Top 10 Reasons Apache Pulsar Beats Kafka

pandio

Apache Pulsar now emerges as the superior messaging solution, consistently beating Kafka in the full spectrum of relevant benchmarking challenges. Pulsar provides an enterprise level multi-tenant, high-performance message solution with the freedom of open-source. And Pulsar delivers the ultimate competitive combination of high-throughput streaming with flexible message queuing to unify several technologies in one messaging solution. This combination simplifies the integration of message streaming models as well as message queuing. Apache Pulsar regularly benchmarks the lowest latency among all competitors in a fully scalable messaging solution. Here are the top 10 reasons developers and project managers prefer Apache Pulsar...

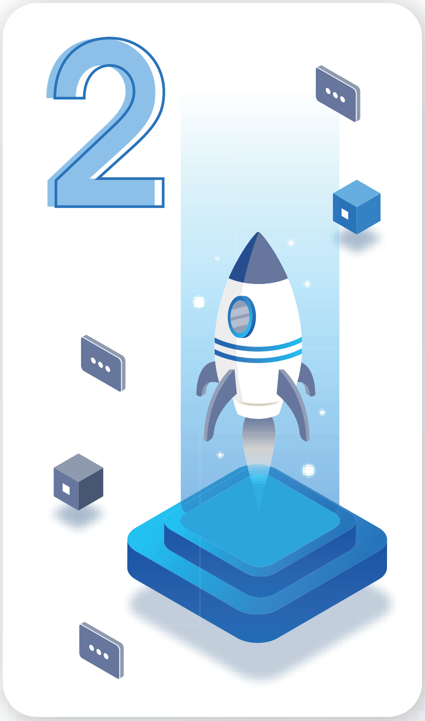
Combining Streams Queues and PubSub All-In-One

The technical advantages of Apache Pulsar over Kafka are abundant. Much of this edge arises from Pulsar's unification of multiple messaging technologies. To begin, Apache Pulsar provides standard message queuing tech including:

- ✓ Competing consumers
- ✓ Fail-over subscriptions
- ✓ Simplified message fan out.

Apache Pulsar tracks client read position in a topic automatically. Pulsar stores this client read position in its unique, high-performance distributed ledger called Apache BookKeeper. In fact, this innovation is one of several which give Pulsar its competitive advantage.

While Kafka is limited in this sense, Apache Pulsar handles many of the use cases of a traditional queuing system such as RabbitMQ. By contrast to most messaging solutions, Pulsar supports both real-time streaming and message queuing in one platform.

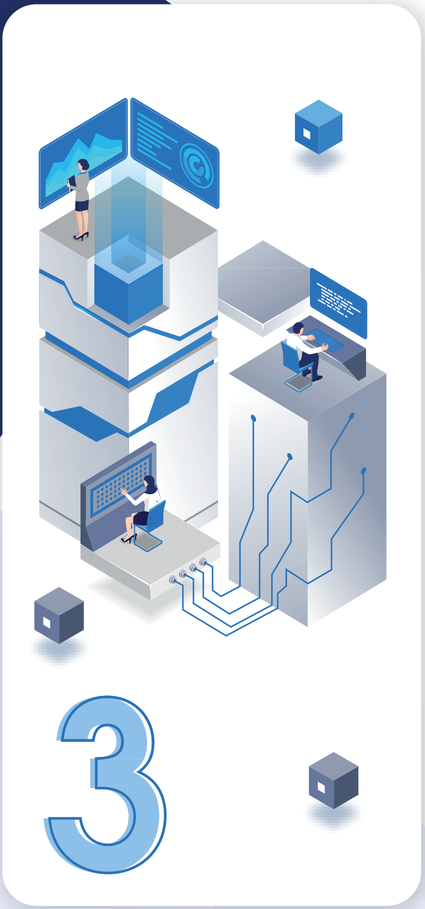


Apache Pulsar Consistently Benchmarks Faster than Kafka

For project managers and developers in the design stages of evaluating messaging solutions for an application technology stack, a critical look at speed and performance benchmarks is important. Is the benchmarked sample code really equivalent to your intended application? The design of the benchmark and the initial settings and configuration of the competing messaging platforms must be accurately understood in order to make the benchmark relevant to your application.

To enable independent benchmarking of important performance metrics, Open Messaging created a test suite to compare Pulsar and Kafka. A Linux Foundation project, Open Messaging provides project managers and developers this critical tool for objectively measuring the latency and other factors important to the Kafka vs. Pulsar competition. Additionally, GigaOm released independent research showing that Pulsar performed 2.5 times better than Kafka in throughput, with 40% lower latency.

Other benchmarks which compare the latency and performance of Kafka vs. Pulsar variously force syncing to drives versus async methods. Pulsar clearly wins these benchmark comparisons. However, as mentioned earlier, it is essential that you know the actual relevance to your own app; the benchmark outcome may be affected by a nuance of your app. With that in mind, you will make the correct choice. Ultimately, speed and reliability determine the competitive success of apps built on messaging platforms and Apache Pulsar use cases consistently demonstrate its superiority to Kafka.

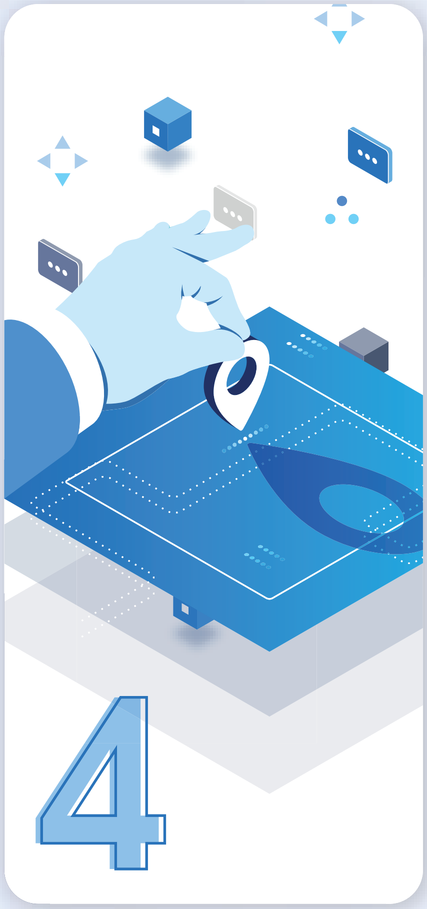


Decoupling Storage and Computation

An important innovation in messaging technology was achieved with Apache Pulsar's multi-layer architecture. In particular, Apache Pulsar architecture decoupled data processing and data storage, often called "storage and compute." Pulsar separates these layers to achieve significant performance advantage: the stateless "broker" manages data serving, while "bookie" nodes handle data storage. This feature allows a broker to be stateless and scale horizontally.

In this way, Pulsar optimizes Cloud-native factors and improves over other messaging solutions like Kafka, wherein storage is linked to the broker, making it difficult to scale efficiently. By contrast, Pulsar's decoupling strategy yields a number of benefits. For example, it enables the storage and compute layers to scale independently of each other. This elasticity is the essence of the Cloud-native paradigm. In other words, the ability to spin up elastic environments to containers and scale resources to adapt to traffic changes dynamically are the salient features of Cloud computing leveraged by Pulsar.

Thinking along these lines, Splunk chose Pulsar instead of Kafka because the extensibility in Pulsar's decoupled storage enabled Splunk to optimize costs. Broker processes in Pulsar are responsible for all data movement. Producer and consumer data are managed by broker processes. Pulsar's innovative collocated "BookKeeper Bookies" running on a variety of hardware infrastructure ultimately store data from brokers. This performance nuance in Pulsar's architecture persuaded Splunk to leverage Pulsar as an enterprise-level messaging solution.



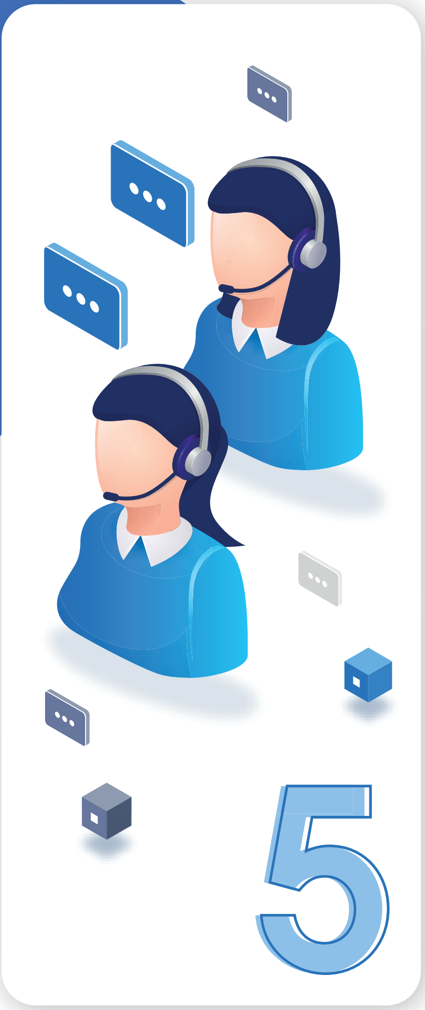
Geo-replication is native to Pulsar but not Kafka

Pulsar natively replicates messages across clusters of a Pulsar instance, and tenants are easily configured to do so within Pulsar's distributed messaging service. For this reason, Pulsar is superior to Kafka, because Kafka requires another tool and additional steps to achieve equivalent geo-replication. Here is a great illustration of Pulsar's inherent geo-replication features.

Replicating across regions is inherently complex, which is an important reason to use a pulsar stream processing because it natively handles latency, coordination, and resource management most efficiently. Using a hosted Apache pulsar service is even better because Apache Pulsar consulting experts take care of many common operational nuances for you in advance.

With Pulsar, it is built into the very core of the application. With Kafka, it is an afterthought. This means that configurability and efficiency are sacrificed. With Kafka it also requires a separate tool that simply links two completely separate Kafka clusters. Pulsar allows a single instance to be geo-replicated, which offers a lot of flexibility such as replicating only a single namespace or tenant.

Geo-replication is slower and more complicated with Kafka because another tool called MirrorMaker must be implemented to achieve it. MirrorMaker replicates messages across datacenters and cloud regions to simulate what Pulsar does inherently. This means that Kafka may be sufficient for backup or recovery functions, but it is not optimal for geo-replicating messages to all consumers across all clusters as Pulsar is.



Multi-Tenancy Clusters

The clever innovations of Apache Pulsar include native multi-tenancy. This feature provides a secure shared operating environment to multiple software applications called "tenants." Another essential Cloud-native concept supported by Pulsar, multi-tenancy includes:

- ✓ Policies
- ✓ Roles
- ✓ ACL's

...which can be associated with namespaces and topics, for the purpose of making a clear distinction between tenants. Tenants may require this functional distinction for various reasons. One tenant in a software enterprise may use a development namespace, while another tenant may work in test automation. The departments in a big box retailer could each be represented as separate tenants. The credit card group of a bank might be one tenant that is distinguished from the loans department.

Pulsar's Multi-Tenancy functionality provides an organization with the ability to operate one global cluster which securely isolates data and hardware by department and thereby enhancing data security - a crucial Cloud priority. This creates several advantages, including:

- ✓ **Overhead** — Multi-tenancy reduces infrastructure and maintenance planning.
- ✓ **Cluster Size** — Pulsar architecture optimizes both horizontal and vertical scalability.
- ✓ **Replication** — Pulsar's Stateless brokers and BookKeeper innovate with n-mesh replication and geo-replication.

With Kafka, tenants have to be managed manually as an internal concept applied across topics & partitions manually. There is no central concept that makes organization of this straightforward.

Stateless vs. Stateful Stream Processing

Let's look at a more technical aspect of messaging in which Pulsar really outshines competitors. First, we need to define a couple of terms: stateless and stateful. In stateless stream processing no dependency exists between the current event and any previous events. Every incoming message event will therefore be processed without maintaining state information about previous messages. Stateful stream processing, on the other hand, implies a dependency between the current message event and previous events. In this case the state of prior messages may influence the processing of the current message. Why is this important?

Pulsar Functions take stream processing to the next level by supporting both stateful and stateless events, which renders messaging operationally serverless. You can perform transformations, routing, and windowing with distributed state, and much more. Additionally, Pulsar takes care of running these functions operationally. Pulsar *Brokers* and *Bookies* implement stateless stream processing, which means, among other things, that scaling is a simple matter.

By contrast, with Kafka you must build Functions yourself using the Kafka Streams SDK. You must then operate it yourself by deploying it separately from Kafka. This also means that the state has to be managed yourself as well. Additionally, the latency is increased because of the physical distance between the messages and the functions.

Storage Tiers For Intelligent Retention

While topic backlogs may expand to an unmanageable volume in other messaging platforms like Kafka, Pulsar resolves this issue with the implementation of tiered storage architecture. The function of tiered storage is to retain older message backlogs intelligently by moving them from Bookkeeper to less expensive storage according to design.

Kafka, by contrast, may retain message backlogs indefinitely, but it does so using partitions. The partitions are limited to the disk size and therefore do not capture the essence of a true Cloud solution. Developers will have to design ad hoc code to manage older message retention beyond the physical partition size. Apache Pulsar uses Bookkeeper's distributed storage to solve this problem and make scaling storage easy.

Tiered storage allows you to extend backlog storage seamlessly to other Cloud services like AWS S3. Cluster storage can then extend infinitely without further coding or consideration. Tiered storage also presents no further complication to the user. If requested data happens to be in AWS S3, it is retrieved through the same request method.

Pulsar Supports Unlimited Topics

An important limitation of Kafka is that it does not support a large number of topics. Kafka's design limitations include

- ✓ Broker is structured
- ✓ Storage tied to the broker
- ✓ File handlers opened with partitions
- ✓ Need to rebalance data when scaling

Resolution of these design limitations arrived with Apache Pulsar, which is designed to support millions of topics. Brokers that handle inbound data and requests for data are stateless and decoupled. This means they can scale horizontally and support as many topics as resources allow. Backing the topics on the storage layer are Apache Bookkeeper instances, which also scale horizontally. Pulsar handles this coordination for you with Apache Zookeeper, which makes the number of topics you can support theoretically infinite.

Suppose, for example, that you want to model customer data on an event stream. If you model each customer within a unique topic, then events will be chronologically ordered in Pulsar's architecture. Thanks to Pulsar's innovative storage layer, millions of customers can be scanned quickly for state without additional cost, increased latency, or memory overhead. This is a significant improvement over Kafka's limited topic range.

Zero Message Loss

Message loss prevention is a critical design component of Apache Pulsar stream processing engine (SPE). In order to guarantee zero message loss, often said "effectively once," the messaging application and/or the SPE must reconnect to the messaging system at a point before the failure reprocesses the data. With this design in place, Pulsar supports zero message loss and zero message duplication. Except in rare cases, Pulsar corrects errors so fast that corrected messages will still appear to users in chronological order. Apache Kafka was not designed for mitigating data loss completely. It is possible due to its design that under certain failure conditions, messages can be lost. Here again, we see the clear technical superiority of Pulsar over Kafka. Such technical advances should come as no surprise since Pulsar evolved intentionally to correct Kafka issues.

Apache Pulsar Gains Enterprise Momentum

Although Apache Kafka captured a first-mover advantage upon release, and is better known in engineering circles, the initial edge is now much less relevant. Important reasons for this momentum change include:

- ✓ Frequent new releases of Pulsar - now quarterly, on average.
- ✓ Major enterprises now choosing Pulsar include: Capital One, Verizon, Splunk, Salesforce, OVH, Tencent, and Overstock.
- ✓ Extraordinary media exposure including hundreds of new articles, videos, training decks, white papers from major enterprises, a 36 session conference, 25+ organizations, and 600+ sign ups, and a flurry of daily activity in the official Slack channel, all praising Apache Pulsar.
- ✓ 289 contributors to the Apache Pulsar community, and growing.

We at Pandio are directly involved in the rise of Apache Pulsar and continue to drive market awareness and adoption through education, strategic partnerships, and by offering our hosted messaging solution. A number of high profile clients and familiar brands will be announced by the end of the year. As we are talking to many large enterprises it is becoming clear that broader market adoption is simply a matter of time.

A True Cloud Native Arises

Imagine discussing a game-changing idea for your business, and then realizing that, in order to implement it, you will need to add yet another technology to your stack. Increasing the complexity of technology adds risk to development as potential points of failure. The learning curve now includes new libraries, complications to the CI / CD pipeline, and more work for QA and increased test automation issues. Apache Pulsar anticipates and consolidates these technologies into one platform.

Furthermore, the performance improvements of Pulsar are such that new tools are now emerging to assist the migration from Kafka and other technologies to Pulsar. Kafka-on-Pulsar, (KoP) for example, created by OVHCloud and StreamNative enables you to migrate a Kafka app to Pulsar without code revision. Yet another tool enables RabbitMQ app to use Pulsar functions remotely.

In light of the accelerating developments around Apache Pulsar, CTOs now frequently find hosted Pulsar solutions, like that from Pandio, highly beneficial. This is primarily driven by the increasing pressure to accelerate the adoption of AI in an environment where data is growing exponentially and human talent remains an ongoing bottleneck.

Pulsar Outshines Kafka

We have now seen that Apache Pulsar clearly arises as a superior alternative to Kafka.
Here is a synopsis of some the truly outstanding features native to Pulsar:

- ✓ Separation of compute and storage
- ✓ Lightweight brokers that scale horizontally
- ✓ Bookkeeper storage nodes that scale and provide storage control
- ✓ Millions of topics
- ✓ Geo-replication on the namespace level
- ✓ Inherently Cloud native
- ✓ Low operational expenditure
- ✓ Richer functionality

pandio