and while moving. (TLS only encrypts messages while moving.)

IBM MQ supports a wide range of applications, systems, and services and can enable organizations to send data from on-premises to the cloud, providing real-time data back to head offices on all types of data points, such as application usage, warehouse stock, and transaction volume.

Differentiators

There are other MQ and message streaming solutions on the market. Some of the other more commonly used ones include RabbitMQ and Amazon MQ.

RabbitMQ is one of the most popular open-source message brokers. It is lightweight and easy to deploy on-premises and in the cloud. It supports multiple messaging protocols. RabbitMQ can be deployed in distributed and federated configurations to meet high-scale, high-availability requirements. It is the original implementation of the AMQP protocol and is based on an architecture where the messages are queued on the central nodes before sending them to the destination.

Similarly, Amazon MQ is often described as a managed message broker service for Apache ActiveMQ. As such, it makes it easy to set up and operate message brokers in the cloud. It is configurable to allow different broker network architectures to be designed based on messaging, performance, and high availability needs.

There are varieties of MQ offerings. Most are based on open-source technology. As with any open-source solution, that means there are pluses and points to consider. On the plus side, a user benefits from the vibrancy of an open-source community. Typically, any issue that is found or new capability that is needed is quickly addressed and added. On the points to consider side, the business is responsible for many aspects of deployment and management. That is time-consuming, and many businesses may lack the internal expertise to use such systems.

On the other hand, IBM MQ is enterprise-grade messaging. It helps simplify the integration of diverse applications and business data across multiple platforms. It includes enterprise-grade messaging capabilities that ensure the secure and reliable movement of information.

Chapter 3:
What About Kafka?

Businesses that seek to be real-time enterprises are increasingly turning to Apache Kafka to support data streaming. It is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

Kafka works well as an alternative to a traditional message broker, solving the problem in a different way. It does not replace an existing message broker; it complements it. To that point, message brokers are used for a variety of reasons, such as to decouple processing from data producers, to buffer unprocessed messages, and more). In comparison to most messaging systems, Kafka has better throughput, built-in partitioning, replication, and fault tolerance, which makes it a good solution for large-scale message processing applications.

Benefits abound

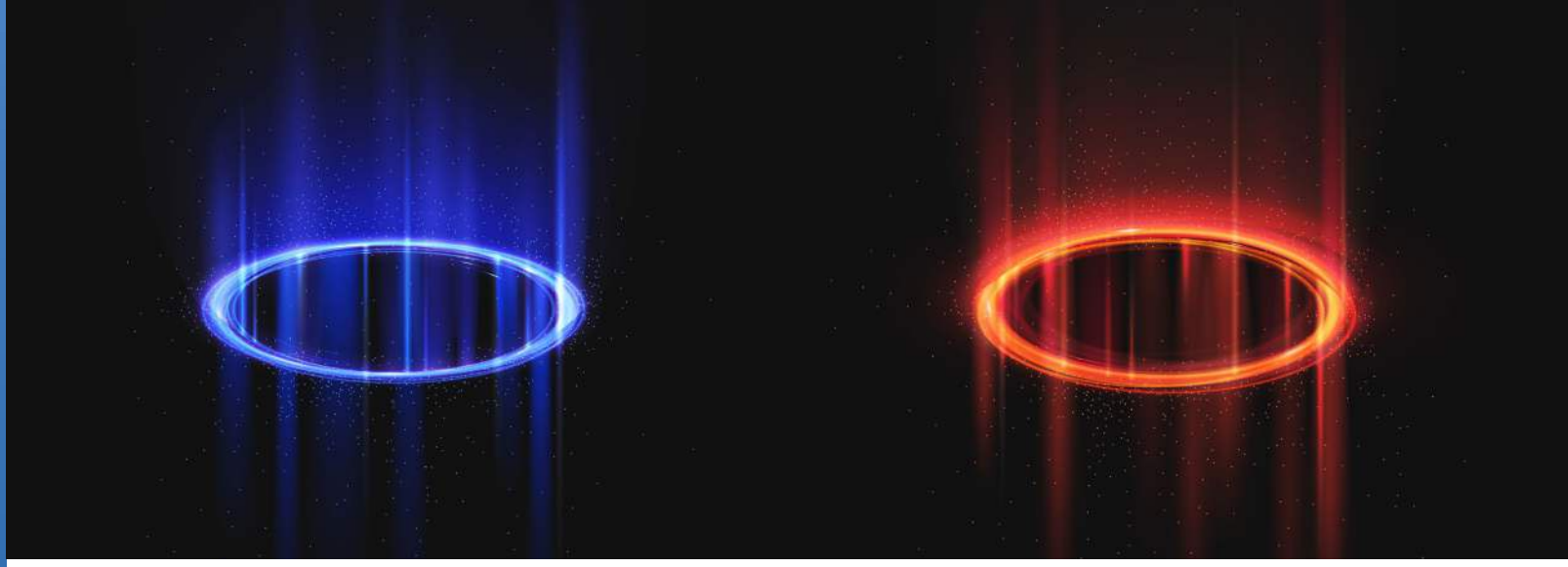
Kafka offers several benefits for businesses seeking to make use of data streaming. It is able to support high-velocity and high-volume data. That translates into high message throughput, which is needed in many modern applications. Additionally, it is capable of handling those messages with very low latency (in the range of milliseconds). Again, this is required in most real-time apps.

Kafka offers fault tolerance in that it is inherently able to overcome node/machine failure within a cluster. It provides message durability in that messages are never lost, and they can be replicated. And Kafka is highly scalable. Additional nodes can be added without incurring downtime. Furthermore, the distributed architecture of Kafka makes it scalable using capabilities like replication and partitioning.

Many businesses turn to Kafka as an alternative to traditional message brokers. It can handle thousands of messages per second and in low latency conditions with high throughput. In addition, it permits the reading and writing of messages into it at high concurrency. General use cases and application areas where Kafka is often used include:

- **Stream processing:** Many Kafka users process data in processing pipelines consisting of multiple stages where raw input data is consumed from Kafka and then aggregated, enriched, or otherwise transformed for further consumption or follow-up processing.
- **Event sourcing:** Event sourcing is a style of application design where state changes are logged as a time-ordered sequence of records. Kafka's support for very large stored log data makes it well-suited for use as a backend for an application built in that style.
- **Commit log:** Kafka can serve as a kind of external commit log for a distributed system. The log helps replicate data between nodes and acts as a re-syncing mechanism for failed nodes to restore their data.

While Kafka has the ability to scale horizontally and vertically, there are architectural choices that must be considered, especially when looking at capacity and surges in capacity. For example, financial market data systems have had to address this for many years. Microbursts in trading can cause large volumes of data that need to be delivered with very low latency. The underlying architecture must handle microbursts. Kafka allows partitioning across nodes and nodes to be added to help scale, as more throughput is needed.



Chapter 4:
MQ and Kafka: Why it is Not Really One Versus the Other

The demand to modernize and create new applications is outpacing developer resources in many companies. One approach to allay that issue is to give business users the capability to carry out the needed integrations at the heart of such development efforts. A critical aspect of such enablement is reusable integration, and that is part of broader industry trends.

Many organizations are turning to low-code/no-code development methodologies that help non-technical staff graphically build applications. Often these technologies make use of composable elements that are reusable blocks of code that perform a simple function. For instance, a composable element might be an API that exposes a specific service on a legacy system or delivers uniform access to data in a relational database. Reusable elements can also include connectors that provide context-rich access to commonly used applications, databases, and services.

The concept of reusable elements can be extended to reusable integration templates. For example, an integration template would offer a set approach to provide predefined entities and field mappings to enable the flow of data from source to destination. It might simply be a standard way to move data from point A to point B, or it could also transform the data from the source so that it could be used in a destination application.

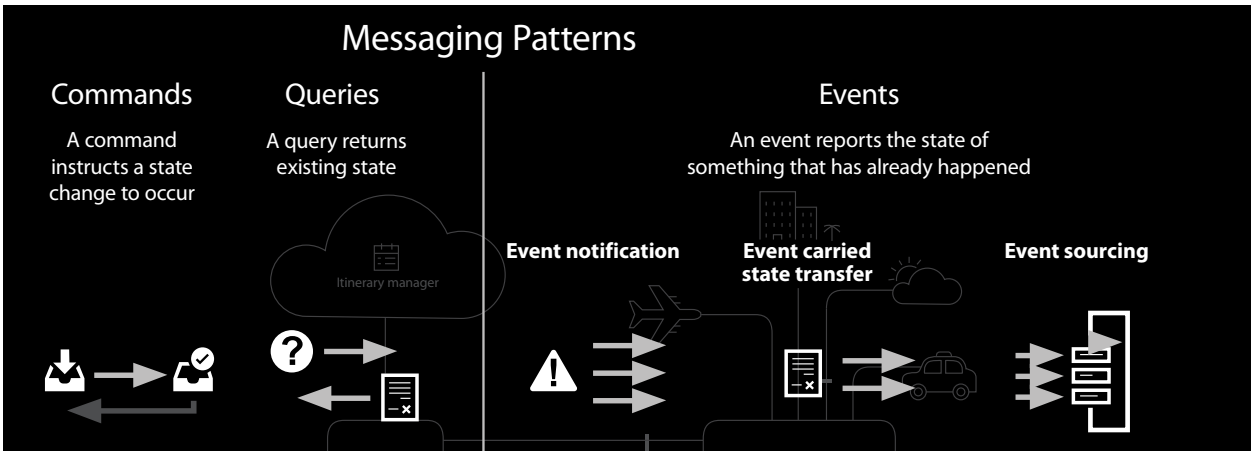
Any reusable asset, whether it is an API or a template, gives users in different business units a company-approved and standardized way to perform functions with only a simple configuration required instead of a bespoke creation. One immediate benefit is that a company does not have different business units reinventing the wheel over and over every time the same connectivity or data sharing is done.

Using IBM MQ versus Kafka is often couched as an either/or. However, in most cases, it is not a hard black-and-white choice of one or the other. Each has its sweet spot, depending on the application. And in many scenarios, a case could be made for using them together. In fact, there are synergies when IBM MQ is used alongside Kafka. Let's take a deeper look.

In general, Kafka is often used for applications that demand high throughput. On the other hand, IBM MQ is best suited for applications with similar requirements that also require a high level of reliability and cannot tolerate message loss.

But that is a gross simplification. A better way to analyze the situation is to look at the **messaging patterns** that an application must support.

Messaging patterns can be put into three general categories: commands, queries, and events. An application using each has very different requirements.



Commands: A command instructs a state change to occur. For example, make this purchase, deduct this money from my account, and book this flight. In such systems, messages must get through, no questions asked. And the system must be secure, distributed, and support queued workloads.

Queries: A query returns an existing state. For example, where is my order, what is my balance, and what time is my flight? A system that uses queries must be real-time, highly available, secure, scalable, and simple to use.

Events: An event reports the state of something that has already happened or changed. For example, when was my order shipped, what day was my check deposited, and when did a flight depart? An event might also include a change of state. For example, when did a parameter cross a threshold, or when did a static measurement change?

Unlike commands and queries, there are multiple uses for events, each with different messaging patterns and requirements to support those patterns. These patterns include:

- **Event notification:** A system supporting event notification must be real-time, scalable, and lightweight. It must be simple to use, invisible to the application, and highly available.
- **Event-carried state transfer:** A system supporting this messaging pattern must be scalable and lightweight. It must also be simple to use and secure.
- **Event sourcing:** A system supporting event sourcing must maintain an ordered projection of the data for efficient, repeatable processing. It must also be secure and scale for high data retention.

Obviously, there is some overlap in requirements across all the various messaging patterns. And different features and capabilities of IBM MQ and Kafka stretch across those various requirements. That makes each suitable for different messaging patterns.

IBM MQ is ideal for messaging patterns that include commands, queries, event notifications, and event-carried state transfer. Kafka is well matched to messaging patterns that support event sourcing, event-carried state transfer, and in some cases, event notification.

So, there are messaging patterns such as event-carried state transfer and event notification where both IBM MQ and Kafka are useful. And other messaging patterns where one or the other is best.

Another way to determine when to use IBM MQ or Kafka is to examine architectural patterns. What is important is how messages and events are being used by an application or system.

IBM MQ is a good fit in systems where messages and events are used for communication and notification. Such applications and systems rely on messages and events to communicate between services in real time, not just within single applications but across organizations and between them.

Kafka is best for systems that use events for data persistence. In such systems, events represent past state changes. Retaining a record of that change enables replay and blurs the line between the role of messaging and data storage.

Basically, IBM MQ and Apache Kafka are both messaging technologies that take different approaches to satisfy different requirements.

Closing thoughts

Enterprise messaging is a critical aspect of real-time operations. However, different real-time systems and applications have different messaging requirements. IBM MQ and Apache Kafka offer distinct features and capabilities that match specific requirements.

Apache Kafka is designed for a set of high-throughput, partitioned topics, where applications are implemented to follow an event streaming ordered pattern. Deviating from that can result in losing many of Kafka's performance and availability benefits.

IBM MQ is designed to support a more diverse set of application scenarios than Kafka. It supports a broader mix of messaging patterns, styles, and requirements.

Depending on an application's needs, one may be preferred over the other. But in most cases, there can be a need for both. And when that is the case, help is available. In particular, businesses can integrate IBM MQ with Event Streams Kafka as part of Cloud Pak for Integration using Kafka Connect with IBM MQ Kafka Connectors. In such a deployment, businesses can get the best of both technologies as needed by their applications.

RTInsights is an independent, expert-driven web resource for senior business and IT enterprise professionals in vertical industries. We help our readers understand how they can transform their businesses to higher-value outcomes and new business models with AI, real-time analytics, and IoT. We provide clarity and direction amid the often-confusing array of approaches and vendor solutions. We provide our partners with a unique combination of services and deep domain expertise to improve their product marketing, lead generation, and thought leadership activity.

IBM Automation Automating your business intelligently is no longer a choice, it's an imperative to remain relevant and competitive. IBM Automation helps businesses develop and implement a strategy that automates entire business and IT processes, not just stand-alone tasks, so that your whole enterprise can run more efficiently.

