EPISODE 370

[INTRODUCTION]

[0:00:00.3] JM: Apache Kafka is an open source distributed streaming platform. Kafka was originally developed at LinkedIn and the creators of the project eventually left LinkedIn and started Confluent; a company that was building a streaming platform based on Kafka. Kafka was very popular but it's not easy to deploy and operationalize, that's why confluent has built a Kafka as a service product so that managing Kafka is not the job of an on-call dev-ops engineer.

Neha Narkhede is the CTO of Confluent and she's been on the show twice before to discuss Kafka. In our last interview we discussed event sourcing and CQRS with Kafka. In this episode we explore some more common enterprise uses for Kafka, and Neha talks about the engineering complexities of building a managed Kafka as a service product.

This was an interesting episode because it started to give insight to how Confluent is building a business which is not something that we discussed in previous episodes. It was mostly discussing just the engineering and opportunities that people can use when they're building things on top of Kafka, so it's quite cool to see Confluent evolving as a company and releasing a product that will no doubt make it an extremely success business.

[SPONSOR MESSAGE]

[0:01:29.9] JM: To build the kind of things developers want to build today, they need better tools, super tools, like a database that will grow as your business grows and is easy to manage. That’s why Amazon Web Services built Amazon Aurora; a relational database engine that's compatible with MySQL or PostgreSQL and provides up to five times the performance of standard MySQL on the same hardware.

Amazon Aurora from AWS can scale up to millions of transactions per minute, automatically grow your storage up to 64 TB if need be and replicate six copies of your data to three different availability zones. Amazon Aurora tolerates failures and even automatically fixes them and

continually backs up your data to Amazon S3, and Amazon RDS fully manages it all so you don't have to.

If you're already using Amazon RDS for MySQL, you can migrate to Amazon Aurora with just a few clicks. What you're getting here is up to five times better performance than MySQL with the security, availability and reliability of the commercial database all at a 10th of the cost, no upfront charges, no commitments and you only pay for what you use.

Check out Aurora.AWS and start imagining what you can build with Amazon Aurora from AWS. That's Aurora.AWS.

[INTERVIEW]

[0:03:08.7] JM: Neha Narkhede is the CTO of Confluent, she is returning to Software Engineering Daily for her third episode. Neha, welcome back to Software Engineering Daily.

[0:03:18.9] NN: Thank you. Thank you for having me.

[0:03:21.6] JM: It's always great to talk to you about Kafka because it's a really popular project. It's really a popular product that people use, and I always like to do a bit of a refresher on what — When we have these technical product discussions, what is it? What are we discussing?

Let's ease into the topic that we're going to be discussing today, which is going to be Kafka as a service and what you're building at Confluent and just talk a little bit about why Kafka is useful, and let's start with messaging. Explain what pub/sub messaging is.

[0:04:00.9] NN: For the cool kids, pub/sub messaging is basically like Slack for all your company's data and applications, where a channel is a topic, users send messages to Slack channels much like applications would send messages to topics in pub/sub messaging. A more technical explanation is that pub/sub messaging is a messaging pattern where a set of publishers, they send messages to topics instead of sending data directly to intended receivers. What the receivers do is they subscribe to topics and get access to data.

There is this fundamental decoupling that is introduced by a pub/sub model where things that publish data and things consume data, they don't know about each other, and hence, are fully isolated from changes on either side.

[0:04:51.2] JM: Most companies don't start out with this message queuing pub/sub infrastructure, this different channels of doing multicast messaging. Is there some — Most companies, they start out with web app or with some sort of internal service that they're offering, or external service they're offering. What is the point at which a company — Let's just talk about a typical web app company. What is the point where they get to where they need some sort of messaging system?

[0:05:23.3] NN: I think messaging queues have been around for a really long time and the primary use case is point-to-point communication between applications. The moment you have a handful of applications, there's going to be this need that arises where these applications need to share data in an ongoing fashion. Typically that's where messaging queues come in.

Now, they're useful but there are several downsides to using message queues that way. Basically, the same data needs to go to multiple applications, then the sender application is forced to send it several times. One have to tie them with every MQ that it has between those apps. What I've seen is, like overtime, this leads to basically N-squared connections where every application ends up talking to several others which really doesn't scale anymore. It is unmanageable and lossy. Beyond a handful of applications, message queues come with a lot of overhead. That is basically why Kafka was born.

[0:06:30.1] JM: The type of issue that you're describing is, for example, if I'm running an e-commerce company and a customer purchases something, you want that purchased, the purchasing service to communicate with several different services, like the logistics service, and the accounting service. You need to send this message to multiple areas of the application. You just said that Kafka became particularly useful for that type of use case. Why was Kafka different from the other messaging systems that came before it?

[0:07:11.7] NN: Kafka overcame a lot of fundamental drawbacks of message queues and also presented a ton of strengths that was needed to excel in a world of different applications that

talk to each other and in a world of distributed systems where you have lots and lots of those versus just one database and a one warehouse.

Kafka is really different from a lot of messaging systems in a number of different ways. It is built from the ground up as a distributed system, so it is horizontally scalable and highly available. What's different about Kafka is that it is a cross between a file system, a messaging system and a database. It is really powerful in the ways it can be put to use that go well beyond what to point-to-point message queue can do. It can store several terabytes of history. It can scale to several hundreds of megabytes of messages per second, support low latency of communication in the middle of all that.

Looking for a new term to describe these capabilities, and what we came up with is that Kafka is a streaming platform in the sense that it can do messaging, it can do stream processing and it can help you build the streaming data pipelines. At the heart of it, what it does is it greatly simplifies the application architecture because your N- squared connections go away. All that devolves to is applications tap into a central event bus, like Kafka , and then type into it to get data. That simplification is at the heart of Kafka's popularity.

[0:08:49.9] JM: In the world where we're doing these point-to-point communications, if I had a micro services architecture and, like I said, there is an order that comes in and the purchasing service has to communicate with the logistics service, and the payment service, and the accounting service, all these other services. In the world where we're not using a message bus, like Kafka, I would have to say, "Okay, I'm going to send one HTTP request to service-X, one HTTP request service-Y, one HTTP request service-Z," and so you just get all of these different requests. Explain how that contrasts with what we're doing with Kafka and why — Just explain why it's more efficient, and I guess you could describe more broadly how a company with a microservices architecture might use Kafka and why that's useful.

[0:09:43.6] NN: Yeah, I think that's a great question because there are so many parallels between the data integration problem that led to the birth of Kafka, which is that a lot of applications and systems need to share data and that leads to these N-squared connections. We are seeing the same thing played again in the adoption off microservices because, basically, the same problem when it comes to your applications that they need to send data and your user

sign-up service needs to talk to your purchasing service and accounting and billing service, and to me it's the same N-squared connections problem replayed again.

The role that Kafka plays or the efficiencies is one is that you can only send once and consume many times. That's useful because, exactly the example you gave, that one event happens in your company and a lot of downstream applications need to respond to it and do something. Now the “do something” part might change, but the respond to that event bar doesn't change. In a response to user sign-up, you might need to send an email, or in the response to user sign-up you might need to put it in a newsfeed of other applications and other users and so on.

One of the advantages is that it’s super-efficient in the sense that you send it once, Kafka records it as an event and in an ordered log and then it automatically becomes available to any application that needs to subscribe to it.

The other efficiency is that it leads to a forward compatible application architecture. Let me explain that a little bit because I think it's really important. As applications evolve, you sometimes, or at many times, you cannot predict what are the future applications that are going to need access to the same data. A great example is you might have a web app today that might send a certain event that is needed downstream and tomorrow you actually don't know that you might need to build a mobile app, or in the future you might need to expose different external APIs.

What will happen with Kafka is that it allows the publishers and subscribers to be fully decoupled from each other, all the downstream services, they don't need to know that, “Oh! Now, there’s a mobile app, and now the same data is coming from three different places.” All it knows is that it's happily subscribing from Kafka. This decoupling that's introduced and in this forward compatibility that it enables where future applications — Essentially, you have the of zero cost to enable future applications. That's really the key benefit mapping in.

[0:12:24.0] JM: When the consumers of a Kafka channel are pulling messages off of the queue, does that pulling process often happen — Is it one by one or is a consumer pulling off chunks of messages and caching them locally and then processing them one by one? If we’re talking about like purchases coming in, if you had a high throughput of messages coming in or maybe if

we're talking about IoT, that might be an easier explanation, because IoT, you think of this super high volumes of data and different services are going take different lengths of time to process different events. Maybe you may have this different pace at which different services can process those messages. What is the typical pattern of consumption of consuming messages?

[0:13:22.9] NN: That's a good question. I think of Kafka works really well in both of those cases. One case where you have a really high throughput of events coming in and the second case where you have these trickle events coming in. In both cases, you want to be efficient in the sense that you want to get a balance of throughput and latency.

What Kafka does underneath the covers is at the API level you can consume in batches or one at a time. Underneath the covers, there are lots of efficiencies applied at the Kafka layer. Kafka pulls messages in batches at a time and it cashes locally as you would imagine. It also has a long pole API. When you have these trickle events coming in, you can say, "Just wait at the server-side until an event comes by or until a timeout passes." It handles the trickle case really well as well.

[0:14:17.2] JM: How do people manage the different ways of integrating between the Kafka queue and the consumer application? If I've got a mobile application and a database and all these different types of consumers that are pulling data off of Kafka, is there a standard way that they are integrating with that message bus?

[0:14:42.4] NN: Yeah. The standard way is the open source Kafka protocol. The protocol is implemented by a number of different libraries. There is the consumer library that's available in not just Java, but if you use it through a complement platform, then in Python and in .NET and Go and so on, then there's a REST API that's useful for all the other HTTP services.

Furthermore, there is the connect API which really encapsulates a lot of the hard parts of source and destination connections between Kafka. Depending on what application you're building, whether it's a data pipeline or simply a consumer application, you might choose to use any of these APIs that all implement the Kafka protocol underneath the covers.

[0:15:45.0] JM: [SPONSOR MESSAGE]

Hosting this podcast is my full-time job, but I love to build software. I'm constantly writing down ideas for products; the user experience designs, the software architecture, and even the pricing models. Of course, someone needs to write the actual code for these products that I think about. For building and scaling my software products I use Toptal.

Toptal is the best place to find reasonably priced, extremely talented software engineers to build your projects from scratch or to skill your workforce. Get started today and receive a free pair of Apple AirPods after your first 20 hours of work by signing up at toptal.com/sedaily. There is none of the frustration of interviewing freelancers who aren't suitable for the job. 90% of Toptal clients hire the first expert that they're introduced to. The average time to getting matched with the top developer is less than 24 hours, so you can get your project started quickly. Go to toptal.com/sedaily to start working with an engineer who can build your project or who can add to the workforce of the company that you work at.

I've used Toptal to build three separate engineering projects and I genuinely love the product. Also, as a special offer to Software Engineering Daily listeners, Toptal will be sending out a pair of Apple AirPods to everyone who signs up through our link and engages in a minimum of 20 work hours. To check it out and start putting your ideas into action, go to toptal.com/sedaily.

If you're an engineer looking for freelance work, I also recommend Toptal. All of the developers I've worked with have been very happy with the platform. Thanks to Toptal for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:17:49.4] JM: Okay. We've motivated why people are using Kafka today. I want to gravitate towards the reason we're having this conversation, which it Kafka as a service. I think the way to get there is to discuss some of the issues that people don't want to deal with with Kafka, like these things that we've discussed where, "Okay, you get to decouple the producer of an event and the multitude of consumers of an event. That's great."

You might get new consumers of an event source that the event creator doesn't need to worry about, that's great, that you can just add a new mobile app that's consuming this data source and the creator of the data source doesn't even have to worry about it. That's fantastic.

Of course, what are the problems and the issues that people experience when they're deploying and managing Kafka that are less pleasant to deal with?

[0:18:51.1] NN: Some issues are related to operations. As you mentioned, deployment and management off the Kafka server and cluster itself, and other issues are around management of schemas and management of data pipelines. Separating those two issues and attacking the first one, which is that there is a certain amount of overhead that comes with deploying and managing any distributed system, and Kafka's is just one of them. You need skilled SREs who understand the nuances of a distributed system are able to deploy and operate the system for 24/7 availability. These skilled SREs that also have Kafka knowledge, they are really hard to find. Unsurprisingly, there is a huge industry-wide trend to move to managed services, to move away from the business of buying machines and hiring these expensive hard to find people to manage these services and instead focus on the application business logic.

There are some things that are unique to Kafka as well that you need to be skilled at in order to operate it 24/7. Some of those problems are classic sort of data systems problems where if there's some kind of traffic patterns that change that need you to load balance the data differently, then staying on top of detecting that issue and then running a set of tools to be able to automatically balance that data out, that's just one example of an issue that is unique to Kafka.

Coming back to the issues around development and management of the data itself, those are attacked through the — We experience this firsthand when we deployed and managed Kafka ourselves, is there's an issue around management of schemas, there's an issue around just having visibility into end-to-end monitoring for your data. What we've done is two-fold. For the management issue, we've just launched Confluent Cloud so that you can use the entire Kafka and Confluent platform as a service. For the latter, we continued to invest in product tools to ease the management of schemas and be able to provide a UI where you can view the end-to-end sort of SLA on your data itself beyond just Kafka.

[0:21:17.0] JM: Can you describe that term schema management more?

[0:21:20.4] NN: Yeah. For any data that is sent through Kafka there is some metadata that describes the data itself. Now, that metadata could be in JSON or it could be in Avril or Protobuf or any of these other serialization formats. Kafka itself is a schema-agnostic system. All it understands is bytes in and bytes out.

Ultimately, if you want to use it a companywide scale, you need to worry about changes to your metadata. What if an upstream publisher just deletes a field that some downstream consumers depended on, and so the moment you push an update to this publisher system or application, all these downstream consumer applications break. That's a classic schema management problem that companies face often times 10 steps down the line. When you have just two applications and the same developer managing those two applications, no one worries about this. The moment you have 10 or 20, you suddenly realize that this is a problem.

We built a thing called schema registry that allows a central registry of all schemas that allows applications to detect whether you're introducing incompatible changes, and it's an optional thing that you can choose to use that's available to Confluent open source.

[0:22:47.3] JM: Will people put that somehow into a continuous integration pipeline or something, like check against the schema registry if I'm making an update to my database schema. What is the process — If I have downstream consumers and I'm writing some sort of service and I'm about to push a breaking change, when do I get notified by the schema registry? How do I get prevented from making that breaking change?

[0:23:18.8] NN: Yeah. What you do is there's a little wrapper on the client site that just sits with the Kafka client if you choose to use the schema registry, and then there's a server-side, which is just the registry. If you use this and when you try to push the change that introduces that perhaps back wasn't compatible change to your schema, what it does is the Kafka client tries to talk, tries to register that new scheme in the registry. Before it can send any data, it gets an act from the registry and then it cashes that. If their acknowledgment is like, "Oh no, this is a breaking change," then it won't be able to send those messages.

Essentially, it comes down to your philosophy off handing these changes is for years together the philosophy around ETL was that you just push any data in and then there's a lot of cleaning under downstream. What we said is with streaming pipelines, it's really hard to get any value you're your streaming pipeline. What's better to do is instead enable a clean data in, clean data out service that's what schema registry would allow you to do.

[0:24:25.7] JM: I take it, you're not a fan of versioning schemas.

[0:24:31.7] NN: This, in fact, versioning schemas underneath the covers, because what you want to also be able to do before you push the change is be able to go to this registry and just try out your new schema and know what the previous version was. A utility service on top of that is be able to — As a data owner, or the business owner, just be able to go in this registry and look what the changes were between two different versions. It does do versioning underneath the covers.

[0:25:01.9] JM: Help us understand the hosting model here and what we're getting out of like a Kafka as a service thing, because I think what a lot of people do these days is they have their own Kafka deployment that is running on AWS along with the rest of their infrastructure that's running on AWS or Google Cloud or whatever it is, mostly AWS. What would we get out of the Kafka as a service, Confluent cloud? What is the business offering?

[0:25:38.5] NN: The business offering is that Confluent cloud is a fully managed streaming data service that offers Kafka as a service today and in the future will offer the entire Confluent platform with schema registry and the REST proxy and lots of connectors. That way it offers that is essentially just like any other managed data service where what you do as a user is specify what you need in terms of compute storage availability and just depend on Confluent cloud to then run your Kafka server or your schema registry fully managed manner.

[0:26:21.8] JM: Is it still running on servers that the buyer purchases from AWS?

[0:26:30.5] NN: Not. It is a fully managed service in the sense that we take care of provisioning or what's underneath the covers on the cloud provider they're [inaudible 0:26:38.4]. In some

sense it's just like any other multi-tenant hosted fully managed service that a cloud provider might offer except that it is cloud provide agnostic and it will support all the open source Kafka APIs and the open source Kafka protocol.

[0:26:56.2] JM: If I'm developing on a cloud provider like AWS or Google. Let's say i just have the web app. I' in that early web app stage and I'm getting to the point where I want to have some sort of message queuing infrastructure, and I can go with one of these managed messaging products from AWS or Google, why would I choose Confluent cloud in contrast to one of those services?

[0:27:25.4] NN: One is that you're probably looking for Kafka, so Kafka is one of the most popular streaming data services out there. There's a large set of users that is just looking for Kafka, but Kafka as a service wasn't available really from Confluent, and so that was a gap. But there are several other advantages as well. One is that if you use Confluent cloud, you not only get Kafka as a service, but because it supports the open source Kafka protocol and APIs, you actually get access to the entire prolific open source ecosystem of Kafka. All the connect connectors, all the clients, all the tools that the community has developed around Kafka, you still have the optionality to pick and choose and use those tools along with Confluent cloud. We thought that that's a major advantage that we must offer a community with this service. That's one.

The other is the variety of tools that will be available from Confluent cloud around Kafka, be it schema registry or be it connectors to S3 DynamoDB and other services. That's the other. The final one is optionality of moving your application to another cloud provider if you choose to, because you will be programming your applications to open source Apache Kafka APIs. If you choose to move from one cloud provider to another, it really doesn't lock you in to the proprietary APIs if you choose to use one of the proprietary services.

[0:29:06.4] JM: Yeah. For a while I've thought about why AWS, they have Kinesis, which is sort of like a pub/sub thing, and Google has, I think, Google cloud pub/sub, which is a Google pub/sub thing. For a while I was wondering like, "Why don't these companies just focus on a Kafka offering?" I think it's probably because — Amazon had some internal thing and then they were like, "Let's just like make this available as a service," and Google probably said the same thing.

They were like, “Well, we know how to run this thing because it works internally. Why don’t we just offer that and, plus, the proprietary so maybe we get some lock-in advantages business wise.” Do you have a sense for how those products qualitatively differ from Kafka?

[0:29:56.2] NN: A; I think the Kinesis versus Kafka one is simpler because it's more of an apples to apples comparisons. Kinesis, the way we saw it evolved, it was very much influence from the initial Kafka paper, but there are several key differences. One is in just the scalability of the system. Kinesis throttles you at a very low, sort of five messages per second rate. Beyond that, you have to add shards and that increases the cost of the system, versus Kafka can scale from 0.1 MB per second all the way to a hundred megabytes per second if you want it and the cost is still comparable there.

There are several other features that are missing. In the last couple of years, Kafka has evolved from being just a pub/sub system to a full disputed streaming platform. Just like features like lock compaction, where you can keep data for an infinite amount of time. It compacts based on case or having access to connectors to all these different systems, or even being able to do stream processing on Kafka. A lot of these features are not available.

In the latest release of Kafka, we are enabling exactly one's guarantees end-to-end, and so that's going to be a major differentiator between any of the services and Confluent cloud and, hence, Apache Kafka.

[0:31:20.7] JM: That's an interesting point. I think it’s probably worth going into that exactly once guarantee discussion a little bit. Explained why exactly once is an important feature.

[0:31:31.0] NN: In any kind of distributed request response system, like Kafka or even otherwise, there is this inherent problem where if you send a data or you wait for an acknowledgment, that the acknowledgement could fail for multiple reasons, whether it’s a network failure or some other reason. If you're the publisher, you really don't know what happened. Perhaps the message was already written on the server side and the acknowledgement was lost, or maybe the message wasn't written at all.

As a publisher, what you can do is resend data, and that's the reality of any RPC. What happens when you resend is in some of these cases you end up sending the same message which gets written twice and a duplicate introduced.

This is relevant to a lot of Kafka applications, because Kafka is used in mission-critical applications where people depend on the absence of duplicates. For example, Kafka is used to process credit card payment payments. You definitely want that to be exactly once. Kafka is used in billing pipelines and you definitely want that to be exactly once. It's used in ad impression counting.

A lot of business-critical use cases need this guarantee that, “Hey, I just want to send my data, and I hope that it will be processed exactly once. This was the motivation behind putting in so much work in Kafka to make it exactly once. Prior to that, Kafka guaranteed in order delivery with at least once guarantees.

[0:33:06.5] JM: Why is that such a hard — I feel like we might have discuss this on a previous episode, but if we did, I forgot. Why is it such a hard distributed systems — I’ve have read some distributed systems papers and I know like every problem in distributed system is hard. Maybe you could talk about why exactly once processing in particular is so hard, and maybe some explanation of the algorithmic hoops you have to jump through.

[0:33:34.4] NN: Yeah. The example I gave, you can sort of consider a simple case with that example, which is just a single partition point-to-point problem. If you had a single partition and you had only one publisher and only one subscriber, that you still have to deal with this duplicate issue. But the solution for that particular problem is slightly simpler. It's just called idempotence where you detect or you keep track of the ID of the publisher and you have a sequence number for your messages. If you end retrying the message with the same sequence number, with the same producer ID, then you can easily detect duplicates and get rid of those.

Kafka is a sharded system, so it has multiple partitions and a several publishers that could write to multiple partitions and several subscribers that could consume from them. In a case where you have several partitions, then the guarantee that you're looking for devolve to, “Can I write a set of messages atomically across multiple partitions so that all the subscriber either see all of

those messages or they see none of those messages," and that guarantee is particularly hard to provide. You can imagine, you have to keep track off of the transactions. We call it transactions, but is not quite database transactions. What we're trying to provide is in-order atomic writes across partitions. That is significantly harder. It's much harder to even and explain how we solved it, but you can imagine that the easier problem itself is solve through just idempotence, but when multiple shards are involved, it becomes much harder. Essentially, you're doing distributed transactions. At the same time, keeping track of ordering.

This one step further in Kafka that builds on these two guarantees that I just described which is idempotence and atomic writes, to provide exactly once stream processing, and that's the real, I would say, killer feature that the community is looking for is they want to do these read process write kind of operations. That's essentially what stream processing is on top of Kafka; you read messages, you process them, and you write them back, and you want that reprocess write operation to be exactly once.

In order to provide that guaranty, we need to rely on these previous two guarantees, and we are happy to know that in the upcoming 011 release in June, that we have tackled all these three problems. Any stream processing application that's written using the streams API will be end-to-end exactly once. That's a really powerful guarantee.

[SPONSOR MESSAGE]

[0:36:36.7] JM: Spring is a season of growth and change. Have you been thinking you'd be happier at a new job? If you're dreaming about a new job and have been waiting for the right time to make a move, go to hire.com/sedaily today. Hired makes finding work enjoyable. Hired uses an algorithmic job-matching tool in combination with a talent advocate who will walk you through the process of finding a better job.

Maybe you want more flexible hours, or more money, or remote work? Maybe you work at Zillow, or Squarespace, or Postmates, or some of the other top technology companies that are desperately looking for engineers on Hired? You and your skills are in high demand. You listen to a software engineering podcast in your spare time, so you're clearly passionate about technology. Check out hired.com/sedaily to get a special offer for Software Engineering Daily

listeners. A $600 signing bonus from Hired when you find that great job that gives you the respect and the salary that you deserve as a talented engineer. I love Hired because it puts you in charge.

Go to hired.com/sedaily, and thanks to Hired for being a continued long-running sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:38:08.5] JM: When the Confluent team is trying to solve these really hard distributed systems problems, I'm really curious how this proceeds, because in academia, I think it's typically — You have like one or two people and they're just stay up really late, drinking a lot of coffee and thinking about this and writing it out maybe on just blank sheets of paper with really long screeds and then maybe occasionally get in a room together and they whiteboards some stuff and then eventually it comes out in the form of a long paper, and the paper is really descriptive at it uses really precise language.

Now, like these days, I think a lot of the disturbed systems breakthroughs are happening within companies in team formats. How does the communication around solving a really nitty-gritty problem like that occurred? Does it happen on Slack or does it happen in — Are you white-boarding? How do these these discussions get resolved?

[0:39:09.6] NN: That's such great question, because I've witnessed a really amazing small, not particularly research scientists or whatnot, a set of engineers solve this along with the community, which is really the key sort of win here is that we not only brainstorm this in a small team, but we did that successfully with the community, and I'm really proud of that.

It happens in stages. There are some engineers that kneel down sort of the guarantees and then think about given the current architecture of Kafka and these guarantees that we want to provide, what are the sets of solutions that are reliable and what are the trade-offs and which trade-offs are acceptable to our users and which trade-offs are not. I think the team tackle this through not only meticulous sort of product-oriented thinking around this problem coupled with constraints around architecture, but this team was particularly good at communication with the

community. We have this thing called Kafka Improvement Proposals that are the community writes for any big change in Kafka. I would really courage people who are listening to this podcast to go to read the [inaudible 0:40:26.5] for exactly one stream processing in Kafka and go look at the discussion. I think it is one to emulate and it's one to just follow. It's incredibly hard problem that was solved with a community of people. I'm so proud to work with these engineers here at Confluent.

[0:40:44.0] JM: Wow! Okay, yeah it sounds like kind of the level of detail and complexity that happens in, for example, like in the Linux kernel, like in those mailing lists when people get really emotional and they really go in the detail about like, “Here’s how you should implement some file system thing.” It sounds like you can have the same kind of conversations not necessarily around lines of code in the Linux kernel, but you’re having them around, “Here's how you solve this theoretical —It's a theoretical problem.” You're not looking at code, right? You're looking at like a theoretical issue.

[0:41:21.4] NN: Yeah. We looked at making a traditionally theoretical set of guarantees into a practical feature for a large community and an existing widely adopted system, and I think that's where the wind lies is this was made practically possible and that came through a lot of constructive communication and a lot of deep thinking.

[0:41:48.6] JM: As you get a lot of customer volume on Confluent cloud, you start to manage a lot of Kafka clusters. You’re the CTO of Confluent, so you’re going to be solving a lot of interesting problems around how you manage the — What happens when you're just managing tons and tons of Kafka clusters. You must be watching all of the Kubernetes developments quite closely because, I'm sure there's a lot of stuff around management that you can learn and leverage from the Kubernetes community. Can you talk some about the technical architecture of what you're doing to host and manage tons of Kafka clusters at scale?

[0:42:35.8] NN: Yeah, sure. It will be difficult to do just describing the architecture in great detail, but let me summarize the various building blocks. First off, our design philosophy for Confluent cloud was to build it from the grounds up to be cloud agnostic and we made that decision in the early days and that has influenced the design of every building block.

There are so many different parts to bringing this product together. I think there's a provisioning service that takes the throughput retention availability requirements from the user, disk capacity planning, provisions the Kafka resources and appropriately sized easy two instances, and there's a lot of sort of expertise around Kafka and understanding AWS in order to automate the provisioning service.

Then there is a bunch of monitoring and alerting services that collect data and metrics from various levels write from EC2, to the operating system, to JVM, to Kafka, so that we can monitor at these various levels to really automate and reduce the meantime to recovery.

Then there is a bunch where Kubernetes and Confluent cloud and sort of the parallels come in is it around the failure recovery and auto healing parts where we have to respond to — We've build these layers that respond to various failures and then rebalance data in Kafka as needed, be able to re-provision those instances under the constraints of what the user provided, whether they want within lazy replication or cross lazy replication in order to bring the system back to full operation in an automated fashion. We have we have a lot of focus on cloud agnostic behavior and automation around manual operation in order to scale our ability a startup to run this massively large Kafka installation.

[0:44:34.5] JM: To jump back to what a user, what a developer who wants to run their Kafka on Confluent cloud, what they want. To jump back to that, if I'm using the Confluent cloud service, I pick my throughput, my retention and my replication and you just mentioned that those knobs that the user tunes, those lead to ways that you're going to provision and configure the auto healing and the scalability of the underlying infrastructure. I'd kind of like to dive into that as an issue, but let's start with the surface level for what the user wants out of that. If I'm a user, I'm building some application, I pick these features; throughput, retention, and replication, what do I want from those different knobs?

[0:45:30.2] NN: What users want to not think about is how the capacity plan Kafka brokers, how to manage them, and how many you need and how large should they be. Instead, what they want to think about is how much compute do I need for my application, which is expressed in throughput in megabytes per second. How much storage I want for my application, which is expressed in retention, which in Kafka could be a day or 30 days or even more and what sort of

availability do I want to pay for? Whether it is within a single lazy replicator twice or across you know where AZs availability zones are datacenters in Amazon terminology, and whether I want that much availability for my service or not.

What we've done is allowed, sort of absorbed the user behavior and concluded that users want to express the needs of their application in parameters they understand. What they don't want to think about is how that translates into how many Kafka brokers you need, how large should they be, and how should that change over time?

In Confluent cloud, what we concluded is what you specify are those three parameters and you hit Go, and that's what we promised as Confluent cloud operators, as uptime availability and performance guarantees that you might expect, and we'll do the hard work off thinking about how the capacity plan. Believe it or not, I think of the most frequently asked question in the Kafka community is, "Well, how do I know how many brokers or how many partitions I need and with Confluent cloud that will entirely go away?" Not only that, but a lot of other problems that you have to overcome even after that.

[0:47:25.1] JM: Can you talk a little bit about the mapping between if I'm a user and I set some levels of throughput and retention replication, how does that translate to what you're doing at the auto scaling and the auto healing level of your infrastructure?

[0:47:42.3] NN: At some level, the intelligence lies in there is some golden sort of ratio that we've accumulated from observing tons and tons of deployments between how many partitions for broker work well and what kind of poor broker throughput works well for a certain instance size. A lot of these provisioning observations that we've you accumulated over time and we've taken those and then built the service out of it that figured out, given your requirements, how do we —Knowing the limits of a certain broker or a certain size cluster, how would we best spread out your data not only today, but note that in SAS services, users might start one place and then grow into another tier, and so what we have done is we've thought about as users grow, how would the quotas for your particular tenant be set so that, A; you don't overcome and overwhelm all the other users as well as you get the room you need to grow over time. Of course, the billing side of the thing where it keeps track of how much you've used and, hence, how much you will pay at the end of the month.

[0:48:58.0] JM: Yeah, that's the advantage of having been involved in Kafka since the earliest days is you've just seen so much. Picking how knobs of throughput retention replication map to actual lower-level configuration stuff, that's not something that you're going to find a proof for theoretically. That's more something that is going to come from experience.

[0:49:26.5] NN: That's exactly right.

[0:49:30.2] JM: The last conversation we had was about CQRS and event sourcing and why Kafka is useful for those sorts of things. In that episode, we talked about how if you have this event stream, you often have multiple databases that are going to be updated in response to that event stream. For example, if you have an event on your event queue, you might have your elastic search index that gets updated and then you have your MySQL database that gets updated. You have all these different data sources that get updated and something needs to handle the process of doing that update. I've had a couple of conversations recently with people who are building serverless frameworks and, basically, the idea is, "Oh, you can use these AWS Lambda services or some other function as a service to update the consumers that are consuming the event stream. Have you been doing any interesting work around the functions as a service stuff, or maybe you could talk some about what your opinions are for what's the best system of processing those events when you just have those lots of small event updates?

[0:50:47.0] NN: Yeah. Let me just sort of describe what was my understanding of function as a service. It's all about running some backend code without managing your own server systems or your own server applications. Often times, the way it is expressed is these functions are code snippets that are triggered by event types that might be defined by the provider.

In Amazon Lambda, these stimuli or triggers could include file updates in S3 or they could be scheduled tasks or messages that are added to a message bus, like Kinesis. Certainly, I think a system like Kafka and where the parallels are drawn here is that it stores much of the streaming data in a company. It makes sense as one of those stimuli or triggers where you can write your little functions where they're on in some kind of Lambda kind of service or a service that might be available on Confluent cloud, and you use Kafka as one of those sources for those events.

There are some limitations on such functions the way it is defined by Lambda. In the sense that it has to be stateless, there are certainly stateless computations but there are lots of stateful computations as well and it cannot take more than five minutes to run. While that is useful for some cases, it cannot do any kind of long-running stream processing or transformations.

A lot of the thinking and the work we are doing is around — It's twofold. One is what is the AWS Lambda equivalent that will allow you to run these stateless, short-lived transformations on top of Kafka and they're just — Providing Kafka as a source Lambda might be the easy answer. The harder part is what kind of service do people need to be able to do these long-running stateful steam processing kidn of transformations, like developers love to do using the Kafka's streams API or Spark or similar services.

When that support is available for doing that in Confluent cloud, I'm sure the entire Kafka base will be very happy as they will be able to do both stateless as well as stateful processing easily on commons sort of compute layer.

[0:53:11.1] JM: Okay. That API is not really there yet, or that functionality is not built into the Confluent cloud yet, but you're starting to think about how you might offer that to customer.

[0:53:23.1] NN: That's exactly right. I think that makes a ton of sense giving that Kafka is primarily used for stream processing. That will be a very natural set of things that you might expect from a hosted Kafka as a service product like Confluent cloud.

[0:53:38.5] JM: Have you seen any cutting-edge Kafka use cases lately that have surprised you? Like maybe companies that are doing heavy machine learning pipelines or something else?

[0:53:50.1] NN: The most cutting-edge or exciting use cases I've seeing are around IoT and around creating these intimate real-time customer experiences. For instance, major automotive company along with several others, they're investing in this connected car initiative where thousands of sensors will feed onboard processors on cars, cars would report data in a streaming fashion through Kafka allowing these companies to alert drivers to avoid accidents

and also to be able to do this real-time traffic routing across the entire country. I think that's pretty exciting.

Similarly, around the real-time customer experiences, cruise ships are using Kafka to create a digital real-time customer experience both shipboard as well as offshore. If we're booking activities, dinner reservations, applying gambling credits, it's very similar to the talk you might see at Kafka Summit New York, the keynote that talked about event-driven banks, where a lot of banks' commodities, they are getting commoditized actually. And a lot of the differentiation is around providing a more intimate, a more useful real-time customer experience.

I think that's a pretty big shift in sort of the applications of Kafka from just the low-level message messaging to sort of providing this high-level business value around customer experiences and competitive advantage.

[0:55:23.1] JM: Cool. Neha, it's been great talking to you. Do you have any closing thoughts on Kafka or Confluent cloud or what you're working on right now?

[0:55:35.0] NN: Closing thoughts are we're really looking forward to exactly once being available in the Kafka community. I'm really looking for Confluent cloud to be generally available. Right now, we've started on building early customers, and the way to sign up for that is go to confluent.io and signup.

I think the last thought I'd leave people with is Kafka Summit SF is coming up in in August and spaces are filling up quickly. We have an awesome agenda lined up and I hope to see the community there.

[0:56:05.6] JM: Cool. It it's really fun to watch Confluent thriving. I've been watching it for several years now and it's just exciting to see more infrastructure providers getting a lot of traction because I just think it's really helpful to the software engineering community as a whole when there are more and more providers. It's also just interesting to see you maybe three or four years ago, it would've looked like really niched, like, "Okay, is a company built around Kafka?" Now, we're seeing, "Oh, it's actually a huge area of innovation," and it makes complete sense to have a provider that is focused entirely on this technology.

[0:56:51.1] NN: Absolutely. I think I'm looking forward to seeing how it actually plays out in real businesses.

[0:56:57.6] JM: Okay. All right, Neha. It's been a pleasure.

[0:57:00.5] NN: It's been a pleasure too. Thank you.

[END OF EPISODE]

[0:57:04.8] JM: At Software Engineering Daily, we need to keep our metrics reliable. If a botnet started listening to all of our episodes and we had nothing to stop it, our statistics would be corrupted. We would have no way to know whether a listen came from a bot, or from a real user. That's why we use Encapsula to stop attackers and improve performance.

When a listener makes a request to play an episode of Software Engineering Daily, Encapsula checks that request before it reaches our servers and filters bot traffic preventing it from ever reaching us. Botnets and DDoS are not just a threat to podcasts. They can impact your application too. Encapsula can protect your API servers and your microservices from responding to unwanted requests.

To try Encapsula for yourself, got to encapsula.com/sedaily and get a month of Encapsula for free. Encapsula's API gives you control over the security and performance of your application. Whether you have a complex microservices architecture, or a WordPress site, like Software Engineering Daily.

Encapsula has a global network of over 30 data centers that optimize routing and cacher content. The same network of data centers that is filtering your content for attackers is operating as a CDN and speeding up your application.

To try Encapsula today, go to encapsula.com/sedaily and check it out. Thanks again Encapsula.

[END]