EPISODE 914

[INTRODUCTION]

[00:00:00] JM: A new software product usually starts with a single database. That database manages the tables for user accounts and basic transactions. When a product becomes popular, the database grows in size. There are more transactions and more users. A company grows around that product and the company starts to accumulate more data in different sources. There's analytic systems, time series databases and the logging tools, and all those tools start to generate data. Moving this data between different systems starts to become complicated. Apache Kafka is often used as a system for moving data between these different systems performing transactions and generating aggregations and summaries of these large quantities of data.

Robin Moffatt works Ct confluent, and he has written numerous articles about how to move data between systems and design effective workflows for data pipelines. Robin joins the show to talk about modern data platforms and databases and the patterns for using Apache Kafka to connect those systems to each other.

If you're interested in learning more about how companies use Kafka, the Kafka Summit in San Francisco is September 30th through October 1st. Companies like LinkedIn, and Uber, and Netflix will be talking about how they use Kafka.

Full disclosure; Confluent, which is the company where Robin works, is a sponsor of Software Engineering Daily.

[SPONSOR MESSAGE]

[00:01:36] JM: Cox Automotive is the technology company behind Kelly Blue Book, autotrader.com and many other car sales and information platforms. Cox Automotive transforms the way that the world buys, sells and owns cars. They have the data and the user base to understand the future of car purchasing and ownership.

Cox automotive is looking for software engineers, data engineers, scrum masters and a variety of other positions to help push the technology forward. If you want to innovate in the world of car buying, selling and ownership, check out cox autotech.com. That's C-O-X-A-U-T-O-T-E-C-H.com to find out more about career opportunities and what it's like working at Cox Automotive. Cox Automotive isn't a car company. They're a technology company that's transforming the automotive industry.

Thanks to Cox Automotive, and if you want to support the show and check out the job opportunities at Cox Automotive, go to coxautotech.com.

[INTERVIEW]

[00:02:56] JM: Robin Moffatt, welcome to Software Engineering Daily.

[00:02:57] RM: Thanks so much for having me.

[00:02:59] JM: We've done many shows on Kafka. We've also done many shows on databases. We have not done much coverage of the connection between the two. Why would I want to integrate a database with Apache Kafka?

[00:03:14] RM: That's a great question. I suppose most people have got data, have some databases. Whether, say, a legacy thing and then moving more to keeping their data in Kafka or in other systems, or simply because they're building applications where they got some data residing in a database. They want to bring that data together with data coming from other places, maybe some microservices they're writing. But almost always, there's a database in the picture somewhere, and a lot of time people want to bring that data into Kafka.

[00:03:40] JM: Give an example of why I might integrate a database with Kafka.

[00:03:45] RM: Sure. So databases have been around for ages. They'll been around for a long time, yet people use database to keep their data in. But other time, that data, they also use with the events that are flowing through Kafka. So that could be offloading data from a database through Kafka to an alternative data store. It could be using data you got in a database to use to enrich events that are flowing through Kafka. But almost always, there's database figures in the picture somewhere and you want to get that data into Kafka.

[00:04:14] JM: Could you define the term ETL, because I think this is a term we're going to be using throughout our conversation?

[00:04:21] RM: Sure, and that's a really interesting question, because I think the answer is changing to this. The historic answer, it means extract, transform and load, and it was what we did between relational databases. We took data from a transactional system typically. We extracted it from there. We transformed along the way. We maybe de-normalized it. We conformed the dimensions. Did the typical data warehousing type stuff, and then we loaded it to our data warehouse.

But nowadays, ETL is more broadly describing how people work with data. There are so many more different data stores out there and sources of data that a lot of the time people are doing ETL without really realizing it. They're taking data from one place. They're modifying it and they're putting it somewhere else.

So I think the traditional definition of ETL is becoming somewhat obsolete almost, and actually people need to think more broadly about what they're doing through the lens of ETL and the kind of operations we do through it.

[00:05:17] JM: So traditionally, you've got a transactional database. Maybe you're running a bank or you're running a ridesharing company and you've got this database that's actually accepting financial transactions. It's accepting user transactions. This is the database that's actually your source of truth for your users, your customers as they're interacting with your main system. Then if you want to do things like data science or aggregations of the data, you probably want to put it in a different database for a number of reasons that we've covered. The process of getting that data from the source of truth database, from your operational database into whatever your analytical system is, that is a changing story, if I understand you correctly.

[00:06:07] RM: Absolutely.

[00:06:07] JM: So how has the ETL process changed and why has it changed? Have there been some new technological developments that have caused it to change?

[00:06:17] RM: I think the key thing is the idea of events, and to almost as much of an extent, the idea of real-time processing. So I think one of the great misunderstandings around Kafka and stream processing in general is that it's about real-time. If you think, "Well, I don't need real-time, then I don't need something like Kafka or an event streaming platform." But that's kind of looking at things back to fronts.

The reason that people are changing how they do ETL to be using an event streaming platform is by building you a data around events. You can actually model much more accurately what happened in your business. So every single interaction with your business, whether it's sales, or clicks on a website, or rides in a rideshare, they're all just series of events. Those events you can aggregate up into a standard view of the world through facts and dimensions. But that stream, that sequence of events also describes important operations and characteristics to what's happening in the business.

So by capturing those events, using something like Kafka enables us to get much richer view of the data and also do so in their real-time. So you get an immediate reaction to what's happening, but also a greater understanding of what's happening as well. So that's why people are moving more towards these things, using Kafka for ETL, because it gives the richer insight to the data.

[00:07:34] JM: Where are events being created and where are events being stored?

[00:07:39] RM: Events are being created all around us, and there are some exceptions. But broadly speaking, all data is a series of events. It's just that we usually choose to store them in an implementation detail as lumps of data in a database somewhere. But it all started off as a series of events. Clicks on a website are a series of events. Transactions at a bank are a series of events. Almost all data is a series of events. Anytime data is created, it's created as an event.

How we store those events is then up to us, and we can store them and persist them in Kafka. We can take them from there. We can aggregate them and write them anywhere else we want to. But something like Kafka as an event streaming platform enables us to persist those events.

[00:08:18] JM: Give me a high-level architectural picture of what an ETL pipeline is. So we've talked about transactional databases. We've talked about the role of Kafka a little bit, and we've talked about some databases that you might want to put ETL data into, like data warehousing kinds of systems. Can you zoom out and give me an architectural picture of the different components of this ETL pipeline?

[00:08:48] RM: Sure. You've got your original database or databases, and this is another of the reasons why people are moving towards different architectures that you often have different sources. You then need to get the data out typically done using change data capture. That's then streamed through Kafka or a streamed processing platform. The enrichment is then done on that data as it passes through. So that's applying those transformations.

With that transformed data, or indeed that raw data, that can then be streamed out to one or more different targets. So the simplest example would be a single transactional system streaming the data into something like Kafka and streaming it straight through unmodified into, for example, S3, with Athena on top to do your analytics.

But more people are realizing the benefits of actually once you got that data in Kafka, you can transform as it passes through written back into Kafka and then consume it to multiple places. So you could write it to S3 if you're a cloud analytics. You could also lend it to an on-premises database, like PostgreS or something like that. But you can also use that same data to drive applications. So you kind of start to get away from this idea of having separate analytics systems and separate applications, and the kind of the never the two shall meet. But actually you can use the same data to drive your analytics and your applications, because it all comes from the same events.

[00:10:13] JM: The ETL process has historically been thought of as a batch process. So, for example, maybe we, on a nightly basis, take all of our transactional data and batch it into a data warehousing system and then we can do analytics and aggregations on this data warehousing system, because it's better for doing these large scale queries. Of course, there're a number of problems with this batch approach. One, as soon as you decide where your batch is going to be, you are going to start losing data that has happened since the batch. Two, if you want to build up-to-date applications on top of that analytical processing system, then it's going to be outdated. I mean, there are other problems with the batch mode of thinking. Can you explain why ETL makes more sense as a streaming process, as a constantly updated process rather than a batch system?

[00:11:16] RM: I think the answer to that is to turn it around on its head and to say, "Well, why would you batch it up? Why would you wait an artificial period of time before you process your data that happened?"

In the old days that was because while we had to wait for the shops to shut their doors and the point of sales around that reconciliation and to send the data to a mainframe because of all these technical reasons. Nowadays, we don't have those technical limitations. We have the ability to process data as it happens with all of those business benefits of having an up-to-date fresh view of the data that can drive up-to-date analytics, but also drive transactional applications. We don't have to segregate the processing and say, "Over here we have some stale data for analytics. Over there we got a separate hookup for applications needing the real-time data." Hooking into that transactional database directly and having two hits on that source data. We can actually take the events as they occur and we can use them to drive both analytics and applications.

[00:12:16] JM: What does the streaming ETL workflow look like in contrast to a batch ETL workflow?

[00:12:24] RM: So in some senses, it's the similar concept. You've got brining your data in. You've got cleansing it. You've got joining it, enriching it, aggregating it. Some of the differences are around how you handle things like time, because in a batch, you know what your time window is. It's once a day or once every hour, and it's clearly defined.

With streaming ETL, you need to reason more clearly about time. Are you talking about the event time, or the system time, and also the time windows that you apply to the data. But by and large, once you've kind of understood the concepts of the streaming components of it, it's still the idea of taking some data, applying the standard analytical processes to it. So conform your dimensions and enrich it and so on and put it somewhere else, or indeed, drive analytics directly from that stream processing system.

[00:13:11] JM: So if you have this data in a data source, like a transactional database. Maybe it's in Mongo. Maybe it's in PostgreS database. You want to do some stuff to that data. You want to perhaps put the change data capture log into Kafka and then you want to do additional processing on that data. Can you describe the process of getting that data from your transactional database into Kafka? Give the engineering breakdown of what I need to do to do that.

[00:13:46] RM: Sure. The technique you're going to use is change data capture, and there're actually two different types of change data capture, CDC. The one which most people think of when you say CDC is log-based change data capture. Going against the transaction log of the source database. Taking the events from the transaction log. But there's also query-based change data capture, and this is where you pull the source database to try and determine what's changed since you last polled it.

So both ways of finding out what has changed with different pros and cons. So log-based change data capture gives you much greater fidelity of the data. Every single change in a particular database or a table is captured, every delete, every update, every insert. If you use query-based change data capture, it's much easier to set up, but it means that you're going to miss certain elements of a data. You can't capture delete, because you can't query a database for data that doesn't exist anymore. You're also going to potentially miss changes to the same record that occurred multiple times within the polling window. So you poll the database every five seconds and the data changed four times within that five-second window, you only captured the latest state. You don't capture those in between changes.

So then it comes down to, "Well, what are you using that data for?" If all you want to do is mirror the state of your database over to a target data store for analytics maybe, perhaps it doesn't matter. But if you're looking to create event-driven applications based on the state changes within the data every time a customer is created, every time their address has changed, every time an order status changes and so on, then you're going to want to use log-based change data capture, because then you capture every single event after the databases transaction log.

[00:15:29] JM: Once that data is in Kafka, how do you make use of it?

[00:15:35] RM: You can take that data from Kafka and you can stream it straight through to a target system. So the way that we typically do integration with Kafka on other systems both inbound and outbound is using Kafka Connect. So Kafka Connect is part of Apache Kafka. It's one of the APIs within it and it's configuration file base. You don't have to write any code or such, and it lets you do streaming integration. So you can stream data in using CDC, for example, into Kafka. You could stream that data straight out to a target system. So you can say I'm going to capture every single change from our source system. I'm going to land all of those events down to S3, or up to S3, or to BigQuery, or to a local PostgreS, or Oracle instance. Wherever you want to put that data, you can do it straight through using Kafka Connect.

You can then also process that data within Kafka using stream processing technologies such as Kafka Streams, which again is part of Apache Kafka. Using something like KSQL, which is a streaming SQL project working with data within Apache Kafka, and the outputs of those stream processes go back into Kafka topics, which can then be synced using Kafka Connect down to target systems. All those topics can be subscribed to by applications that you write that want to use that data.

[SPONSOR MESSAGE]

[00:16:58] JM: Looking for a job is painful, and if you were in software and you have the skillset needed to get a job in technology, it can sometimes seem very strange that it takes so long to find a job that is a good fit for you.

Vettery is an online hiring marketplace to connects highly-qualified workers with top companies. Vettery keeps the quality of workers and companies on the platform high, because Vettery vets both workers and companies. Access is exclusive, and you can apply to find a job through Vttery by going to vettery.com/sedaily. That's V-E-T-T-E-R-Y.com/sedaily.

Once you're accepted to Vettery, you have access to a modern hiring process. You can set preferences for location, experience level, salary requirements and other parameters so that you only get job opportunities that appeal to you. No more of those recruiters sending you blind messages that say they are looking for a Java rock star with 35 years of experience who's willing to relocate to Antarctica. We all know that there is a better way to find a job.

So check out vettery.com/sedaily and get a $300 sign-up bonus if you accept a job through Vettery. Vettery is changing the way people get hired and the way that people hire. So check out vettery.com/sedaily and get a $300 sign-up bonus if you accept a job through Vettery. That's V-E-T-T-E-R-Y.com/sedaily. Thank you to Vettery for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:18:47] JM: Once you have your Kafka data pipeline setup, does it constantly update and then you can reliable have those syncs, the read side of this data pipeline? Is this data going to be constantly updated?

[00:19:04] RM: That's right. As an event comes into Kafka, it will get streamed out. If you're not doing any processing on it, it will get streamed straight out to the target. So if you're using something like Elasticsearch, for example, you're going to see that almost instantaneously. You can also have you aggregates populated that way. As long as your target system allows item-potent updates, then your aggregates will update in place. So if you're building an aggregate of what are my current sales within the last hour, for example, then the aggregate key, the value for that key will get overwritten each time and will just increase per hour. Any late arriving data would also get updated against the appropriate key as well.

[00:19:41] JM: Okay. So you've given another example here of a search system. So you could see potentially taking your transactional data store, and if you wanted to have search indexes created against that transactional data store and you want to have those search indexes constantly updated, you could have the change data being buffered from your transactional data store into Kafka and then you could have your search system reading from Kafka and updating its search indexes.

[00:20:12] RM: Yes, absolutely. Again, that's one of the common use cases for Kafka, because since Kafka persists its data for as long you want it to, whether that's forever, whether that's for 10 minutes or based on a certain amount of size. You persists that data in Kafka and you can use it multiple times.

For example, you populate your search index. If you search index goes bang or if you want to scale it out and populate additional instances, you replay that data from Kafka down to the target. But you're also updating it constantly from the source system so it gives you that ability to populate and maintain those caches. But that same data that you're using to populate your search indexes, you can also be using that same data to populate a graph database or an analytics database. However you want to use that data, you can use it multiple times from that same topic in Kafka.

[00:21:02] JM: I want to ask a question here, why do we actually need Kafka? If we're just trying to get our data from a transactional data system into a data warehouse or trying to get our data from a transactional system into a search index, why not just throw that data straight into the search index or straight into the data warehouse? Why do we need this middleware of Kafka?

[00:21:23] RM: That's a good question and it's one that does come up, because on the surface of it, it seems like, well, that's the obvious thing to do. We're introducing apparently additional complexity by using Kafka in the first place.

The answer is that it's never just one pipeline. It's never just the set of data is used in this one place over there. It's always, "Well, this data here needs to populate a search index, and we want to use it for analytics, and we want to share it with this other department, and we want to use it to drive this application."

So you have two options. You use something like Kafka, which lets you take a real-time feed of those events from the source system and use it multiple times concurrently against different target applications and consumers, or you start to rebuild yourself a spaghetti architecture by taking the data from a source system to one place, copying it to another and making it available to another and hooking all these different dependencies together, which then makes it very difficult to change any of those pieces to scale them out. If you want to replace one of those or remove one of them, you have all of these nasty dependencies.

So Kafka lets you decouple those dependencies. It also gives you many additional benefits, like providing buffering between your source and targets. So it does make a lot of sense once you actually start examining the alternatives to using it.

[00:22:38] JM: One other thing that you can do in Kafka is data enrichment. So if I buffer my change data capture log into Kafka, I can have data enrichment processes running over that data that's in Kafka. Can you explain the role of data enrichment?

[00:23:00] RM: Sure. So this lets you take events as they come in which are going to be from any source system or application. So it could be stuff happening in a database. It could be applications writing directly to Kafka. Those events, you can then start to process. You can start to cleanse the data as it comes on and filter out by records. You can start to reshape it and modify the schemas. You can start to join it to other data. That could be data from a different database, from a different system, from a flat file, from another application. But because it's all in Kafka, it can then be worked with together regardless of where it came from. So you can start to join in and enrich your data. You can start to concatenate and create derivations of that data and doing it all within Kafka.

[00:23:42] JM: The processing of data within Kafka is often done through a system called Kafka Streams. So stream processing is the act of doing changes to streams of data that are coming into your system. There's a large number of ways that you can perform stream processing. You could just set up a simple Python script to just read data off of Kafka and make some changes to that data and write it back to Kafka. You could use a distributed streaming framework like Flink. You could use Spark streaming, or you could use Kafka Streams. Why should people use Kafka Streams or could you present some of the different alternatives that people can use to do streamed processing against data in Kafka?

[00:24:33] RM: Sure. There are various different ways to do it, as you said, and you listed some of the common ones there. To an extent, there's not one right one. There's going to be technical reasons why you may choose one over the other. There's also going to be pragmatic reasons, like scales and experience. But Kafka Streams, it's part of Apache Kafka. It's a Java library. So you just bring it into your existing applications.

By being part of Apache Kafka, it benefits from much tighter integration when it comes to things like security, things like transactional processing in exactly one's semantics. So those are some of the reasons why often people start out on a greenfield project just using Kafka Streams, because it makes a whole ton of sense. If they already have things like Flink in place or they already are massively brought into Spark streaming, they may well choose to use those unless they hit up against some of the technical limitations, which think, "Well, actually we'll reevaluate this and we're going to use Kafka Streams."

One of the other common ways that people do stream processing using Kafka is using KSQL, and this doesn't require any coding or such at all. That's just using a SQL type language to interact with a data and declare stream processing applications. So that means no Java, no setting up kind of Spark clusters and stuff like that. So it's another good route that people do often take.

[00:25:52] JM: When we're taking the change data capture log from our transactional database and we're getting it into Kafka and we're going to use that change data capture log to update sync databases, are we getting the entire database? Are we getting the entire change log into Kafka? That sounds like putting a lot of data into Kafka.

[00:26:15] RM: It depends entirely on how you configure your CDC tool. So if you're using something like Debezium, which is very good popular open source Kafka connector. It supports things like PostgreS, MySQL and so on. You simply say, "I'm interested in these particular tables." So you could put in the whole database if appropriate. But often times it'll be, "Well, I want the sales table. I want the product table. I want the customers table," and you just get the events related to those particular tables. But it comes down to how you configure the particular integration.

[00:26:43] JM: Let's take the search example. So like let's say I want to have my transactional database – I get a search index built over that transactional database. If I'm doing that, am I starting by seeding the search index with a copy of my database or is the entire process based around that change data capture log? Do I have the entire historical change data capture log? I guess I'm wondering you want to have this search index that you're going to be updating overtime to be a reflection, to be a searchable reflection of the transactional database that you have. But when you're bootstrapping that search index, do you just take the actual database or do you just take the historical change data capture log?

[00:27:32] RM: So the way that many of the tools work is they'll will take a snapshot of the current database state. They record the SDN, the points in the transaction log at which that was taken. So kind of queries the system. Then from there on in, capture any of the events out of the transaction log. So it's kind of current state just through a a selector server plus transactional log from that point in using the SDN to make sure you don't miss anything in between.

[00:27:55] JM: Okay. Let's talk a little bit about this interface between a database and Kafka, and that's Kafka Connect. That is the interface point of how you're getting your data from one of these sources into Kafka or from Kafka into one of these syncs. Explain what Kafka Connect is.

[00:28:14] RM: Kafka Connect in a nutshell lets you do streaming integration between source systems on Kafka and new target systems. It's just configuration file-based. So you don't have to write any code, and it's part of Apache Kafka. So if you're wanting to get data from Kafka down to HDFS, Kafka to S3, from a database into Kafka, from Kafka to Elasticsearch. All of the possible permutations of pipelines you can think of. Almost always, you want to be doing that through Kafka Connect.

So Kafka Connect solves a lot of the problems of integration between systems. It does things like fault tolerance. It's distributed system is built on top of Kafka Semantics itself. So you can scale it out. It also handles the more tricky things, like schemas and offsets. All of the kind of things that if you decide to write it yourself, as many people unfortunately start off by doing and saying, "Oh! Well, I'll write myself a Spark job to get this data from here to here, or I'll write a Java program to get data from this database into here." All of those tricky little things, Kafka Connect has solved already. So it's a solved problem, really.

It also has additional capabilities, like transforming the data as it passes through. So whilst we've talked about stream processing frameworks for the kind of the more advanced stuff. Kafka connect also has the ability of transformations on the data as it passes through, and it's called single message transforms. You can do some pretty cool stuff. You can start to mask the data or job fields or enrich them or change data types. All through Kafka Connect, all through configuration files. So it actually makes it much more accessible to many more people other than just those who are going to write a Java program or write a Spark bit of code.

[00:29:57] JM: Oh, that's useful. So you can do begin the transform process at the entry point of data from the source into the Kafka Connect system.

[00:30:08] RM: Absolutely. So, for example, if you're pulling in data from a table with hundreds of columns, which is not uncommon, and you say, "Well, actually, I only need a subset of those." You could just drop out all of the additional ones, or if you've got data coming in and it's got personally identifiable information, it's got credit cards, it's got addresses, and you don't need that data. Actually holding that data within Kafka then has additional implications. You can say, "Well, we'll just drop those columns out.
At the same time, this column over here, let's cast this and change the data type. This one over here, let's add in some metadata, some lineage information about where the information is coming from." Again, all just through configuration files.

[00:30:45] JM: Kafka Connect runs as a distributed process. So there's multiple nodes for a Kafka Connect process. Is it always true? Can you explain the parallelism model of Kafka Connect?

[00:30:59] RM: Sure. So Kafka Connect, and this is an important point to make actually, does not run on your Kafka brokers. Nothing runs on your Kafka brokers except possibly ZooKeeper and then just plenty of people that would disagree with us as well. But Kafka Connect runs separate from your brokers. You can run a single instance of Kafka connect if you want to. But as soon you want fault tolerance, as soon as you want additional capacity, you then deploy additional Kafka Connect workers and deploy them as part of the same group.

Kafka Connect will then distribute the workloads across those instances. If you have a single connector, like a sync process, thus taking data from a Kafka topic out to a target, you can have multiple workers which will then form a Kafka consumer group and read that data in parallel. If you're brining in data from a database, for example, Kafka Connect can parallelize that ingest and read from multiple tables at the same time. So the parallelism is defined within Kafka Connect and lets you scale that out.

[00:31:57] JM: When would you need that parallelism? Is that only if you have a database that's changing really, really rapidly?

[00:32:05] RM: So it depends on what kind of throughput you want to get on your data. If you're quite happy, just let Kafka Connect sit there and chug through the data, then I guess you don't need us. But it gives you that ability to do so if and when you decide you want to get the data in, if the data is being created at a greater rate than is being ingested on a single worker node.

[00:32:23] JM: So now that we've talked through some of the finer points of this streaming ETL process, could you zoom out again and describe the streaming ETL process? Let's say I've got a transactional Mongo database. That's my source of truth database. I've scaled up overtime and now I want to build a search index on top of that database. I want to use Kafka as the middleware. Describe the end-to-end process of getting that data from Mongo into a searchable index.

[00:32:56] RM: Sure. There are two different answers to that. I'll give you the simple one first and then I'll come back to the second one. The simple one is you deploy Kafka Connect. You use the Debezium connector, which is a plugin for Kafka Connect. So I should have mentioned, Kafka Connect is a plugin-based architecture. So you plug in the appropriate connector for the technology. So we plug in Debezium. We use the change data capture for Mongo. That's changed the data into a Kafka topic. We can then stream that data straight through out to our search index, again using Kafka Connect perhaps with the Elasticsearch sync connector. So that's two different JSON files. The first one defines the source connector. The second one defines the sync connector.

In the middle if we want to, we can write some stream processing using something like Kafka streams or KSQL. If we wanted to start modifying that data dropping out records, enriching records and so on. But the simplest is simply a source connector, a sync connector.

The second answer to that is that you don't always want to use Mongo as your source of truth. You may well decide to architect your application to write your data to Kafka first, and then Kafka would write the data to Mongo as well also then on to other systems, such as Elasticsearch. So it's always worth evaluating if it's appropriate to write your target system and then ingest from there into Kafka or actually write to Kafka and Kafka pushes the appropriate systems at which you want the record of that data.

[00:34:20] JM: So there are all these different connectors that have been written in the ecosystem. So you've got connectors for all the different popular databases. How hard is it to write these connectors? So somebody who's sitting inside of Confluent, the Kafka company, who is writing all these integration points for the Kafka connectors?

[00:34:38] RM: So Kafka Connect is part of Apache Kafka. It's an open API, and there are different parts of the Kafka Connect framework that's important to understand when you start to get down to this level. The connectors themselves, which define the integration between Kafka Connect and the source or target technology. So understanding how to read [inaudible 00:34:59] for MySQL, or understanding how to send records to Elasticsearch. Then within Kafka Connect, you have these transforms which are mentioned. Again, that's an open API. You can write your own transforms in Java. There's also something called converters.

So converters separate out the business of connecting to source or target technologies from the business of how you're going to serialize your data in Kafka or de-serialize your data out of Kafka. Are you going to use something like Avro with its rich support for schemas? Are you going to use something like JSON? Hopefully you're not going to decide to just use CSV and strings, but unfortunately some people do. But those separate components make it easier to build these different connectors, and the connectors themselves simply have to worry about how do I connect to the source or target technology and interface those records, getting them in, or taking records and pushing them out.

[SPONSOR MESSAGE]

[00:35:59] JM: As a programmer, you think in objects. With MongoDB, so does your database. MongoDB is the most popular document-based database built for modern application developers and the cloud era. Millions of developers use MongoDB to power the world's most innovative products and services, from cryptocurrency, to online gaming, IoT and more.

Try MongoDB today with Atlas, the global cloud database service that runs on AWS, Azure and Google Cloud. Configure, deploy, and connect to your database in just a few minutes. Check it out at mongodb.com/atlas. That's mongodb.com/atlas.

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

[INTERVIEW CONTINUED]

[00:36:55] JM: We're talking here about using Kafka as this middleware data platform that we're going to be building additional data systems on top of. In some ways, I see Kafka as replacing some of the roles of the "data lake". The data lake is the term that has historically been used for HDFS or perhaps for S3. The place where you're dumping all your data and then you may suck in that data and do things with it later on. Do you see people reconfiguring the way that they frame their data platform by replacing some of the roles of the "data lake" with Kafka?

[00:37:40] RM: Yes, I think that's a good way to look at it. I think there's still and always will be a role for having a bunch of data sat around, whether it's in S3 or even just in relational database that people can go to and run ad hoc queries and just kind of generally munge around with. Where I think people are correctly realizing that this makes less sense, is when they're then hooking it up to other systems. So I would say a distinct anti-pattern would be taking transactional data, writing it to HDFS, to S3, and then writing other systems which use that data. Those systems should be taking the data from Kafka, which should be hooked up to the transactional system.

I supposed it's the ad hoc nature of the data lake support. So I think that's still a valid use case. When you replay data out of Kafka, it's done so sequentially. So it's fantastic for using to drive applications. It's fantastic for populating target systems. If you're doing the equivalent to a full table scan, that's not going to perform so well. There are other systems which are more suited to that kind of access pattern to the data.

[00:38:43] JM: Tell me more about how people with late stage data systems where they've already got a data lake set up. How are they augmenting their data platform system with Kafka? Are they moving certain workflows away from a data lake-centered workflow towards one that is centered around Kafka? Just tell me that's how evolving.

[00:39:10] RM: I think generally the adaption patterns of Kafka start off with particularly use cases. This is not a rip and replace approach really with the technology, because I think like you mentioned earlier, people quite will say, "Well, I've got this system and this system. Why don't I just connect them to each other?"

You start to realize the benefits of it when you have a driver perhaps from needing the events themselves, when you need the near real-time nature of those events. So different use cases with prompt the adaption of Kafka, and then once Kafka is in place, it's used to kind of just tends to accelerate and people realize the benefits of being able to hook in to the system as the source of data for their applications for populating their systems instead of having to hook up these kind of spaghetti architectures from place to place to place. So it tends to evolve on top of what's there already rather than necessarily rip and replace and kind of get rid of existing things.

[00:40:04] JM: The replacement of a spaghetti architecture, this is something that pub/sub systems have been solving for a longtime. We've had older pub/sub systems for Kafka and they have solved something similar where you have a producer. You have a number of consumers that want to see data from that producer, and the way that an architecture often starts is you've got a producer that is sending messages directly to each of these consumers. By creating a pub/system, you centralize that logic and you allow it to be more easily propagated, because you just say, "Hey, I'll publish a message to this pub/sub system," and then all the necessary consumers can read from that pub/sub system as they need to. Why did Kafka take off? Why was Kafka able to supplant some of the legacy pub/sub systems? What did Kafka do differently than the other pub/sub systems?

[00:41:09] RM: I think one of the bickeries in this is the fact that it's a distributed system. So it scales and doesn't have the issue of the many of the previous ones in which you would end up with having to create multiple instances or kind of hub and spoke architectures with the associated problems.

The fact that at its heart it's this distributed commit log. It's a very, very simple concept that events are up-ended to this end of the log and anyone can read from that log from any particular offset and scan through and consume the messages rather than having to have subscribers in advance declaring that they want to receive messages. So, again, supporting the scalability side of things.

I think the fact that Kafka itself has these abilities to do processing as well means that you can do more within it. The other thing is that Kafka is based around events. Events are different from messages. Events give you the stability to share a notification as well as state. So not simple a message saying someone bought something and it's a transient message that's been deleted and you have to persist that state somewhere else, usually in a database and things get messy already.

An event, so someone bought something and here's what they bought. If you have the events, you can reconstitute the state. Within Kafka Streams, within KSQL, you have the concept of a table, and a table actually comes from a Kafka topic. It's simply saying, "Looking across these keys, what's the current state?" We can aggregate across those. So Kafka becomes your source of truth in what happened in your business. You can take that data and put it somewhere else as your secondary copy, but your master view of the data can actually come from those events. So I think it's simply more powerful, more scalable than what was there previously.

[00:42:54] JM: You've written about other patterns, patterns that we've not explored yet. One pattern that I've seen you write about is bout ingesting sys logs into Kafka. Can you explain what a sys log and why it might be useful to ingest those into Kafka?

[00:43:10] RM: Sure. This is already an interesting one. It came from a home setup and a server, I was playing around with and different virtual instances within it. I thought, "Will it be useful to just take all of the sys logs?" So whatever is generated by the system and any applications on the system writing to the local log to be able to centralize those logs and then start to be able to inspect them and process them.

So often times there's a lot of background noise. But I found it interesting to tape a home server, it's connected up to the internet to see how often you got brute-force SSH attacks against it. By taking those sys log events, stream them into a Kafka topic and then writing some stream processing against that. I could start to analyze what was happening. Because it's event-driven, I could then generate alerts from that and just let push out notifications saying, "Hey, in the last five minutes there have been 10 attempted accesses to the server."

Again, this is just written using Kafka Connect. In this case, using KSQL. So I didn't have to write any code to do that. Kafka Connect sits there and it lessens for sys log connections. Get writes into a Kafka topic. KSQL takes that Kafka topic, it processes it, it filters it for certain conditions such as the string you get in sys log when there's an SSH attempts and then performing the aggregations on top of that. Then that's writes back to a Kafka topic, which Kafka Connect then hooks up and pushes out the notifications on.

[00:44:32] JM: Is Kafka more broadly useful as buffer or a middleware system for log data?

[00:44:39] RM: It's certainly is. It's used in so many different ways, because it is so flexible and adaptable. So I mentioned earlier on the idea of database offload. That's one of its huge use cases. But it's also widely used for this idea of aggregation, log centralization, processing, because you can do that processing on the events as they arrive.

[00:44:59] JM: You've also written some about how Kafka can fit into a data pipeline that also includes data warehousing tools. So Google BigQuery. Maybe you've got Snowflake at the end of this. Can you describe some patterns around involving Kafka with your data warehouse?

[00:45:18] RM: Sure. So wherever your data is coming from, whether it's coming from a straightforward relational system that you've got on-premises, whether it's from a cloud-based platform. Whether it's from your own applications writing into Kafka directly, you have your transactional events in Kafka. You can then enrich them and transform them as you want to. Then Kafka Connect can push that data to the appropriate analytics platform. So it can push it to BigQuery. It can push it to Snowflake. There are connectors to both of those that's a passive Kafka Connect framework. So you can have that real-time data aggregated and enriched if you want to using stream processing, but then landing an aerial time in these target data warehouses.

[00:46:01] JM: I have read several of your articles where you frequently use Oracle databases in your examples. I don't know if that has to do with your background in particular database. But do you encounter a lot of Kafka users who have Oracle installations?

[00:46:19] RM: Yeah. So my background is with Oracle. I used to work doing consultancy with Oracle analytics projects. So I've got a naturally affinity to it. It's interesting I think how the apparent market is changing or usage with Kafka is changing. Originally, a few years back when I started looking at it, it was very much the cutting edge stuff. So those people are really heavily interested in the cutting edge distributed systems. The traditional enterprise, things like Oracle got less of a mention.

Now, you see more and more conferences and meet ups on community platforms. People asking about integrating with Oracle, with Teradata, with SQL server because that level of adaption with Kafka is there, that it's actually making huge inroads into the enterprise. It's not just a cool kid's toy anymore. It's a serious platform that many enterprises are adapting, and enterprises tend to have enterprise databases.

[00:47:13] JM: I think there are people who would like to move away from their Oracle installations, because Oracle can get quite expensive and the database market has gotten really competitive. So there are a lot of good alternatives to Oracle. Do people use Kafka to migrate away from Oracle?

[00:47:33] RM: I'd certainly agree. There's definitely a shift to that. If it's a straightforward migration? This question does come up quite a lot at conferences. If it's simply a one-time big band, then Kafka is not necessarily appropriate way to do it, because there are plenty of migration tools out there on the market. Where Kafka fits very well is where you want to retain an existing system without impacting that, but take changes from that system and start pushing them to the new one. So you can run side by side.

So people use this approach and moving from one database to another. They also it in moving from on-premises to a cloud platform or to running separate cloud platforms. Because you've got your events from your source, whether it's a database, whether it's an application writing to that database into Kafka, those events can then be pushed to whatever your target one is and you can run side by side without impacting your original application. Then once you're good to go, you can switch off that source. So Kafka gives you massive benefits there.

[00:48:28] JM: You've written some about the blurring lines between analytical and transactional systems. So this is like where historically you might have been running a nightly job to aggregate maybe a recommendation system. Maybe you've got your transactional data system, and in order to build recommendations based on that transactional data system, you do an ETL job into a data warehouse, and then you use the data warehouse to generate those recommendations.

But these lines are blurring, because we want to do more and more up-to-date analytical workloads. Can you explain how the patterns that you're seeing around Kafka are fitting into this improved latency of the analytical workloads?

[00:49:21] RM: I think historically we've always separated out the technologies, because you had to. I mentioned earlier, sometimes technology forces how we design systems. In the past, you had to decide, "Am I building a transactional system? Do I want to be able to get the data in quick but it will be slow to read? Am I building an analytical system?" It will be quick the data out, but it might be slower to get it in." You have to choose.

So organizations and teams grew up around this way of thinking, and developers and engineers had this way of thinking. It's like, "Well, I'm an application developer. I need application technologies. Nothing to do with analytics. I can ignore those data warehouses and stuff.'

Kafka brings the data, brings the events front and center. Now applications can access that data in aerial time. Analytics can also be driven by that same data. So I think the move away from doing things in separate worlds is actually it's happening. I think it's happening slowly because you still have this in-built way of people have a thinking about, "Well, I'm building this application. It must be an application type architecture. I don't need to think about analytics. I'm not going to integrate with analytics, except for a one-time bachelorette once a night out of it." Now it's much more tightly integrated."

[00:50:33] JM: You've been at Confluent for a while now. What's something new that you've learned about Kafka in the last year?

[00:50:39] RM: That's a good question. I think that my answer to that would be around KSQL, which is a product I spent a lot of time working on last year. So that was built on top of Kafka Streams. So understanding a lot more around how Kafka Streams has its concepts of handing streams of data. Being able to instantiate the state on top of that and how you can actually do that through KSQL. It's probably my main learning from the last year.

[00:51:02] JM: Okay. Last question. Do you have any reflections on what it's like to be at a rapidly growing infrastructure company? Because Confluent is growing really quickly.

[00:51:13] RM: Yeah. It's a fantastic place to be. If you want to just kind of turn up and do something and then go home and not care about it, then I guess you could do that, but you're not going to get the most out of it. There's some super, super smart people, some super nice people as well. It's just a very welcoming, friendly place to be where there's a lot of very exciting work going on. I feel privileged to have a job where the technology is awesome. I can talk about it enthusiastically because I genuinely think it's really, really good, so does that one other company and the people working on it are really smart. It's just an exciting place to be.

[00:51:46] JM: Okay. Well, Robin Moffatt, thank you for coming on the show. It's been really fun talking to you.

[00:51:50] RM: Thank you so much.

[END OF INTERVIEW]

[00:51:59] JM: Software can improve our lives, but the business motivations of software sometimes conflict with user desires and may hurt us instead of helping us. Rehack is a reverse hackathon that uses humane design to tackle this conflict. Rehack is devoted to making today's technology healthier, fairer and more humane with as little business comprises as possible.

How can our software improve our psychological state rather than stressing us out? How can we redesign our software products to make social interactions more productive and enriching instead of introducing feelings of isolation and inauthenticity. Rehack emphasizes humane design, usability and positive mental health.

Rehack is being hosted by Princeton University this November and they're looking for sponsors who are interested in supporting their mission. For $500 to $3,000, your company can support Rehack with resources for the hackathon. Event specifics can be found on recheck.co. That's R-E-H-A-C-K.co. Rehack hopes to use higher level product thinking, UI and UX design and human computer interaction to find ways to improve today's tech products.

I'm a fan of what Rehack is doing and I think it's a great idea for a hackathon. I think it's very positively motivated. If you can sponsor them, they could really use the help. Go to rehack.com to learn more about Rehack and details on how to support them.

Thanks to Rehack for being a friend of the show, and I look forward to seeing the awesome projects that come out of Rehack.

[END]