EPISODE 588

[INTRODUCTION]

[0:00:00.3] JM: Message broker systems decouple the consumers and producers of a message channel. In previous shows, we've explored ZeroMQ, PubNub, Apache Kafka and NATS. In this episode, we talk about another message broker, Apache Pulsar. Pulsar is an open source distributed pub/sub message system originally created at Yahoo. It was used to scale products with high volumes of users, such as Yahoo Mail.

There are three components of a Pulsar deployment; the Pulsar broker, which handles the message brokering, Apache bookkeeper which handles the durable storage of the messages. and Apache zookeeper. which manages the distributed coordination. Lewis Kaneshiro joins the show to describe how Apache Pulsar works and how it compares to other messaging systems like Apache Kafka. Lewis is the CEO of Streamlio, a company that builds messaging and stream processing systems for enterprises and it uses Pulsar in its core product.

[SPONSOR MESSAGE]

[0:01:12.0] JM: At Software Engineering Daily, we have user data coming in from so many sources; mobile apps, podcast players, our website, and it's all to provide you our listener with the best possible experience. To do that, we need to answer key questions, like what content our listeners enjoy, what causes listeners to log out, or unsubscribe, or to share a podcast episode with their friends if they liked it. To answer these questions, we want to be able to use a variety of analytics tools, such as Mixpanel, Google Analytics and Optimizely.

If you have ever built a software product that has gone for any length of time, eventually you have to start answering questions around analytics and you start to realize there are a lot of analytics tools.

Segment allows us to gather customer data from anywhere and send that data to any analytics tool. It's the ultimate in analytics middleware. Segment is the customer data infrastructure that has saved us from writing a duplicate code across all of the different platforms that we want to analyze.

Software Engineering Daily listeners can try Segment free for 90 days by entering SE Daily into the how did you hear about us box at sign-up. If you don't have much customer data to analyze, Segment also has a free developer edition. But if you're looking to fully track and utilize all the customer data across your properties to make important customer-first decisions, definitely take advantage of this 90-day free trial exclusively for Software Engineering Daily listeners.

If you're using cloud apps such as MailChimp, Marketo, Intercom, Nexus, Zendesk, you can integrate with all of these different tools and centralize your customer data in one place with Segment. To get that free 90-day trial, sign up for segment at segment.com and enter SE Daily in the how did you hear about us box during signup.

Thanks again to Segment for sponsoring Software Engineering Daily and for producing a product that we needed.

[INTERVIEW]

[0:03:41.9] JM: Lewis Kaneshiro is the CEO of his Streamlio. Lewis, welcome to Software Engineering Daily.

[0:03:46.9] LK: Thanks so much, Jeff. Thanks for having me. I really appreciate the time. I'm excited to talk about Streamlio.

[0:03:52.1] JM: Absolutely. Let's start with Pulsar. Apache Pulsar is this messaging system. Explain the purpose of a messaging system.

[0:04:02.4] LK: Sure. The purpose of a messaging system is really connecting data, connecting users of data in a very flexible manner. What a messaging system allows is data to come in from really any source. We think about data coming in with our customers as sources, such as mobile apps, it could be generated from sensor data, it could be user-generated, it could be eyeballs, it could be clicks.

A lot of data that is coming into a messaging system really has a destination. The destination will be a consumer of that data. The consumer of the data can change. It can be a number of different users downstream, that are actually unknown at the time that the data is being generated. A messaging system really allows for a layer that allows developers to access that data in a very flexible manner.

[0:04:57.2] JM: There are two messaging models. We've explored these in recent episodes about Kafka and NATS. There's the queuing model, there's the publish-subscribe model. Can you describe why these two types of messaging exist?

[0:05:13.5] LK: Sure. We actually take the unified approach at Pulsar, where we unify queuing and pub/sub, or streaming. The reason why both exists is up until now, actually up until Pulsar, those systems are serving different use cases. In queueing, order does not matter. What really matters is that an event, or a message that's traveling your queueing system really has a requirement to be processed. It doesn't matter whether or not it's processed immediately before or after any certain event. As a result, you can have a number of different clients that can process these events in parallel.

What you can do is scale out the performance of a queueing system by having many more customers ingest that data, but the order doesn't matter. That's very different than pub/sub type systems, where the order does matter. Pub/sub you have mentioned Kafka, but we also have a very similar model with Pulsar, which allows that data in order to be replayed played to different consumers and serve use cases where the ordering does matter, and the history matters and orders matters.

With Pulsar, the model that we have that we've developed really allows both queuing and streaming use cases. What's exciting about that with customers and really groups that are familiar with queuing and streaming systems, they usually have multiple systems that are dealing with both use cases. Queuing, you could have ActiveMQ, or different messaging queuing type systems. RabbitMQ, RocketMQ, those types of systems come to mind.

Then streaming use cases, what enterprises are doing is bringing in a separate system, typically Kafka, you mentioned NATS as well. The way that we approach the space is these are

enterprise use cases. From the start, from day one Pulsar was designed to handle both, and so we simply have queueing semantics, where consumer groups can connect on to a topic, so not delving too deeply into vocabulary, but a topic is simply that that messaging type of system, and you can either have a queueing use case where messages are processed out of order. It can be scaled to parallelize wise and really deliver on whatever performance is needed.

Then once a message is processed, it can be deleted. As a result, you don't have a queue that continues to grow indefinitely. You're actually able to consume out of that queue and delete the data, if that's needed. On the pub/sub type of model, what you can do is have the data remain in the system, and so that's the other very exciting thing about Pulsar, where it's a true event storage system underlying Pulsar, we can get into architecture. Underlying Pulsar is a true stream storage system called Apache bookkeeper.

That allows users to store data really definitely. Simply by adding nodes, you can you can have a streaming system that has as much data as you would like to retain. What that allows us to do with enterprises in use cases is really explore both queuing and streaming in a single system that that we feel is really the future of event-driven architecture at enterprise scale.

[0:08:47.4] JM: We will explore that architecture in a bit. I want to start from a little bit more foundational level. There are people out there listening, that probably just think of application development in terms of request response. You've got a client device, maybe it's your laptop, maybe it's your smartphone. You do something like click the button on Facebook and you just imagine that that request gets hit to Facebook's servers and it executes some like method. Then the liking target has been liked and it gets updated on your client device, it's just a request response.

In that mindset, the listener might not be sure, why would you need a messaging system? My client device is just making a request to the server and a response is being received. When does an application model need a messaging system?

[0:09:46.8] LK: That's a good question. When we think about it, it's really a matter of scale. If you have a very, very, very simple query, queue and response, basic question response and the system is built around only that architecture, you want to click a light button to anyone to hit a

server and you want to come back, there's really no flexibility in that model. Let's say you have business – a BI use case, where that click data also wants to be analyzed in some sense. That would actually require either a server-side analytics, and so you want to be storing all of the click data and the response data in order to get analytics on that data, or on maybe the client side you need some method to be analyzing that data.

What's interesting is in that response, is it's difficult to scale that type of architecture and it breaks that flexibility mindset that I talked in the beginning. When you put a messaging system, a true messaging system in that type of query response pattern, what you're allowed to do is now use that topic, that data stream to do a lot of different things. A lot of those different things is really where enterprises are gaining value with their streaming data.

Some types of value around that is analyzing, so having real-time, or streaming analytics, you could do a lot of different things with that clickstream data, with data scientists and being able to analyze that data, being able to enrich and augment that data. What a messaging system allows you to do in Apache Pulsar in particular, is you're allowed to create just-in-time transformations on that data. As the clickstream data comes in, you can augment it with a number of different services within your enterprise. You could be hitting a number of different databases, for example. You can be doing predictions on say, gender, or maybe different user-generated data and make a prediction on what that user is doing and be able to deliver back a much richer experience to the user, rather than having a very simple almost static architecture that is a query response.

[0:12:04.0] JM: Right. You mentioned the – you can have basically the consumer and the producer having different loads. For example, if we're talking about the like button, maybe you have thousands and thousands of people that are liking stuff throughout the day and there's different workloads, there's different workload capacities throughout the day, so the number of people liking stuff throughout the world may vary. It's useful to have this scalable buffering infrastructure, so that if you have the server that is reading those likes and processing those likes.

The server may not necessarily have to scale up or down, you just have the messaging infrastructure in between scale up and down, the messaging infrastructure buffers all those likes

and the server that's processing those likes just processes them as it would like to. In addition to that, then since you have this buffer that is containing all the requests, if there's other consumers that you want to respond to that like button, other than just the like processing server, you can have these other consumers. This is something that the guest I had on, who was talking about NATS was emphasizing, was the fact that you don't ever know, you don't ever want to assume who is the recipient of a message.

Maybe there's other people within face – if you're talking about the Facebook design and the Facebook infrastructure, maybe there's other people that want to know every time a like gets submitted to Facebook servers, maybe there's people in the analytics team, or people in the reactions team that want to know about this data. If you share the messaging system between them, then they can both use that infrastructure, that that queue of information and that's why you're talking about the storage system underlying this. Now that we've outlined the basic idea of a queueing system, why was Pulsar originally built?

[0:14:10.4] LK: Sure and that's great overview. What's exciting about messaging systems in general as you point out Jeff, is there's a lot that you can do with this incoming data. Traditionally, in the era of slow data as we like to call it, the data was stored. The data comes in, it's stored in a database, maybe a data lake and it takes a lot of time in order to get that data out and actually deliver value on that data.

When you have a decoupled system, and many different teams are able to use that data, either in real-time as it's being generated, or streaming across an organization and having multiple teams use that same data, that data becomes very valuable. That's really where Pulsar came into being. When Pulsar was created, there were a number of existing streaming solutions and queueing solutions that were already on the market, or really open source.

Apache Kafka as you've mentioned before, was open source about eight years ago. Hadoop era technology where it is focused on use cases, moving data, data motion, that were eight or nine years ago. When Pulsar was created, it was really created to solve enterprise-grade use cases from day one. What that means is for listeners that don't know Pulsar too much, it was created in Yahoo and open sourced about a year and a half ago.

At Yahoo it runs all real-time, basically all real-time messaging services at Yahoo. At the beginning, Pulsar was really created to focus on a number of different capabilities. One is durability, so data is replicated and synced to disk. Ordering and delivery guarantees, it was focused on, but an enterprise-grade requirement that Pulsar solved for Yahoo was this idea of geo replication.

Having multiple data centers and needing that data to be available at all data centers immediately is a very different use case. It's a very different system that's needed. That was a day one requirement to Pulsar. The other is multi-tenancy. This idea of a single cluster, a single messaging cluster serving the entire enterprise, so you can think of a global and enterprise wide Pulsar cluster is needed when you consider the number of teams that want to access this data.

Going back to really why messaging systems exist and the idea of why we're excited about messaging systems now is the idea that a number of teams want to access this data. Having multi-tenancy as a day one requirement really required in a new system. That's where Pulsar was created, so zero data loss, geo replication at Yahoo scale, and multi-tenancy, and that is the development of Pulsar as is really serving both queueing and and pub/sub or streaming use cases.

[0:17:16.3] JM: What happens when a Pulsar cluster is spun up for the first time?

[0:17:23.6] LK: What happens when a Pulsar cluster is spun up for the first time, Pulsar is actually leveraging another Apache project; Apache bookkeeper. Bookkeeper as I mentioned is true distributed stream storage. What that means is very fast zero data loss, geo-replicated system. When Pulsar spun up, what it's doing is connecting to bookkeeper and bookkeeper underlying bookkeeper is also connecting to Apache zookeeper.

When it's spun up, what Pulsar does is enable – there's many different ways to spin a Pulsar cluster, but what Pulsar allows you to do with multi-tenancy is to have a namespace. With that namespace, you're allowed to select what region, what cluster and what team that topic is being written to and accessible to. That actually allows security authentication, everything that you would expect in an enterprise system is spun up when you spin up a single Pulsar cluster.

As you begin to connect up, you have producers and consumers per topic and that's per namespace. You can obviously spin up a default namespace and that's something that we added from prospect demand, because what they really wanted was a system that was as simple to use as possible. You can have default namespaces, but you can also connect up with a multi-tenant system, which allows isolation at all levels.

[SPONSOR MESSAGE]

[0:19:08.8] JM: If you love Software Engineering Daily, I think you'll also love the Google Cloud Platform Podcast. It's a podcast about Google Cloud Products, how they're built and how you can use them. Really it's all about the changes that are going on in software engineering, as told from the point of view of Google engineers.

You can find the Google Cloud Platform podcast at gcppodcast.com. You'll hear from Googlers like Vint Cerf and all about Tensorflow and Firebase and BigQuery, from the high-level use cases to the low-level implementation. The GCP Podcast has all of that covered.

Find the Google Cloud Platform Podcast at gcppodcast.com and subscribe to it wherever you get your podcasts.

[INTERVIEW CONTINUED]

[0:20:00.2] JM: The three components of a Pulsar deployment that are worth addressing the Pulsar broker, which handles the message brokering; Apache bookkeeper, which handles the storage and zookeeper which enables the distributed coordination. Describe these three components in more detail and outline their purposes.

[0:20:24.5] LK: Sure. Pulsar's architecture is really a separated broker as you mentioned and bookie system. The broker system, what that's allowing is really a separation of the read and write path. As a broker ingest data, it connects up with zookeeper, which is managing the bookies. The bookies manage the write path to disks. That's physical disk of syncing to disk in a segment-based architecture.

Within this, the zookeeper cluster is simply managing the coordination and metadata of the system itself, but what this architecture allows is the independent scaling of brokers and bookies. That really comes into the cost equation. Again, enterprise requirements from day one really require the system to be able to scale, scale efficiently.

As more and more data comes into the system and if there's use cases that require longer and longer storage, what's key about Pulsar is we simply need to add nodes and bookkeeper nodes, and that's really bookie nodes, which are much cheaper than adding brokers. With that architecture, you're able to scale and we're able to scale without rebalancing. We hear that that's pain points with other systems.

It also allows to scale basically as the enterprise scales, or for example if you have spiky data, you could have spiky increased data events around, say the Super Bowl, that with other systems it's required to do very costly rebalancing, or partitioning of nodes. With Pulsar and the separation of architecture, it's very simple to add nodes. That's the separation of brokers on Pulsar that are managing writes to bookies.

[0:22:24.2] JM: Okay. Help me understand, what happens when a message is sent from a publisher? Maybe you could outline a typical use case for the pub/sub system. I mean, or the queuing system, I guess it's unified in the Pulsar world. Just describe when a message is sent from a publisher and consumed by a consumer, walk me through the different steps that happen.

[0:22:51.8] LK: Sure. When a producer writes a message, it's going to write that message to a topic in Pulsar. Again, that topic is in a namespace. When it's written to a topic in Pulsar, the broker will then take care of connecting with a bookie in bookkeeper, and the bookie will take care of replicating and writing that not only message, but that segmented message to disk.

What we mean by segmented message is that message can be efficiently stored and replicated up to basically a configure a number of times, and so you could have in a particular use case, and let's call it a financial use case, where the message is being written is an actual transaction. That message needs to be acknowledged when it is written to a disk and replicated a certain number of minimum time.

What the system can do is configure save five writes, and a message will be written up to – well, the message will be written five times, but you can configure the minimum acknowledge number. You could have a number of five writes, but minimum acknowledgement of three, for example.

Once the bookie successfully writes to disk three times, then it will reply back with an acknowledgement. Once that message is written to disk, you can have a consumer consume that message and the consuming message will again access a broker, and that broker will access the bookie written to disk.

You can also have that message in in-memory, and so it can be a very, very fast consume. The great thing about Pulsar and writing to bookkeeper is that we have latencies on the order of some five millisecond latencies at the top end. Very, very fast writes to disk, and so you have a very efficient zero data loss solution for any message that's acknowledged. We can delve deeper into the architecture, but that's the route. The producer writes to a broker, the broker then writes to a bookie, which is replicated to a configurable amount of time. The consumer then will consume that message again by accessing the broker. Producers and consumers can be served by different brokers which again allows for very fast reads and writes.

[0:25:26.0] JM: When there's a high volume of producers that are writing to the Pulsar cluster, is there any scalability that needs to happen?

[0:25:37.0] LK: Any scalability that needs to happen, if the producer is writing too fast, you can obviously scale up the number of brokers. If the bookies are writing to disk and disks are filling up, you can again, increase the number of disks simply by adding nodes. It was really built to be very efficient to scale architecturally. In that sense, producer writes is easy to scale up and consumers as well.

[0:26:13.4] JM: There is this typical use case of enrichment. If I for example have a Fitbit, I'm wearing a Fitbit and it's broadcasting my location data on a regular basis to some server, and I want to enrich that – let's say it’s lat/long coordinates. It's just broadcasting lat/long coordinates to the server. On the server side, they would want to enrich those lat/long coordinates with

location data, for example. This is a typical example of when you would want to do an enrichment and it's a typical example of when you would want to buffer up your messages in the pub/sub system and do the enrichment, and then perhaps write it back to another topic, the enriched version of the topic.

When the consumer actually consumes the message, it has enriched data, like the zip code and the state and all the other location data that goes along with lat/long, but maybe you would not want to have broadcasted from the original message from the Fitbit to the cluster. This use case of enrichment seems like a great purpose for these rich pub/sub queueing systems. We've talked about in our shows about Kafka. If I wanted to enrich messages by reading a topic and then writing to a new topic, can you do that in Pulsar and what are the best practices for doing it?

[0:27:45.1] LK: Yeah, absolutely. That's again, as you mentioned, a very common use case. With Pulsar, we have functionality called Pulsar functions, which serves as lightweight compute. We like to call it stream native computing. What this does is fill up, or really solve almost a microservices-type architecture. As the data comes in from the Fitbit, that has user data, but maybe not lat/long data, that data wants to be enriched. Perhaps the enrichment data is in an aerospike database, for example.

As the message comes in and is buffered up within the same system and actually on the broker itself, Pulsar functions is in Java and Python. What you can do is per message, query a database if needed, have that result come back within the same system, enrich the data so maybe join that to hold together with a lat/long and whatever other data you want, perhaps gender, user data, what status or a class that user is, and then write to an output topic downstream. That topic can be based on – it could be even based on a machine learning model, which is a very common design pattern that we're seeing.

Within the same system, and that's pretty key with Pulsar functions in the way that we've designed it, was to be as simple as possible for a new developer to come on to the system, not need to learn any new API, not to need to know functional programming, for example, and be able to solve that exact use case. Enrich data, write out to a topic, or any number of topics, or topic that is predicted based on some machine learning model that the Pulsar functions is able

to access, maybe in a switch statement and have that data then, that enriched data be accessible downstream.

That use case is something that we see more and more often. It gets back to the entire idea of how multiple teams may want to enrich incoming data for their own use cases, and then write that data to another topic and expose that enriched data to other teams downstream.

[0:29:59.8] JM: Okay. I want to get into some comparisons, because we've done shows, like I said on Kafka, we've done shows on NATS and Google pub/sub. How do you look at Pulsar's place in the market of these different distributed queuing systems?

[0:30:18.2] LK: When we look at the market, and going back to when and why Pulsar was created to address some of the limitations of systems that were available and actually quite popular at the time, one of the core differentiators that we see is Pulsar accessing true distributed stream storage in bookkeeper.

What that allows is a distributed event storage system that allows any amount of history to be stored. By solving storage first Jeff, what we found is that it's actually unlocking a number of new use cases, that quite frankly with limited storage would be very difficult. For example, since you want direct comparisons with Apache Kafka, we here it's very difficult to store many, many days' worth of data, or store, or increasingly store data.

With Pulsar, again what you do is you simply add nodes, you can scale up the duration in which you want to store data. What we're finding is users are using longer and longer data periods to train and retrain machine learning models, for example. Another request that we've had that we're delivering on very shortly is this idea of tiered storage.

What tiered storage allows is data accessed in the streaming fashion, a pub/sub fashion can simply exist in Pulsar, but as data rolls off, or becomes older, let's call it after two or three weeks, we can configure that data to be automatically stored in a tiered fashion externally, for example in S3. What this allows is a developer, she can access the same data with the same streaming paradigm, but that data can exist in S3, in bookkeeper/Pulsar and be available without changing the paradigm.

Compared to other systems, we find that's unique, because with other systems you're actually required to unload the data yourself, or manage the data yourself, rather than accessing this idea of any amount of streaming data over any amount of time. Again that's geo-replicated, accessible in any region.

Because we're a replicated system, that's where we find the comparison most often against Kafka. We do see comparisons on the queuing side, call it to ActiveMQ and RabbitMQ. Again, where we're coming from is a very flexible unified system.

We know that NATS is a newer system. Although, we don't see it as often because – and stop me if I'm wrong, but we don't see the replicated, the storage of data yet. The NATSio use cases, we haven't come across that as much. Honestly, where we see a lot of excitement and attraction around Pulsar are with users that have maybe found some limitations in existing systems, like Apache Kafka, or they're looking for a unified system, or they're looking for a use case solution that is really leveraging some of the features and functionality that Apache Pulsar was created for. Namely, zero data loss, fast and durable type storage, geo-replication for sure, and multi-tenancy.

[0:33:41.0] JM: Interesting, so really the use case that you're going for is the durability and accessibility of the messages that are being written to that queue.

[0:33:56.0] LK: That's correct. If we take a step back and why are we excited about this phase, if you look at typical enterprise architectures now you tend to see large data stores, that could be Hadoop, that could be a data lake, that could be this, we like to call arrow slow data solution, where data is simply stored and in our strong opinion, that the value of that data decreases over time.

The most exciting data and enterprise is the most recent data. That allows use cases real-time enrichment of data that is returned back to the user, so real-time interaction with customers. As a result, there's been new systems like messaging and streaming systems that are trying to access that real-time and streaming use case, but then that data is then offloaded on to say a data lake, or different databases.

What Pulsar allows is this almost unification of real-time and stored data. It's the unification of messaging and storage. Really, we're able to look at data as simply data. We're not looking at data as data in motion and data at rest, because what we're really doing is unifying that and that's at the core of why we're excited about Pulsar as a system, because we're no longer talking about simply a messaging system, or simply pub/sub.

We're actually allowed to go after use cases that are really looking at what an application developers want, which is I just want my data. I want enriched data in a streaming fashion, in real-time, but also accessing historical data. That's really what we get excited about and use cases that need that in an enterprise-create fashion. Zero data loss. It's not an okay solution. It was it was really architected and designed from day one to solve these core enterprise use cases.

[SPONSOR MESSAGE]

[0:36:03.1] JM: If you are on-call and you get paged at 2 AM, are you sure you have all the data you need at your fingertips? Are you worried that you're going to be surprised by things that you missed, errors, or even security vulnerabilities because you don't have the right visibility into your application? You shouldn't be worried. You have worked hard to build an amazing modern application for your customers. You've been worrying over the details and dotting every I and crossing every T.

You deserve an analytics tool that was built to those same standards, an analytics tool that will be there for you when you need it the most. Sumo Logic is a cloud-native machine data analytics service that helps you run and secure your modern application.

If you are feeling the pain of managing your own log event and performance metrics data, check out sumologic.com/sedaily. Even if you have your tools already, it's worth checking out Sumo Logic and seeing if you can leverage your data even more effectively with real-time dashboards and monitoring and improved observability.

To improve the uptime of your application and keep your day-to-day runtime more secure, check out sumologic.com/sedaily for a free 30-day trial of Sumo Logic. Find out how Sumo Logic can improve your productivity in your application observability whenever you run your applications. That's sumologic.com/sedaily.

Thank you to Sumo Logic for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:37:47.4] JM: Does Kafka have – because I think Kafka does retain all that event data that is written to Kafka, but maybe does it write it to disk more often? Is it slower to access than in Pulsar?

[0:38:03.9] LK: It's definitely slower to access. We have benchmarks, third-party benchmarks through [inaudible 0:38:08.3] and a project called open messaging that includes a benchmark, but not only is it's slower, but there is a configurable time to live, right? You're basically configuring the length of time that you want the data to exist in the system, because you can't have an infinite queue. What we're finding with users of Kafka that may be having problems with Kafka is this idea of needing to buffer up the amount of data that they really need.

Let's say they want three days' worth of data, they have to buffer up that amount of data to say seven days, just to allow a no data loss type of solution. With Pulsar, it's very simple to add nodes as needed. It allows us to lower the total cost of ownership both through higher throughput, easier operations, and this idea of when or if an issue occurs, you can simply add on nodes at the time that it's needed, scale up the system because it's very simple to do.

When that event is over, it could be planned, again like the Super Bowl, or it could be unplanned, call it for maybe a data center goes down, or if there's some issue on the application side, where data needs to buffer up. With other systems, again as users are considering a change from their system to Pulsar, those are the issues that we hear about most often. That's really where we're excited that Pulsar solves these Daemon.

[0:39:42.9] JM: I see. That infinite queuing idea, so I think the idea is if I connect a new application to my event bus at this event queuing system, and we're talking about either Pulsar, or Kafka, or NATS, or whatever it is, sometimes you're going to want to replay every single event from the beginning of time, in order to get this new system up-to-date.

If you don't have all of that data in the queuing system, then you're not able to do that and you would have to reload it from S3, or from some other data lake that you have all your events stored in. Are you saying that in Pulsar by default, you have all of this data just saved, that's it's just sitting in the queueing system, in Apache bookkeeper?

[0:40:39.8] LK: This is where that idea Jeff, of tiered storage comes in. Rather than having it sit in bookkeeper, which you can. You could obviously scale up the system and have data sitting in bookkeeper, if you wanted your bookkeeper cluster to grow that large. With tiered storage, what you're allowed to do is behind the scenes offload data that is beyond that configured period, and store that in a more cost-efficient manner.

That data can be stored in S3, but access through the same streaming APIs as if the data was in the same system. Really we're getting to the idea of just accessing data and your application developer does not need to know where that data is sitting. Really the data could be sitting three, five, 10 days, or months in a bookkeeper cluster if that's needed. If you want to replay data from the beginning of time as you said, and you had tiered storage configured into S3, for example, then you're able to access all of that data and the data and bookkeeper seamlessly.

The key point there is that we're abstracting away where the data is residing and we're not limited to the smallest disk, for example, which is a core limitation we've discovered with other systems. That's really where that sophistication of Apache keeper and true stream storage is coming from. The segment-based architecture is taking the same logical view, and so an append-only log storage replicated system, but on the physical side what it's doing is using segment-based storage. What that allows us to do is very efficiently store older data in whatever storage in a tiered fashion the user needs.

[0:42:31.9] JM: Does Kafka have a tiered storage model?

[0:42:34.9] LK: Honestly, I do not think so. That would be left to a Kafka expert. There is ways to unload data obviously from Kafka into S3, or any other database. To access it again, the application developer needs to reload that data into Kafka, and then access data as if it was in Kafka.

That actually came up as a request for large Kafka users, that were asking for this idea of saying, “Yes, we love the streaming paradigm and streaming data,” but there's actually limitations and those limitations come out on the architectural side. Those architectural issues are exactly what Pulsar was created to solve.

[0:43:23.1] JM: What about the cloud provider solutions; Google cloud pub/sub and Kinesis?

[0:43:28.8] LK: Sure. Google pub/sub we're familiar with, because it comes up quite often. What we have found with pub/sub is ordering, especially geo-replicated ordering becomes questionable. Definitely on the on the cost side, we have a significant cost reduction compared to Google pub/sub, and Kinesis as well.

Kinesis does have a tiered storage type of functionality called VCR. The VCR Kinesis functionality is something that is very similar in some ways to tiered storage. What's nice about that is Kinesis really doesn't expose where it's storing the data, or how it is allowing users to replace streams. Against Kinesis and especially the most common pattern we see is Kinesis to Lambda, or even SQS to Lambda. We see enterprises using Kinesis, SQS, Lamda and maybe SMS, a lot of different systems. What we deliver on is ease of use there, where we're actually combining multiple different systems and deliver that on a much lower cost.

[0:44:38.0] JM: I think the bigger picture of what you're building Streamlio is not just a system with Pulsar. Pulsar is just one component of it. Pulsar is the messaging pub/sub queuing component of a larger system for queuing and processing the data that is queued. There are these varieties of streaming systems. I think two or three years ago, there was just this huge swarm of streaming systems, where you had Spark, Spark streaming, Storm. Heron from Twitter, which I think is what you use at Streamlio, and the purpose of a streaming system is to be able to read in these high volumes of data and process them in a distributed fashion.

You've got this this architectural pattern where you have all of these high-volume of events that are being queued up, and then you have a streaming system that processes batches of events that come in, or maybe they process each event that comes in one by one. Twitter Heron is what you are using for that that stream processing. Explain what Heron does.

[0:45:52.8] LK: Sure. You're absolutely correct that at the Streamlio level, and this goes back to actually one of your earlier questions around microservices, there is a directed acyclic graph, a DAG pattern of stream processing that was pioneered by Storm. There are systems before it, but really a lot, a wider adoption system with Storm.

Within Twitter, a Twitter size data and demands, there was actually a lot of architectural issues that came up with Storm. Storm was open sourced by Twitter, but then replaced within Twitter by Heron. Heron, really what it did was solve a number of those operational issues at scale, and was open sourced by Twitter about two years ago, and was recently added to Apache as an incubating project back in March.

What Heron does is a distributed stream processing system that can read data from any number of messaging, or queuing systems, join and enrich that data together and be able to do large-scale aggregations; things like counting up hashtags and tweets, or running all real-time compute and advertising at Twitter scale.

You're absolutely right that a few years ago, there was a number of stream processing systems that came out, but what we're actually seeing with customers and wider adoption of streaming systems is that Heron does solve large-scale use cases very well. Twitter scale use cases will require systems like Heron. What we actually found with prospects and customers is that Heron and larger scale real-time compute systems were actually a bit heavyweight for the use cases that are really going into production now.

That very simple augmentation use case from Fitbit that you described, it's definitely possible to solve that with Heron and be able to augment that data with existing connectors in the Heron system itself. What we've done is actually simplify that down and be able to leverage Pulsar functions and do that in a stream native manner.

What we're finding is that yes, Heron is an exciting system and part of the Streamlio platform, but really large banks, IOT use cases and social media mobile company is that Twitter scale are really excited about our capabilities on Heron, especially Heron on Kubernetes. As you see the enterprise's adopt Kubernetes, this is the perfect solution for extremely large-scale stream processing.

What we also found was we wanted to be appealing to enterprises at all levels, and so that's where definitely Heron is possible for over a stream processing perspective, but we're actually very excited about Pulsar functions that's able to not only do the stream native compute use cases, but also the ingestion and writing of data. Connectors based on Pulsar functions and a very simple system.

As you can see that use cases around queuing and streaming link to Apache Pulsar are growing and while we have real-time compute and a true stream processing engine in Heron, we have found quite frankly that that's very appealing to extremely large enterprises, but more efficiently solved very simply lower total cost of ownership, smaller team with Pulsar and Pulsar functions.

[0:49:46.2] JM: I was just at KubeCon in Copenhagen and one of the things that makes me pretty optimistic about companies that are selling to enterprises, the vendors that are – software vendors that are selling to enterprises, is there is – so first of all, there's so many enterprises that want to buy this stuff, right? There's a huge variety of flavors of those enterprises. There's banks, there's oil companies, there's consumer packaged goods companies, there's startups that may not have a core competency in software engineering, so they're looking for a solution that is easier to use for them.

Then in the Kubernetes world, you just see this thousand flowers – well not thousands, but there's a lot of vendors. There's space for all these vendors, because the economics of being a vendor are quite good if you can establish yourself with an enterprise and build a relationship with that enterprise, they're probably not ever going to take out your enterprise, your software solution that you've made for them.

I'm really curious, because when I walk around these expo halls at these conferences, it is so interesting seeing the different positioning, the marketing. I mean, I know a lot of people listen to

this don't really care about marketing, but if you're ever in a position where you're buying enterprise software for your company, you will care about the marketing, and you're going to walk around one of these expo halls and you're going to see 50 different Kubernetes vendors offering what sounds like the same thing, and you're going to scratch your head and be like, "What the heck am I doing?"

It sounds like at Streamlio, you have a little bit less competition, than perhaps the managed Kubernetes provider market. There are competitors, there are a variety of these pub/sub and streaming solution providers. Tell me about the go-to market strategy. How do you position yourself? How do you make yourself more accessible to the right enterprises? How do you find your customers?

[0:51:48.2] LK: Sure. That's a great question Jeff, because it's definitely crowded mainly because the number of use cases around streaming, to some extent real-time, but really streaming are exploding. There's different industries that want to access streaming data in a real-time fashion. What we're finding is that enterprises want to deliver value on both real-time and streaming data, and re-engineer their architecture to be more microservices-oriented, more flexible and gone are the days of applications that are difficult to scale and are difficult to scale with independent components.

Where we see and really the vision of Streamlio when we first started was the idea that accessing fast data is too complicated. It's too complicated for small enterprises especially, but it's also very complicated and almost impossible for large enterprises that maybe can't hire out an entire platform team to stitch together perhaps a number of different open source and proprietary and maybe even custom solutions and integrate that into their enterprise roadmap somehow.

Being able to deliver an intelligent, fast data platform to enterprises of all sizes is really where we position Streamlio and the vision of we're taking Streamlio as a company. The go-to-market strategy and where we find a number of customers, in the beginning we actually believe that this stream processing and compute piece would actually resonate with existing stream processing users and a number of different features of say Pulsar and bookkeeper would be useful in that solution.

Again. in in the reality of what we saw in the vendor space was a number of vendors that were stitching together open source, or proprietary pieces that really weren't really weren't making fast data easy and seamless. What we ended up finding was that there were a number of different issues that we began discovering with users of different systems.

Enterprises that were having existing problems with say Kafka became really interested in Pulsar, especially around zero data loss, multi-tenancy and geo-replication in particular. In open, source say compared to mirror maker, or we know there's a startup behind Kafka that has a proprietary piece for geo-replication. Then definitely on green fields, as you mentioned walking around KubeCon and expos, you see a number of vendors into space, but really it's around the opportunity and value of fast data.

We do see that Streamlio where we believe this idea of slow data is ending and every enterprise needs an answer to how are we as an enterprise going to access that value of fast data. What we have found is that a managed solution, really a multi-cloud solution but a managed solution is where enterprises want to begin with, and long-term their roadmap definitely has one cloud provider at a minimum, but more often we're seeing multi-cloud providers in addition to an on-prem solution.

We think the future for hybrid cloud is very bright, and hybrid cloud solutions definitely need a messaging layer to move data between each of those platforms. Then within each platform geo-replicated as well, which is again a perfect enterprise day one use case for Pulsar.
Managed solution and long-term, we have seen a lot of excitement around a SaaS offering. Internally, that a intern on the system that is a multi-tenant solution. Leveraging that and delivering direct cost savings back to enterprises looking for a cheaper alternative to access fast data is definitely exciting. Then all the way to the edge and we have gotten into the conversation on what the edge actually means, but in our minds an edge cluster, so a cluster that exists geographically near to the edge and aggregating data, what we are seeing is Kubernetes deployments, or even standalone modes, we do integrate with nomad, for example at the edge.

This idea of a fast state of fabric that connects edge to on-prem to a multi-cloud hybrid deployment is where we're seeing enterprises know that their roadmap must take them and

Streamlio is perfectly positioned to start out very simply, either on-prem, or in a managed solution type of deployment. Also perfectly positioned, for the hybrid data fabric type of deployment that we see use cases developing both now and exploding in the future.

[0:56:55.1] JM: All right, well I'm glad we got to edge computing as well. We've ticked all of the Software Engineering Daily buzzword bingo – not that edge computing is a buzzword. I mean, this topic fascinates me and the question of how you shuttle data and communicate data between these endpoints that are maybe on the ocean, or across an oil rig, and there's variable levels of connectivity, but the volume of data is extremely high. This is a large greenfield set of problems, and I'm looking forward to seeing how you're going to tackle it.

[0:57:37.7] LK: Exactly. I mean, again Jeff, this this is where lightweight compute and Pulsar functions fits in perfectly. Having a Pulsar cluster running at the edge, Pulsar functions are running on the existing broker, so it's not a separate cluster that needs to be managed. The common use case at the edge, let's call it in connected cars and the edge, or Wi-Fi towers is the idea of filtering down data. There's too much data at the edge to send everything back to an on-prem cluster.

Let's say you want to step down and filter out that data 1/10th, or 1/100th, that's a perfect use case for lightweight compute and stream native computing and Pulsar functions. Really one of the reasons of why we developed it the way that we did to have an extremely light footprint, very easy to manage operationally. Then a Pulsar node, Pulsar edge node again fits into the multi-tenant requirement. That multi-tenant requirement simply is an edge tenant that you're able to spin up and spin down from the main cluster, connect seamlessly.

We don't see geo replication needs at the edge just yet, but that's definitely a possibility if it's needed. Again, you're seeing that for IOT use cases, Pulsar functions and Pulsar in general are leveraging all these great functionality that it was created for as a day one requirement at Yahoo.

[0:59:06.6] JM: All right. Well, Lewis Kaneshiro, thanks for coming on Software Engineering Daily. It's been really great talking to you.

[0:59:10.4] LK: Thanks so much for the time, Jeff. Thanks again for your interest in Streamlio and Pulsar.

[END OF INTERVIEW]

[0:59:18.7] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plugins. Use the value stream map to visualize your end-to-end workflow. If you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on the fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team, who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations.

You can check it out for yourself at gocd.org/sedaily. Thank you so much to ThoughtWorks for being a long-time sponsor of Software Engineering Daily. We're proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]