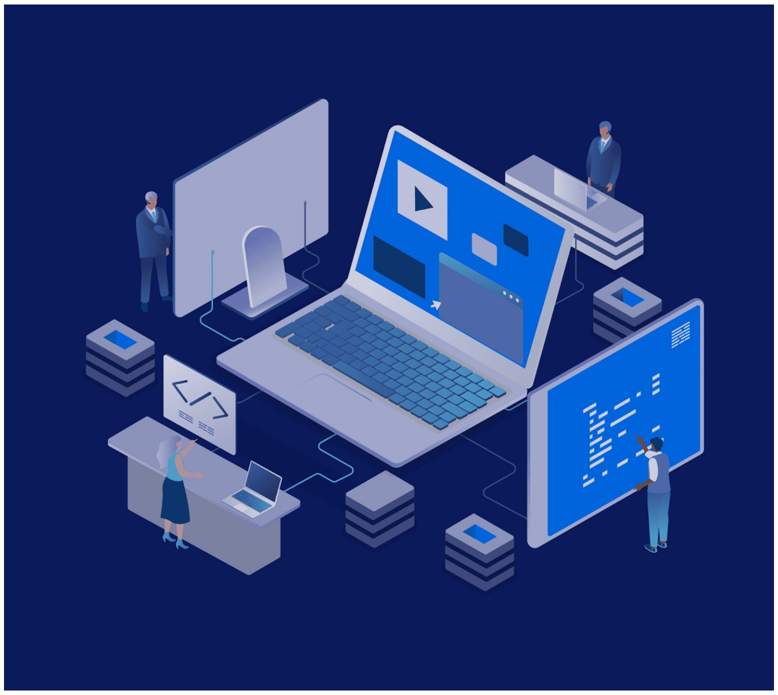

The Complete Guide On Why Companies Should Use Apache Pulsar Instead of Kafka

Real-time, continuous data feeds that power systems and applications are increasingly critical for businesses and organizations of all sizes today.

For nearly a decade, many organizations have relied on Apache Kafka, an open-source distributed software platform, to handle those data feeds.

More than 30% of the Fortune 500 rely on Kafka, so that’s why you may be thinking about using Kafka, too.

And why not?

Kafka powers thousands of companies, but is it right for you?

You may be surprised there’s a better option. If you’re thinking about going with Apache Kafka, you may be making a costly mistake for your company. Here’s why.

What is Kafka?

Apache Kafka is open-source software used for distributing messaging technology. It was developed by a team at LinkedIn and became an open-source tool in 2011 and has become a valuable tool to help companies and individuals build real-time data pipelines and streaming applications.

Kafka connects to external systems to import and export data in real-time and allows users to build applications that can react to or transform data streams. It communications with servers using a TCP protocol.

Kafka operates on a cluster of one or more servers. These servers may be located in different data centers. The software stores record streams in what in categories recognized as “topics.” Every record processed by Kafka includes a value, timestamp, and key.

Kafka partitions topics in cluster nodes and thereby processes massive volumes of real-time message streams. These partitions allow the system to increase its throughput and speed up processing time.

Kafka allows users to:

- Publish records
- Stream records
- Store records streams
- Process record streams

Kafka APIs

Kafka has five APIs: producer, consumer, streams, connector, and admin.

- **Producer:** Enables an application to publish records to one or more Kafka topics
- **Consumer:** Enables an application to subscribe to one or more topics and then process the record stream that's created.
- **Streams:** Enables an application to be a stream processor, meaning it transforms input streams into output streams.
- **Connector:** Enables creation of and running of producers and consumers to connect topics to existing systems or applications.
- **Admin:** Enables management and inspections of Kafka objects, including brokers and topics.

Partitions and Message Streams

Kafka, in simple terms, is a messaging solution for data. But today we often talk about “big data,” but when it comes to Kafka how big is big? That’s where Kafka begins to fall short.

Earlier, we talked about how Kafka identifies record streams as topics. Each topic has a category name or feed name and that’s where the records are published. Some topics can have no consumers that subscribe to it, while others could have millions.

Within Kafka, each topic is maintained in a partition log. That partition contains a record sequence that can be constantly appended. Every partition gets an ID number to identify it within the partition. Those records are then retained based on a user’s specific retention policy. So, if your company needs to retain each record for five business days, then the record remains available on that partition for five days and then it’s discarded to give you more space on your partition.

The benefit of this is that you can store a large volume of records for each topic across many servers, instead of being limited to space on a single server.

Introducing Pulsar

But what happens when you add machine learning and AI to the mix? In most traditional messaging streams, when data moves through the system, it reaches an end point and that’s usually the end of the road for the data. But with machine learning and AI, there is a constant feedback loop, meaning the data may be used repeatedly over an extended period of time.

With AI and machine learning, you need to process increasingly large amounts of data, for example IoT information, web clicks, etc., and that data should be ingested quickly.

AI is amped up in every possible way; so for that data, it means you need more software nodes and more nodes mean more systems and more systems mean more data.

Apache Kafka just isn't that good at scaling at these levels.

The ability to scale is crucial for businesses and at these volumes, Apache Kafka falls short.

Hosting messaging is the answer here, and that's where the new—and better—solution comes in: **Apache Pulsar.**

What is Pulsar?

Kafka wasn't built to be cloud-native, but Apache Pulsar was built for cloud-like scenarios. Like Kafka, Apache Pulsar is open sourced. Pulsar offers the three core messaging patterns – pub-sub, message queuing, and event streaming, in one messaging solution. It was also developed by a large company—Yahoo—and is now available through the Apache Software Foundation.

So how does Pulsar work?

Pulsar is used for sever-to-server messaging.

Pulsar was developed on the publish-subscribe (pub-sub) pattern. Within Pulsar, an application can feed data into the system. That’s called a publisher. An application can then consume that data from Pulsar, which is called a consumer.

Within Pulsar, messages are published to topics. Consumers can subscribe to those topics, process the messages that come in, and then acknowledge when processing is complete.

Pulsar retains those messages until they’ve been successfully processed.

Today, Pulsar is used to support real-time event streaming, stream processing, microservices, and data pipelines.

Understanding Pulsar Components

Let’s take a closer look at the components of Pulsar to better understand what they are and what they do.

Messages: These are Pulsar’s basic units. Each message has a value or data payload. That value represents the data the message carries. Users have the option to tag messages with keys, which can help with topic management. You can also optionally add user-defined properties.

Each message also has a producer name. That’s the name of the producer that created the message. By default, producers get default names, but users can adjust those and apply their own.

Every message within Pulsar is part of an ordered sequence for it’s topic. This controls the ordering of a message within a sequence.

Messages also have a timestamp to indicate when it’s published and there’s an optional event time that can be attached by applications to note when an event occurs, for example, indication of when the system processed the message.

Producers: A producer represents the process used to attach to a topic so messages can be published to a Pulsar broker for processing. The producer sends messages two brokers two ways: synchronously or asynchronously. For sync send, the producer waits to be acknowledged by a broker after sending a message. If there isn’t an acknowledgement, the producer assumes the send failed. With async send, the message goes into a blocking queue with immediate return. Then the client library sends a message to the broker in the background. If a queue is full, the producer might be blocked or there could be an API call failure.

When a producer publishes a message, Pulsar will compress it to save bandwidth. Some supported message compression types include SNAPPY, ZSTD, LZ4, and ZLIB.

Producers can also bath messages and they’re tracked and stored as batches, not individual messages.

Broker: In Pulsar, a broker is a cluster component. Pulsar’s clusters usually use multiple brokers. The brokers run dispatchers to handle message transfers as well as a server for topic lookup and administration.

Consumer: A consumer is what Pulsar uses to attach a topic through a subscription so it can receive messages. The consumer gets those messages from a flow permit request sent to a broker. Similar to producer send methods, consumers receive messages through sync and async. When a consumer successfully receives a message, it sends an acknowledgement back to a broker. The messages are stored until all subscriptions acknowledge it, then it's deleted. You can keep messages longer if you need to by configuring Pusar's message retention policy.

If a consumer doesn't successfully consume a message and needs to attempt to consume the message again, it can send the broker a negative acknowledgement to let the broker know it needs to resend the message. Messages can also be set to redeliver automatically by using Pulsar's automatic redelivery setting.

Topics: Topics are the channels Pulsar uses to transmit messages from the producer to the consumer. Names for topics are structured URLs. Each topic name contains the following elements:

- Topic type, either persistent or non-persistent
- Tenant
- Namespace, which is used to group related topics
- Topic

In Pulsar, you don't have to manually create new topics. If a message is processed through a topic that doesn't exist, Pulsar can automatically create a new topic under namespace through a topic name.

BookKeeper: You'll hear a lot about BookKeeper in Pulsar. Apache BookKeeper is the log storage service used by Pulsar. Essentially, it's how Pulsar stores your data.

Why Would You Need Kafka or Pulsar?

Many companies use distributed messaging platforms to help them react to their customer and business needs in real time.

Kafka began as a way to track user activity on websites. It enables tracking of site activities, for example, page views and searches, and can enable systems to process that data for processing, monitoring, storing, and reporting. When you're tracking down to a granular level, say, for example, each click a user makes on one page visit, you generate a high volume of messaging data.

Today, Kafka and Pulsar can do much more than just track website activity. The systems are used to monitor operational data, collecting and processing log activities, processing data pipelines, sourcing events, and more.

Here's an example. Let's say you have multiple applications for your critical business operations. In most cases, historically, data is siloed in each system. Or, you have to couple these systems together in a house-of-cards way to get them to work together. Platforms like Kafka and Pulsar change the way you process and consume data from your applications.

Who Uses Kafka?

Originally created by LinkedIn, Kafka is used by large organizations around the globe. Here are a few examples:

LinkedIn uses Kafka for activity stream data and other metrics for products like it’s newsfeed and LinkedIn Today.

- Twitter uses Kafka as part of its Storm stream processing
- Netflix uses Kafka for real-time monitoring and event processing.
- Pinterest uses Kafka for log collection
- Spotify uses Kafka for log deliveries
- Cisco uses Kafka in its Security Operations Center (SOC)
- Shopify uses Kafka for logs and event testing
- Ancestry.com uses Kafka for event log processing
- Hotels.com uses Kafka for real-time event tracking
- MailChimp uses Kafka to power its data pipeline

Some other high-profile companies that use Kafka:

- Goldman Sachs
- Microsoft
- The New York Times
- Intuit
- Target

Who Uses Pulsar?

Created by Yahoo, Pulsar was built in 2013 and became open-source in 2016.

Pulsar, although gaining momentum, is a bit less known than Kafka, but not because of lack of functionality or product value. When LinkedIn open-sourced Kafka, it was backed fairly quickly by the team at Confluent, which was supported by the original LinkedIn developers. That team quickly raised funds and were able to successfully market Kafka.

Conversely, when Pulsar emerged open-source from Yahoo Japan, it did not have that same commercial backing, which means it’s been a little slower to pick up brand awareness and adoption at the scale Kafka did.

But Pulsar has already been adopted by businesses around the world including:

- BestPay
- BrandsEye
- China Telecom
- Clever Cloud
- Comcast
- EMQ
- Giggso
- Globalegrow E-Commerce
- IoTium
- MaxKelsen
- Mercado Libre
- Narvar
- Ndustrial.io
- Nutanix
- Okcoin
- Onde
- Overstock
- ProxyClick

- StreamNative
- Tencent
- The Hut Group
- Toast
- Verizon Media
- VIP Kid
- Yahoo Japan

Check out a full list of Pulsar users at pulsar.apache.org/powered-by.

Why Choose Pulsar over Kafka?

Scalability has always been an issue for Apache Kafka. Kafka wasn't designed to work in the sort of environments created by AI and machine learning. While Kafka can be patchworked to work, some users think making those adjustments for big data and millions of messages within Kafka is like building a house of cards.

In Pulsar, however, resource utilization is better and so is through-put. Pulsar, on average, has about 2.5 times the through-put as Kafka. In simple terms, the plumbing pipeline in Pulsar is bigger and that means it costs less to do the same amount of work and processing within Pulsar compared to Kafka.

There is also significantly less latency in Pulsar compared to Kafka. Pulsar can scale to more than a million topics with little latency (< 5ms) for publishing.

In Pulsar’s recent User Survey Report, more than 40% of users said that Pulsar and streaming platforms bring increased agility to their businesses and it unlocks new use cases. Just more than 30% said Pulsar also helps reduce costs.

Other benefits cited included improved customer experiences, more accurate and faster decision-making, reduced risks, new customers, and more sales.

Customers said they’re choosing Pulsar over other platforms because of its design, scalability, reliability and fault-tolerance, with additional acknowledgement of its cloud-native availability and simplification of operations.

How Companies Use Pulsar

We mentioned in the Pulsar survey respondents said the platform unlocks new business use cases. That same survey revealed that the top three use cases for Pulsar are asynchronous apps, building core business applications and extraction, transformation and loading.

Here are other ways companies use Pulsar:

- Application monitoring
- Business intelligence reports
- Communications
- Machine learning
- Backed analytics
- IoT
- Recommendations

Pulsar Multi-Tenant Benefits

The Pulsar survey also took a look at its top features. Not surprisingly, pub-sub took the lead with 72% of responses, but multi-tenancy was in the second spot.

Multi-tenancy in Pulsar means the platform can serve multiple applications, often referred to as tenants, in a shared environment. It simplifies architecture and management and that over time, can help reduce operational costs.

Pulsar Reliability and Durability

Pulsar is also more durable than Kafka with less message loss. Here’s an example of its resiliency:

Jack Vanlightly, principal software engineer with Pivotal Software, attempted to force message loss in a series of tests on Apache Pulsar clusters. He used Blockade to kill off nodes in an attempt to slow down a network and lose packets.

Here’s what it looked like.

First, he conducted a control scenario where he published 2,000 messages with no disruptions. He ran the test five times without any “chaos actions” to ensure all messages were read in the correct order and weren’t duplicated. He did this test five times, which took about 20 minutes. There was no message loss.

Next, he initiated seven different scenarios in an attempt to cause message loss in the Pulsar clusters. He ran these tests:

- Kill the topic broker owner
- Kill a bookie in the ledge ensemble
- Isolate the topic owner broker from ZooKeeper
- Isolate the topic owner from ZooKeeper with deduplication enabled
- Isolate a bookie from ZooKeeper
- Kill multiple bookies
- Kill Qw bookies

The results?

No missing messages. No out-of-order messages.

Vanlightly said after his review that Pulsar is “very robust so far and hard to misconfigure.”

It’s interesting to note that Vanlightly ran similar tests on Kafka, including 11 testing scenarios, and was able to introduce message failure into the clusters.

You can read more details about his Pulsar tests here and the Kafka tests here and here.

Pulsar Queuing, Streams and Pub-Subs

There are different types of messaging systems like queuing, steaming, and pub-sub.

Apache Kafka only streams messages. It can’t queue on its own. To do so, you need a separate piece of technology, like RabbitMQ.

Pulsar, on the other hand, can queue and stream. It’s like getting two products in one, which you need for big data.

Pulsar Retention and Recall

Kafka can’t separate data storage from compute, and that’s an issue for companies that have audit and compliance requirements that mandate data retention.

With Kafka, that means you have to maintain your stored data yourself because Kafka doesn’t separate the data you need stored from the dating you’re using.

Pulsar, however, has a tiered storage option. Let’s say your data retention policy is to keep 30 days’ worth of messages on your Pulsar cluster. You can configure the system to offload data after 30-days into other storage.

And, because your stored data gets to stay within the Pulsar system, if you need to review it, Pulsar can pull it for you, without having to use additional systems to access that data.

Pulsar gives you infinite retention options that just aren’t available with Kafka.

Pulsar Increased Resource Utilization

Earlier we explained how Kafka partitions all topics to increase through-put. Those partitions, over time, equate to more servers and ultimately more expenses for your company.

Here’s an example: Let’s say you analyze your data processing needs for a daily average. You discover that some days, your maximum messages processed every second reached 500,000. But on most days, your average is about 300,000.

With Apache Kafka, that means you'll need to have a cluster of servers that can handle 500,000 messages. So you have to pay for enough CPUs to handle those 500,000 messages, even if you're not using them most of the time. When your data actually becomes "big data," you have to add on additional servers, thereby rapidly increasing your costs. That's because with Kafka, every CPU you use (or need to retain for usage), you have to pay for.

With Pulsar, on the other hand, you pay for the space you need, as you need it.

Why is this important?

Because your goal is to ensure that you're using 100% of the systems you're paying for. If you've purchased enough server space on Kafka to process that maximum 500,000 messages we mentioned earlier, but most days you're only actually using space for 300,000 messages, then you're paying for services you don't always need.

On average, Apache Kafka utilization is 25% at best. The larger your data set and needs become, the worst utilization you get. Traditionally, that can be at best 23%. Some large companies have less than 15% utilization of their Kafka servers. That means moving away from Kafka to Pulsar with even minor utilization improvements can save you significant amounts of money.

Additional Benefits

Here are some additional benefits of selecting Pulsar over Kafka:

- End-to-end encryption for improved security

- Uses quorum replication to ensure messages aren’t lost, compared to Kafka’s followers’ acknowledgment approach
- Brokers are stateless; Kafka’s are not
- Supports authentication, authorization, isolation, and quotas
- Has persistent message storage through Apache BookKeeper, which creates isolation between read and write functions
- Uses REST Admin API for administration, monitoring, provisioning, and tools
- Supports flexible messaging models with APIs for Go, C++, Java, and Python
- Created for geo-replication between data centers in multiple geographic regions
- Push message consumption model compared to Kafka’s pull model
- Index storage architecture compared to Kafka’s log storage model
- Better scalability
- Past message querying with SQL

A Better Data Journey

In addition to data storage and retention, real-time message streaming data has an often complex journey within systems, networks, and business functions. That journey is generally supported by a team of designers, developers, and engineers who create the path your data takes within these systems and manages it over time.

In a typical scenario, for example, you adopt a new software solution, like Kafka or Pulsar, so you need to configure it. Let’s look at that configuration like a slider. On the left are all the features that make the product durable. On the right are all the features that make it perform well.

Often, businesses will adjust that slider based on individual needs. For example, if your focus is on durability, you may care less about performance, so you might move your slider toward durability, meaning you’re fine if the software performs slower as long as you never lose your data.

Over time, as you grow and process more data, your team must constantly tweak that slider with the goal of keeping your focus on durability. Or, if your priorities change over time, your team then has to make adjustments to improve performance. If you’re using an open-source solution like Kafka or Pulsar, that constant tweaking often requires your team to get bigger so you can deal with all your data and those adjustments.

But what happens when you’re a small or mid-sized business or a larger organization where your team is already stretched?

Neural Networks Save the Day

A neural network, like the one within Pandio, which is built on Apache Pulsar, can help you manage your message distribution and data management with a focus on connectivity, optimization, and automation.

Essentially an efficient neural network can replace—or supplement—the team and manual tasks and adjustments.

We’ve talked about how Kafka and Pulsar process messages, so let’s look at it now in relation to a neural network.

A neural network looks at all the messaging within your system in a granular way. That includes everything from storage and message distribution to partitions and servers for scaling. With a team managing these systems, you have to do constant static evaluations to determine if you have enough resources to meet your scaling needs and if the system functions as you want it to, but an efficient neural network can do that for you. It uses the messaging system itself to report on the effectiveness of your model in real time and on a per individual component messaging basis.

Instead of making manual adjustments, the network consumes your data and predicts what it should be doing.

For example, the neural network determines you need more memory. Instead of your team making those adjustments, the system dynamically resizes memory allocation for you.

Or, let’s say your cluster needs to change. Using reinforced learning, your neural network can make that change for you, then evaluate whether or not that change was positive or negative and whether it worked or not. Over time, it learns whether or not it’s making the right decisions and improves performance. Essentially, you can predict the best configuration for your entire messaging system on a real-time basis.

In 24-hours of learning, your neural network can easily become more efficient than a team of well-skilled individuals.

Pandio's Hosted Apchae Pulsar is designed to leverage machine learning. It rapidly maps data so AI can easily digest it. It then enables customers to use that knowledge to quickly build out complex workflows to make business decisions based on predictive models.

Pandio for Big Data

As more companies want to adopt machine learning and AI, they struggle to adopt a robust and efficient messaging platform. Many say it's a bottleneck to adopting AI because they get to a fork in the road where they have to make difficult and costly decisions.

Option 1: Do it in house and find a way to build a highly-skilled, efficient team and solution. If you don't already have trained professionals on staff, you'll have to invest money to hire them, train them, and get them up and running. If you're working with existing staff and you need to train them and improve their skills, you're looking at delays of possibly months while your existing staff learns more about your specific needs and the skills required to meet your objectives.

And sadly, like many technology-related jobs, there's scarce human capital within the industry.

Option 2: Remove the operational burden and work with an experienced team using an industry-innovating platform to reduce your time, expenses, and resource drag.

The reality is messaging is already hard to do and it gets more complicated with AI and machine learning are involved. Distributing messaging is a key path to AI and machine learning and as businesses and technology improve going forward, it’s going to become increasingly important.

Pandio builds on the strengths of Apache Pulsar to help companies solve big data challenges. It’s the first truly open AI and big data ecosystem. Pandio’s storage layer is backed by Apache Bookkeeper. It provides infinite retention and recall at unparalleled speed.

From a machine-learning and AI perspective, Pandio uses machine learning to integrate with data to iteratively train and then deploy game-changing models.

Pandio’s technology facilitates rapid build-out of API connections so you can efficiently integrate large amounts of data and then normalize, deduplicate and map that data to a storage solution either on-premises or in the cloud.

Earlier, we talked about improved resource efficiency, particularly related to data storage, to reduce costs associated with messaging systems. With Pulsar, you use fewer resources, and a Pandio Pulsar/neural network combination means you could increase your utilization by 97%.

A 3-4% reduction in utilization expenses for large enterprises could result in millions of dollars in savings.

On average, Pandio drives about 10 times cost savings for companies by maximizing scalability in the cloud while helping customers avoid high-cost setbacks that often prevent teams from growing exponentially.

Challenges Ahead

According to the Pulsar survey, in 2020 60% of organizations plan to deploy more systems or applications that use Pulsar and almost 40% plan to increase their Pulsar budget. Their challenges will be with whether they build those systems in-house or work with an outside team, like Pandio.

Many companies are hesitant to work with outside companies or adopt new platforms because they don't want to give up control of their data and processes. That's the result of how many managed services operate, leaving the client with little direct control of their applications.

Pandio is different. It's not a black-box scenario. Clients remain full control and can adjust configurations and settings as needed. Create an issue on a change or update? No problem. The Pandio team is there for support and course corrections as needed. Or you can let the Pandio team run your system with full confidence they understand your goals and requirements.

If you're struggling to decide whether Apache Kafka or Apache Pulsar is the best fit for your business, reach out to a Pandio advisor for support. We'll be happy to explore the pros and cons based on your specific business needs and are happy to help.