EPISODE 670

[INTRODUCTION]

[0:00:00.3] JM: Netflix users stream terabytes of data from the cloud to their devices every day. During a high-bandwidth, long-lived connection, a lot can go wrong. Networks can draw packets, machines can run out of memory and the Netflix app on a user's device can have a bug. All of these events can result in a bad user experience. There are other errors that can occur that do not disrupt the user experience.

Netflix runs thousands of machine learning jobs, logging servers and other pieces of internal infrastructure, customer service dashboards, CICD pipelines and AB testing frameworks are all software that are built internally at Netflix. When an error occurs in one of these places, engineers need to be able to diagnose and debug that error.

Observability is the practice of using logs, monitoring, metrics and distributed tracing to understand how a system is working. Kevin Liu is a Senior Software Engineer at Netflix with the Edge Insights team. He joins the show to talk about adding observability across the microservices deployed at Netflix. We also talk about how to manage high volumes of logging data effectively using stream processing. Kevin brings a ton of historical context to the table. He's worked at Netflix for several years and I really enjoy talking him about this applied perspective on the topic of observability.

[SPONSOR MESSAGE]

[0:01:38.0] JM: DigitalOcean is a reliable, easy-to-use cloud provider. I've used DigitalOcean for years, whenever I want to get an application off the ground quickly. I've always loved the focus on user experience, the great documentation and the simple user interface. More and more, people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU-optimized droplets perfect for highly active frontend servers, or CICD workloads.

Running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance. The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily. As a bonus to our listeners, you will get a $100 in credit to use over 60 days. That's a lot of money to experiment with.

You can make a $100 go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure and that includes load balancers, object storage, DigitalOcean spaces is a great new product that provides object storage, and of course computation. Get your free $100 credit at do.co/sedaily.

Thanks to DigitalOcean for being a sponsor. The co-founder of DigitalOcean Moisey Uretsky was one of the first people I interviewed and his interview was really inspirational for me, so I've always thought of DigitalOcean as a pretty inspirational company. Thank you, DigitalOcean.

[INTERVIEW]

[0:03:44.8] JM: Kevin Liu, you are a senior software engineer on the Edge Insights team at Netflix. Welcome to Software Engineering Daily.

[0:03:51.4] KL: Thanks. Thanks for inviting me, Jeff.

[0:03:53.5] JM: You wrote part of an article about observability at Netflix, and observability is a newer term, though it encapsulates some behaviors and some practices and technologies that we've been doing for a long time. My sense is that observability represents a shift in behavior, as well as some newer technologies. What kinds of shifts in technologies and behavior does observability that newer term, what does that represent?

[0:04:24.5] KL: I think it represents just the increasing complexity of the systems that we build today, especially the introduction of microservice architecture. Instead of having one big app and just watching what that app is doing, now we need to look at what a bunch of services are doing. I've heard for example at Uber, they've got over 2,000 microservices. I mean, how do you know what's going on in a big system like that? We also have quite a number of systems in Netflix too.

[0:04:50.6] JM: Whereas before, it might have been do you have logging and metrics, etc. instrumented? Now it's more you have these tools, but you have so much sprawl to your organization. Are you actually observing what's going on? Can you actually grapple with that scope?

[0:05:11.4] KL: Right. Yeah.

[0:05:12.1] JM: How has the tool chain for dealing with that massive scope of services and their respective monitoring and logging and metrics and so on, how is that different than what Netflix used maybe two or three years ago prior to this explosion in services, or in scope?

[0:05:32.5] KL: Before we started with a time-based database – time series database that kept a bunch of metrics and we could look at graphs, we could also plot these metrics using multiple dimensions. We build a lot of dashboards, but the dashboard started becoming really big. I mean, we had to watch a lot of systems, a lot of metrics. How do we quickly more determine what the root cause of an issue is? That's where more tooling came in to look at different signals, not only just those metrics, but looking at logs, looking at request traces and then deducing what exactly is going on.

[0:06:07.2] JM: Who uses observability tools at Netflix? Is it every engineer? Is it just specific operations people?

[0:06:13.1] KL: Operations and a lot of engineering teams. Also customer service uses them too. They're on the phone with customers trying to troubleshoot streaming issues and they use a bunch of our tools.

[0:06:24.1] JM: You have many different customers for these sets of tools. How do you evaluate whether you're going to build a new tool and how do you evaluate build versus buy? Because I know that Netflix takes a lot of stuff off the shelf, but they also are completely willing to build something if necessary.

[0:06:41.1] KL: It depends on what the use case is and who the customers are and what the requirements are. We could talk a little bit about mantis. I think that's one of the internal tools that we decided to implement ourselves, versus taking something off the shelf like Apache storm or spark.

[0:06:55.5] JM: Sure. Dive into that.

[0:06:57.8] KL: Because of Netflix's growth in traffic and complexity, we were thinking about how do we get smarter about quickly determining the root cause of streaming issues. We could take a batch processing approach, but that might take too long. We thought about outages. Outages five years ago would affect only a certain number of members, but now that same type of outage could affect a lot, many, many more millions of users. We need to make sure that an outage lasts as short as possible.

Having that control was important to us. Also cost was important and speed. Basically, cost and speed we focused on for mantis. We wanted something that was cost-effective, meaning that we don't want to run something in batch and it's running thousands of instances all the time. We wanted to scale it appropriately based on what somebody was looking at. We wanted to look at stuff as quickly as possible. Instead of having a batch process running looking at only one thing, when somebody needs it, when an outage happens, they could easily formulate a query, mantis could within seconds startup analysis job to figure out exactly what the root cause is and look at different logs, that kind of thing.

[0:08:13.0] JM: To set some more context for talking about mantis and your observability stack, Netflix has been growing for a long time and there are certain challenges that you get to with observability when the data gets really large, especially when we're talking about data that has questionable value, like logs. It's an open question, how much data from your logs you actually want to save and where you want to save it, how fast is the access time need to be? You need to be able to query this data in some time horizon, but you don't know what that time horizon is going to be, or it might vary from application to application. Talk about the challenges of having observable and queryable data as that data gets a really big.

[0:09:02.9] KL: Right. Yeah, as the data gets really big, cost becomes an issue, just storing terabytes, or even petabytes of data can be very expensive. Then we'll just store what you need based when you need it becomes important. Can't store everything. We started storing about two weeks' worth of data for some logs and that became too expensive at some point, so we cut it down to a week. Some other logs we even cut down to only four days.

With mantis, everything is kept in memory. You only need it when you query for it. If you want to save it off for later, mantis has that ability too. You can just save off whatever logs that you found into persistent storage.

[0:09:43.4] JM: Let's say I have a service, maybe it's the codec service. It's creating a bunch of different versions of a movie. Whenever the codec service gets called and it's probably spitting out tons of logs and mantis is consuming those logs, mantis is doing something with those logs. What is mantis doing at a high-level with these raw pieces of log data?

[0:10:10.9] KL: Right. When you start a mantis query, you tell it exactly which log you want to look at, which field you want to look at and want conditions that you want to match on. That query actually gets uploaded to in this case, the codec itself. As a mantis agent that has a list of queries and that will look at the stream of logs coming off that machine, or off the bunch of servers in this case, and only send those logs that match your query to mantis, which then does processing and you can look at the logs there.

[0:10:43.1] JM: Sorry. Maybe we could get a better picture for how this data is being generated, where it's being generated. I imagine it originates on the actual node, the microservice node. Then where does the logging data go to from that node and how does it eventually enter this mantis zone?

[0:11:00.6] KL: Once a log matches your query on the node itself, the node will send it to a mantis agent, which is a separate cluster entirely that collects the logs and then you can query against whatever that mantis agent is receiving from the node.

[0:11:17.3] JM: Okay. Does every microservice, or the vast majority of microservice nodes run this query filter component, or agent on your service node and then that's sending it to the mantis node?

[0:11:30.1] KL: That's right. Yeah. That agent is part of every hap in Netflix.

[0:11:34.3] JM: Okay, cool. Then when it gets sent to the mantis node, the mantis node is doing further – sorry, do you say it's doing further processing, or the mantis node is going ahead and storing it in-memory and the mantis system is more of the in-memory storage system?

[0:11:51.6] KL: Right. It just stores the logs in-memory and then you can additionally run another query on that stream if you want to filter it down even further. You can chain your search into multiple queries.

[0:12:03.6] JM: Is the in-memory storage system built off of something, like Redis or Memcached, or did you get, like roll your own query and memory storage system?

[0:12:15.7] KL: You could roll your own. You could do just whatever is in-memory, or yeah, if you've wanted to use Memcache, or Redis, or whatever you could if you wanted to. In fact, I can think of some internal jobs that do do that and others that just memory is good enough, like on node memory is good enough.

[0:12:30.8] JM: Is the difference there the durability, or the failover cases?

[0:12:35.4] KL: Right. Yeah. How long do you want to keep that data around? Is a single node, its memory capacity enough to keep what you need, or do you need to save it off and keep it for a day or two, for example?

[0:12:45.5] JM: Something I had a lot of trouble with understanding and I think some of the listeners do have trouble understanding is what a stream is. You have this series of log events that you get from your microservice node, that gets sent to the mantis nodes and they exist in an abstraction of a stream. What is a stream?

[0:13:07.8] KL: It's just a sequence of events, or logs as they happen.

[0:13:13.3] JM: Yeah. How does that differ than an abstraction like a table, like a database table?

[0:13:19.6] KL: Yeah, so database table would just store your logs probably in a single file, or multiple files. This is an idea that it's a finite set, or it could be a growing set. Whereas, a stream is just you're looking at a sequence of events as it's coming. It could be potentially an infinite stream of events.

[0:13:37.7] JM: Okay. As this stream of events is coming in, if I'm in charge of building observability tooling for my service, what kinds of additional things, what I want to set up on top of the mantis node that is accumulating this stream of events from my service.

[0:14:00.2] KL: As you mentioned, you could store the stream of events in a database for querying later. Let's say you need data for a month, even a year for example, you couldn't keep that all in-memory, so you do need persistent storage. Then being able to query that quickly can be important for just user experience of using those observability tools.

[0:14:21.2] JM: People get to choose how long they want to store their log data, and based off of how long they want to store it, maybe they want to write it to persistent disk or put it in Redis, or choose-your-own-adventure.

[0:14:35.6] KL: Right, exactly based on a use case. Yeah.

[0:14:38.1] JM: Okay. And is this mantis cluster used – do you have one that's devoted to logging data, or is this also used for data science jobs?

[0:14:47.2] KL: It can be used for any use case. Yeah, it's pretty flexible.

[0:14:51.8] JM: Can you describe in more detail the design? I'm not sure how familiar you are with mantis, but the design decisions between mantis and these other systems, like storm or spark, the other stream processing systems?

[0:15:02.3] KL: Yeah. It just goes back to the cost-effectiveness and the speed. A lot of these other systems, they can read a stream of data from Kafka, Apache Kafka like a messaging queue, or a database, but mantis was built for speed so it doesn't use a messaging queue, it doesn't use a database. It gets the logs directly from the app nodes themselves. Speed was important and then just being able to scale the mantis agents as needed, so you can size up when traffic increases, when everybody's watching during the evening. Then scale down when people start going asleep. A lot of these other stream processing systems, they don't have auto scaling.

[0:15:44.0] JM: As the nodes scale up, do your querying semantics – does querying mantis remain fast? Can you do MapReduce jobs across these different mantis nodes?

[0:15:54.6] KL: Well, MapReduce implies doing batch processing. Really it's just – the query doesn't change. It's just the system can handle an increase of the data rate and then the query will still work.

[0:16:06.7] JM: Tell me more about the set of tools. We've covered the basic use case of I've got a service set up and it's streaming logs into mantis, and then I can do things with those logs. Maybe I want to generate metrics, maybe I want to set up alerting, maybe I want to build a tool for customer service representatives. Help me understand what you are doing at the edge insights team and what kinds of tools you're building on this set of infrastructure?

[0:16:35.4] KL: Sure. Besides mantis, there's also a metric system called Atlas, which is also open sourced. I mentioned that before, that's used to build dashboards and to look at metrics. Then we're also doing some very basic machine learning inference type tools that look at all these metrics, as well as logs and try to come up with some English explanations about what's going on.

For example, one of our tools looks at a bunch of logs, as well as a customer streaming account, to decide for a given playback session. Let's say you tried to watch House of Cards, some episode of House of Cards. For some reason, you only got standard definition video and you're wondering, “Okay, well why did I get standard definition video? I have a streaming plan that supports HD video.”

One of our tools looks at various logs and looks at your streaming account, looks at your network bandwidth to figure out why did you get standard def only and not HD. We also expand that quite a bit to 4k. 4k videos is used quite popularly now with a lot of 4k TVs out there. We can also deduce, “Okay, well why didn't you get 4k video? Why you're only getting HD video.”

[SPONSOR MESSAGE]

[0:17:52.0] JM: Data holds an incredible amount of value, but extracting value from data is difficult, especially for non-technical, non-analyst users. As software builders, you have a unique opportunity to unlock the value of data to users through your product or service. Jaspersoft offers embeddable reports, dashboards and data visualizations that developers love. Give users intuitive access to data in the ideal place for them to take action within your application.

To check out Jaspersoft, go to softwareengineeringdaily.com/jaspersoft and find out how easy it is to embed reporting and analytics into your application. Jaspersoft is great for admin dashboards, or for helping your customers make data-driven decisions within your product, because it's not just your company that wants analytics, it's also your customers that want analytics.

Jaspersoft is made by TIBCO, the software company with two decades of experience in analytics and event processing. In a recent episode of Software Engineering Daily, we discussed the past, present and future of TIBCO, as well as the development of Jaspersoft. In the meantime, check out Jaspersoft for yourself at softwareengineeringdaily.com/jaspersoft. Thanks to Jaspersoft for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:19:25.0] JM: What will be the process of actually debugging that? If a customer called a customer service representative and said, “Hey, I'm not getting high-definition video.” What's the series of steps where the customer service representative is able to identify that issue and what's happening in the data pipeline on the backend that allows the customer service representative to diagnose, or work with other people to diagnose that issue?

[0:19:49.8] KL: Yeah. One of our internal tools shows one webpage for a given playback session. You can immediately tell why a customer is not getting 4k video. On the backend, it actually is aggregating logs from different systems to make that conclusion. It's looking at things like your streaming plan. Actually, each count can have up to five profiles and on each profile, you can actually indicate what type of video quality you'd like. You could just let it Netflix do its thing and just give you the best video quality, or you can actually tell it, “Well, I'm on a mobile phone plan. I don't want to use too much network bandwidth. Therefore, give me not HD.”

We actually check the setting that you use to play a given title. That profile, let's say had the lower setting and we can immediately tell, “Oh, you have that lower setting on that profile that's why you didn't get HD.”

[0:20:38.6] JM: Okay. Is there a set of interactions that you want to occur, or does the – can the customer service representative do this entirely themselves? Do they need to go out and talk to an operations person to help them figure out that issue?

[0:20:51.7] KL: No, they don't need to talk to anybody. That's the beauty of [inaudible 0:20:53.7] tools. We want to make them as self-service as possible, so any user coming to the tool should be able to root cause an issue just using the tool without having to talk with anybody.

[0:21:05.7] JM: Who builds the tool for the customer service person?

[0:21:10.0] KL: Customer service uses a suite of tools. We build one of them. They have another that they use when they're on the phone with the customer. It has other pieces of information, like what payment method they're using, that kind of thing. Different teams at Netflix build different tools for customer service, including us.

[0:21:27.1] JM: Okay. You mentioned Atlas. What is Atlas and how do you use it within your operations?

[0:21:31.8] KL: Atlas is our open source metric system. At the backend, it has a time series database so it can store metrics based on time and also multiple dimensions. You can build graphs, you can build dashboards using Atlas. Let's say I wanted to watch, or monitor, you mentioned codec before, I’m just going to take an example. Let's say we're watching anything encoding pipeline, we're just encoding a bunch of content from our studios for example. You could build a dashboard that just looks at what's happening throughout the day and you can see a graph throughout the day about what's happening with the encoding pipeline.

You can also use Atlas to do other things, like we were talking about different microservices. You can look at each microservice and watch its CPU load. You can look at its network bandwidth usage, bytes in, bytes out, that kind of thing. You can also use, I mentioned multiple dimensions. Let's say you wanted to watch, or monitor playback traffic for, I don't know, the Apple iPhone, the iPhone X for example. You could look at the iPhone X and drill into just playbacks for a given country, you could drill in to, I don't know what else, a given title. For example, let's say you wanted to know on the iPhone X who's watching House of Cards episode 1, you could do that too using Atlas.

[0:22:48.9] JM: Now there's a difference there between monitoring your own node activity and monitoring your user’s delivery mechanism. In one case on the Netflix app, you want to be able to – that somebody downloads and they're watching a video, you want to have some sense of crash reporting and crash, or just data reporting that gets shuttled back to Netflix via some mechanism. In another, the data is generated a little bit closer to home, to your – you have the straightforward, I guess mantis architecture, or some other system where you can just deploy the agent that the data reporting agent on the node and have it be reporting back to some system. What's the difference in shuttling data between a user agent device and a service device across Netflix's infrastructure?

[0:23:47.0] KL: Right. The backend servers have a more reliable network connection, but for all the customer devices yeah, that can be – there's so much variation in ISPs, right? What customers use as their ISPs, or their cellular network. The device has got to be more resilient and sending back telemetry. The Netflix app will do things like it will batch up, let's say logs on the device and send it out. If it can't send it successfully, it will try again later. There's a little bit more resilience on the client app. We also have some of that retry logic also on the backend, as well that we – I don't think it occurs as much as it happens on the client devices.

[0:24:27.0] JM: Is there a user insights service that is accumulating all this data, like it's occasionally getting batched from the user devices and it’s getting sent to Netflix and then accumulates in some service, some particular service?

[0:24:39.9] KL: Yes. I can think of one thing that we call device logs. A particular service handles those logs and stores them in persistent storage.

[0:24:48.3] JM: Okay. Well, in contrast, the model where you are just trying to monitor the codec service, or something else that is on your infrastructure, you want to be able to accumulate telemetry about those individual services so that you can diagnose for example a JVM issue, or a database issue, or a resource constraint on the CPU, or the network, or the disk. You want to have this different telemetry instrumented. To some degree, you want to make it turnkey for the microservice developer to have this stuff easily deployable. You want some configurability of course, but you also want some straightforwardness. What's the standard model for deploying this instrumentation? Does it just come with the service? Is it a service mesh thing? Tell me about that deployment system.

On every service that somebody deploys at Netflix, if I'm a random developer deploying a microservice, I assume there's some instrumentation that just comes out of the box, thanks to me working in Netflix, and it helps instrument my application and it helps me debug it and it probably is the data that goes into Atlas, it helps me get logging, perhaps help shuttle my logs to mantis. What are the things that are bundled into that service, or help me understand what the agents that get deployed with a typical microservice at Netflix are.

[0:26:18.0] KL: There's a group here called platform engineering and they develop all the libraries that go out with every Amazon AMI, every image that gets deployed in the Amazon Cloud. The mantis agent is one of the libraries, that’s included one of the libraries that gets deployed with every AMI. Similarly, there's an Atlas client also that gets deployed. Yeah, so there's just various clients that get deployed within every app.

[0:26:43.2] JM: What does the Atlas client give you, or give you a particular application?

[0:26:47.1] KL: It gives the ability to log any metric that the app needs and the Atlas client will take care of sending those metrics off to Atlas.

[0:26:57.3] JM: Okay. If I'm writing an application, then I can insert a debug statement, or a log this thing statement and I can tell Atlas, "Hey, I want this metric to be remembered here."

[0:27:12.6] KL: Yup, that's right.

[0:27:14.0] JM: What's the pathway that it follows, it just gets sent to Atlas and then what system is it stored in?

[0:27:21.9] KL: That's owned by the Atlas team and I don't know too much about that.

[0:27:25.0] JM: Okay. All right, fair enough. You've talked a little bit about machine learning. How is machine learning used in this observability infrastructure?

[0:27:33.1] KL: We use it for primarily looking for trends in the data. For one of our tools, we look at – I mentioned device logs earlier. We look at the device logs coming off various devices and we try to group them. For example, I used the iPhone X as an example earlier. We could do is let's say, a customer calls in and they had a streaming problem on their iPhone X for Orange is the New Black let's say in episode. What our machine learning will do is it will observe that an error happened on the given account for Orange is the New Black, let's say episode 1.

We'll actually go and look at other accounts to see if there's any correlation with other iPhone X's for that title. If so, it can alert the user of the observability tool that they can tell, "Oh, this is not affecting just this account. We also notice this is affecting other accounts as well."

[0:28:27.5] JM: Okay. This is something that your team built on top of Atlas, or was this something that the Atlas team built?

[0:28:33.4] KL: This is something we built ourselves.

[0:28:36.3] JM: Okay.

[0:28:37.1] KL: It doesn't involve Atlas.

[0:28:38.8] JM: Okay. The time series database, I think you mentioned time series database and that was within Atlas, so I guess that's another thing that's not really under your scope, right?

[0:28:48.8] KL: That's right, that's right.

[0:28:50.0] JM: Okay. Tell me more about what you do on the edge insights team and what you're building today.

[0:28:57.2] KL: I'm a full-stack developer, so I do both the frontend and the backend. On the backend, I aggregate a lot of data. I also think about algorithms and different trends to figure out errors. On the frontend, I build different tools. We've used different frameworks. We've got for example, an Ember.js-based app. We use ReactJS for other tools as well. What I'm working on today, it's a lot of that trend analysis, it's distributed tracing.

We want to on top of distributed tracing, what I mentioned before is deriving intelligent conclusions on those traces. A trace will tell you basically the topology of a request. Let's say when I hit the play button, what happens? I can see all the microservices that serve that request, but we want to add on top of that is let's say that play request fails for example. Let's say a customer account has exceeded their stream limit. Let's say they had a four-stream plan and they've tried to play on a fifth device. We want to be able to surface that in addition to just the trace. There's a lot of additional logic that we build on top of the trace and on top of the logs that we look at. That's primarily what I've been working on.

[0:30:10.2] JM: Now how do you decide what to work on? You're in the midst of seeing these different customer problems and building all these different tools, so distributed tracing for example, these distributed traces of various different services, distribute trace can tell you how a service bounces through a bunch of downstream services. Then you can look at a collection of distributed traces for any given service over time and you could for example, build a flame graph with it. By looking at the flame graph, you might be able to derive that there's some problem.

In order to know what flame graph to look at, or what service to study, or what problem in your infrastructure to study, you need to have some knowledge of the problems that are manifesting across the organization. It almost sounds like you're a researcher and you look into specific problems that are happening around the infrastructure, or you get wind of them somehow and then you look into them and you inspect them and you try to find ways to either observe them more comprehensively, or actually figure out how to remedy those issues. How do problems bubble up to you and how do you find what to research and what issues to work on?

[0:31:31.6] KL: I meet regularly with a lot of the teams that use our tools, and I get a lot of feedback, and people give me feedback via e-mail, via meetings. We also reach out to our users. We know who our internal users are. We reach out to them and say, “Oh, hey. How's it going with our tool? Do you find it useful? We just deployed that new feature. Are you finding it useful?” They give us a lot of feedback based on features that we deploy and then also as problems arise for them, they bring use cases to our team and we think about how to address their use case with our tooling.

[0:32:04.6] JM: You've been at Netflix for I think eight years, is that right?

[0:32:08.6] KL: Mm-hmm. Yeah, a little over eight years. Yeah.

[0:32:10.5] JM: When did the edge insights team get formed? What was the impetus, because it's a very specific type of team.

[0:32:18.0] KL: Yes. I don’t remember. We originally were just one team, the streaming infrastructure team and we built all the backend servers that serve streaming. At some point, I looked at one of the internal tools that we had and it didn't have very good user interface. I basically rewrote the user interface for a half-day project. I ended up winning award for that and ending up owning it for since then. I think that was what? 2012. It's been six years.

[0:32:46.5] JM: As a fortuitous hackathon.

[0:32:49.4] KL: Yes. I can't remember when the team formed after that, because that we were also working on other server apps at the same time.

[0:32:57.1] JM: That UI was for what exactly?

[0:33:00.9] KL: It was a tool to allow you to introspect what was going on during a screening session. You could look at what was happening on your iPhone X for example, when you're watching an episode of a show and just look at all the device logs that were coming off the device, you could look at what the server was logging. It was a very basic tool at the time. It didn't have any of that intelligence that we've added since then.

[SPONSOR MESSAGE]

[0:33:31.2] JM: Thank you to our sponsor Datadog, a monitoring platform that unites metrics, distributed request traces and logs in one platform. Datadog has released new log management features and trace search and analytics to help you explore and analyze your APM data on the fly. Plus, Watchdog is a new auto detection engine that uses machine learning to alert you to performance anomalies without any setup.

Datadog includes out-of-the-box supports for more than 250 technologies, including PostgresSQL, AWS, Kubernetes and more. With rich dashboards, algorithmic alerts and collaboration tools, Datadog can help your team troubleshoot and optimize modern applications.

Start monitoring with a 14-day trial and Datadog will send you a free t-shirt. You can go to softwareengineeringdaily.com/datadog to get that free t-shirt and to try out Datadog with all its new features. That's softwareengineeringdaily.com/datadog.

[INTERVIEW CONTINUED]

[0:34:47.6] JM: When I think of streaming video to somebody, to me it seems like a challenge that is pretty hard to introspect effectively, or you really have to probably study it to understand a good strategy for introspecting, because there's so much that goes on between the client and the server. It's such a high bandwidth transmission and such a long-lived transmission that diagnosing a problem seems really hard. Can you tell me some of the tips and tricks for monitoring a high bandwidth connection process like this? I'm sure there's somebody listening who is like a game developer, or a VR developer and it would be really useful for them to understand this better.

[0:35:32.9] KL: Yeah, yeah. It goes back to observability. It's just how well can you look at a systems outputs and infer what's going on inside. For a high-bandwidth app, I mean, it's doing a lot, but how do you build your app so that it is emitting events, or logs so that when you look at those events and logs, you know exactly what your app is doing at that time. Let's say you want to look at, played a two-hour movie, you have to have enough logs and events to tell you what exactly happened during that two-hour window; to be able to store it, to be able to query it easily.

It's quite challenging, especially when you think about how many millions of Netflix members watch every day and being able for customer service for example to help those customers when they have a streaming issue. How do you look at one specific streaming session and be able to tell the customer, “Oh, hey. I know why you're having an issue.”

I guess, advice would be yeah, it's all about think about the observability. What about the internal state of your app do you need to expose, so that if somebody were to ask you about it later, you know immediately about what's going on in your app.

[0:36:43.7] JM: Do you have a mental flowchart for first, check the network logs and a second, check the JVM logs, or something like that? Is there some sequence of priorities you have?

[0:36:57.0] KL: It really depends on the use case. I think for streaming, people are concerned about what was the network bandwidth, how did the network bandwidth change over time. Yeah, well how about the device? You mentioned load on the device, CPU load, what was going on the server. Was the server loaded? Was it healthy at the time? I mean, all that needs to be taken in consideration with what's happening on the client, what's happening on the server, what's happening on the network.

[0:37:20.9] JM: Your process for diagnosing it, is it you just look at data sets and graph them and look for outliers, or what's the process of diagnosing of waiting through all that data?

[0:37:35.8] KL: Yeah. Basically, outlier detection, anomaly detection, something abnormal happened. How do you know if something abnormal happened? Well you need some baseline of what normal means, so during a normal streaming session we know what that looks like. We know what a bad streaming session looks like for example, when you get the reload spinner on the screen and your playback suddenly just freezes and then it starts up again after a few seconds. It's called a rebuffer. We watch out for signals like that.

The device does signal when it has a rebuffer, or when it runs into other issues, like when you get an error code on the screen, we know when that happens as well. We know when something bad happens, we know what that looks like. We start with that, we look for trends. I gave the 4k HD example. I mean, you could playback fine in HD, but really you were expecting 4k. The device won't send any errors. It'll just say, "Oh, I just played in HD. I didn't play it in 4k." We want to look at well, we noticed that the title is offered in 4k, we noticed that your device supports 4k, but why didn't you get 4k? We look at those cases as well, not just the error cases.

[0:38:44.1] JM: The streaming team, I guess you were getting that going, was that six years ago, did you say?

[0:38:50.1] KL: That was when I first started in 2010, so that was eight years ago.

[0:38:54.1] JM: Okay. Eight years ago. How have the challenges of building streaming infrastructure and streaming observability, how have those things changed as Netflix has matured so significantly?

[0:39:05.1] KL: Yeah, it's just gotten more complex. There's just more product features over the years that got added. For example, offline viewing so you can download a title on to your iOS, Android or Windows phone. That was a huge feature that got added that affected a number of systems, and we needed insight into what was going on with offline viewing. You can think about it like you're on an airplane and you're watching something on your phone, you have no network connection, the app is running, you're watching a two-hour movie. Ideally, you like to know what happened during that two-hour time window.

When you come back online, when you connect to the internet again, some telemetry does get sent to the Netflix servers and we can record what happened. If you ever ask about – if you as a customer I mean, call about like, “Oh, hey. How come I saw a bunch of rebuffers for example? Why did that happen when I was watching that episode?” We can look at our data and tell you why.

Just the increasing complexity over time as product features as a product, as a Netflix app evolved over the years, we've had to incorporate more and more telemetry, more logs, more events. It's just complexity has gone up.

[0:40:12.7] JM: One thing we touched on earlier when we were talking about mantis was the fact that different customers, or internal customers, people who are using mantis, or using different aspects of data infrastructure, they are in some sense I believe at Netflix, responsible for their own cost controls, for their own budgeting. Can you tell me how cloud cost controls are administered and how they're budgeted, give to apply for stuff? How do cost controls work at Netflix?

[0:40:49.8] KL: They're reviewed, but there's another team that handles that and I don't know the details there. We do know that the cost of our own servers and our storage for example, and we review that periodically. As far as like, you're asking about planning, that I don't.

[0:41:05.2] JM: It's never been an issue for you where somebody said you're spending too much, you need to control things.

[0:41:10.9] KL: In the past, some of the logs have gotten up there and we did get that feedback, “Look. Oh, hey. Can you really reevaluate, do you really need to store all that data, for example?” That has happened in the past. Yes.

[0:41:22.1] JM: Let's go back to distributed tracing. There are more and more companies that are instrumenting distributed tracing. Why is distributed tracing useful?

[0:41:31.7] KL: Because of the increase in microservice architecture is just really hard to see all the contributing servers that service your request. One of the top uses of distributed tracing is just knowing the network topology. When a request gets submitted to one server, it hands it off to a bunch of other servers. Just knowing what servers were involved helps a lot. Tracing also is helpful to know about latency. Let's say somebody changed some code somewhere on some microservice and that caused the overall response latency to go up. You'd ideally like to know that and which service contributed to the increase in response time.

It's topology, it's latency that I hear are the top two values of distributed tracing. We also want to add stuff on top of that trace too. We were talking about the 4k HD use case, or just the more English explanations of what's going on in the system, why something went wrong, why didn't you get 4k, why didn't you get HD. We want to add that logic on top of the trace too.

[0:42:32.0] JM: Say more about that. What do you mean by that?

[0:42:34.1] KL: It's just we think we can provide more value just to the users of our tools, than just looking at a trace. Normally we look at a trace, you know what the topology is, you know how long it took for each microservice to respond, but it doesn't tell you anything else than that.

We also want to tie together other things, like logs for example. Like, "Oh, we noticed this microservice was involved in this distributed trace. Let's go look at its log and see what it emitted. Can we deduce anything from the log that it emitted and surface that to our tool user?"

[0:43:06.5] JM: Can you give an example of what that would look like? Maybe talk about the 4k video example in more detail on how you would use, let's say, there were some – again, distribute trace for those who are a little unfamiliar is a series, if you have some service A call service B, service B call services C and D, each of those call other services, you can have this graph of services that are called and each of them has some amount of time that they spent in this distributed trace, and then over time that the traces will resolve. Maybe some of them time out, maybe some of them miss for some reason, but the end result is you have a trace of the top-level request and you can gain all sorts of insights about that.

Now what you were just saying was maybe if a microservice developer is delving into their distributed trace, they want additional insights on top of just looking at the trace, looking at the visual representation of the trace. Maybe you could potentially give them an insight to say, “Hey, you hit service X and we know that service X is associated with this database,” or has this other issue. What's an example of when you could surface additional insights through that distributed trace debugging process?

[0:44:22.1] KL: Right. A good example is that 4k use case. For a lot of Windows PCs, they need the KB Lake processor for example that supports HEVC hardware encoding – decoding. Sorry, decoding, but not all Windows laptops do that. Let's say when you hit play, we can show you the trace, but also we can also look at the why a device – does it support that hardware decoding, or not? That only comes from a signal from the device.

The device itself knows whether it can support that video decoding or not. That way we can – we show the trace, we show that okay, this is the service that decided. You asked for 4k, but we can't provide it to you because we know your device can't support it. In order to arrive at that conclusion, we actually needed to look at a log that came from the device.

[0:45:11.9] JM: I think of observability teams as hand-in-hand with ops teams, or devops teams, or on-call. Do you spend much time interacting with ops teams, or you mentioned other teams, but I'm wondering if you use any tooling, or if you have any best practices for interacting with other teams, if you use Slack, if you use, I don't know project management tools. What are your best practices for interacting with all these other teams?

[0:45:39.7] KL: Yes, we interact with operations teams. In fact, we have a central operations team and they give us feedback periodically. We have Slack. As issues arise, if it's urgent, you have normally, they'll reach out via Slack. Yeah, best practices it's as needed. I mean, if it's urgent, then Slack is a wonderful tool, real-time chat. Other best practices, just meeting regularly, we also have tried user surveys, where we ask about specifically one of our tools and how people use it. If they have any suggestions for improvement, for example and we collect a survey feedback that way and incorporate that into our future plans via surveys, Slack, there's also just e-mail and we also just in-person meetings.

Oftentimes, other teams are so busy that they don't have time to respond to survey, so we make time to meet with them. Sometimes we've been meeting quarterly with some teams. Other teams, maybe every six months, something like that.

[0:46:37.6] JM: Coming back to distribute traces, because there's something I forgot to mention. As we talked about earlier, the logs from services get stored in mantis and you can have this, build your own query adventure on mantis nodes. Is the same true with distributed tracing requests, or do you have a different storage and querying system for those?

[0:47:00.4] KL: It's a different storage and querying system for the distributed trace data.

[0:47:04.0] JM: Okay. Were you involved in building that? Are you familiar with them?

[0:47:06.7] KL: Yeah, I worked on it. Have been working on it. It's been evolving. The trace data comes off the app service nodes and it gets published to Kafka. From Kafka, we read all that, all the trace events and we actually have a mantis job that actually looks at all those events and stitches the trace together. That mantis job is responsible for storing the trace data, the transform trace data into a persistent storage.

[0:47:35.2] JM: Okay. After it goes into the persistent storage, actually what are you using for the position storage? What database are you using to query those distributed tracing requests?

[0:47:45.1] KL: Right. We used to store in Elasticsearch, because of Elasticsearch’s flexibility. As the volume grew, Elasticsearch had a huge ingestion delay, because Elasticsearch needs to keep indexing all the documents that you're giving it. The size Elasticsearch cluster grew in addition, because the Elasticsearch cluster is spending so much time ingesting the data, indexing it, read response times actually got really high, so high that people gave us feedback and said, “This is too slow for me. Your tool is basically unusable, because it's taking too long for me to pull up the trace.” At that point, we switched to Cassandra which has much better ingestion write rate and also read rate. We're on Cassandra now. It's been working wonderfully since we moved.

[0:48:30.0] JM: Now, is it harder to query Cassandra, because Elasticsearch has really flexible querying semantics from that indexing.

[0:48:39.4] KL: Right. We needed to plan ahead on all the use cases to query that trace data. We have a limited set of queries to query against trace data. Those few use cases work for everything right now. We can build indexes as we need. It's basically in Cassandra. You just add another entry that says, "Okay, somebody wants to look up a trace by this identifier. Let's map that identifier to the identifier that we use, our trace identifier." Once we establish that mapping, then we've got another way to look up the trace data.

Yeah, in contrast Elasticsearch where you just say, "Well, I can index on all these fields and I can be able to query on any of them." Elasticsearch definitely gives you that flexibility, but again, there is a downside of the ingestion rate. All that processing needs to happen to index all the incoming documents.

[0:49:23.4] JM: Okay. Distributed traces, each one you can consider a document and it's high dimensional. It's got a bunch of different services that it touched, it's got some information about each of those contact points with the different services and the process of indexing, the collection of distributed tracing requests, you could be indexing on any number of those dimensions. If you were to naively do it in Elasticsearch, it's going to take a while and it's going to be indexed on – you're going to build indexes across all of those dimensions. Cassandra, you just pick and choose the dimensions that you want to index on?

[0:49:58.8] KL: That's right. Yeah.

[0:49:59.8] JM: What are the most high signal indexes, or the most regularly queried indexes?

[0:50:06.0] KL: I mentioned a trace identifier before, so you can query on our trace identifier. You can also query by a device identifier. Let's say you know that iPhone X is identifier, for example, you can query on that and get all the traces for that device.

[0:50:20.5] JM: Cool. Well, I know we're nearing the end of our time. I think we've covered most of the results of your article, most of the details of your article. Is there anything else you want to add about observability at Netflix, or your experience over the eight years there?

[0:50:37.1] KL: It's always challenging and continues to be challenging thinking about how to design and implement each service, so that it can expose enough of what it's doing, so that you can know what – how that service interacts with all the other services who also need to emit their state as well. How do you tie everything together? It's always challenging. Different teams can have different approaches. How do you work well together, so that you can tie everything together to know exactly what happened during some conceptual event like a playback session, for example, when talking about playback?

There can also be other concepts for the future too. For example, when you first start up the Netflix app, you get to see rows of boxart images. What if one of those images didn't render? Why did you get recommended this particular category of movies? There's just a lot of questions that we have and we're thinking about how do we apply our observability learnings to these new use cases.

[0:51:35.2] JM: Oh, yeah. Well, the question of why did I get recommended this movie? That's a whole new world of observability. How do I observe the black box of machine learning models that I'm being fed?

[0:51:47.3] KL: Yeah. All the recommendations, algorithms, yup.

[0:51:49.4] JM: Wow, interesting. It sounds like your work is – it's almost like a technical product designer at this point. You know Netflix infrastructure pretty well and you're basically going around and talking to teams to be like, what do you want me to be to build for you? What do you want me to look in to? It’s like you're just talking to these different internal customers and seeing what needs to be built.

[0:52:07.7] KL: Yeah. It's an interesting mix of getting a specific feedback on specific use cases, but yeah, also you're right. I mean, thinking about what we think users will need.

Sometimes the users don't know what they want and we have to just think about and come up with the best experience that we think would be useful in troubleshooting issues. Yeah, it's an interesting combination.

[0:52:27.2] JM: Well Kevin, thank you for coming on Software Engineering Daily. It's been really great getting your experiences and your lessons learned.

[0:52:34.2] KL: Yeah. Thanks for inviting me, Jeff.

[END OF INTERVIEW]

[0:52:40.3] 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 e-mail 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 talk 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.

To find out more about sponsoring the show, you can send me an e-mail or tell your marketing director to send me an e-mail jeff@softwareengineeringdaily.com. If you're a listener 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.

Send me an e-mail at jeff@softwareengineeringdaily.com. Thank you.

[END]