MCKNIGHT
CONSULTING GROUP

Total Cost of Ownership Comparison

Messaging and Streaming Data Platforms

February 2023

Sponsored by

Synadia

Contents

Executive Summary **4**

Messaging and Streaming Platforms **6**

Kafka 6

NATS 6

Subject-Based Addressing 6

Streams vs Topics 7

Consumers 8

Delivery Guarantee 8

Scaling 9

Testing Methodology **10**

Test Architecture 10

Kafka 10

NATS 11

Test Frameworks 11

Test Results for Performance Equivalence **13**

Limiting Factors 13

Kafka 13

NATS 14

Performance Equivalence 15

TCO Calculations **18**

Infrastructure Costs 19

People Time-Effort Costs 20

Conclusion **23**

About Synadia **24**

About McKnight Consulting Group **25**

Executive Summary

Over the past decade, streaming data has risen in popularity and adoption as the demand for near real-time responsiveness and decision making has increased by consumers, businesses, and emerging markets.

Streaming data technologies allow for efficient, scalable, and durable exchange of data between two or more parties with use cases spanning brokered collaboration, data ingestion, and stream processing.

The current market leader in the space of data streaming is Apache Kafka, which was originally developed within LinkedIn and open sourced in 2011.

During the same year, the first open source version of NATS.io (NATS) was released. NATS was originally developed as a stateless messaging technology supporting low-latency and real-time communication patterns such as publish-subscribe and request-reply.

In March 2021, NATS introduced a persistence subsystem called *JetStream*. With this subsystem, the use cases NATS supports overlap those that were once only applicable to Kafka.

The focus of this report is on benchmarking throughput between NATS with JetStream enabled, and Kafka, taking into account the total cost of ownership (TCO), including infrastructure and estimated personnel.

The OpenMessaging Benchmark framework was used as the basis for testing several configurations for Kafka and comparable NATS configurations were used.

Our findings suggest that NATS has an 88% lower TCO while achieving up to 185% higher throughput by broker count and up to 128% higher throughput by total IOPs with comparable setups to Kafka.

In addition, installing, configuring, utilizing, and maintaining NATS is significantly easier. When infrastructure and personnel costs are combined to determine total cost of ownership, NATS requires significantly less work.

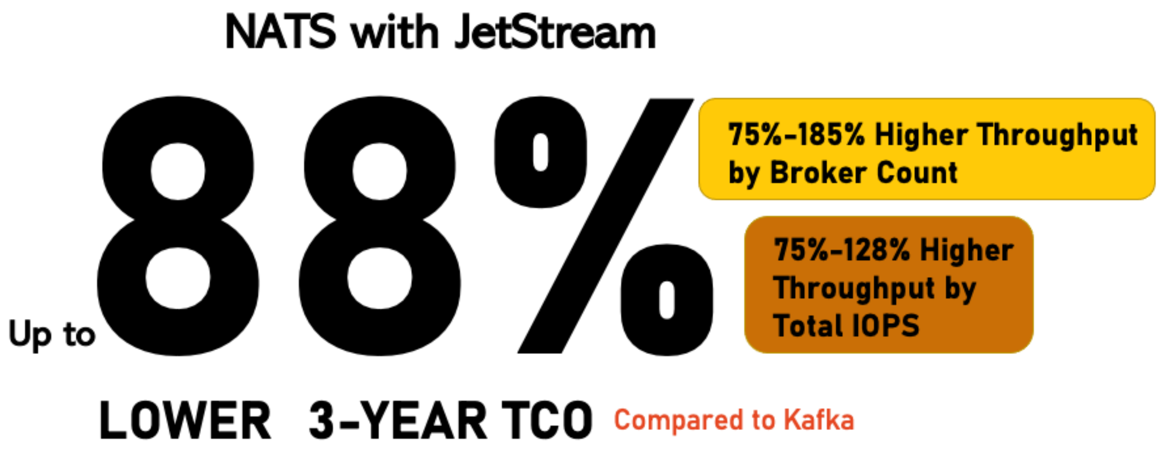

NATS with JetStream
Up to 88%
LOWER 3-YEAR TCO
Compared to Kafka
75%-185% Higher Throughput by Broker Count
75%-128% Higher Throughput by Total IOPS

Figure 1 - NATS TCO Comparison with Kafka

Messaging and Streaming Platforms

Kafka

Kafka is an append-only write-ahead log (WAL) that does not allow for targeted message deletions, but instead relies on compaction to reduce the size of the log. It is a distributed log system that is based on topics, with each topic containing one stream of messages. Each message is associated with a key, which is used for distributing the message across the system.

Kafka is an offset-based replay system, meaning that messages can be replayed from a certain point in the log. This is useful for applications that need to process data in order, such as streaming applications.

Kafka also supports deduplication, limited to a single session. This means that messages sent within the same session will not be duplicated, but messages sent in different sessions may be duplicated.

Kafka is a powerful distributed log system that is used for streaming applications.

NATS

NATS is an open-source messaging system written in the Go programming language and has been deployed by millions of developers globally. The core NATS server is based on a subject-based addressing architecture that allows decoupling of microservices and a highly scalable “fire & forget” messaging backbone, supporting patterns like request-reply and publish-subscribe. JetStream as an embedded persistence layer extends the core NATS qualities of services to include streaming, at-least-once & exactly-once delivery guarantees.

Unlike Kafka you can delete specific messages and have “constraints”. For example: *keep only up to exactly X, where X can be one message per subject in the stream*. Individual messages in the stream can be erased as well as the ability to compare and set make it more similar to a data store than just a Write-Ahead Log.

Subject-Based Addressing

NATS is based on support subject based addressing (SBA), a powerful tool for messaging applications. It allows for messages to be captured into streams using subject token wildcards, without the need to pre-define the topics. This makes it easier to filter messages and create sub-streams.

When producing messages, the key with an optional unique ID, is naturally included inside the subject, allowing for as many tokens as needed. This makes it easier to filter messages when consuming them from a stream. The filtering happens at the server level, only the matching messages are sent down to the consuming application, and that it leverages indexing for speed.

Kafka is limited to replaying from an offset or using Kafka Streams to create filtered sub-streams. With NATS, client applications can 'query' the stream using subject filters, pushing the filtering back to the nats servers. This avoids full stream scans by the client application, as indexing is used to avoid full stream scans by the server.

NATS provides hierarchical filtering using a positional wildcard denoted by "*", or an "any" wildcard denoted by ">". This filtering can be combined with time or sequence-based offsets for the ability to start scanning in a specific point in the stream.

Streams vs Topics

With the introduction of JetStream, NATS introduced the "stream" primitive which is analogous to a single-partition Kafka topic. Like a Kafka partition, a NATS stream is the unit of total order, replication, and distribution. A NATS stream is declared with one or more subjects *bound* to it, and messages published on the overlapping subjects will be received and persisted by the stream.

Each stream can be configured with a number of options such as file-based or in-memory persistence, replication factor, placement in the cluster, and retention policy, including enforcing max message age, total stream size, as well as queue or interest-based semantics.

In addition, streams can be configured as a "mirror" of another stream, that can be placed in a location closer to consuming application that requires, for example, offline access. Another option is "sourcing" streams which can multiplex multiple streams for consumption or can demultiplex a single stream into multiple streams. This use case is common when aggregating data in the cloud from streams created in multiple edge locations.

In contrast, Kafka topic configuration is primarily centered around compaction and partition replication behavior.

Consumers

As a parallel to the stream, NATS introduced the “consumer” primitive which is conceptually similar to a Kafka consumer group. Although NATS has the concept of a “subscriber,” which is initialized by a client application at runtime and does the work of *processing* messages, a NATS consumer acts as a *view* of a stream and, on behalf of the subscribers receiving and processing the messages, keeps track of messages consumed, acknowledgements, and redeliveries etc.

Any number of consumers can be created per stream and are uniquely identified by a name. They can be ephemeral or durable, be configured to start at a particular sequence or at a particular timestamp and they support *server-side filtering* using an overlapping subject with the stream. Additionally, they can be configured to be push or pull-based, depending on what is appropriate for the use case.

As noted above, a NATS stream does not have the notion of partitions in a Kafka topic. One constraint with a Kafka consumer group is that there can be no more consumer group members than there are partitions in a topic. This is not the case with a NATS consumer. A queue-based push consumer can have any number of subscribers as well as a pull-based consumer can arbitrarily scale up or down the number of subscribers freely.

Delivery Guarantee

Exactly once in data processing is a guarantee that a message is processed only once, even in the event of a system failure. This is an important concept in distributed systems, as it ensures that data is not lost nor duplicated.

NATS consumers automatically manage acknowledgements and redelivery of messages. Multiple kinds of acknowledgements are supported including ack, nack, term, and in-process, which control the back-off and redelivery behavior of each message.

Configurable time window message deduplication is built-in to NATS streams, which works across client application sessions. A unique message ID header is supplied at publish time, which makes this possible. Support for *infinite* message deduplication at the stream-level has recently been added which relies on including an identifier in the subject and specific stream configuration.

Kafka consumer groups support *ack all*, and the client application needs to do its own *seek* for re-deliveries. Kafka’s built-in message deduplication feature is limited to messages sent within a single session. NATS supports synchronous double acknowledgement called an *AckAck* of the consumption of a message feature, while with Kafka the client app must store both the offsets and the results of the consumption to some outside system in order to be able to store them atomically.

Scaling

As already described, the unit of ordering, replication, and distribution in NATS is a stream, whereas in Kafka it is a partition within a topic. In both systems, replicas can be placed on a deliberate set of servers in the cluster.

In order to scale writes horizontally beyond a single stream, NATS provides a logical way called *subject mapping* to deterministically map subjects to streams, which is analogous to a Kafka partition.

Kafka natively supports partitions as part of the topic configuration and provides mechanisms to rebalance partitions if the number changes.

Testing Methodology

The purpose of our benchmark testing is not to declare one of these streaming platforms a "winner" by showing it is faster, performs better, has higher throughput, or lower latency. This type of testing is not as helpful in a total cost of ownership study, because it only takes into account the edge of usage extremes, and not day-in-day-out performance. Rather, we attempt to show what configurations of these platforms and their underlying infrastructure offer equivalent performance.

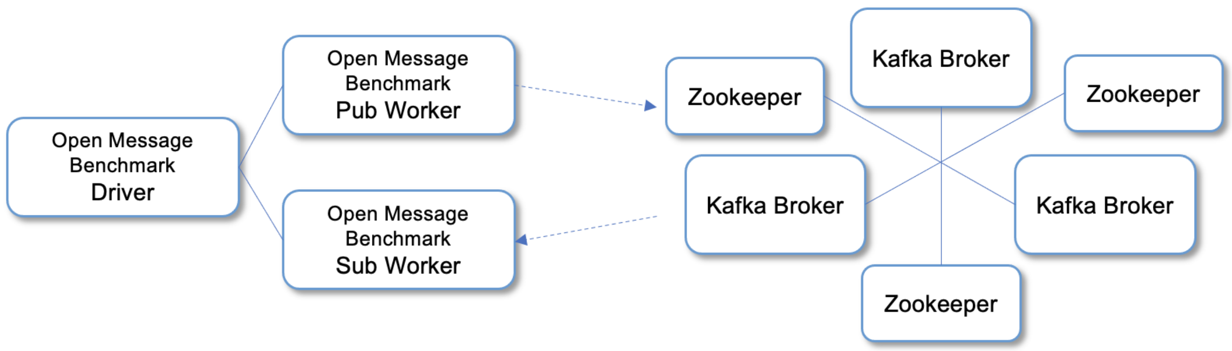
Additionally, we normalized the performance of these platforms based on the performance of a 3-node cluster of NATS running JetStream and writing three replicas of logs to disks as the "lowest common denominator" for an equivalent use-case to Kafka.

Test Architecture

To conduct our field test of these platforms, we setup an architecture consisting of the following components:

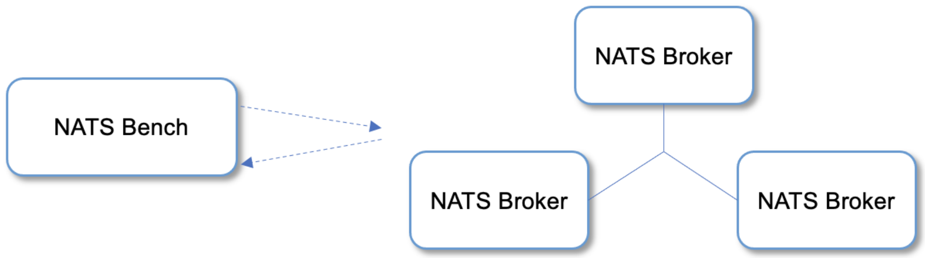
Kafka

Component	v	Instance Type	Instance Details	Log Disks
Apache Kafka Brokers	3.3.1	c6gn.large c6gn.xlarge c6gn.2xlarge	2 vCPU 4GB RAM 4 vCPU 8GB RAM 8 vCPU 16GB RAM	various gp2 and io2 between 3,000-50,000 total IOPS
Zookeeper Brokers	3.7.1	c6gn.large	2 vCPU 4GB RAM	N/A
Open Messaging Benchmark Driver		c6gn.large	2 vCPU 4GB RAM	N/A
Open Messaging Benchmark Workers		c6gn.2xlarge	8 vCPU 16GB RAM	N/A


Open Message Benchmark Driver
Open Message Benchmark Pub Worker
Open Message Benchmark Sub Worker
Zookeeper
Kafka Broker
Zookeeper
Kafka Broker
Kafka Broker
Zookeeper

NATS

Component	v	Instance Type	Instance Details	Log Disks
NATS Brokers	2.9.11	c6gn.large c6gn.xlarge c6gn.2xlarge	2 vCPU 4GB RAM 4 vCPU 8GB RAM 8 vCPU 16GB RAM	various io2 between 3,000-15,000 total IOPS
NATS Bench	2.9.11	c6gn.2xlarge	8 vCPU 16GB RAM	N/A



Test Frameworks

For the field tests, we used two different frameworks—the Open Messaging Benchmark (OMB) and NATS Bench. OMB[1] is a popular open-source tool for testing streaming and messaging platforms. While it does have a testing suite for NATS, it does not natively support JetStream and lacks different testing configurations. OMB is largely a "messaging" benchmark tool and can only perform pub/sub workloads.

NATS Bench[2] is a utility distributed as part of the NATS CLI which can benchmark both messaging and streaming workloads. However, it produces the same metric — throughput, in terms of messages per second. We configured NATS Bench to behave as closely to OMB as possible.

For the tests, we made the testing parameters constant, while varying the parts of the Kafka and NATS configurations to determine equivalence. Those testing parameters were:

[1] https://openmessaging.cloud/docs/benchmarks/
[2] https://docs.nats.io/using-nats/nats-tools/nats_cli/natsbench

Parameter	**Kafka**	**NATS**
Message payload size	1KB	1KB
Publish batch size	1MB[3]	100
# or workers	various (2 – 10)	various (2 – 10)
# of partitions	matched the # of brokers	N/A
# of replicas	1 or 3	1 or 3
Purged stream between runs	new topic each test	yes

[3] linger.ms (time to wait before sending a batch to a Kafka broker) needed to be changed in order to achieve low latency; we set it to 1 ms.

Test Results for Performance Equivalence

To create a use-case scenario to give us a basis for our pricing calculations, we assessed an approximated equivalent performance. To define "performance equivalence," we sought a configuration for each platform that could publish and deliver bursts of 500,000 1KB messages a second. There are edge cases in the streaming world of enterprises who need to sustain 1M plus messages per second. This is not our use case. Our use case is day-in-day-out workloads of 10,000 to 100,000 messages per second, but occasionally needs to burst up to 500,000. When self-provisioning infrastructure, the burden lies with administrators and engineers to determine what the high-end usage scenarios are and plan accordingly.

We also were most interested in factors that have operations and cost implications:

- Number of servers (cluster size)
- Number of vCPUs (instance size)
- Number of replicas (high availability and fault tolerance)
- Storage disk performance* (disk provisioned IOPS)

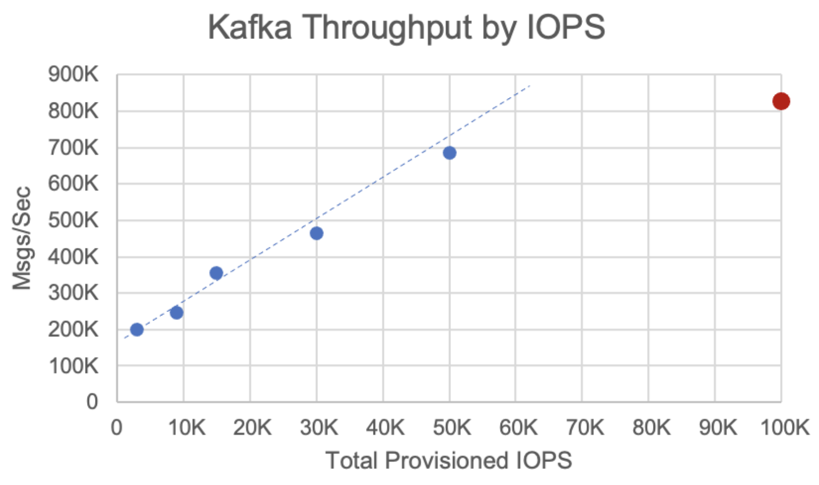
While disk storage volume is a cost factor, it is not related to performance. Storage retention requirements will vary widely by use case and organization.

Limiting Factors

Before distilling the results, we made a few informative observations when it came to results.

Kafka

Most workloads we threw at Kafka were I/O bound. Kafka scaled linearly as we added more provisioned IOPS to its persistent io2 disks. However, once we exceeded 50,000 total IOPS (5 brokers each with 10,000 IOPS disks), the linear scale pattern broke, as can be seen in the figure.



Kafka also scaled linearly with the number of broker nodes. We exceeded our 500,000 messages per second threshold with 5 brokers; thus, there was no need to try larger cluster sizes.

NATS

The workloads on NATS streams were mostly CPU- or network-bound. We exceeded our 500,000 message per second requirement with only 3,000 total IOPS on a single node. As we added IOPS to the disks, we noted the CPUs were still carrying the bulk of the work and the throughput did not increase with additional IOPS; thus, we were able to leave the IOPS at 3,000 for each disk in the cluster.

In addition to testing NATS with NATS Bench using the same conditions as Kafka with OMB (pub/sub only and disk persistence), we tested some of the other scenarios we thought our readers might find interesting.

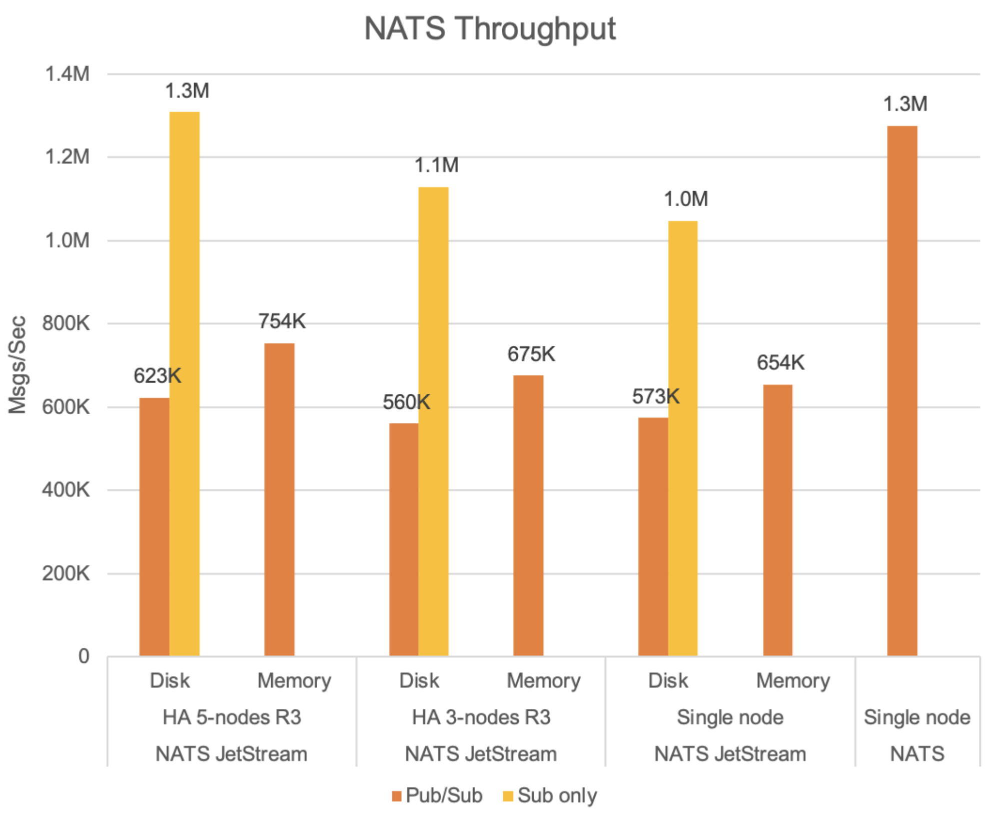
First, we tested NATS with memory persistence, instead of disk persistence. As one might expect, the throughput was higher, but because our process was CPU-bound (and not I/O-bound), it was not overwhelmingly higher.

Second, we noted that publishing messages to the stream is more intensive than reading them. With NATS Bench, you can publish a number of messages to a stream (or streams) in one test run and then subscribe to the stream and read them in a follow-on run. Unsurprisingly, NATS could subscribe and read messages 4 times faster than they can be written, which is good news for enterprises with a few publishers and many subscribers (which we find is typical).

The OMB we used does not allow separation of publish and subscribe functions. You must do both at the same time.

Third, we also tested core NATS (without JetStream). This resulted in over 1M messages per second, which is also good for those who only require high message raw throughput without persistence or robust stream requirements. NATS with JetStream enabled covers both the messaging and streaming needs in one solution. Not all messaging needs to be persisted or is even applicable with streaming (i.e., microservices).

The adjacent chart highlights and compares the additional NATS testing we performed.

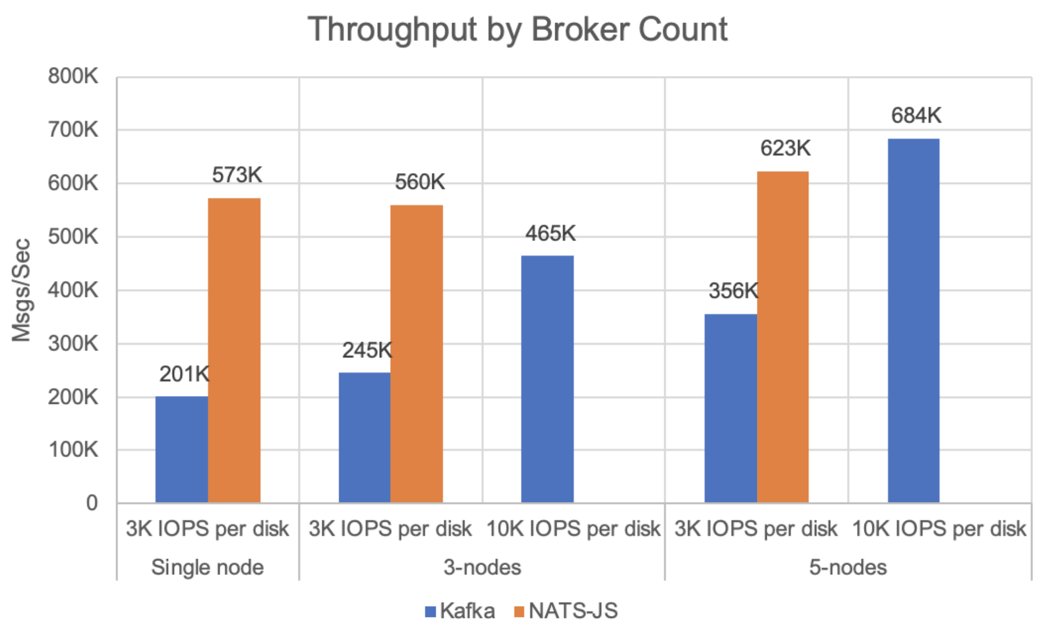


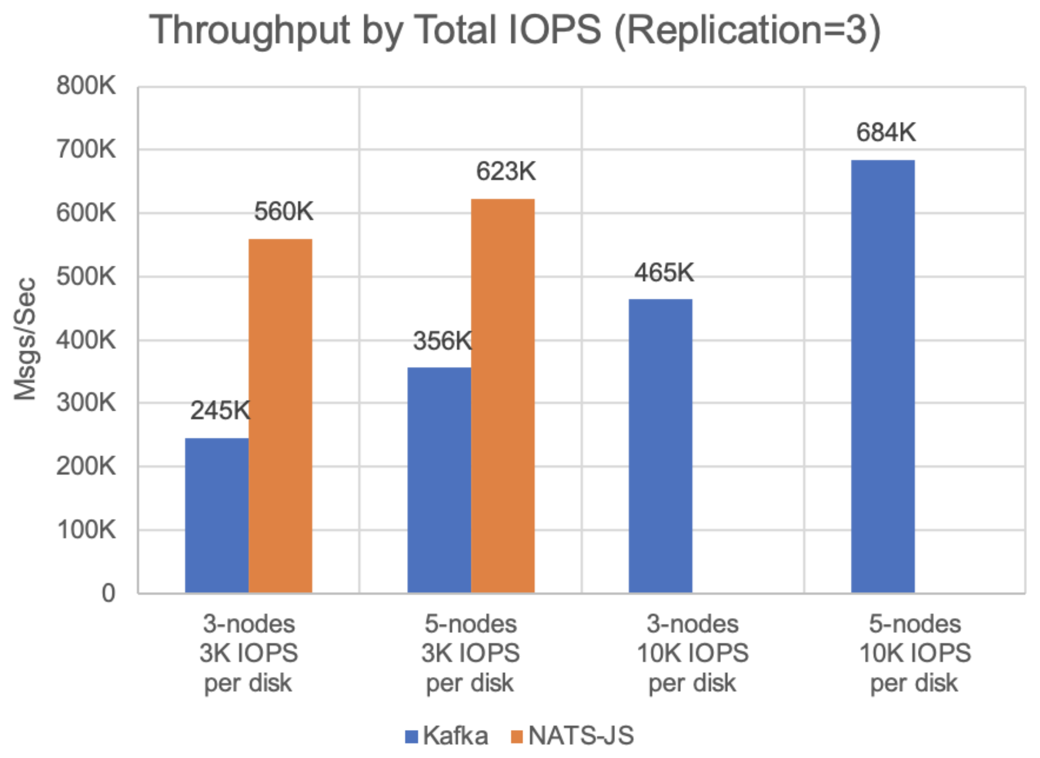
Performance Equivalence

Our testing revealed a comparable configuration for both Kafka and NATS to be used in our TCO calculations.

It is worth saying again, testing streaming data in the cloud is challenging. Configurations may favor one vendor over another in feature availability, virtual machine processor generations, memory amounts, optimal input/output storage configurations, network latencies, software and operating system versions, and the workload itself. Also the performance that you get from a cloud provider is relative to the cost. For example, you can pay for more IOPS.

The charts below compare the testing results we found when comparing various cluster sizes and provisioned IOPS of these platforms.





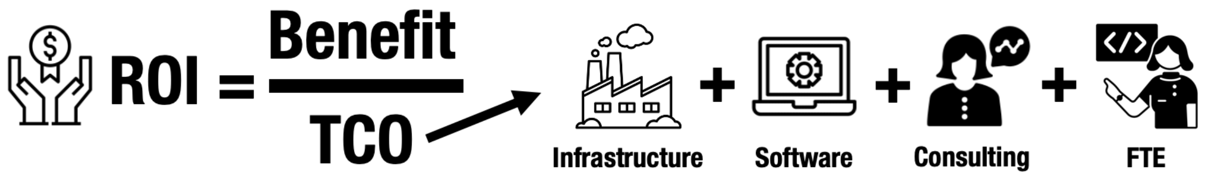
We did not need to test NATS with 10,000 IOPS, because adding additional IOPS to our provisioned disks did not increase throughput or make sense cost-wise, since increasing IOPS on disks increases the cost significantly. For sake of comparison, we noted both 3 and 5 broker nodes of both Kafka and NATS with the former's disks provisioned for 10,000 IOPS each and the latter's disks at 3,000 IOPS produced an equivalent performance. Note that the replication factor was set at 3 for these configurations, except the single node one.

TCO Calculations

Calculating the total cost of ownership (TCO) and return on investment (ROI) in projects is something that happens formally or informally for most enterprise programs. It is also occurring with much more frequency than ever. Sometimes, well-meaning programs will use ROI calculations to justify a program but the measurement of the actual ROI can be a daunting experience, especially if the justification ROI was assessed lightly.

This section will focus on the platform costs, including the ever-important development, operations, and maintenance costs. If you are doing a full ROI for these projects, you would need to consider cost of money, a probability distribution, the *n*-ordered benefits and determining and using only what is tangible.

When projects are done in an agile fashion with functionality metered out, as we suggest, it can be difficult to say when initial project costs end and costs go into maintenance. We will use the usual enterprise standard and draw the line between initial costs and maintenance around the point where most of the functionality is delivered. In this context, it is very important to consider both the accumulated costs to this point as well as the "maintenance" costs for scaling, enhancements and updates on an ongoing basis afterwards.



Costs will fall into these categories:

Infrastructure – Infrastructure is the cloud hardware required to run the platform. In most scenarios this is a separate cost through one of the major cloud providers. In our calculations, we use the most appropriate AWS specifications (disk storage, instance types and sizes) for each platform to achieve comparable performance.

Software – Software is the cost of running the platform from the vendor. In most cases, we use the on-demand hourly rates for software. However, some vendors only price their offering monthly, yearly, or with traditional perpetual-use licensing. In the case of this study, we omitted software costs since both Apache Kafka and NATS are open source and freely available. There are, however, enterprise and as-a-service offerings of these platforms, which are not free. Organizations interested in the enterprise offerings will need to consider those costs as well.

People Time-Effort (Consulting) – Consulting models vary widely, but many projects utilize consulting to a high degree for the initial implementation. In our calculations, we only calculate consulting costs as a time-boxed expense depending on the type of project or its size. For example, a streaming implementation at a small organization may only require 6 months of consulting, but the Infrastructure, Software, and FTE costs will go on for the life of the project.

People Time-Effort (FTE) – Employees contribute to the initial implementation and largely to the maintenance of these projects. Their time and effort cannot be discounted. Choosing a platform with a lower time-effort burden to build, operate, and maintain will free these people's time and energy to work on more strategic projects within the organization.

Infrastructure Costs

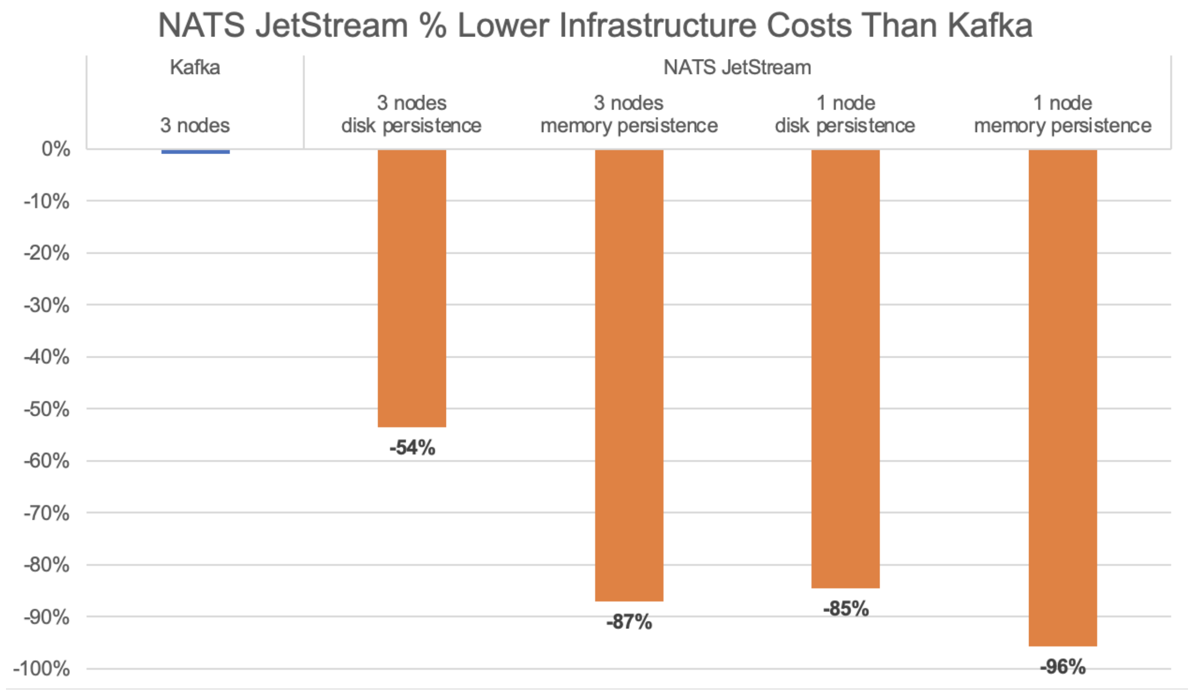
The following table breaks down a comparison of infrastructure costs based on the configurations we tested.

TABLE 1	**Kafka**		**NATS JetStream**				**NATS**
Infrastructure Costs	**5 nodes**	**3 nodes**	**3 nodes disk**	**3 nodes mem**	**1 node disk**	**1 node mem**	**1 node**
Compute							
Brokers	5	3	3	3	1	1	1
Broker Size	c6gn.xlarge	c6gn.xlarge	c6gn.xlarge	c6gn.xlarge	c6gn.xlarge	c6gn.xlarge	c6gn.xlarge
Broker Per Hour	$0.17	$0.17	$0.17	$0.17	$0.17	$0.17	$0.17
Zookeepers	3	3					
Zookeeper Size	c6gn.large	c6gn.large					
Zookeeper Per Hour	$0.09	$0.09					
Compute Per Month	$820	$568	$378	$378	$126	$126	$126
Storage							
Disks	5	3	3		1		
Volume (GB)	1,024	1,024	1,024		1,024		
IOPS	10,000	10,000	3,000		3,000		
GB Per Month	$0.13	$0.13	$0.13		$0.13		
IOPS Per Month	$0.07	$0.07	$0.07		$0.07		
Disks Per Month	$3,548	$2,334	$969		$323		
Total Per Month	**$4,368**	**$2,902**	**$1,347**	**$378**	**$449**	**$126**	**$126**
3-Year Infrastructure Cost	**$157,246**	**$104,459**	**$48,508**	**$13,624**	**$16,169**	**$4,541**	**$4,541**

There are a few assumptions worth noting. The size of each disk was chosen at 1TB, but each organization is likely to have different storage volume requirements, based on their retention and persistence requirements. Also, we used compute and storage rates from AWS US East 1 region at the time of this report. Also, we omitted the root operating system disks for the cluster nodes, which are usually small and are lower cost general purpose storage which would contribute an insignificant amount of cost overall.

Even with purely in-memory streams in NATS, there is still the meta state across the cluster that needs to be maintained which requires storage. It is negligible, but a provisioned disk would still be required.

Also note the differences between the Kafka and NATS. NATS does not require Zookeeper nodes. Also, core NATS, or NATS streams only using in-memory persistence, would not require provisioned IOPS data disks at all. These differences are shown by empty cells in the table above.



With the configurations we tested for equivalent performance, we found NATS requires less infrastructure costs, which vary by whether disk or memory persistence and high availability are required.

People Time-Effort Costs

People's time and effort costs are more difficult to precisely calculate and must be estimated, given the varied nature of possible operations and maintenance scenarios, projects, competing priorities, and other factors. Thus, we will attempt to make some objective inferences based on what we know of the ops and maintenance differences of Kafka and NATS.

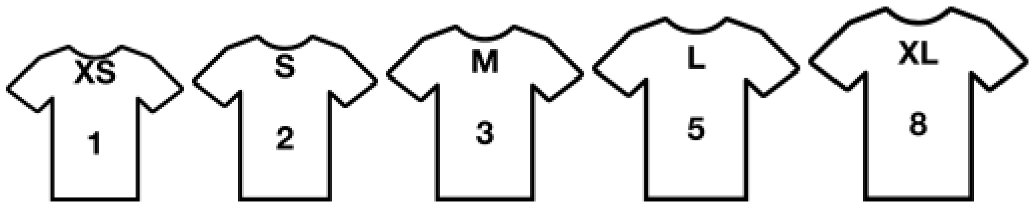
We took inspiration from an agile scrum development project management approach. The agile methodology is much bigger of a subject than the scope of this paper, but for its purposes, we used a few concepts from an agile methodology in this study which may be familiar to readers.

- Story – A story is a specific task that needs to be completed to move forward in development of a new piece of the project or to complete an operational objective.

- Story Size – A story is sized according to how much time and effort required to complete it. Sizing a story appropriately is an art and a science that requires some experience.

- Story Points – Regardless of the sizing method used, the story size is typically expressed as a numeric value to quantify story work and completion.

To quantify size and points, we used both the T-shirt size method and the Fibonacci sequence. We went from the extra-small (XS) size to the extra-large (XL) size. The chart below shows each size and the Fibonacci number assigned to it.



In our study of these platforms and witnessing their usage in the field, we have observed a number of development and operational differences in Kafka and NATS. The following table is a work breakdown of common build and ops tasks and the differences in their time-effort.

	Kafka		NATS JetStream	
Setup Cluster	S	2	XS	1
Tuning Cluster	L	5	XS	1
Configuring Zookeeper	S	2		
Configuring Kafka Streams	L	5		
Configuring MirrorMaker	S	2		
Build Effort Story Points		**16**		**2**
Creating filtered sub-streams vs. subject-based addressing	M	3	XS	1
Adding Brokers	S	2	XS	1
Repartitioning new brokers	XL	8		
Removing Brokers	S	2	XS	1
Repartitioning after removing brokers	XL	8		
Ops Effort Story Points		**23**		**3**

Hands down, NATS is easier to set up, configure, use, and maintain. The resulting time-effort ratio for *developing* on NATS compared to Kafka is 0.125 (one-eighth), while the *operational* time-effort ratio is similar at 0.13 for NATS over Kafka.

Again, this is an over-simplified extrapolation using some of the most basic build and operational tasks. There are many others you will likely consider.

Using those time-effort ratios can give us an estimate of the time-effort costs for our TCO equation. The following tables give a reasonable comparison, given those time-effort ratios we calculated.

To assign a monetary value to people's time, we used a salary of $150,000 a year for FTE and $200 per hour for consulting. This rendered a blended hourly time-effort rate of $116.

People Costs	**Internal Staff**	**External Services**	**Blended Rate**
Average Annual Cash Compensation	$150,000		
Burden rate	22%		
Fully Burdened Annual Labor Cost	$183,000		
Standard Work Hours per Year	2,080		
Hourly rate	**$88**	$200	
% Mix of Labor Effort in Production	75%	25%	
Build Labor Cost per Hour			**$116**

Thus, we can use that amount and our calculated time-effort ratios to determine a personnel cost for using Kafka versus NATS.

Migration/Build Costs	**Kafka**	**NATS**
Ops/Admin FTE	1.0	0.125
Total FTE	1.0	0.125
3-Year Production Costs Total	**$549,000**	**$68,625**

Production Costs	**Kafka**	**NATS**
Ops/Admin FTE	1.0	0.130
Total FTE	1.0	0.130
3-Year Production Costs Total	**$549,000**	**$71,609**

Both cases reveal a difference of about 87%.

When we put both infrastructure and people cost together to calculate total cost of ownership, the following table shows our results. Yours may vary.

	Kafka		**NATS JetStream**			
	5 nodes	**3 nodes**	**3 nodes disk**	**3 nodes mem**	**1 node disk**	**1 node mem**
3-Year Total Cost of Ownership	**$1,255,246**	**$1,202,459**	**$188,741**	**$153,857**	**$156,403**	**$144,775**
% Cheaper than Kafka 3 nodes			*84%*	*87%*	*87%*	*88%*

People-cost often make up the bulk of the expense in enterprise data projects. As you can see in our analysis, the cost of infrastructure makes up only a small fraction of the TCO.

Conclusion

The majority of the workloads we threw at Kafka were disk I/O bound. As we added more provisioned IOPS to its persistent io2 disks, Kafka scaled linearly. Once we surpassed 50,000 total IOPS (5 brokers each with 10,000 IOPS disks), the linear scale pattern broke down.

NATS workloads with JetStream were predominantly CPU-bound or network I/O bound. We exceeded our requirement of 500,000 messages per second with only 3,000 total IOPS per node. As we added IOPS to the disks, we observed that the CPUs were still handling the majority of the work and that the throughput did not increase with additional IOPS; consequently, we were able to maintain the IOPS at 3,000 for each disk in the cluster.

NATS requires less infrastructure costs than Kafka. The costs vary depending on whether disk or memory persistence and high availability are required, based on the configurations we tested for equivalent performance.

NATS is significantly simpler to install, configure, use, and maintain. The time-effort ratio for developing on NATS versus Kafka is 0.125 (one-eighth), while the operational time-effort ratio for NATS versus Kafka is 0.13. So what takes a day with NATS will take about 8 days with Kafka. When we put both infrastructure and people cost together to calculate total cost of ownership, a NATS implementation is substantially less effort.

About Synadia

Synadia was founded in November 2017 by Derek Collison to build the connective fabric for modern distributed systems. The company is behind the popular open source messaging & data streaming software NATS.io, used by millions across the globe. Synadia's diverse customer base ranges from Fortune 500 enterprises in finance, retail, industrial manufacturing, healthcare to innovative startups in fintech, IoT, AI, ML and gaming.

About McKnight Consulting Group

Information Management is all about enabling an organization to have data in the best place to succeed to meet company goals. Mature data practices can integrate an entire organization across all core functions. Proper integration of that data facilitates the flow of information throughout the organization which allows for better decisions – made faster and with fewer errors. In short, well-done data can yield a better run company flush with real-time information... and with less costs.

However, before those benefits can be realized, a company must go through the business transformation of an implementation and systems integration. For many that have been involved in those types of projects in the past – data warehousing, master data, big data, analytics - the path toward a successful implementation and integration can seem never-ending at times and almost unachievable. Not so with McKnight Consulting Group (MCG) as your integration partner, because MCG has successfully implemented data solutions for our clients for over a decade. We understand the critical importance of setting clear, realistic expectations up front and ensuring that time-to- value is achieved quickly.

MCG has helped over 100 clients with analytics, big data, master data management and “all data” strategies and implementations across a variety of industries and worldwide locations. MCG offers flexible implementation methodologies that will fit the deployment model of your choice. The best methodologies, the best talent in the industry and a leadership team committed to client success makes MCG the right choice to help lead your project.

MCG, led by industry leader William McKnight, has deep data experience in a variety of industries that will enable your business to incorporate best practices while implementing leading technology.

www.mcknightcg.com