TIBCO

Choosing between Kafka, Pulsar, and Other Messaging Technologies

As the pace of business and change increases, application communication and integration have become significantly more important. A hardened, proven, tightly coupled communication infrastructure is at the foundation of a truly digital enterprise that can quickly react to change.

The most promising approaches to digital communication in recent years have come from the open-source community where developers have collaborated to provide solutions to common challenges in building a digital world. One of these solutions is Apache Kafka, which was built to provide distributed messaging for log management and stream processing.

Because meeting the ever-changing needs of your business requires careful consideration, in this whitepaper, we define several commercial and open source options and set out their pros, cons, and information about their complexity and cost of ownership.

A Brief History of Messaging

Starting from the time that someone needed one computer to communicate with another, "messaging" describes digital communications between systems.

Arpanet, the first wide-area packet switching network, used network layer protocols like Ethernet, TCP/IP, and UDP for system-to-system communication, which were the beginnings of messaging.

As Arpanet evolved, and more systems were interconnected to become today's Internet, communications principles started percolating up from the network layer to the application layer. With these advances came layers of abstraction that simplified how systems connected and communicated, which became the goal of messaging technologies.

> Over the years, new communications protocols were invented for different types of communications. Specifications and protocols like Java Message Service (JMS) and Data Distribution Service (DDS) came in the late 90s and early 2000s. Many applications began using protocols like HTML and HTTP for more than what they were originally designed. Everyone was looking for that one protocol that would work for everything—like AMQ and AMQP (page 7)—but the reality is that there will never be a single approach to communicating. One person's efficiency is another's demise. Digital communications will always be an amalgam of multiple approaches and paradigms.

Apache Kafka

Open Source Software Solution

To understand Apache® Kafka®, you need to understand where it came from. Developed by LinkedIn and donated to the Apache Software Foundation, Kafka was originally designed as a common framework to handle high-throughput and distributed workloads for streaming logs and other real-time data feeds.

While the concept of high-throughput messaging isn't new, Kafka brings a new approach to solving the challenges of data distribution and data resiliency that are built on the traditional concepts of pub/sub messaging. Producer and consumer applications send and receive data using topics (metadata) that allow brokers to do the routing. Kafka is unique in how it manages data persistence and tracks consumption. It distributes brokers, and segments topics, into partitions that can be balanced and redistributed by administrators as more capacity and scale is needed.

Unlike other real-time messaging systems that store data based on durable consumption, Kafka persists data (and consumption metadata) based on Time To Live (TTL), an approach that allows applications to consume data from any point in the persisted data stream (replay those streams on demand) and use consumer offsets to track which data has been consumed.

TTL provides native support for data replay where other systems usually require out-of-band techniques to accomplish this.

Key Characteristics of Apache Kafka

REQUIRED SKILLS	Understanding of messaging and underlying operating system functions, like storage and networking communications. Additional understanding of open source software such as Apache Zookeeper, MirrorMaker, etc.
COMPLEXITY	Relatively easy to use out of the box. Complexity increases when features like security, replication, and global distribution are required.
PROS	• Long term data persistence • Distributed data streaming • Data replay services • Higher throughput
CONS	• Multiple systems to manage (Brokers, Zookeepers, MirrorMakers, Connectors, etc.) • Replication not natively built into Kafka brokers • Management and monitoring can be challenging as the infrastructure grows • Topic partitioning • Secure communications not designed in and is difficult to implement. • Node partition balancing and leader selection • Community contributors are largely (90%) from a single organization
TOTAL COST OF OWNERSHIP	Apache Kafka is simple and relatively easy to get up and running initially, especially for small to medium-sized projects. Open source doesn't mean free, and growing Kafka to enterprise-scale requires dedicated support staff to maintain the infrastructure. A number of commercial vendors, including TIBCO, offer Apache Kafka support and maintenance.
PERFORMANCE HIGHLIGHTS	**Volume: HIGH** 100,000+ messages/second **Latency: AVERAGE** Average of 10 ms **Scalability: HIGH** Clusters can scale both horizontally and vertically **Global Distribution: YES** Possible with third-party add-ons

Apache Pulsar

Open Source Software Solution

Developed by Yahoo, Apache® Pulsar®, like many other messaging solutions, is built on the concept of publisher and subscriber clients that leverage topics for data access. However, Pulsar provides a storage system for both real-time and historical data analysis.

In many ways, Pulsar is similar to Kafka, but the foundation for enterprise scale and deployment differentiates it. Natively built to support all the data distribution paradigms that traditional messaging solutions need to provide, Pulsar also supports the ability to manage stream processing functions directly in the broker infrastructure. This is very appealing to users looking for less complexity when deploying large scale, global infrastructure. Pulsar's distribution at enterprise scale provides out-of-the-box support for multi-tenancy and data replication as part of the core infrastructure, allowing for simplification in growing application usage and adoption over time.

Key Characteristics of Apache Pulsar

REQUIRED SKILLS	Basic understanding of messaging and underlying operating system functions, like storage and networking communications. Additional understanding of OSS like Apache Zookeeper, Apache BookKeeper, etc.
COMPLEXITY	A simplified encapsulated approach where all functions are centrally accessible, reducing complexity when scaling to enterprise levels.
PROS	• Long-term data persistence • Multi-tenancy and data replication • Flexible security implementations • Much higher performance • A more centralized approach to integration and streaming functions • Very broad community support, multiple contributors from multiple organizations
CONS	• Initial setup can be more daunting • While centralized, there are a number of components • Not yet as widely deployed as other solutions
TOTAL COST OF OWNERSHIP	Apache Pulsar takes a bit more effort to get up and running, but once deployed, it scales to enterprise levels very well. Open source doesn't mean free, and running Pulsar at enterprise scale typically requires dedicated support staff to maintain the infrastructure. A number of commercial vendors, including TIBCO, offer Apache Pulsar support and maintenance.
PERFORMANCE HIGHLIGHTS	**Volume: HIGH** 100,000+ messages/second **Latency: AVERAGE** Average of 10 ms **Scalability: VERY HIGH** Clusters can scale both horizontally and vertically **Global Distribution: YES** Native support for global distribution and data replication built-in

Eclipse Mosquitto (MQTT)

Open Source Software Solution

Like many other messaging solutions, Eclipse Mosquitto was built and designed with a specific purpose in mind. It is uniquely different in that it was built for the Internet of Things (IoT), specifically to support MQ Telemetry Transport (MQTT).

MQTT was developed as an OASIS standard with input from multiple organizations with many years of experience in messaging and data distribution. Organizations like IBM, Microsoft, TIBCO, and many others, contributed to the specification, which has become one of the standards for IoT communication. Developed to leverage MQTT exclusively, Eclipse Mosquitto provides a simple broker approach to deploy lightweight messaging suitable for internet-connected devices that usually have low power consumption and intermittent network connectivity. MQTT allows for a pub/sub, topic approach to communications for devices like phones, controls, sensors, and microprocessors.

Key Characteristics of Eclipse Mosquitto (MQTT)

REQUIRED SKILLS	Knowledge of the MQTT protocol and specification
COMPLEXITY	Very easy to set up and deploy. The MQTT protocol can be a little complex depending on the usage requirements.
PROS	• Simple setup for messaging to devices in seconds • Purpose-built for IoT • The protocol defines message structure, making it easy to integrate with other systems
CONS	• Limitations for large scale enterprise adoption • Infrastructure data persistence can be a challenge • Designed as a gateway communications protocol that should be integrated into larger backend systems
TOTAL COST OF OWNERSHIP	Eclipse Mosquitto provides a simple way to provide purpose-built communications to IoT devices. It is easy to deploy and maintain, but like any open-source solution, cost increases from supporting and maintaining the infrastructure as it scales. IoT applications have the potential to grow rapidly, and supporting this rapid growth requires more investment in application development and the supporting infrastructure.
PERFORMANCE HIGHLIGHTS	**Volume: HIGH** 100,000+ messages/second **Latency: VARIABLE** Depends heavily on deployment architecture and network devices **Scalability: HIGH** Designed for large scale device communication, cluster scalability can require additional resources **Global Distribution: NO** Built for device interconnectivity; clusters are not designed to scale for global communication but to provide global aggregation of data

Java Message Service (JMS)

Open Source Software & Commercial Solutions

Developed by a large consortium of enterprise software companies and software developers with the goal of providing a vendor-neutral approach to pub/sub messaging, Java Message Service (JMS) was designed to simplify communications and application development for Java programming. The early goal was to provide a common framework and interface for sending and receiving data in a Java-centric world. In the late 1990s, JMS became the de facto standard for application communication for the Java programming language; However, the end goal of vendor-neutral messaging was never fully achieved because the specification only defined the application programming interface (API) to leverage a JMS system.

The JMS specification, while by definition only applying to Java, quickly grew beyond Java because its features and functions were needed for all types of enterprise communication. It didn't define the wire protocol or many of the implementation details that the JMS infrastructure needed; therefore, it became a standardized way for applications to interact with a JMS compliant messaging system. Each implementation was unique and provided additional functionality that was not defined. This meant that switching from one JMS implementation to another was not as simple as originally imagined.

Defining the JMS specification, however, ushered in a new era where common patterns were expected to be available for enterprise-class messaging systems. Flexibility in delivery types for broadcasting data to large numbers of consumers versus more pointed delivery for applications needing queuing semantics became common features of most messaging systems. In addition, the ability to define how data persistence and distribution occurred and the agreements for when a message was processed, became common functions of messaging after the advent of JMS.

Key Characteristics of Java Message Service (JMS)

REQUIRED SKILLS	JMS specification knowledge is a plus
COMPLEXITY	JMS systems are fairly simple to use and deploy. Since the system is built on a defined specification, the operational behavior is fairly well defined for most scenarios, but understanding all the pieces of the specification can be daunting.
PROS	• Well defined due to JMS specification • A very broad set of delivery modes, semantics, and features • Purpose-built for large scale Java communications

Key Characteristics of Java Message Service (JMS)

CONS	• Has grown to support many heavyweight operations • Most implementations provide unique extensions that are highly valuable but not interchangeable • Specification requirements tend to make the protocols for data exchange chatty and heavyweight
TOTAL COST OF OWNERSHIP	JMS has been around for a long time, and there are both commercial and open-source solutions available. Knowledge of the JMS specification typically lowers application development costs as the interface is well defined and well known. Scaling JMS infrastructure can be challenging and at times requires large numbers of servers, and most enterprise operations require JMS infrastructure to be set up for disaster recovery or high availability, which adds significant complexity and cost.
PERFORMANCE HIGHLIGHTS	**Volume: MEDIUM** 10,000+ messages/second **Latency: VARIABLE** Depends heavily on deployment architecture and latency of the persistence engine can vary from 10s to 100s of milliseconds **Scalability: VARIABLE** Designed for large scale deployment but typically requires larger-scale server infrastructure to support largely scalable environments **Global Distribution: YES** JMS is designed for large scale deployment but typically requires larger-scale server infrastructure to support largely scalable environments, and complex routing to support global architectures

AMQ / AMQP

Open Source Software & Commercial Solutions

Like JMS, Advanced Message Queuing (AMQ) and Advanced Message Queuing Protocol (AMQP) are specifications designed to provide a common framework for data exchange between applications. Unlike JMS, AMQ and AMQP define both the application programming interface (API) and the underlying network communications layer. By defining the underlying protocol, AMQP does something JMS cannot: provide a true vendor-neutral approach to message distribution.

A common messaging paradigm both at the API layer and the network protocol layer gives developers and organizations a neutral way to implement a messaging infrastructure without having to invest in multiple messaging systems. It's equivalent to the world agreeing that we are all going to speak the same language. So, no need for error-prone translations from one language to another and no more expensive time and investment in learning how other languages communicate. But what language is best for universal communications? How do we incorporate all the efficiencies and subtlety of each unique language's communication into a common universal language?

Will everybody be willing to sacrifice the language they are comfortable with for one that will take a large amount of effort to learn, adapt, and master?

This is the challenge of AMQ and AMQP. In trying to be everything for everybody, many sacrifices in efficiency and functionality have to be made. For some applications, these sacrifices for universal communication definitely make sense; For others, the sacrifice is just too great.

Key Characteristics of AMQ/AMQP

REQUIRED SKILLS	AMQ and AMQP specification knowledge needed.
COMPLEXITY	AMQ and AMQP can get highly complex quite fast. A universal approach to data exchange at both the API and protocol level means that the options for data distribution grow exponentially.
PROS	• Defined to provide a universal, vendor-neutral approach to message exchange • Allows organizations the flexibility to deploy infrastructure that is easily exchanged • Specification means operational behavior is very well defined
CONS	• Specifications are very heavyweight, due to all the unique universal requirements • Doesn't allow many specializations for individual application requirements • Plays to the lowest common denominator, which isn't necessarily the best choice for all applications • Value proposition comes from universal adoption • If AMQ is not used as the API, multi-language compatibility is lost • Incompatibility
TOTAL COST OF OWNERSHIP	AMQ and AMQP could be a great approach to provide universal communications; However, to provide this there has to be an organizational standard that all communications are required to use. In some organizations this is possible; in many others, it is not due to unique requirements. If universal adoption is possible, the TCO is low. If not, the TCO grows significantly as AMQ/AMQP becomes an integration protocol that is fairly heavyweight and needs support and maintenance to provide a common point of message integration.
PERFORMANCE HIGHLIGHTS	**Volume: MEDIUM / LOW** Average 1,000+ to 10,000+ messages/second **Latency: AVERAGE** Average of 10 ms **Scalability: VARIABLE** Similar to JMS, AMQP is designed for large scale deployment but typically requires larger-scale server infrastructure to support largely scalable environments. **Global Distribution: YES** Similar to JMS, AMQP is designed for large scale deployment but typically requires larger-scale server infrastructure to support largely scalable environments and complex routing to support global architectures.

High Volume / Low Latency

Open Source Software & Commercial Solutions

One of the benefits of standardization is that everything becomes a level playing field. And for some companies, leveling the playing field would eliminate their competitive advantage. Organizations in industries like financial services increase profitability and productivity using applications that make decisions faster, process events faster, or distribute more market data faster than others.

This is where the specialization of high volume / low latency messaging has its appeal. Thirty years ago, market data distribution for electronic trading was limited in scope, and the technology at the time was limited in scale. The high-performance network infrastructure was lucky to be capable of 10 megabits a second, and low latency was described in seconds. Today, network performance has increased by four orders of magnitude, and 10 gigabit networks and real-time or near real-time application responsiveness are commonplace. Data distribution has to be less than 50 microseconds, and some applications require nanosecond response.

Today, many of the features and functions of high volume / low latency messaging have been incorporated into more traditional enterprise messaging offerings. For example, open-source solutions like Apache Kafka and Apache Pulsar describe themselves as high volume / low latency. Purpose-built commercial solutions like IBM LLM, TIBCO FTL software, and Informatica/29west LBM all went to war in the early 2000s with a low-latency race to zero with many of them now providing broad enterprise functionality built on their high volume / low latency heritage.

The biggest key to leveraging messaging for high volume / low latency functionality is defining what high volume and low latency means to your enterprise. One organization's low latency is another's high latency. So knowing what can be done with a given solution, and how far that solution can scale to meet the demands of extreme data volumes, is key.

Key Characteristics of High Volume / Low Latency

REQUIRED SKILLS	Typically for extremely low latency and high volume, extreme tuning of the underlying operating systems, networks, and applications is required. Knowledge in threading, kernel tuning, and network tuning is a plus.
COMPLEXITY	To achieve highest performance, the solutions can get fairly complex in what can be optimized and what tradeoffs need to be made. Most solutions do not require using the more complex layers unless application requirements demand that level of performance. Typically solutions built for high volume/low latency are fairly easy to use in basic operations and can be tuned to meet high demands with increased complexity.

Key Characteristics of High Volume / Low Latency

PROS

- Built for some of the most demanding requirements and performance
- Handles workloads for all or most all application types
- Designed to scale as infrastructure grows
- Typically can use a peer-peer communication paradigm, removing network hops

CONS

- Defining the needed level of performance can be challenging
- Some solutions lack enterprise features needed for enterprise-wide deployment
- Complexity can increase quickly as demands for performance increase
- Requires well-architected publishing and subscribing applications to keep up with and leverage the advantages

TOTAL COST OF OWNERSHIP

These high volume / low latency solutions typically perform very well for the task at hand. As demands on performance increase, complexity typically does as well, meaning more investment needs to be made in deploying, optimizing, and maintaining the infrastructure. Leveraging low latency / high volume messaging as the nervous system for enterprise communication provides a high ceiling with regards to growth, but choosing the right solution that can meet all requirements can take time and effort.

PERFORMANCE HIGHLIGHTS

Volume: Extremely HIGH
Average 1,000,000+ messages/second

Latency: Extremely LOW
Average 50 microseconds

Scalability: HIGH
Designed to scale both infrastructure and client applications to process and distribute large volumes of data with extremely low latency

Global Distribution: YES
Many solutions provide global distribution but require careful architecture deployment to maintain performance, typically architectural guidelines are provided to deploy these solutions in a globally accessible way, with trade-offs

Websockets / Mobile Messaging

Open Source Software & Commercial Solutions

One of the first areas that traditional enterprise messaging needed to extend into was for supporting web and mobile communications. Arguably, this requirement was a precursor to cloud messaging and IoT because mobile devices required a lightweight approach to data delivery and needed variable infrastructure. With the advent of WebSockets and HTML5, a new approach to data delivery for web and mobile devices became available that allowed for a natural extension of enterprise messaging features to web and mobile.

Websockets offered a simplified approach to providing bi-directional communications for web and mobile applications, but like standard sockets, a publish/subscribe abstraction layer became very appealing with its much simpler approach to communicating and scaling web and mobile applications.

The biggest value WebSockets and mobile messaging brings is messaging-based communication native to the devices that need to use it. For web-based applications, messaging is extended as native WebSockets in Javascript or node.js models. For mobile devices, WebSockets and mobile messaging extend to the native interfaces for those devices, Android Java for Android devices, and iOS C and Swift for Apple devices. Flexible communications natively supported by the consumption medium provides native support for push notifications. Enterprise functionality allows enterprise organizations to integrate and extend existing architectures to mobile communication and integration.

Key Characteristics of Websockets / Mobile Messaging

REQUIRED SKILLS	Websockets / mobile messaging solutions tend to be fairly easy to stand up and use. The interfaces tend to be natively defined for web applications and mobile devices, meaning that the normal skill sets for developing applications on these types of interfaces apply.
COMPLEXITY	The overall complexity of Websockets / Mobile Messaging tends to be low. It is designed to be very easy to setup, deploy, and service. Typically, complexity comes in handling the large number of connections that these types of applications require.
PROS	• Lightweight and easy to use • Typically provides native device application development bringing messaging to the device seamlessly • Simplifies communications between front-end and back-end systems leveraging native bi-directional communication to web and mobile devices
CONS	• Typically not as robust in the enterprise message feature set that may be needed for some types of applications • Communications protocols can be heavier weight due to connection management and network connectivity requirements • Failure operations need to be considered in highly mobile environments and when devices and not reachable long term
TOTAL COST OF OWNERSHIP	Typically Websockets / Mobile Messaging solutions are designed to provide simplified communication for web and mobile devices and integrate with large scale enterprise solutions. This means that if the only requirement is to provide messaging to these types of applications, the cost of ownership is relatively low, however like most systems these solutions need to tie into an enterprise backbone that requires additional components.

Key Characteristics of Websockets / Mobile Messaging

PERFORMANCE HIGHLIGHTS	**Volume: MEDIUM to HIGH** Depends on the deployment model data structures used **Latency: HIGH** Latency can be very high depending on the networks and systems used **Scalability: HIGH** Typically provides high connection scalability but can require many nodes/servers to do it **Global Distribution: MEDIUM** Most solutions are designed to provide internet based communications, but the deployment of these solutions can limit global reach and availability

Cloud Messaging

Commercial Solutions, Some Built on Open Source

Cloud messaging offerings are the newest offerings. With the rapid growth of cloud services, many organizations are looking to leverage the cloud not only for hosting application infrastructure, but for communications infrastructure as well. Because of this, many cloud providers offer simple communications protocols for application development and integration. In addition, many traditional messaging approaches are now available as either hosted services or deployable as containers into cloud environments.

The challenge is that there are so many options depending on the cloud service and the application requirements. Another question is the level of integration needed with existing on-premises systems, and whether multi-cloud support is needed. Moving a communications nervous system from on-premises to cloud/multi-cloud can expose a lot of components that may or may not be suited for the Cloud.

Where cloud messaging solutions tend to really stand out is in new application development. As cloud-native services are being built and deployed, a cloud-native communications infrastructure purpose-built for these applications makes communication simple. Cloud messaging initially provides a very fast and easy approach to enabling communication for cloud-based applications.

Key Characteristics of Cloud Messaging

REQUIRED SKILLS	Managed services require very little skill to get up and running. Deploying cloud solutions in non-managed environments typically requires basic knowledge of cloud deployment options and containerization models like Kubernetes.
COMPLEXITY	Cloud messaging offerings tend to be fairly easy to use. When more complex operations are needed, cloud deployment of enterprise solutions can be used.
PROS	• Built for cloud-native communications • Easy to use, deploy, and maintain • Managed services are readily available • Managed services require no additional infrastructure
CONS	• Many solutions do not provide the wide breadth of features that traditional enterprise messaging solutions provide, like data recovery and persistence • Multi-cloud deployment requires a neutral approach that allows for deployment into any vendor's cloud • On-premises integration can be challenging, depending on which cloud messaging offering you choose, especially if an on-premises messaging solution is already deployed • The latency of transmission can become challenging depending on the location of service deployment
TOTAL COST OF OWNERSHIP	Cloud messaging is a very low-cost way to provide native messaging support without the overhead of deploying infrastructure to support communications channels. Leveraging cloud messaging typically is a great, low-cost approach for new application development or when you need to extend the reach of applications to internet-based services.
PERFORMANCE HIGHLIGHTS	**Volume: HIGHLY VARIABLE** Depends heavily on the deployment architecture and location of services **Latency: HIGHLY VARIABLE** Depends heavily on the deployment architecture and location of services **Scalability: HIGH** Designed to be highly scalable on demand **Global Distribution: HIGH** By design, cloud messaging is globally accessible and typically globally distributed

Conclusion

Today more than at any other time, enterprises face a difficult challenge when selecting a messaging communication offering. While a single solution has a low TCO, no one solution can meet all the demands for all applications. Messaging has to be more holistic to fit specific and varied application requirements—including for high performance/low latency event processing, streaming data for streaming analytics, microservices for native integration among disparate applications, IoT applications, and much more.

The TIBCO Messaging platform fully supports almost all the approaches described in this whitepaper. It takes on the burden of natively integrating all communications solutions and allows application developers to select the approach that makes the most sense for the requirements without sacrificing performance or needing additional coding. And with its native information exchange, interchange, and data transformation, TIBCO Messaging gives you the flexibility to build a fully-integrated communications nervous system to unlock the data throughout your enterprise.

With nearly 30 years' experience deploying and maintaining some of the most complex communications infrastructures in the world—and now with full enterprise support for open source and the other messaging technologies in our portfolio—TIBCO can help you deploy and maintain pretty much any solution you chose.

Global Headquarters
3307 Hillview Avenue
Palo Alto, CA 94304
+1 650-846-1000 TEL
+1 800-420-8450
+1 650-846-1005 FAX
www.tibco.com

20May2020