EPISODE 521

[INTRODUCTION]

[0:00:00.6] JM: When you go to a website where a video is playing in your video lags, how does the website know that you are having a bad experience? Problems with video are often not complete failures. May be part of the video loads and plays just fine and then the rest of the video is buffering. You've probably experienced sitting in front of a video waiting for it to load as the loading wheel mysteriously spins. Since problems with video are often not complete failures, troubleshooting a problem with a user's video playback is not as straightforward as just logging whenever a crash occurs. You need to continuously monitor the video playback on every client device and aggregate it in a centralized system for analysis.

The centralized logging system will allow you to separate problems with a specific user from problems with the video service itself. A single user could have bad Wi-Fi or have 50 tabs open with different videos, and that would be their fault. To identify problems that are caused by the video player or the company that is serving the video rather than a user, you need to capture the playback from every video and every user.

Scott Kidder works at Mux, where he builds a streaming analytics system for video monitoring. In this episode, Scott explains how events make it from a video player on to the backend analytics system running on Kinesis and Apache Flink. Events from the browser are constantly added to Kinesis, which is much like Apache Kafka. Apache Flink reads those events off of Kinesis and MapReduces them to discover anomalies.

For example, if 100 users watch a 20 minute cat video and the video stops playing at minute 12 for all 100 users, there's probably some data corruption in that video and you would only be able to discover that by assessing all users, because if you looked at only one user and you noticed a problem at minute 12 for that user, well, it's equally likely that the user could have bad Wi-Fi or have some problem. That's why you need analytics on everybody.

Scott and I discussed the streaming infrastructure that he works on at Mux as well as other streaming systems like Spark, Apache Beam and Kafka. This episode is one of a short series

we're doing about streaming data infrastructure. I wanted to do some shows in preparation for the Strata Data Conference in March in San Jose, and I'll be attending there. Thanks to a complementary ticket from O'Reilly. O'Reilly has been kind enough to give me free tickets to these conferences since Software Engineering Daily, and I did not have the money to attend any conferences. So I am always thankful to O'Reilly in their support of Software Engineering Daily.

If you want to attend Strata, you can use the promo code PCSED to get 20% off. Also, if you want to find all of her old episodes about data streaming, big data, Hadoop, you can check out the Software Engineering Daily app on iOS or android, which has all 650 of our episodes and it's in a searchable format. We've got comments, related links. It's all open-source, which you can find by going to github.com/softwareengineeringdaily. I hope you check it out and download the app or join our open- source community. We try to be very welcoming to anybody that wants to come in and contribute code.

With that, let's get to this episode of Software Engineering Daily.

[SPONSOR MESSAGE]

[0:03:44.4] JM: Apps today are built on a wide range of backends, from traditional databases like PostgreS, to MongoDB and Elasticsearch, to file systems, like S3. When it comes to analytics, the diversity and scale of these formats makes delivering data science and BI workloads very challenging. Building data pipelines seems like a never ending job as each new analytical tool requires designing from scratch.

There's a new open-source project called Dremio that is designed to simplify analytics on all these sources. It's also designed to handle some of the hard work, like scaling performance of analytical jobs. Dremio is the team behind Apache Arrow, a new standard for in-memory columnar data analytics. Arrow has been adopted across dozens of projects, like Pandas, to improve the performance of analytical workloads on CPUs and GPUs. It's free and open-source. It's designed for everyone, from your laptop, to clusters of over 1,000 nodes.

Check out Dremio today at dremio.com/sedaily. Dremio solved hard engineering problems to build their platform, and you can hear about how it works under the hood by checking out our interviews with Dremio CTO, Jacques Nadeau as well, as the CEO, Tomer Shiran. And at dremio.com/sedaily, you can find all the necessary resources to get started with Dremio for free.

I'm really excited about Dremio. The shows we did about it were really technical and really interesting. If you like those episodes or you like Dremio itself, be sure to tweet @dremiohq and let them know you heard about it from Software Engineering Daily.
Thanks again to Dremio, and check it out at dremio.com/sedaily to learn more.

[INTERVIEW]

[0:05:45.3] JM: Scott Kidder is an engineer at Mux. Scott, welcome to Software Engineering Daily.

[0:05:49.7] SK: Awesome. Thank you, Jeff. Glad to be here.

[0:05:51.8] JM: Yeah, we've done a few shows about topics related to what we're going to discuss today. You are an engineer at Mux, and I've talked to the founders of Mux about what they're building. It's basically technologies around delivering online video or monitoring online video, and I've also done a show about Flink, and today we're going to talk about the intersection of the two, but let's start with the idea of delivering online video. What is hard about that? Why is it hard for YouTube or Funny or Die, or nbc.com, all these companies that they broker in video? What's hard about what they do?

[0:06:33.9] SK: Great. Yeah, so what makes delivering video difficult is there are so many different client-side technologies and network technologies and server-side technologies that are involved in the encoding, management and delivery, and ultimately the playback of video. There are just a lot of points in the process where things can go wrong. It's not as simple as plugging in an HDMI cable into your television and having it playback seamlessly. It's is a complicated process.

So what we're doing at Mux is providing tools to give video publishers insight into that process and ultimately provide a better experience for their customers.

[0:07:16.4] JM: As somebody's delivering video, like if I'm cbs.com and I'm delivering a video to a user, what analytics can be generated in that process of the streaming that are going to help resolve potential issues? What is the monitor touring output stream look like and where is it being generated? Is it on the client side? Is it on the server side? What are the analytics that we're concerned with here?

[0:07:42.6] SK: Great question. So we've got analytics that we get from — In the case of some CDNs, we actually get information about time to first byte. So that's the amount of time that it takes for the actual video content to start streaming to the end-user device. We also have information about the round-trip time, which is the total amount of time that it takes for a video segment or manifest or some other video asset to ultimately be delivered to the client and. Then we have client-side metrics that include all kinds of different metrics. It kind of represent the quality of experience for the end-user, things like buffering, quality shifts up and down as it's adapting to changes in the bit rate that's the sustainable for that client connection.

So we actually get those metrics transmitted from the client devices themselves to the Mux service, and over the life of the video playback session, will accumulate all of those metrics, and once the video view is finished, we'll kind of flatten them out and provide a complete picture about what that particular user experienced while they were watching a video.

[0:08:49.5] JM: What are the requirements for building an end-to-end video analytics monitoring system?

[0:08:57.6] SK: Right. At the client-side, the first thing that you need is a client SDK, a library that allows you to integrate with the video player. So we support HTML 5 and Video.js web players. So we can actually tap into events that the player is seeing as it's downloading video assets and playing them. So that's the client SDK side. We also have native SDKs that allow us to capture video events for playback in native apps.

So the SDKs periodically send beacons that indicate information about what is happening during that video playback session, and so those beacons are just sent over a plain HTTP to collectors that we have, which quickly throw the beacons on to a Kinesis stream. So, in that cases, it's like a distributed event log that is sharded. So all of the events associated with the particular playback session go to the exact same shard, and then we have processors that then read from a specific shard and they kind of aggregate all of the beacons associated with many different video sessions. So then once those video views finish or they encounter some sort of error, we then — Like I mentioned earlier, they kind of flattened it out and compute these things, like the overall buffer, the amount of time spent buffering during playback, identifying any errors that happened during playback, and we're then able to compute a picture that represents what that particular video view look like.

[0:10:29.2] JM: If I understand correctly, let's say you've got 10 users that are watching a video about a cat, and those 10 users all are watching that video in the browser, and the browser is the player — The browser player is logging events to your Mux endpoint, and the Mux endpoint is ingesting all of those events. It's sharding charting them by the individual user, but it's also got — What was the word you used? The tags or the markings — What was the word you used to associate the different videos — To make sure that you're sharding by the individual user, but you want to be able to correlate that they are all watching, all those different user are watching the video.

[0:11:18.4] SK: Yeah. So we've got a shard where it's sharded by the video view ID. So a single user could have and can and will probably have multiple views, but all their views go to the exact same shard. So it's all sharted deterministically. Then we've got processors that are pulling beacons off of a shard.

[0:11:37.5] JM: Beacon.

[0:11:38.3] SK: Yeah. So that the beacons represent just a snapshot in time indicating what's happening in the player at that given moment. So then we accumulate all the beacons over the life of that video view and then we collapse is all once the view is finished.

[0:11:55.9] JM: Right. So with this sharding schema, you basically have a way to differentiate if a user — If an error is happening within a video, then you can determine if it is correlated among a user, like within a user's identity, and if all the videos that that user watches has this particular problem, maybe it's a problem on the user side. But if you can figure out that 10 different users watching the same video have that problem, then it's probably a problem with the video.

[0:12:31.9] SK: Right. So when we started Mux, we were able to make those sort of conclusions using an off-line batch process that at hourly intervals would do these sorts of roll ups that are able to compute those types of observations, but we need the ability to provide error rate alerts that were more timely than just an hour. So that's kind of what led us into Apache Flink.

What we did with Apache Flink was as soon as a video view is over, is finished or there's been some sort of error during playback, our processors then send information about that video view to a second Kinesis stream that is read by an Apache Flink app. So Apache Flink was — We're looking at different stream processing systems, and Apache Flink just really fit the bill for what we need to do, which was have very low latency, also high-availability, the ability to read from not just Kinesis, but also Kafka, other stream sources, and it's worked really well for us in building this anomaly detection error rate alerting application.

[0:13:48.1] JM: Okay. I want to get to Flink eventually, but I want to start a little bit earlier in the pipeline, because this term streaming, when I first started doing some shows about streaming, I really had a whole lot of trouble understanding the difference between streaming and batch, and I think that the best way to actually disambiguate it is just to really dive into the pipeline and explain how it works.

So you've got these just events that are being thrown on to Kinesis, and Kinesis is a distributed, scalable, queuing service pub/sub service, much like Apache Kafka, but it's hosted by AWS, and I think for all intents and purposes we can think of it as very similar to Kafka. So that's familiar to people who have [inaudible 0:14:36.5] about Kafka. Would you say that's accurate?

[0:14:37.9] SK: Yeah, that's perfect. Yeah.

[0:14:40.3] JM: Yeah. So you get all these events and they're thrown on to the Kinesis queue. Do you call the Kinesis queue a stream as well or do you just call it a queue?

[0:14:50.6] SK: It's a stream.

[0:14:51.8] JM: Okay. Just a stream of events, basically.

[0:14:54.4] SK: Because in the case of queue, an item that has been added to the queue is ultimately removed from the queue and once it's removed from the queue, it is gone forever, most likely. But in the case of a stream, you can actually have multiple consumers on that stream that are at different locations in the stream. That's one of the most powerful aspects of Kinesis and Kafka is their ability to have multiple consumers that are reading at different points in the stream and are able to process a stream in different ways and not be — They're not subject to the same limitations that you have with a traditional queue.

[0:15:34.0] JM: Indeed. So we can think of the Kinesis stream as the stream, and then Flink is the reader or the processor of the events on those streams. Would you say that's accurate?

[0:15:47.1] SK: Exactly. Yeah.

[0:15:49.7] JM: This Flink streaming technology, this is one of a bunch of different stream processing/batch processing technologies that can be used basically for processing raw events for doing correlations among raw events, for doing aggregations among raw events. Basically, the earliest days of this kind of technology was to Hadoop, which did it in batch.

[0:16:16.8] SK: Right. Just a traditional ETL extract transform load process where you throw data on to a disk and then you process it in batch off-line with potentially a lot of latency, but you get deterministic results.

[0:16:31.8] JM: How would you characterize the movement from the earliest days of Hadoop, Hadoop MapReduce, to whatever we have now with this cornucopia of different streaming frameworks?

[0:16:44.6] SK: Right. So it all comes down to just needing data as quickly as possible, needing to interpret and draw conclusions from data as soon as it's generated. In the example that I mentioned earlier with the one hour delay and in processing video [inaudible 0:17:00.3] events, it was just too long, and I think of a lot of companies are facing similar requirements where data is being generated and it needs to be processed as quickly as possible, and streaming platforms like Apache Flink really enable that.

[0:17:16.2] JM: Why is it so much easier to do it in batches than in streams?

[0:17:21.4] SK: Well, that's a great question. I think that batch processing is a concept that has been around for a very long time and a lot of people are comfortable with it. Obviously, you get deterministic results in making those calculations as opposed to Lambda architectures where you end up having kind of like a speed layer. I've seen it called the speed layer, and then a bash layer, where the speed layer is meant to serve like a frontend application and provide probably like, often times, imprecise results. Then a batch layer, which then kind of comes around later and then cleans up the results. Lambda architectures is like that. Have been popular in the past built with Hadoop and Apache Spark, but I think that the idea of like building two applications essentially, building a speed layer and a batch layer is undesirable, and Apache technologies, like Apache Flink, make it possible to have the accuracy that you get from the batch layer while also being able to have the speed that you get from the speed layer in like a Lambda architecture and only develop a single application. It’s really the best of both worlds.

[SPONSOR MESSAGE]

[0:18:43.2] JM: A thank you to our sponsor, Datadog, a cloud monitoring platform bringing full visibility to dynamic infrastructure and applications. Create beautiful dashboards, set powerful machine learning based alerts and collaborate with your team to resolve performance issues. You can start a free trial today and get a free t-shirt from Datadog by going to softwareengineeringdaily.com/datadog.

Datadog integrates seamlessly with more than 200 technologies, including Google Cloud Platform, AWS, Docker, PagerDuty and slack. With fast installation and setup, plus APIs and

open-source libraries for custom instrumentation, Datadog makes it easy for teams to monitor every layer of their stack in one place, but don't take our word for it. You can start a free trial today and Datadog will send you a free t-shirt. Visit softwareengineeringdaily.com/data dog to get started.

Thank you to Datadog.

[INTERVIEW CONTINUED]

[0:19:51.1] JM: If you asked somebody from Apache Spark or Storm, the technologies that are not Flink, why their technologies have evolved in such a way or were created in such a way, where if you wanted to do some sort of streaming system, you would probably want to do a Lambda architecture where you have stream processing that may be imprecise and you have batch processing to reconcile that imprecise stream processing. What would they tell you about like why they made the trade-offs that forced them to sort of have this Lambda architecture? Like help me understand — Let's frame what the trade-offs are that those systems are making versus Flink.

[0:20:37.8] SK: Sure. Apache Spark is a very mature project and that was actually a consideration that we had to take into account when we were choosing whether to use Apache Flink or something else. It's a bit of a gamble taking a risk with a less mature project, like Apache Flink. There are a lot of engineers who are familiar with Apache Spark. It's very widely known. So I think that Apache Spark has that going for it.

Traditionally, was started out as purely a batch processing system, and they've added support for streaming, which for a while was referred to as micro-batching, where it essentially acts like a stream processing platform, but under the hood is really just processing data in micro-batches, which in a way can be — That can be said of just about any stream processing platform, but the difference with Apache Flink is that it's a stream processing platform first. So it’s kind of the inverse, right? It’s stream processing platform that can operate as a batch processing platform, whereas Spark is the other way around. It was batch first and then they implemented stream processing on top of that batch processing system.

[0:21:54.2] JM: When I first started doing shows around these topics, I think I made a mistake and that I was thinking that these were interchangeable frameworks. Sort of like you look at the different frontend frameworks, like React JS or ViewJS or AngularJS. You really only pick one of these. They're kind of the same siblings. They have very similar ways of looking at the world and you're probably not going to combine them. But I think it's a different story with the streaming framework. It's like my understanding of Spark is that basically what Spark did differently was instead of having this Hadoop mentality — Hadoop mentality is you run this big Hadoop job and after it's done with that job it has to right to disk.

The idea was Spark is like you load your data into a working set in-memory and you can do an operation on it, which is sort of a batch operation, but then it sits in-memory after that operation is done and then you might want to do another big operation on it and it it's going to continue to do these operations over a working set in-memory. So you could consider these big, bulky batch operations, but because it's all in-memory, it's faster. Would you say it's an accurate description of Spark?

[0:23:11.9] SK: Yeah. It's a traditional ETL type of job where it's just operating on the data that's in-memory, and that's true of Flink as well. Reads from a stream source, does a MapReduce style operation on it, and has the potential to keep those results in-memory or send them to an external system or have those feed into another job. I think that's true of Spark as well.

[0:23:39.4] JM: So Flink can do a big MapReduce over a bunch of events that are sitting in Kinesis, but it can also do something where it's got some working set in-memory and it just ingests one-off events to continue building some sort of in-memory record. Is that right?

[0:24:00.3] SK: That's right. Yeah, and I'll give you an example from the error rate alerting app that we built at Mux. So we send information about video views that had errors or didn't have errors to this Apache Flink app, which then does a map operation on a couple of different breakdowns. So it'll do a map operation on video title. So My Cat Video might be like a particular video title, and we want to see what the error rate is for that particular cat video.

So we'll accumulate a fixed number of video views, like a hundred video views for that title and then we'll calculate the error rate that we observed for those 100 views for the known error

types for that particular customer property. So suppose we find out that from those 100 views, we had a 5% error rate. Well, what makes us a little challenging is different customers have different normal error rates. So a 5% error rate might be really good for one customer, it might be really terrible for another. So we've been able to avoid defining static error rate thresholds by accumulating at a second step, accumulating observed error rates for each customer property. Then we can do a calculation to find out what their statistically normal error rate is and then compare this most recently observed error rate against that kind of definition of what normal was. Then use that determination to decide whether to open an error rate alert or close an error rate alert.

So what's really cool is it we're able to bring on new customers, observe what their normal error rates are, which can and do change over time without any manual configuration required on our part or on the part of our customer. That's all possible with Apache Flink, because it accumulates all this data that it keeps in-memory as part of the job state.

[0:26:04.8] JM: By the way, that's pretty awesome moat you're building there, because if you're doing all these analytics on all these different videos, you probably have an idea of how video performance works. Like a pretty differentiated dataset of how video performance works agnostic of a specific customer.

[0:26:21.7] SK: Yeah. We're applying that kind of approach to our next product, which is a video delivery product.

[0:26:29.5] JM: Which I've seen, by the way, which is amazing, and the potential of that is just huge.

[0:26:35.3] SK: We're not just looking at error rates for a particular customer an finding out what normal is for them. We have plans to look at error rates for different types — Or not just error rates, but also quality of experience metrics for types of videos, different geographies, different CDNs and really optimize the encoding and delivery of video for all those different factors, and Flink plays a very important role in that.

[0:27:03.7] JM: Yeah. Okay. So I'm enamored with the business, but I guess — So back to the anomaly detection. So let's say company — Let's say CBS uploads this new breaking news cat video and a bunch of people start watching it instantly, and I guess they start watching it and then as soon as they start watching it, you're starting to get a shard of a new type of video and you're instantly starting to aggregate those views into a new Flink system, a new Flink memory partition or something. Maybe you could just tell me what happens when CBS uploads a new video and you're starting to monitor that video because you want to be able to alert them if something is wrong with it.

[0:27:51.5] SK: Right. So with our existing data product, what happens is they've got to a web player or a native player that is instrumented to send us information about video views that happen on their site or in their app. Playback starts, we start getting beacons with a video ID or a view ID rather that is globally unique. We'll start accumulating all those video view beacons, an error occurs or the video view finishes. We'd send all the details to a Kinesis stream that is monitored by Apache Flink, and Apache Flink then does a map — It performs a map function on that video view to map it to a particular video title and also to a customer property. So we can actually detect kind of site-wide issues for a customer like across all of CBS not just for that particular cat video title, but also across an entire customer site.

So we'll add that video view to a map operation, which accumulates a fixed number of video views. So we use counting windows. So accumulate video views until we reach a certain target number, like a thousand video views or a hundreds video views, at which point we'll then do a reduce operation, which then performs all — It calculates all the error rates for the different error types that we know to exist for that customer. Then we compare those error rates against what is statistically normal for that customer and then we decide whether to alert.

So all of these is just happening continuously, just that map operation of accumulating all these video views, and then collapsing it with the reduce, and then comparing it against normal. That's just happening all the time, and Flink just does a phenomenal job of making that really simple. They've got beautiful APIs that made — Developing the system really, really intuitive and easy.

[0:29:46.5] JM: Although your example would bring up something — I did a couple of interviews with people from the Google Dataflow Project, and one of the things that they say is that this

whole idea of batch versus streaming is kind of a red herring or a misnomer or whatever, and when I think about your example, you've got all these views that are getting mapped and then you have this periodic reduce function. It seems like the reduce is kind of a batch, that's like a little batch job.

[0:30:14.9] SK: It is, yeah. So there is some latency that's introduced there, just happened to accumulate video views and then do the reduce operation. So you do have to strike a balance and picking, in our case, the size of the window, the number of video views to accumulate before you do the reduce operation, because that does affect the — Like in our case, that affects the speed of reporting. Too large of a window and it could take a long time to actually fill.

Yeah, dataflow team is doing a lot of interesting work, developing APIs that are in compatible with the Beam spec, which Flink aspires to, and to a large degree is compatible with. So that was also a big selling point on Flink was its compatibility with the Beam API and it being also inspired by the [inaudible 0:31:09.3] design paper they came out of Google and influenced Google Dataflow.

[0:31:14.4] JM: Okay. Beam is one of the most confusing things I have ever tried to report on. They were explaining it to me and I just did not understand it. It's like something like — So it's like an API they introduced where you can make your — Oh my god! Can you just explain Apache Beam? Just give an explanation. See if we can —

[0:31:38.9] SK: Just a quick disclaimer. I've never actually used Apache Flink via the other Beam API, but its goal is to have a set of APIs that are portable across stream processing platforms, like Apache Flink, Google Dataflow. So the idea is, in theory, you should be able to write an application against that uses the Beam APIs and then actually have it run on Apache Flink or Dataflow with little to no modification. So it's very powerful in that way.

[0:32:13.9] JM: This is, I guess, useful because we're talking — I mean, we just talked about how you can basically describe these same operations to Spark, or to Flink, or to Hadoop, or to Storm, or to Dataflow. You could describe these operations to each of these different systems. You're probably going to get different latencies. You're probably going to get different — Like how much it costs to run. So there's probably a high degree of variability among the different

systems that you could run them on, but they should be able to run them all. These are all basically [inaudible 0:32:52.5] complete distributed processing systems.

[0:32:54.0] SK: Yeah, exactly, and that's really — One of the really cool potential things, I'd love to see somebody provide the ability to submit a Beam application to — Or an application that uses the Beam APIs to a runner, a set of runners, maybe one that's using Spark, one that's using Flink, one is Dataflow, and provide information about the cost, the operational cost, the performance, portability across cloud providers and really just be able to make a decision that way. Instead of it being a Flink versus Spark versus Dataflow type battle. Just be very pragmatic about it and simply choose the tool that's best for that job.

[SPONSOR MESSAGE]

[0:33:46.4] JM: Amazon Redshift powers the analytics of your business, and intermix.io powers the analytics of your Redshift. Your dashboards are loading slowly, your queries are getting stuck, your business intelligence tools are choking on data. The problem could be with how you are managing your Redshift cluster. Intermix.io gives you the tools that you need to analyze your Amazon Redshift performance and improve the toolchain of everyone downstream from your data warehouse.

The team at Intermix has seen so many red shift clusters that they are confident that they can solve whatever performance issues you are having. Go to intermix.io/sedaily to get a-30 day free trial of Intermix. Intermix.io gives you performance analytics for Amazon Redshift. Intermix collects all your Redshift logs and makes it easy to figure out what's wrong so that you can take action all in a nice intuitive dashboard.

The alternative is doing that yourself, running a bunch of scripts to get your diagnostic data and then figuring out how to visualize and manage it. What a nightmare and a waste of time. Intermix is used by Postmates, Typeform, Udemy and other data teams who need insights into their Redshift cluster.

Go to intermix.io/sedaily to try out your free 30-day trial of Intermix and get your Redshift cluster under better analytics. Thanks to Intermix for being a new sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:35:32.2] JM: Okay. Now I get it. This feels gratifying, because now I get it, and it seems like nobody is saying who won the database war. Was it PostgreS? Was it Cassandra? Was it HDFS? Was it CockroachDB? Was it InfluxDB? That would be of radicular argument.

[0:35:50.2] SK: Right, exactly. Yeah. They've all got their own strengths and weaknesses, and it's really about finding the best tool for the job.

[0:35:57.9] JM: Right. So Flink's differentiation, or at least one of its differentiations — I think I saw you say this or it was written in a documentation somewhere or something. Flink has the ability to handle unbounded event streams with exactly once event processing. Am I remembering correctly?

[0:36:17.9] SK: Yeah, that's. Yeah.

[0:36:19.6] JM: Okay. Why is that hard to implement?

[0:36:23.1] SK: It's hard to implement because not all systems have the check pointing capabilities that Flink has. Actually they say exactly once processing, but what they mean is at least exactly once or something to the effect, because what happens is you end up having Flink applications that have the callouts to other systems, and maybe they write a record to a database, so they put a message on another Kinesis stream. Those side effects can't really be undone. So in that way it's like you could end up having a Flink application that has a side effect that is repeated multiple times. But when you've got a Flink application that's running, it actually — And it reads a message from a stream. It keeps track of its position in that stream and it keeps track of all the operators that are part of the application, the Flink application graph, what their particular state is. So then when the Flink application is checkpointed or save pointed to durable storage like — So Flink applications typically use HDFS to persist their internal state.

You might configure Flink application to checkpoint it every five minutes or 10 minutes or something like that. And so what it does is it sends information about it. It writes information about it's state to durable storage, and it also updates information in Zookeeper about where it is in the stream and the location of the most recent checkpoint data the written on HDFS.

So then if the Flink application stops, or it needs to be restarted, or if you have to upgrade, etc. What happens is the Flink job manager and test manager is they actually load their state from that previous save point or checkpoint that's on HDFS, and they'll resume. There might be some period of time from when the last save point was taken, which could lead to reprocessing of messages and side effects, like I mentioned earlier, writing a record to a database again or putting an additional message on a queue, but that application continues to process without interruption.

[0:38:41.3] JM: Tell me what I misunderstand about the system. No, no — That was actually really good. A Flink processing pipeline is this basically a directed acyclic graph of different computational steps. You've going to have these different computational steps in your data pipeline and at each computational step you do some processing and the data is then ready for the next step, and at different intervals, you're going to want to checkpoint the data that has been processed in a given step two disk in case something crashes and you're halfway through a full pipeline being finished. Do I have it right so far?

[0:39:26.2] SK: Yeah. That's right.

[0:39:27.0] JM: Okay. so when this check pointing occurs in one of the nodes in the dag, it's going to be checkpointed to HDFS. Does that block things? Like if you have to checkpoint at one of these nodes in the gag, does that block processing or is it some kind asynchronous thing?

[0:39:45.6] SK: It's asynchronous. It's really quite beautifuf the way that Flink has implemented it. You can think of that as standing — Imagine yourself standing at a stream and drop a marker into the stream. You drop something that floats and it flows down the stream and different points along the way, different operators that are part of this graph, they see the marker and then they checkpoint their state.

So meanwhile, upstream, from where that marker was placed, it can actually continue processing. They can still continue reading, but meanwhile the marker flows down the stream of the application and then once it reaches the end nodes, all the syncs, t hen that checkpoint or save point operation is considered complete. But it doesn't block operators that are upstream of where the marker currently is from continuing to operate.

[0:40:42.5] JM: That is beautiful. That's pretty cool. So is it problematic to do all that checkpointing or does it get garbage collected or something?

[0:40:50.9] SK: It's not really problematic. It introduces a small —

[0:40:54.3] JM: It's just disk, I guess. Disk is cheap.

[0:40:56.3] SK: Well, it's disc, but it's network I/O. So depending on the size of the data that you're keeping in-memory, the checkpoint operations can actually start to impact the performance of your app.

So, recently, Flink added support for partial checkpoints. So just doing the deltas from the last time it did a checkpoint, and that's lead to a huge improvement and performance, because if you're dealing with an app that — A Flink that has on the order of gigabytes or terabytes worth of data that's stored in- memory, check pointing that or save pointing that every five minutes means potentially transferring all of that data to HDFS, and that's very expensive both in terms of storage and network I/O, etc. Just being able to checkpoint the deltas is very valuable.

[0:41:53.4] JM: Really good descriptions there. Part of the streaming architectures that tends to happen is you often have a processing system that pulls data off of the Kinesis or Kafka, like mainstream system, do some processing and then write back to the Kinesis on a different data channel or a different shard or partition or — I don't remember what the terminology is, but you write it back to the Kinesis queue or the Kafka queue and then it makes it available to other systems to either pull off for more processing or whatever.

Can you help me understand like when do people want to do all of their processing in a single Flink job versus writing it back to Kinesis and making it available to other people or writing it back to Kafka and making it available for other people? This seems like a burgeoning area that's like pretty important? How do you manage the different channels in your data stream? Does that question make sense?

[0:42:58.0] SK: Yeah, it does. I'm actually going to just relate it to a Flink app that we wrote for a video product that ties in really well with problem that you just described. So we get access logs for our video product from our CDN providers. So every time a video chunk is requested or a video manifest s requested, our CDN provider provides us with a log record that indicates you the URL of that particular piece of media that was requested, performance metrics about it, etc.

What we're doing is we're taking that log record and putting it on the Kafka topic and then we've got a Flink application that reads from that Kafka topic and then it queries a database to kind of enhance that log record, find out information about the video resolution, it's audio and video bit rates, customer information, etc. and it kind of like decorates it, enhances it, enhances that log record and then puts it on a second Kafka topic that will eventually use for billing, because we have like kind of a metered like usage-based billing system for video. So we'll find out how many — The duration of video that was requested or the duration of video that we served for a particular customer over the last month, etc.

But we also use those same log records to feed it into our monitoring system. Actually, after we decorated that log record, we then do a MapReduce operation on it to calculate round-trip time and all these other performance metrics that we're interested in, freshly monitoring the real-time quality of experience for users of our video service. So though those metrics are written into an InfluxDB database that we've got Grafana Dashboards for. Then we've got Grafana dashboards that we show in our engineering area that show the current health of the system.

So this is an example of a Flink application that is serving multiple purposes right. So it's enhancing log records and feeding them into a second Kafka topic to be used for billing, but then it's also helping us with monitoring the health of the service.

[0:45:11.8] JM: Wow. Well, Kinesis is really your master database.

[0:45:15.5] SK: Yeah. We use Kinesis for stream processing in AWS, which is where our data product lives, but our video products, we're trying to have that run in multiple cloud providers. So Google Cloud, AWS. So Kafka really fit the bill there, because it's easy to deploy in any cloud provider, whereas Kinesis, it's an AWS only product. So we're using Kafka and Kinesis with Flink and have had really no problems with either. The Flink team provides for Kinesis and for Kafka has just been phenomenal.

[0:45:54.8] JM: Support, is it like data artisan support, or is it just like through message boards and stuff?

[0:46:00.1] SK: Through message boards. The Flink user's email list is great. They're very responsive on the Jira bug tracker as well. At Mux, we've come committed several features and bug fixes back to the Flink product and they've been very receptive and just really great. Great partners in that way.

[0:46:20.9] JM: Are you deploying Kafka on Kubernetes?

[0:46:24.6] SK: Yeah. So we've got Kafka, HDFS, Flink all running on Kubernetes.

[0:46:30.9] JM: I don't know if you've set up Kafka before, but like was it easier because of Kubernetes or have you talked to people about like — Does Kafka become significantly easier to deploy and manage with Kubernetes?

[0:46:42.1] SK: I think deploying on Kubernetes was our goal from the outset. The vast majority of our services, code that we write as well as just third-party code is deployed using containers that run on Kubernetes. So kind of having like a homogenous production environment is really desirable, and we've not had any performance issues or deployment issues with Kafka yet. Knock on wood.

[0:47:11.0] JM: Isn't that remarkable? Isn't it remarkable how much infrastructure you can stand up these days and how complicated it can be and it actually like works?

[0:47:20.2] SK: Absolutely. Yeah. Don't look too hard. It's all working. All these things are able to talk to each other and it's not an issue. Yeah, it is pretty amazing.

[0:47:35.1] JM: So it seems like people have a pretty good handle over how to work with different microservices. An increasing use case — Well, it seems like these data streams, like being able to keep track of your data streams. Like you just discussed these data streams that you're reusing in different places. Do you have some centralized schema or list of the different data streams that are available maybe with some documentation or descriptions around the data streams? Because these things — You could almost look at these like micro-services where, "Yeah, you've got —" Like if you want to pull this particular type of data stream, you have to know where it is. You have to know like how to access it, right?

[0:48:19.8] SK: Exactly. Yeah. We do keep — We use Kubernetes manifest to configure Kafka with all of the different topics that are available on a particular set of Kafka brokers. So that's documented and those manifests are checked in to source control, and so it's pretty easy to see and know which broker or which topics you'd expect to find on a set of Kafka brokers. But then there's also the problem of knowing how to send and consume data from those topics.

So for that, we use proto-buffing coding for all of the messages that we send and consume from Kafka and Kinesis. Protobuf is very efficient and easy to work with. Yes, so that's — We've got the Protobuf specs that are checked in to source control as well. So it's all pretty clear as far as which topics or streams you can you can publish and consume from and then also how you — What the format of the messages are.

[0:49:20.0] JM: I've done some shows with people who will talk about the systems like Looker, for example, and they'll say that one of the problems that Looker was solving — There's a whole class of these kinds of tools, like Tableau, for example, where it's trying to solve this problem where you've got analysts in accompany, data analysts, and it's really hard to get the data analysts access to the data if they are not super tech savvy, if they don't know how to do a MapReduce job or like a hive job or whatever. I think Looker type tools did a pretty good job of solving these in a batch fashion. I'm wondering if this is entering like a whole new tier of complexity for the analyst. Like how does the analysts — It's very easy-to-understand like, "Okay. If an analyst gets a daily updated CSV or an updated MySQL database or whatever with

a data that they can analyze, that seems very easy to reason about for the analyst, but maybe less so if they've got the streams that are constantly updating that you need to subscribe to them." Is there a usable infrastructure for the less technical user to access that kind of data?

[0:50:29.2] SK: That's a great question. There is a lot of interest from business units like as far as being able to access this data directly. I know that Netflix is using Flink very heavily to basically provide like a self-service type interface to a lot of usage data from the Netflix players so that engineers aren't necessarily kind of serving requests from business groups trying to run one-off queries, trying to make it more of a self-service type system. But that does require a lot of documentation about how to interact with the streams and, yeah, that is absolutely a tough problem.

[0:51:10.7] JM: Okay. Well, I know we're up against time. We did go into your event ingestion and processing architecture in some detail. Just to summarize it for people, you've got these user events that get loaded into Kinesis, they get processed by, for example — Well, they processed by Flink, and then after they get processed, maybe they continue through a long Flink job. Maybe they get written back to kinesis. Eventually they get put in the PostgreS or InfluxDB or they get put back into Kinesis maybe for further processing or just for access for the billing team or whatever. T there's a whole lot more about the whole event ingestion architecture, the MapReducing and so on in these talks that you've given, and I'll put those in the show notes.

But since we're up against time, I just want to ask like a little bit of a high-level question. So you've been working in video infrastructure for a long time. You were with Matt and John at their previous company, which was basically another video infrastructure company. Do you have any big takeaways from what the canonical engineering problems in video are?

[0:52:22.3] SK: Yeah, John was one of the founders of Zencoder, which was one of the first cloud-based video encoding services. Matt has a lot of video experience well. Has worked on Video.js and has a deep understanding of what it takes to — What some of the problems are with actually playing video and why it remains a difficult problem.

One of the biggest problems is just the diverse landscape for the types of devices that we watch video on, the networks that are involved in delivering video. Some are fast, some are slow. It's a complex landscape out there and it's really tough for producers of video to understand all of these problems. They know that they have a high quality input video and they wanted to look as good as possible on end-user devices. Picking bit rates and picking resolutions is not for the faint of heart, and traditional online video platforms, it's a decision that producers would have to make once and then in many cases they're stuck with those decisions for years to come unless they want to re-encode their video.

What we're trying to solve at Mux is really feature-proofing your video. So you'd give us your highest quality video and we'll deliver it optimally now and well into the future. And so that's really what sets us apart.

[0:53:40.9] JM: Well, I can't wait until YouTube is not the only game in town for sharing video, because they think that's like the big elevator pitch of how Mux could be really big, is like making video really easy to share without having to use the kludgy YouTube playe.

[0:53:59.8] SK: Yeah, and no disrespect to YouTube.

[0:54:02.4] JM: Nothing against YouTube.

[0:54:02.9] SK: They do a phenomenal job, but there are a lot of the video publishers and just media publishers that would like to be responsible for serving their own content in a way that's not necessarily tied to YouTube or branded with YouTube, but they should necessarily have to sacrifice quality in terms of video or the experience for their users.

[0:54:26.6] JM: Wistia was a pretty early to this game, but it seems like — I think Wistia may be is, I guess, less modular than the Mux, the mux approach, what you guys are going for.

[0:54:38.1] SK: Yeah. Wistia is their traditional online video platform. They've actually been one of our Mux data customers. They've had great success with integrating with Mux data as well.

[0:54:50.4] JM: All right. Well, Scott, really great talking to you. Very technical conversation. I think we did a good job of getting from the high-level to the fairly low level. It was great talking to you.

[0:55:01.3] SK: Thanks. Likewise.

[END OF INTERVIEW]

[0:55:05.6] JM: If you are building a product for software engineers or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an email, jeff@softwareengineeringdaily.com if you're interested.

With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers. I know that the listeners of Software Engineering Daily are great engineers because I talked to them all the time. I hear from CTOs, CEOs, directors of engineering who listen to the show regularly. I also hear about many newer hungry software engineers who are looking to level up quickly and prove themselves, and to find out more about sponsoring the show, you can send me an email or tell your marketing director to send me an email, jeff@softwareengineering.com.

If you're listening to the show, thank you so much for supporting it through your audienceship. That is quite enough, but if you're interested in taking your support of the show to the next level, then look at sponsoring the show through your company. So send me an email at jeff@softwarengineeringdaily.com. Thank you.

[END]