EPISODE 523

[INTRODUCTION]

[0:00:00.6] JM: Kafka is at the center of modern streaming systems; Kafka service as a database, a pub/sub system, a buffer and a data recovery tool. It's extremely flexible, and that flexibility has led to its use a platform for a wide variety of data-intensive applications.

Today's guest is Gwen Shapiro, a product manager at Confluent. Confluent is a company that was started by the creators of Apache Kafka; Jay Kreps, Neha Narkhede and Jun Rao who have all been on the show in previous episodes. In those shows we discussed the inner workings of Kafka. This episode is more about practical use cases and design pattern around Kafka.

Gwen explores a few use cases. First, reconciling data between different servers, a massive international multi-user game, like World of War Craft, needs to keep its users in sync despite the fact that those users are pinging different server locations. Kafka can be used to help reconcile data about the users that are across those different servers. This discussion reminded me of the awesome show that we did with Yan Cui about scalable multi-playered games. Other examples that we discussed include log management, data enrichment and large scale analytics. This was a great conversation. I think you're going to enjoy it as well.

I wanted to mention that summer internship applications to Software Engineering Daily are being accepted. If you're interested in working with us on the Software Engineering Daily open-source project full-time this summer, send an application to internships@softwareengineeringdaily.com. We'd love to hear from you.

If you haven't seen what we're building, what we're looking for interns for, you can check out softwaredaily.com. You can download the Software Engineering Daily app for iOS or Android and you can find all of our episodes. You can find recommendations, discussions, related links and it's all open-source at github.com/softwareengineeringdaily. I hope you'll like it.

[SPONSOR MESSAGE]

[0:02:10.3] JM: Your enterprise produces lots of data, but you aren't capturing as much as you would like. You aren't storing in the right place and you don't have the proper tools to run complex queries against your data.

MapR is a converged data platform that runs across any cloud. MapR provides storage, analytics and machine learning engines. Use the MapR operational database and event streams to capture your data. Use the MapR analytics and machine learning engines to analyze your data in batch or interactively across any cloud, on-premise or at the edge.

MapR's technology is trusted by major industries like Audi, which uses MapR to accelerate deep learning in autonomous driving applications. MapR also powers Aadhaar, the world's largest biometric database, which adds 20 million biometrics per day.

To learn more about how MapR can solve problems for your enterprise, go to softwareengineeringdaily.com/mapr to find whitepapers, videos and ebooks. MapR can leverage the high volumes of data produced within your company, whether you're an oil company like Anadarko or a major fin-tech provider, like Cabbage, who uses MapR to automate loan risk and has done $3 billion of automated loans to date.

Go to softwareengineeringdaily.com/mapr to find out how MapR can help your business take full advantage of its data. Thanks to MapR for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:03:52.8] JM: Gwen Shapiro you are a product manager at Confluent. Welcome to Software Engineering Daily.

[0:03:57.9] GS: I'm super happy to be here.

[0:04:00.0] JM: Yeah, I'm really happy to have you. We've done a bunch of different shows about Kafka and then recently we've done a lot of shows about streaming frameworks and how to use those streaming frameworks and it seems like Kafka is at the center of the story for so

many of these different architectures, whether we're talking about a company that's doing large scale anomaly detection or just queuing up events to be processed in a CQRS pattern. Kafka is really at the center of a lot of it. But you've been around since before Kafka was a thing, and I wanted to get a little bit of your perspective on where we are historically, evolutionarily. You used to do a lot of different consulting. How has data management evolved over the years since you started working in this field?

[0:04:53.0] GS: Yeah, it's been differently a fun journey. So I'd say I started working on data management maybe around 1999, year 2000, and I was working as an engineering in Oracle DBA, and we had big databases and that was pretty how much how you wrote applications. At some point, if you needed to do large scale data analysis, you go to really big databases and you did data warehouses, and it was really — Like the center of your existence is data warehouse architects was those huge data models and those large batch jobs basically running once a day, loading the data in and running those transformations and it takes tons of CPU and if anything went wrong, it's like, "Oh my god! We will never have to report in time on the CTO desk at 8 a.m. in the morning and when you will get [inaudible 0:05:48.0] data." By the time you find anything out, it was kind of too late, because it ran all night and there you are at 7 a.m., and what can I do about it in our job right now?

It was always a struggle, like you had new data sources and you wanted to create new reports and you wanted to, as you said, maybe do some fraud detection, and you have all these information somewhere, and then it turns out that it's not in the data warehouse, because we never modeled it right, or the teams that does collect the right information never really talk to us. Even if it wasn't the data warehouse, maybe it wasn't in the right form, and it's a three month process to find the information [inaudible 0:06:24.4] into that model and then create a report. So then six months later you may have something ready and, I mean, at some point the attitude in the organization is like we can't ask this team for anything. It will just take forever, and they'll ask us for data models and we don't know how to do it and it will just be very painful.

Then somewhere, let's say, 2010-ish maybe, the pendulum kind of swung in the other direction. We had SQLs and we Hadoops and suddenly just get all the data in there and you don't have to do any more delaying, you don't have to do all those transformation. You just dump the data in

and whoever needs to report will just deal with it and create the reports in it. Everyone is just super heavy that they don't have to talk to DBAs anymore.

Then the problem was that now that they had to figure out a lot of contradictory and problematic data formats. Some of it was CSV, some was XML, some was JSON. What do we do with all that? How do we actually generate — Can we trust the data? How do we know it's correct? So it went from something that's very regimented and moved too slowly, something that was basically wild west.

[0:07:34.9] JM: Yeah. As I understand, the evolution — So as you said, you started with this time where you have Oracle and you've just got your big Oracle database, and that might as well be your data warehouse or your data lake or whatever you want to call it, your giant data repository. Then we had the age of one-size-fits-all ending, and because we had motivations to move to different databases and we also wanted to start throwing all of our data in HDFS, because the Hadoop distributed file system is where you put your data if you want it to be processed by Hadoop jobs, which are going to run a lot faster, I'm assuming, than whatever other database query you would run to do this nightly reporting. So that was great for a while. It seemed fantastic, but overtime we realized that this approach of putting everything in these HDFS files and writing them to disk, this data lake or data warehouse and then this nightly reporting, this could be improved upon, and then that's how we eventually made our way towards the data stream being a place where you can have your cake and eat it too in some sense, where you get all your data in this one place maybe with some sort of expiration date, but over maybe a two-week window. You've got all your data in Kafka or in some other queuing durable system and you can do operations over it in a much more accessible, efficient fashion. That's the data streaming model.

So maybe you could just give me a perspective for how you think about that evolution, going from the big database to the data warehouse or data lake. I think those are kind of the same thing as far as I can tell.

[0:09:21.8] GS: They've both batchy.

[0:09:22.3] JM: They're both batchy, yeah, to the data stream. Describe that evolution as you see it and what do you think are the implications moving forward.

[0:09:30.2] GS: Yeah. Basically a lot of driver behind streams was that we wanted speed. We needed events to be processed in real-time or very close to real-time, basically immediately. We definitely did not want to wait for the big batch job to run. We also wanted to be able to share data in organization. We wanted to have this one source of truth that everyone can refer to and also give everyone who wants to create their own version of this truth, the ability to use their own data store was still being connected to a central source of updates and events. This kind of opened a really interesting possibility of connecting what until then was kind of this separate world. So there was people building microservices and events-driven microservices and then use those messaging apps. Basically JMS and all those messaging solutions all relied on transient messaging. So you had those message, but you didn't have the store of messages that you can go back to and do data analysis and do stream processing and, as you mentioned earlier, CQRS, basically having this shard source of truth for the organization.

So I think that's what makes Kafka valuable in so many different domains is that it gives you the speed and it gives you reliable storage in really distributed scalable but a very reliable with [inaudible 0:10:57.5], and then it also gives you the ability to share data far and wide across large organizations. I think that's key, because now groups in organizations really never talk to each other and the people who do data analysis and microservices people never had to really interact, and now they have this shared common view of the world and they can really build things together in ways that they don't get in each other's way, but they can share events and figure out their shared view of the world.

[0:11:31.3] JM: So I can get trapped in a bubble, because I talk to people who are presenting at conferences and people in the Bay Area, and so on, and I sometimes miss out on the pace at which technology is actually being adapted by the bulk of people who could be adopting it. So you take a bank in the Midwest or a manufacturing company in the middle of Canada somewhere, and they've got enough data where they would be making use of something like Kafka. How rapidly are large enterprises adopting it? What's the adoption curve looking like of these kinds of technologies that we've been talking about for a while?

[0:12:16.7] GS: Yeah. I'd say we are now at maybe the early majority. Like one mistake that we seem to make is to look at those large companies as a monolith. You look at, as you said, large manufacturing company in the Midwest and you think that, "Oh! These are all the same." Maybe they employ more engineers than two Facebook, but these are all pretty much the same, but there actually seem to always be some group inside the organization that may as well be in the Silicon Valley. They are user heads. They are probably listening to your podcast and they're like, "Ha! Kubernetes. Let's try to do some Kubernetes."

And they, in a year, those software companies do more projects than a lot of the Silicon Valley. They just work on this massive scale. So at any given point of time, there's probably one project that is trying out something that's new and cool, and if it succeeds, they start going around inside the organization, because we are all human beings. We want to feel influential. We want to share what we learn. [inaudible 0:13:21.9] going around and basically telling everyone, "Hey! I have this new streaming platform and I just built this malfunction detection in this part of the plant and it's working amazingly and I heard that you have your calculate accounting project and maybe you want to try to it in real-time, because I think we actually already know how to get you all the data you need." It will actually jumpstart and save you 5 months in your project.

And I've seen it done over and over again. Basically, like most of the customers I talk to now are really large companies, large financial institutions, insurance companies, some manufacturers, and it's always like the small group that try something, makes it successful and then they just cannot stop running around and telling everyone, "Oh my god, guys! You have to try this."

[0:14:12.9] JM: Yeah. Let's talk through some examples of design patterns in Kafka. There's probably people listening who still — Maybe they haven't worked with Kafka or they've just heard a little bit about it. They know it's a useful technology, but they don't know what it does.

So Kafka is a distributed pub/sub messaging queue. One example of how you could use it, and you've written about this, is reconciling data between different server instances. This gets at the shared data aspect of it. So you give the example of these large multiplayer games. So you've got games where you've got multiple players that are running around in the same world, like an MMO, for example, like World of War Craft. You've got all these different players running around in the same virtual world, but their game sessions are handled by different servers, because

you've got this massive virtual world. So one person running around in one area of the world is maybe not handled by the same server that is handling what goes on in another area of this virtual world, because it's a massive virtual world. They may take actions that impact people who are in the other area of the world. So you want this unified experience. You want the game sessions to be able to interact with one another despite the fact that they are being handled on different servers, because it's this big virtual world where people can interact with each other. That's the whole idea of it. Why is Kafka a good example of being able to help with this use case?

[0:15:43.4] GS: There are few parts of it. The first is like it's highly reliable and, obviously, the last thing you want in the middle of a game is for parts to start crushing and not functioning or even slowing you down a tiny bit. So Kafka gives you this asynchronous experience where you send messages to Kafka, but you don't necessarily have to wait for anything to happen. So you just process events when they arrive.

The other part that is really important is that Kafka has strong ordering guarantees. Obviously if I take a loot and now I have some few items I didn't have before, the last thing you want to happen is someone else to see me have few items and then me taking the loot or things that have been out of order.

So I think that Kafka's strong ordering guarantees are super important to that, and I think that when I talk to people about how to really create functioning applications on top of Kafka, you kind of have to think about where you need those ordering guarantees and when you need data to be processed together in a certain order as a group, for example, everyone who is in this region of the world we need to get their data together, because they can physically — You want to give the experience as they're physically seeing each other while people who are a bit two virtual miles away, you don't need them to get at the events at the same speed, because they don't need the experience [inaudible 0:17:15.0] later wondering. So you can actually separate things based on geographic location. Sometimes if it's credit cards, you need to think about events relating to maybe one store or one card. So you have to think about what data has to go together, what data can be separate, because the more you can actually separate the data and say, "Those things don't have to go in strict order. You can process them in any order."

Now you have the ability to really do massively distributed processing, and that can be like just applying a lot of Kafka streams or spark jobs, processing the data in parallel just to make sure the data is handled faster, but it can also be things that are across the globe. It's very common to have redundant data centers in New York and maybe the other side of the Hudson, if you're a financial institute, and then maybe something in L.A., in Vegas and something in London. Especially in games where speed is important, then locality becomes really important. Having a system that can be globally distributed is really important. So Kafka usually plays a big role in sending messages in a reliable order, high throughput way between just different geographies.

[SPONSOR MESSAGE]

[0:18:36.8] JM: LiveRamp is one of the fastest growing companies in data connectivity in the Bay Area and they're looking for senior level talent to join their team. LiveRamp helps the world's largest brands activate their data to improve customer interactions on any channel or device. The infrastructure is at a tremendous scale. A 500 billion node identity graph generated from over a thousand data sources running a 85 petabyte Hadoop cluster and application servers that process over 20 billion HTTP requests per day.

The LiveRamp team thrives on mind-bending technical challenges. LiveRamp members value entrepreneurship, humility and constant personal growth. If this sounds like a fit for you, check out softwareengineeringdaily.com/liveramp. That's softwareengineeringdaily.com/liveramp.

Thanks to LiveRamp for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:19:43.5] JM: Let's go a little bit deeper on that example, because you explored so many great areas that we could uncover a little bit more on. So the first is we've got these different game servers that are handling different players in different parts of the virtual world, and they may also be handling users in different geographic areas of the world. Maybe you would have a player in China who is exploring the same area of a virtual world as a player in America. Those two players, since their sessions are in different geographic locations in the physical world, they may be handled by different servers.

So for this example of the virtual world, do we have — Help me think about like what is our design for Kafka in this instance. We know that we want to be writing the events that occur for these different players to Kafka topics on the pub/sub system. We know that if I write from one area of my virtual world, we know that that wants to be reconciled at some point with the other area of the virtual world. How does that lead to how we design our Kafka topics? Because topics are where you publish messages too, like I can publish a message to a specific topic, like the North America version of the virtual world, or the South America version of the virtual world. That could be a topic, and then you can also have topic partitions within the topics. Then also, you talked about the different geo-locations. Do we have different instances of Kafka? Do we have one central view of Kafka that gets reconciled between different Kafka instances? Just help me understand, like how would this look architecturally?

[0:21:25.3] GS: Yeah, it's fascinating, because that's one of the places where software design runs into physical realities. Part of it is just speed of light. You can only send events that fast between China and the U.S. Like no matter what you use, if it's Kafka or PGNs or whatever, you can only go that fast. The other part is there're also expenses, like especially if you are hosted on AWS or something similar, suddenly sending events between regions becomes really expensive. So the things that you really don't want to happen in your architecture is that you send any event more than absolutely strictly necessary, which is just once.

The last thing you want to do is to have one Kafka in the U.S. and then every time someone in China needs to know anything, he has to read the event from the Kafka in the U.S., and then obviously you have maybe 20 million players in China and all of them have to copy maybe the same five events from the U.S. and you're basically sending the data from the U.S. to China five times, 20 million times, where you could have just sent it five times and store it locally in China. So for this reason you absolutely need Kafka in every large geography, because otherwise you will be sending the same events over and over and over again.

[0:22:49.8] JM: Different Kafka instances you're saying.

[0:22:51.6] GS: Different Kafka clusters. So you'll have the Kafka cluster of China and the Kafka cluster — Maybe even if you have area that are highly populated and tons of players,

you'll have the Kafka cluster in San Francisco, L.A., New York and Oklahoma, and obviously everything that can be done locally on that cluster is better to just handle locally and never communicate with anywhere else in the world. Some use cases really lends themselves super well toward some data is handled locally and only some data is sent across the globe. So I'm thinking like connected cars examples. Like a lot of decisions can be about something like Lyft or Uber. They can be done within a small geographic area. Like you don't need to know what's going on in China in order to order a cab in New York.

So this geo locality, if you can get away with that, which I agree with you. In video games, you can't really. I mean, people in China would want to interact with people in New York, but if you can get away with that, that's a definitely a saver.

[0:23:59.0] JM: Yeah. Are you saying that there's really not a good way to do the Kafka cluster in Japan reconciling its data with the Kafka cluster in the U.S.?

[0:24:08.2] GS: It will probably be asynchronous. After the fact, I know that we'll have some future plans for trying to improve the geographic capabilities of Kafka, but right now it will be asynchronous, which means that for a lot of use cases, it's fantastic, but for gaming it's probably not going to be fast enough just based on you do something in China and then you send the event the U.S. It does reconciliation, and then you get something back. It may take more than few seconds, or at the very least few seconds. I don't know that's fast enough for videogames. It may be that you can work around that. I know that videogames do a lot of latency hiding just by making educated guesses and kind of fixing stuff behind a simulator. So that could actually work. But if something has to be more real-time, then you probably want to try it for more geographical locality.

[0:25:05.5] JM: Okay. What about the topic question? Like how should you arrange your topics?

[0:25:09.1] GS: [inaudible 0:25:09.5]. Yes. So a super important question and pattern in doing Kafka designs is the idea of topic hierarchies. You basically have, let's say, all the topics that belong to New York. The topic name would start with NYC, and then you have a subset of those topics that belong to players who are playing in this specific area or in this part of this specific

playgroup, and you give them — And that would be, say, nyc.groupa, and then you want to have some information regarding, again, loot. So you have nyc.groupa.loot. Then you also want to capture their chats. So you have nyc.groupa.chat.

You kind of build those logical hierarchies of topics, and the really cool thing that the Kafka consumers give you is that you can subscribe at basically any layer of these hierarchies. If you decide that you want to copy every event that ever happened from New York to China, you just put the consumer that reads every event at this nyc.tar in any of those topics and basically read them and then write them to identical topics in China.

You just want to get all the chat messages maybe because you're writing them to Elasticsearch and have some algorithms that knows how to search them for code of conduct violations. Then you can say, "I want to listen to chat topics." It would be start, because you don't care about any geography at all. [inaudible 0:26:52.7], because you want any group .chat. Now you go to all the chats. So you can really get really flexible if you build those topic hierarchies and you can have a lot of different applications that care about different subsets of topics all consume different topics from within the hierarchy.

[0:27:12.5] JM: So let's go a little bit further with the game example. So you've got this game where different people, the events and different geo-locations are being written to your Kafka queues, your Kafka clusters, and these events might be, for example, a user's geo-location in the virtual world. So maybe you have a virtual world where there's latitude, longitude coordinates for every user in the world, and those events are being periodically sent to the Kafka cluster in this big append only log of, let's say, maybe the topic is user coordinates or something like that. I think that's a realistic example.

So you just append this really long steam of events and maybe you have a retention policy of a week or a couple of days or something, but in any case, at any given time, the stream of events that has been written to on this Kafka cluster is essentially a very long array of user GPS coordinates or latitude/longitude coordinates and you can take any subset of that stream, that array of events, that append only log of events and you could take a subset of it and load it into a steam processing system, and the stream processing system will perform complex logic on those events, for example, doing aggregations or enrichment or other kinds of things. That is my

understanding of how to think about steam processing. Do you think that's a — Is that an accurate way of looking at it?

[0:28:57.1] GS: Yeah, but the really cool part is when you start really enriching the data. That's when it gets really awesome, because like coordinates are coordinates, but then if you say that, "Okay. These coordinates belong to a person's home and this is his workplace and this is his doctor and that's his pharmacy," and you suddenly start noticing that he's going to a lot of doctors who are really, really far away from his home and his workplace. You are like, "Huh? What is he doing?" That's actually super real-world example of how pharma companies are now — They pretty much legally have to detect an, I guess, abuse of prescription drugs.

So knowing about it in real-time, because people — You know where they — You have their home address, you have their work address and you know where they're going to pharmacies and doctors. That's an example of something that if you do it in — It used to be batches, like once every quarter, and now it can happen, like the moment someone goes into a pharmacy, it's like, "Hey, he doesn't live here. That's kind of suspicious. What's going on?"

[0:30:03.7] JM: Wow! It's good to know that not only advertising companies are tracking my every location. Now it's pharmaceutical companies as well.

[0:30:10.6] GS: Yeah, and insurance companies are — Basically, I think they are trying to incentivize you to give them — You know we all have those fitness devices, the FitBits and thing that counts your steps? A lot of them are trying to incentivize their customers to hand over these datas, and this is huge? It's the location and it's also things like how many steps you walk a day. We know it maps very well with high statistical confidence to how healthy you are and all kinds of outcomes regarding your — How much doctor time and resources are you going to consume within your lifetime.

[0:30:47.9] JM: And lots of things that are totally unrelated to insurance that they would probably process and reuse maliciously.

[0:30:54.9] GS: Yeah. It's funny. I don't think it's malicious, but it's definitely commercial. Like whenever —

[0:30:58.9] JM: Commercial. Okay.

[0:31:01.1] GS: I spent a lot of time talking to companies about what they do with their data, and they all have very good aspirations and a lot of, like, "We can use it to find malfunctions faster," and a lot of these things. But the first use case that always like — Well, what's the easiest thing we can delivery? Oh! Advertisements.

[0:31:20.2] JM: Yeah, definitely. Okay. So as we're talking about the enrichment process. So this is something we've covered on some previous shows. A common pattern is if you want to do enrichment. If you got a bunch of latitude/longitude coordinates, maybe you want to reprocess those things and rewrite to the same Kafka cluster the locations, because you can take a latitude/longitude and you can turn that into a location. Maybe this person — Like maybe you want to take the lat/long and — If we're talking about the real-world instead of a virtual world, you could query Yelp real quick and find out what is the business or what is the pharmacy, in your example, what's the pharmacy that's closest to that lat/long? Or is that person at the pharmacy? Can you detect that? Maybe that's your form of enrichment. That's a very basic query that can be made at scale in parallel by a stream processing. You got to choose between your different steam processing system, so I think that's what we'll get into next.

How do you choose between different — You've got Spark streaming, you've got Kafka streams, you've got Flink, you've got all these different stream processing systems that, in some sense, are doing the same thing. They grab a big chunk of data and they perform MapReduce-like operations on that chunk of that data. What are the tradeoffs between different streaming systems that can do that kind of operation?

[0:32:43.2] GS: Yeah, that's a very good question, because from the outside it seems like, "Oh! We have all those steam processors and are all the same." If you talk to anyone who is like in the business and not an engineer, is they will absolutely say, "Oh! It's all the same, just pick something, like maybe the ones that we lack the support contract [inaudible 0:33:01.9] the most." But if you try driving it to the detail, there are differences in the data models. There are differences in maybe supported programming languages. I mean, if you are a Python fanatic, then it kind of narrows your choice a tiny bit.

Then, also in terms of the actual data model and capabilities, they really differ. Even though at the end of the day altering the machines, you can do everything with anything, and they abstractions they provide are quite different and it can make life easier or harder based on what you need to solve and what the language provides.

For example, reprocessing and especially late arrival of data is a huge deal. We talked about those GPS coordinates maybe collected from your fitness device, but if you go offline or the device loses batter and it didn't really send all the data yet, you will get — Next time the person remembers to charge it and connect it to the Wi-Fi, maybe three days later you suddenly get a bunch of events that are three-days-old.

Is it something that you'll need to handle as a special case somehow and figure out how to do them or is it something that your framework just handles? Some [inaudible 0:34:18.8] definitely just handle and some don't.

Similar, one of the harder problems to do is to join two streams. Super popular when you do web searches and you want to do attributions. So what was clicked as a result of which search. You need to basically join two really large steams of events, searches and clicks, or there is a lot of — These enrichment use cases are basically doing really large distributed joints.

Again, you can do it in any stream processing language. I've definitely [inaudible 0:34:54.1] any, but some of them have better abstractions. Like do they make you reach out to different services every time you need to find something out, or do you need to manage your caches, or do they handle a lot of those details behind the scene for you and just give you this one big abstraction?

[0:35:12.1] JM: Another thing that comes to mind is — So something that confused me for the longest time was the fact that when you're taking about a stream processing system, you are processing chunks of this big stream that is often in Kafka, and correct me if I'm wrong, typically it's not like every event that is published to this Kafka queue is instantly pulled into the stream processing system and processed, like you've got a net that's over this fast moving stream and it's just instantly processed as soon as this event comes in. Oftentimes, you've got events that

are coming in in such a rapid pace that you do want to take batches. Like maybe you want to take a batch every two minutes or something and you process it in a streamed processing system, but it really varies. Maybe you would have some topic where events should be published very rarely, like some sort of exception topic, like it captures all the exceptions, and then every time an exception comes in, you would want an stream processing unit to spin up instantly and process it instantly. In that sense, like there's no batch. It's just a stream that's instantly processed. But for many use case, like a high-volume use case, like latitude/longitude events, you would want to wait a little bit and process these chunks. So it is kind of a batch like system. Is that accurate?

[0:36:40.2] GS: There's definitely use cases for a lot of things that are between the batch and they immediately react to every event, and you are right that people definitely want this flexibility. Usually it's not as hard as fast as two minutes. It's ore when there is enough data accumulated or something along those lines. I think this flexibility is important, but it's important to think about it like what drives it. Is it about efficiency? Because in this case you just run benchmarks and kind of decide what you do, or does it actually give value to users that you are processing in those batches?

I think that if you think about it that, "Hey, I need it for performance." A lot of people, what they believe give them — We have basically zero intuition about what makes computers fast and slow. You test things and they're like, "I thought it will spend all my time being bottlenecked on this resource. It's actually the other thing."

So I've definitely seen a lot of times where people are like, "Oh, no! We absolutely have to do it in batches," and I'm like, "Well, I think that you'll see some natural batches just because you pick the data it's accumulated, you process it. By the time you come back to get more data, there'll only be a bunch of data and you don't have to just stop the world and wait for two minutes in order to be efficient. You just process the data that's there and come back to it, and that's what really defines it as streamed processing. The fact that we operate on the data continuously as it arrives and not wait for those fixed time slots. If you do that, I think you'll find that you can naturally have a very efficient process. You don't have to tradeoff your throughput for any kind of real-time streaming experience.

[0:38:33.8] JM: Okay. Now that we've outlined a little bit about what streaming is and how you want to use streaming, let's contrast to stream processing systems. Spark streaming and Kafka streams. Kafka streams, I did a show with Jay Kreps about that, so I'm sure anybody who's curious about that can go check it out. If you can't find it, you can download the Software Engineering Daily app for iOS. A quick plug there. It has all our old episodes.

What I recall from that episode was it sounded like Kafka streams was very useful for this kind of enrichment, where you want very quick processing that's close to Kafka itself. I guess you can do the processing on the same machines that Kafka is one. Correct me if I'm wrong here, but maybe you could just help me understand, because like where a Spark streaming, maybe you would want to do more for tons and tons of records. Like if you want massive parallelism across a ton of records and it's really complex processing, maybe you've got a multi-stage pipeline that you want to all process at once. That's more of a Java Spark streaming. Can you help me like understand how Spark streaming would compare to Kafka streaming?

[0:39:44.0] GS: Yes. So I think the biggest difference would not actually not being scale, because we are definitely running Kafka streams that [inaudible 0:39:53.6] massive scale. I can send later — I don't know how you share your [inaudible 0:39:58.8] with your listeners, but we have talks in Kafka Summit that people share their rather massive Kafka streams deployment.

Really, the biggest difference is how you manage Kafka streams applications versus Spark streaming. So Spark streaming typically runs on this shared cluster. You deploy a big Spark cluster and people run Spark jobs and runs them on to cluster and the Spark cluster is responsible for sharing resources between them and scheduling all those jobs, and basically the engineers have to work with the Spark deployment model, like you give it a jar JAR and you do Spark submit and that kind of how it runs.

The Kafka streams model, it actually — It doesn't have to run on the broker, but the idea is that you write an application, like that same way if you'd write a Java application that has a web server in it and you wouldn't look for a framework on which to deploy it. You wrote this application and you may put it in a container and if you're really modern and you'll say, "I want to run it in Kubernetes," and you'll say Kubernetes will manage my resources. If you don't do it in Kubernetes, it will probably have [inaudible 0:41:10.0] and will manage my resources.

But you basically build it as a normal standalone application. Really, kind of the secret sauce of Kafka streams is that Kafka itself includes a coordination protocol. You basically deploy a Kafka streams application, and if you want more throughput to deploy another instance of this application and you can really go as many of those as you'd like, and all of the know about each other. The coordinates for the secret Kafka coordination protocol, and if one of those application instance squashes, its workload will be automatically handled by another Kafka streams, which makes it incredibly elastic, and we have people who used Amazon's elastic deployment model, which I totally blank out on its name, and we have people running it on Kubernetes.

We're actually working on a blog post. We have KSQL, which is another layer above Kafka streams. Kafka streams is only Java, but SQL is really the most common language in the world. So we have KSQL, which is Kafka streams written in SQL language, and we're working on a blog post on how to deploy and scale it using Kubernetes, because these days, why manage resources on your own and we definitely don't want to reinvent that particular wheel. It seems like one hell of a wheel to reinvent. We're really trying to use the state of the art that is already out there.

[SPONSOR MESSAGE]

[0:42:46.6] JM: When you're building an application, you need it to be fast, secure and always evolving. With Kubernetes engine on Google Cloud Platform, developers can deploy fully managed containerized apps quickly and easily. Google has been running production workloads in containers for over 15 years. Google builds the best of what they learn into Kubernetes, which is the industry leading open-source container orchestrator.

Kubernetes engine combines automatic scaling, updates and reliable self-healing infrastructure with open-source flexibility to cut down development cycles and speed up time to market. To learn more about Kubernetes engine, visit g.co/getgke. That's g.co/getgke. G.co/getgke.

Thanks to Google Cloud for sponsoring Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:43:50.7] JM: Let's talk a little bit about event sourcing, because this is something I talked about with your college, Neha, who's the CTO of Confluent, and event sourcing is another important pattern to cover with Kafka. Maybe you could just describe what event sourcing is and maybe a recent example you've seen, like perhaps, an example, an industry of a customer that has implemented event sourcing that you thought what a good example.

[0:44:16.5] GS: So just to clarify, you're asking me to define event sourcing?

[0:44:20.2] JM: Yeah. I was hoping to get a definition and then maybe an example for how you've seen it implemented.

[0:44:25.4] GS: Okay. First of all, my background is in data engineering. I can give you an example, but I just feel like I'm maybe stepping into — I'm 100% sure there's a lot of semantic words out there that I'll totally be stepping into.

[0:44:39.5] JM: Oh, that's totally fine. I mean, I don't even know if I should be saying event sourcing or CQRS or some other pattern, like I don't care about any of that stuff.

[0:44:47.2] GS: [inaudible 0:44:47.2] that event sourcing and CQRS are totally different things.

[0:44:50.8] JM: Are they? I thought basically they were very closely related, like CQRS is kind of like how — It's like you do that, because you're implementing event sourcing. Like CQRS is you're separating the command and the query, and so event sourcing allows you to do that, because it publishes abstractions. Like you're publishing the abstraction of the event and you're consuming it elsewhere, and I thought those two are like very closely related. But I mean I could be totally wrong.

[0:45:20.2] GS: [inaudible 0:45:19.8]. I think there was like four or five those event-driven microservices patterns that are very clearly related. I think CQRS is definitely one of them. Event sourcing, the way I understood it is where you store events as an immutable stream of events for reprocessing. So if something happens later on and you need to go back and relook at the record, like, "Hey, we gave the wrong insurance quote, and we need to figure out why."

You basically have this really long, highly ordered record of things that you can go back and kind of like, “Okay, what was the first event we had that is related to this law suit and what was the event that happened after that?”

[0:46:05.3] JM: Okay. Event sourcing is more about maybe the fact that you have this durable record of tons of events that you could reprocess in the future.

[0:46:13.1] GS: What was my understanding. Again, I do not claim tons of expertise in that [inaudible 0:46:17.6], but that was the way I saw it.

[0:46:19.7] JM: Okay. So maybe we’re talking about two different things. So the thing I talked about with Neha, for example — I’ll just bring some more clarification to this conversation. So the thing I talked about with Neha was the example of what happened at LinkedIn and sort of what was the instahaul inspiration for Kafka, was the fact that you’ve got the event of somebody changing their profile picture. When they change their profile picture, that profile update needs to propagate to the user profile database and also to the search index, because a search engine is backed by an index and you have to search through that index. So you’ve got to update that index as well. The fact that there was a write that had to propagate to two bases, that was like basically the base case of this really common problem where you have to share these kinds of changes to the system with multiple consumers.

I think that’s a little bit different than the use case for Kafka that we’ve been talking about with, for example, this enrichment. I guess we talked about it a little bit with the data sharing of different game servers. But just to drive home perhaps this idea of using Kafka as a way to propagate an event to multiple consumers. Since I know you work with a lot of different customers, do you have another example for a pattern that you’ve seen for somebody that is using Kafka for this kind of use case?

[0:47:41.5] GS: Yeah, definitely. So I think probably the favorite thing is when people just — A lot of times, they look at just application logs. Like because it’s kind of low risk and very much within the control of an application team or infrastructure team and doesn’t really go to the business, you can just say, “Okay. I’ll use whatever I want to process my application logs, and then Kafka seems like the thing I want.”

Then when you look at it, you want to do multiple things with it. You want the data to go to some kind of Kibana, Elasticsearch kind of thing, so you can search root later on. You also want to trigger alerts if something happens, like has to be immediate. And then you may also have some kind of an organization-wide database at any specific types of events also have to go over there and you take those type of events out of your application logs even though it's not really, because application logs contain more than just errors. It contains clicks. It contains all kinds of datas that will be relevant to other teams.

So you end up having this one stream of events, just my application log, and you dump some of it into Elastic and you have a microservices that does, "Hey, whenever something is severe enough, I'll send an alert," and you have something else that basically send a notification to yet another system saying that, "Hey, someone just expressed interest in this new part of the website and the product team wanted to know."

[0:49:12.4] JM: Yeah, perfect. So how can we fit into this conversation more of a discussion of how people use Kafka streams? Because I think there still is a lot of confusion around like, "Okay. I've got this Kafka pub/sub system. Maybe I'm setting up microservices to listen for publications of events." But maybe there's still confusion, at least on my end, around when you use stream processing pretty much. I know I'm revisiting something we already discussed, but I just wanted to go a little bit deeper to understand kind of what Kafka is building — Sorry, what Confluent is building, because you're building a full kind of management system for streaming on top of Kafka. Maybe you just give me a little bit more of an explanation for what you're building.

[0:49:58.4] GS: Yeah. Basically, in order to do any kind of stream processing or event-driven processing, you need a way to, first, store events, which is what Kafka originally provided. You write events. You can read events and it gives you a more or less long-term storage for those events. Then on top of that, in order to have better abstractions for doing analysis on this data or even for building some of those services, you want to provide something like Kafka streams in order to give you kind of high-level abstraction, and obviously Kafka KSQL on top of it to do even more high-level processing.

[0:50:39.5] JM: Sorry. KSQL, that lets you treat your event stream like you would treat a MySQL database, for example.

[0:50:47.0] GS: I wouldn't do that. I mean, it's probably technically possible, but I think of it the other way around. If you want to do stream processing, if you have this stream of events and you want to either build applications that analysis it, or transform this stream into a stream that is enriched, for example, you could do it in Java and you can do it in Java if you have Kafka streams or if you have Flink or Spark, but it's very likely that not everyone in the organization will know Java. But a lot of people know SQL. So basically do the exact same thing. Do an enrichment or do a transformation or do some kind of analysis, windowed average kind of thing. just select from this stream where the key is something and grouped by a five-minute window kind of thing.

[0:51:41.6] JM: This would be a standing query on top of your Kafka cluster?

[0:51:45.4] GS: Yeah, that's it. It will just be — One of our engineers call it, you query the future, because you basically deploy this query and it just runs, and it's generated [inaudible 0:51:56.2] continuously. So if you look at the output stream in 10 minutes, you'll see 10 minutes worth of transformed stream.

[0:52:06.7] JM: And how is that working under the hood? Is that just read from — Does it just subscribe to all the relevant streams that are necessary to process that query?

[0:52:15.1] GS: Yeah, pretty much. It works more or less like Kafka stream would. It's basically a consumer, producer and a bunch of logic on top. The way it's actually implemented is you get SQL, you parse it, you generate an execution plan, and then you, on-the-fly, you implement this execution plan with Kafka streams operators. Like it says, "Join," so we'll use Kafka streams join. It says, "Group buys." So it means we need group buy-ins in a reduced function, and you basically just does all of these under the hood for you. So you write two lines of SQL and it generates maybe five pages worth of Kafka streams and runs it.

[0:52:56.5] JM: When we're talking about trying to put SQL queries on top of Kafka streams, does it become more important to think about the schema of our events so that we can enforce

that these — Because when you think about SQL, you want to have some schema imposed upon the data that you're querying.

[0:53:17.3] GS: Yeah. I feel schema is always really important. I mean not just when you're querying using SQL. Whenever you access data, you need to know something about the data types you're accessing. You need to know that, "Okay. The key is stringed," and then they have a customer [inaudible 0:53:34.8], that would be an integer, and we talked about the geo-location. We need to know that there are two longs, and one of them represent longitude and the other latitude, and they come in this order. This is super important, right? Like otherwise you're making guesses and someone can make a change and your entire thing suddenly goes up in flames.

[0:53:53.6] JM: So can you go a little bit deeper on the streams platform? Like what else you're building.

[0:53:57.9] GS: Yeah, of course. We talked about how important storage is, and then we talked about that you kind of build processing on top of it, and then we feel like we basically go to customers and say, "What prevents you from doing more stream processing or doing more event-driven microservices?" Sometimes it's like, "Well, we don't have the data." All our data is actually looked up in the main frame.

So being able to integrate this platform is the places where the data already is and the places where data has to end up, say, from our main frame, to Elastic, and to some stream processing system. So we built all those connectors to allow you to do this integration. Basically even have those streams of events in order to process them.

Then, as we're talking about, we just work to see what other problems people run into, and the next problem is that you have five different consumers from a stream of events and someone who produces the streams makes the change to the data they are writing and suddenly you have five applications that are no longer working, because they cannot really the change, which is where schema management really comes in and being able to detect very early on, "Is this chance compatible?" If so, you can put it in. This change is not compatible. If you put it in, it will

break those downstream applications. Be a responsible engineer. Don't break other people's applications. Go and fix your bug right now, and you want to catch it really early in the process.

Then in order to take thing to production, obviously you need to have monitoring. So everything I talk about until now is open-source. The monitoring and manager system is proprietary for Confluent, but we felt it's super important to provide something that knows how to monitor streams of events rather than just systems. So end-to-end, data availability and latency the streams themselves.

[0:55:59.9] JM: Yeah.

[0:56:00.3] GS: That work with any stream processing. It's not specifically tied to Kafka streams.

[0:56:04.4] JM: Okay. Not to open a can of works here, because I know we're running out of time, but you mentioned the connectors, so the Kafka connectors which allow you to shuttle data from one data source to another, and we did a show about the New York Times, and I'm sure you saw this story where they moved — I guess they basically wanted to make all their data accessible through Kafka. Like they pulled all of their old data into Kafka, if I remember correctly.

When I was talking about the LinkedIn example, that was an example where you would want to use Kafka as a place where data goes from Kafka into all these data resources. Like you would write from Kafka essentially to Elasticsearch and to the profile's database. Maybe it's a big Mongo cluster. But with the New York Times example, we're kind of talking about pulling data from a different data source into Kafka. Is that something that's common? Do people want to pull data from all data sources into Kafka, and why would they want to do that?

[0:57:05.9] GS: Oh, absolutely. All the time. Basically, you want to do it, because you want your project to be successful, and all data source already has all the data. So you have some kind of data analysis project and you know that you'll need new data, but going to all those projects that's already writing the data and telling them, "Oh, instead of writing it to this database, now you have to start writing it to Kafka." In many organizations, that's just an impossible battle,

right? You haven't proven the value of what you're doing it yet. Nobody knows if you're going to be successful. So I'm not going to do anything for you.

So in order to kind of jumpstart the process and have a successful project, you go to where the data already is. You put it into Kafka and you do whatever new streams analysis you're going to do and you show results. The nice thing about a lot of connectors is that they can pull all data, but they also do what's called change capture, and they continue capturing things as they go into the database. So every time someone makes an update or an insert to a database, there is an event in Kafka and my new project, streams project gets it, so I can continue doing ongoing analysis and at some point after I have a very successful project, I can go to the people who write data to the database and say, "Hey, why are you even bothering writing to this old database when you could be writing to Kafka?"

[0:58:25.0] JM: Well, that makes sense. Okay. I think we're almost out of time. What are the other projects in the Kafka ecosystem that you're excited about? What else is going on?

[0:58:33.9] GS: Yeah. That's the point where I don't know how much I should be sharing about — It's all open-source, so I guess it's all good. Basically, we see a lot of different companies in the ecosystem doing a lot to make Kafka more efficient in the cloud, and that's super exciting. So you see LinkedIn, for example. They got acquired by Microsoft. They're running on Azure, and suddenly it's like, "Why do we have to store every events three times when we could — We actually have replication given to us by the Azure services." So it's already been — We're basically storing it nine times, because we store it three times and they store it three times.

So can we make some improvements for Kafka to know about the storage system and make more intelligent decisions about that? A [inaudible 0:59:21.2] can we optimize that part of the protocol to deal with the cost of running in the cloud? So we're seeing a lot of activity in that direction, and that's really cool.

From our side, obviously, I'm still super excited about the Kafka SQL. It's super new project. We only announced it in last May, I think, and we keep building and adding capabilities and integrating it with the different projects that we have and we really try to make it kind of a showcase for the entire platform and an easy place to get settled with Kafka. So you can

actually do show topics and get a list of topics so you don't have to learn a lot of different command lines to do all those interactions.

[1:00:01.9] JM: Okay. Well, this has been great, Gwen. I really appreciate you coming on. We really covered a lot of ground, and I always love talking to the Confluent people, because the Kafka — Everything that's coming out of that project is really changing how people manage their data and it's just moving pretty fast. So there's always something exciting to talk about.

[1:00:21.5] GS: Yeah, that was fun. Thanks for having me.

[END OF INTERVIEW]

[1:00:26.6] JM: If you enjoy Software Engineering Daily, consider becoming a paid subscriber. Subscribers get access to premium episodes as well as ad free content. The premium episodes will be released roughly once a month and they'll be similar to our recent episode, The Gravity of Kubernetes. If you like that format, subscribe to hear more in the future. We've also taken all 650+ of our episodes. We've copied them and removed the ads, so paid subscribers will not hear advertisements if they listen to the podcast using the Software Engineering Daily app.

To support the show through your subscription, go to softwaredaily.com and click subscribe. Softwaredaily.com is our new platform that the community has been building in the open. We'd love for you to check it out even if you're not subscribing, but if you are subscribing, you can also listen to premium content and the ad free episodes on the website if you're a paid subscriber. So you can use softwaredaily.com for that.

Whether you pay to subscribe or not, just by listening to the show, you are supporting us. You could also tweet about us or write about us on Facebook or something. That's also additional ways to support us. Again, the minimal amount of supporting us by just listening is just fine with us.

Thank you.

[END]