EPISODE 1476

[INTRODUCTION]

[00:00:01] ANNOUNCER: Streaming data platforms like Kafka, Pulsar and Kinesis are now common in mainstream enterprise architectures, providing low-latency, real-time messaging for analytics and applications. However, stream processing, the act of filtering, transforming or analyzing the data inside the messages is still an exercise left to the receiving microservice or data store, a custom programming exercise likely repeated over and over within an application.

Stream processing tools such as Apache Flink and KSQL database have been around for half a decade, but their complexity has hindered adoption. Decodable's mission is to radically simplify processing on the stream with a SaaS platform based on Flink and only using SQL, which frees up developers to focus on what matters most.

Eric Sammer is founder and CEO of Decodable, and joins the show to discuss the potential of stream processing, its role in modern data platforms, and how it's being used today.

[INTERVIEW]

[00:01:13] JM: Eric, welcome to the show.

[00:01:15] ES: Thanks so much for having me. It's a real pleasure.

[00:01:16] JM: Yeah, it has been a long while. Last time you came on, we were talking about Rocana, which is a previous company that you built. And since then, you've seen the world of data engineering evolve up close. And it's become quite crowded. And I think the area of focus for Decodable, which is your new company, is an area that's crowded as well, which is streaming data.

And I think the streaming data systems today that get the most usage are probably Spark streaming, Flink. There's a variety of others. But streaming, also, it's kind of a broad term. Maybe you could focus in on what area of the stack you're focusing on with Decodable.

[00:02:11] ES: Yeah, I think that's a great question. And I agree with you completely. I think that it's a really complicated and nuanced space with sort of different people with different opinions about how it should work, and different tech stacks, and probably, arguably, different products to sell.

Our position here is that maybe it almost makes sense to start with what we're not. We are not trying to replace the messaging systems; Kafka, Pulsar, Kinesis, GCP, Pub/Sub, those kinds of things. Our position is that, increasingly, people know those systems. They work really well in most use cases. There's an ecosystem around them. And so, I think the most important thing for us is that people don't have to re-platform or any of those kinds of things. We think that messaging is, I don't want to say, a solved problem, but increasingly served by the systems that are out there.

Like you said, the stream processing engines themselves. And I think you're right. Spark streaming, Flink. To some degree, we see some like K Streams or ksqlDB and like those kinds of things. Those things are out there. They are kind of painful in a lot of places. And so, our position is like a lot of people don't want to deal with checkpoint offsets, offset management, state management and sort of the operations of these things. Because these streaming pipelines do look more like micro surfaces than they look like batch data processing jobs. And I think that that's one of a couple of different dimensions in which stream processing is a pain, is that it sort of has like a higher order operational overhead to it.

But our position is like, one, let's up-level it and get people out of thinking about the low-level details of like distributed checkpointing systems, and state management, and job control, and configuration, and task parallelization and those kinds of things. Let's make it a service for people so that they don't have to deploy a whole lot of software.

And like, for better and worse, I'll say, let's pick an interface for people that most aligns with the knowledge that they have. And rather than having people write sort of like DataFlow-style programs in low-level programming languages where they have to deal with serialization and like all these other kinds of details in addition to the business logic, let's just do it with SQL. And that does mean that there are certain things that are hard to express. Certain jobs basically that

I think somebody wouldn't be able to accomplish. But at the same time, it means that mere mortals can build like the 80% of cases that are like filter, parse, transform, restructuring, those kinds of things. We tend to be the processing layer connecting all the source systems to all the destination or sync systems that allow people to self-serve. They don't have to know the guts of Flink or the guts of Spark and like all these other kinds of things. And we do that by kind of up-leveling the developer experience to make it a little bit more tenable.

I think Decodable is to stream processing what like GitHub is to Git, if you like that analogy. We're trying to sort of put like all the workflow and sort of like nice quality of life stuff around it in addition to just being the right way to run stream processing jobs.

[00:05:37] JM: Gotcha. These other platforms, like Kafka streaming, or Spark streaming, or Flink, they have years of open source work that have gone into them to make them resilient streaming platforms. Maybe the APIs aren't perfect. Maybe there's some issues with them. But they do work. Are you describing a system that's built on top of other streaming systems and just provides a SQL API? Or is this an entirely new streaming platform?

[00:06:12] ES: Great question. I think that we are 100% standing on the backs of giants. We use very heavily Apache Flink. We're big fans of Apache Flink. Decodable's core engine is definitely absolutely based on Apache Flink because of exactly what you described. It's been beaten up by Netflix, and Lyft, and Uber and sort of all these different people who have kind of put it through its paces over a long period of time.

And so, even though it's got its warts, right? It's a little bit hard to use, it's hard to deploy, it's got a bunch of different APIs and like these kinds of things, the engine itself is incredibly stable and does exactly what it purports to do, at least in most cases.

The real challenge is the usability of it. And so, we are focused on the kind of abstracting the parts of Flink away that are a little bit challenging and patching over some of what we believe and what some of our customers believe are sort of gaps in functionality and those kinds of things. And sort of obviously stabilizing it, like those kinds of things. We're probably like your typical open core enterprise data company, if that makes sense.

[00:07:27] JM: It does make sense. Now, if you try to sell a new infrastructure product to an established market, it can be hard to ease your way into a trusted position. It's probably maximally hard if you're building a database. If you're building a streaming system, it's a little bit different, because at least people are standing up new streaming data pipelines all the time. Maybe they can try out your system. Maybe you could talk about the prototypical use cases that people are willing to try out for Decodable, where it makes sense.

[00:08:10] ES: Yeah. I mean, absolutely. And I think you're on the money. You can have the best technology in the world. But if it's too painful for someone to try and sort of figure out whether or not it's right for them, it's basically dead on arrival.

For us, one of the core tenants is that we sort of fit nicely into the larger ecosystem. And I think that that's really important. Sort of the existing way that people deploy and build these kinds of systems internally.

One, that means that there can't be any proprietary formats and stuff like that. Specifically, as an example, if you already have Kafka deployed, which we love, and your data's flowing with like AVRO messages, or Protobuf messages, or JSON messages over those Kafka topics, you can add Decodable as effectively just another consumer. What you're sort of getting at, it's a little bit easier with streaming systems. Just another consumer off of that. And build your pipelines. See whether or not that makes sense for you. And if it does, you can sort of slowly replace some of existing pipelines. Or even just add to sort of the existing infrastructure by running those things in parallel maybe for certain users or use cases. And it effectively can coexist with more complex jobs that have to be written in a lower level API. Maybe you have to build Flink jobs for, I don't know, some really complex ML jobs or online training systems and things like that that are hard to express in SQL. Or where you just kind of have pre-existing code or libraries that you want to be able to use, maybe Decodable isn't like the right place to do some of that stuff. And we're okay with that. We love that. It's part of what we do.

Typically, people deploy or use Decodable in two different cases. One is what I would call sort of the core main data platform, where people either don't have maybe Kafka deployed, or Kinesis or those kinds of things because they haven't kind of made that jump to either event-driven services or real-time data integration into data warehousing products and things like that. And in

that case, they're going to use Decodable to connect directly to things like S3 and like all these other kinds of things. And sort of Decodable winds up becoming the place where they build kind of all their pipelines.

The other use case – And I think this one we have seen more commonly. To your point, we don't have the trust of customers the way as a vendor yet, because we've only been around for about a year. And those kinds of things, you have to earn your place in the core data stack. And I think that's a healthy skepticism.

Typically, what people do is they will go to one team within a larger organization and they say like, "Hey –" And it's typically a team that already knows SQL and those kinds of things. And they'll say like, "Hey, why don't you try this for a couple of your pipelines and see if it works? Because it does sort of naturally fit in. And if it does, then we kind of like spread you know horizontally to other teams. And eventually, we may earn our right to be like the core artisanally crafted critical data pipelines that carry like every financial transaction in a bank or something like that." But it does start with basically the end consumers, right? The data science team, or maybe a product recommendation pipeline, or like those kinds of things that sort of are more on the consumption side than the production side in the streaming capacity. And then we sort of – Like I said. We kind of earned the right to move from there.

And this allows people to kind of see it in action. And in fact, the business is designed around this. We have a free tier. So you actually don't even need to call us. Like, sort of stand up a pipeline and just see if Decodable works for you. And if it does, great. If it doesn't, we'd love to learn why. But it's not going to like bring down your entire bank, or your entire insurance company or something like that.

[00:12:21] JM: In building Decodable, you have to build connectors to mediate the streams. Basically, it's the interface between different sources of data, like, messaging services, and storage services, and databases and the streaming data pipelines. There's such a plethora of connectors that you potentially have to build to pull from these different places. I mean, there's entire companies built around data connectors, the Fivetrans and Census' of the world. How do you avoid getting mired in the connector problem?

[00:13:05] ES: Oh my goodness. This is like a hot button topic even within Decodable about where do we need to build, versus where can we actually leverage something that's kind of already in place. I think you're 100% right. The short answer is that we focus primarily on like what we think of as the tier one systems is the messaging systems. And the reason why is a couple things. One, increasingly, database vendors and service providers, like SaaS businesses like Salesforce or Marketo and like those kinds of companies, are basically gaining first-class support for things like Kafka and streaming systems because they're sort of starting to become lingua franca for the way that people think about the network of data.

There are definitely people still think like the data warehouse is sort of the nexus of the world. I'll be honest with you. I think it's one big consumer. But it is just one consumer off of a lot of the streaming infrastructure. That might be another discussion. We build, first and foremost, the connectors to the messaging systems, Redpanda, Kafka, Pulsar, Kinesis, those kinds of systems.

Our second ring of connectors where we're sort of starting to get more involved is the change data capture world and like being able to consume CDC streams and produce into database systems. And there, I'll be honest with you, I don't think we're ever going to be the company that has the long tail of like Oracle 7, and mainframe connectors and those kinds of things.

We're very purposely being thoughtful about the kinds of systems. For instance, Postgres, Mysql and Mongo, Cassandra is probably easy to understand. You start maybe Oracle. But as you get into the like more and more like either traditional database vendors and then legacy systems, I think, to your point, we avoid those like as much as we can and rely on the fact that there are other companies that will build those connectors, because it is high-value. It's just not our business. Build those connectors and they will produce that data to the messaging systems.

And so to us, the messaging systems are becoming sort of the central universe of those things. And certainly, Confluent, for instance, has done an excellent job of building a lot of those really, really complex connectors. We don't want to compete in that space. Like you said, there's other people who have done these kinds of things.

Now that said, I mean, you mentioned Fivetran. You mentioned Census. There's like Hightouch and like all the reverse ETL people and stuff like that. I think that those things are a little bit challenging because they effectively avoid the messaging systems. And I think that that's actually a mistake. I think the fact that like Fivetran wants to produce directly into the data warehouse means that all the microservices and all of the operational infrastructure basically doesn't get access to that data. And it still needs to be sort of like somehow made available to the places where the operational systems get that.

And similarly, with the reverse ETL world. I really wish that Census and Hightouch, for instance, would support getting data from messaging systems, versus the data warehouse. Because a lot of these operational systems like microservices, for instance, that need to produce usage information about billing, need to talk to Salesforce. Forcing that through the data warehouse puts this sort of like, one, it's really expensive compute. And two, it puts the system that doesn't have the same kind of SLA into the hot path.

And so, while these people have the connectors, they're not necessarily accessible to the community of people who need to build those kinds of applications, these operational systems, microservices that are event-driven. Like I said, certainly, a lot of this data does go into the data warehouse. I think I'm not anti-data warehouse. I'm just saying that like I think that operational systems, which, again, like a lot of the Salesforce, Marketo, Eloqua push notifications to customers and like updates around account information and those kinds of things. What you want is central governance. What you don't want is like central database. And I think that, increasingly, that becomes a challenge.

One, we try and limit our development of connectors to what we think of as being sort of critical. And I do think in five years – I think Salesforce, for instance, just started supporting Kafka. I think in five years, every one of the producers of data will natively support Kafka or something like it. I think all the consumers will natively support Kafka or something like it. And I think the database warehouse vendors, like Snowflake, and BigQuery, and like Redshift and all these other folks already have some capacity to ingest from things like Kafka. Apache Pino and Druid, for instance, and a lot of the new-breed low-latency analytical systems which we love already support native consumption from Kafka, or messaging-like systems.

And so, I truly think that we are already where the world is going to end up in a couple of years, where the connectors become less and less important because everybody is starting to sort of think about this as change events or append-only events on streams. But I think we watch this stuff closely. We build connectors.

For instance, we have like an S3 connector, because it's just so common. But like I think we're really thoughtful about where we do this. And because we don't have any of those proprietary formats, your entire Kafka connect ecosystem and all these other kinds of things already work with Decodable. We've managed to leverage what the community of people around these systems have already done.

[00:19:12] JM: You are managing the data pipelines yourself on your own server infrastructure. Correct?

[00:19:22] ES: That's right. Yeah.

[00:19:22] JM: Can you talk about the server abstractions that you're using to host the data pipelines. Like are you using Fargate containers or EC2 instances? Give us a context for the infrastructure.

[00:19:37] ES: Yeah. I mean, I'll tell you, it's a bear. We spent a lot of time thinking about this problem. Under the hood, there's a couple things that we need to do. We need to make sure that people get sort of the right kind of quality of service for each one of these pipelines. We need to be able to handle certain kinds of failures and limit the blast radius of those failures because these are low-latency streams.

Under the hood, Decodable is sliced up into cells. And each cell is limited to a certain number of customers. And so, basically, we sort of always play with these numbers. But let's assume, for instance, that we never let more than 50 customers into a cell. Under the hood, the cell is basically a set of Kubernetes names and spaces that form like a boundary of low-level access control that correspond potentially to node groups, or they could be shared over physical machines.

In AWS, for instance, we happen to use EKS, because EKS helps us sort of like deal with some of those kinds of things without sort of a whole lot of engineering effort. And then we pack tenants or customers into those cells. When you start up a connection or a pipeline in Decodable, what we're actually doing is allocating dedicated containers for that.

Startup can actually take a couple of seconds because we're sort of actually spinning up containers for that. And that cell is all the way down to the Kubernetes level of control. We have the ability, for instance, to say, "This cell is on this Kubernetes cluster." Because like I said, there's a control plane to the cell and the data plane to the cell. And each one of those corresponds to a Kubernetes namespace.

And then we can decide what cells or what namespaces are pinned to what nodes. And in certain cases, we can actually pack multiple cells onto one Kubernetes cluster. Or we could decide that this cell is so important it gets its own Kubernetes cluster. So that we're actually resilient. Or at least we limit the blast radius to the Kubernetes control plane and etcd and sort of like all the stuff that happens with Kubernetes.

And then the Kubernetes cluster is basically a one-to-one relationship with the low-level VPC infrastructure. For instance, this gives us the ability to decide, A, how many tenants do we pack into a cell? B, whether or not that cell is on a shared Kubernetes cluster or a single cell Kubernetes cluster. And C, what nodes that corresponds to? And D, the L2 network segment that those nodes sort of like have access to.

And so, in like an on-demand tier, we may pack many, many tenants together and they might share infrastructure. In the case of like sensitive customers, either for security reasons or for performance reasons, that might be a one-to-one-to-one-to-one mapping, where basically a single cell gets a dedicated L2 segment. And we attach other things to the notion of cell, for instance, credentials. We don't share credentials across cells. In, God help us, the terrible case where there's like a security incident, they're basically trapped within a cell. And they can only sort of operate within a cell.

Over time, we've talked about, for instance, like do we allow the data plane to run under the customer's vpc versus ours? And like maybe we've run the control plane. We've talked about

sort of different permutations of that. And there, that's just the place where we sort of listen to customers and sort of learn how they want to think about it. But we have a bunch of different levers about how densely we pack the cells. How many cells do we have? And we've designed it to be relatively cheap to provision additional cells, right?

I mean, I said, it's as cheap as multiple name spaces on a kube cluster. That's like the two-minute explanation of what the underlying infrastructure looks like. And that architecture, for us, has proven to be really resilient to like weird jobs that do terrible things that can control the outbreak of those kinds of things. Now, we have a much easier problem than low-level Flink, and Spark and those kinds of things. Because the only jobs that customers can express are via SQL. And we know what those operators look like. And we know how to control those things. There's not arbitrary code. And so, as a result, we have like already gotten around like a bunch of things where like somebody writes like a weird UDF or something like that. We don't really have those kinds of problems by design of like the product that we give out to people. That actually allows us to avoid a host of really, really complex problems in a multi-tenancy capacity.

[00:24:37] JM: Has the tires really been kicked on the system? Have there been some really large data workloads that you've thrown onto the system to see how it works?

[00:24:46] ES: Yeah. I mean, obviously, we do sort of the normal sort of enterprise burn-in testing and performance testing and those kinds of things. We also do regular things. Like, we're constantly like upgrading the underlying infrastructure and stuff like that, patching for security, updates and those kinds of things. Like at any given moment, there's always like a node restarting, or joining, or leaving a cluster or something like that. We do continuously deploy and sort of have different deployment gates as different versions of the runtime of the system make it out there. And then we obviously have customers that just beat it up with running lots of small pipelines, a small number of big pipelines, different combinations of those kinds of things.

And we do have different levels of sophistication of customers because, like I said, one of the things we're trying to do is make it easy for people to build these pipelines. If somebody has been building pipelines with like Airflow, and DBT, and Snowflake, they probably don't have low-level knowledge of like Flink, and streaming and those kinds of things. And so, sometimes we get some like pretty wacky SQL that does some really interesting stuff.

And I got to tell you, on the whole, we have seen the system be pretty resilient to this. And to be fair, a lot of that is the hard work of like the Flink community in dealing with those kinds of failures. I don't want to be unfair to the team. Certainly, the team at Decodable does an awful lot of work around state recovery, and job recovery, and sort of retry mechanisms, and like all sorts of deep tuning. But like that's our business. That's our value to people.

And so, if we're going to be good at what we are trying to be good at, it really is incumbent on us to be able to respond to those kinds of failures. We've seen – The unit of work for Decodable is the task, right? A pipeline is allocated in a certain number of tasks, or maximum number of tasks from the user. So you might say, "I'm going to process this amount of data. And I want to allow up to 10 concurrent tasks." And then the query engine is going to decide if it can parallelize up to 10 tasks. And if so, which operators get assigned to which nodes in that job graph?

And in those kinds of cases, we are able to sort of deal with about 80,000 to 100,000 events per second per task, or about 8 megabytes to 10 megabytes per second per task, whichever one you hit first. If you have like small events or large events that can change performance characteristics, obviously, the SQL can change performance characteristics. But that's sort of like our sort of what we think of as the unit of work.

And so, you can imagine people who show up with like a hundred tasks or a thousand tasks deep pipeline, really put some pretty serious work on the runtime. But myself and some of the other folks on the team have quite a bit of experience running Kafka, and Pulsar, and Kinesis, and Flink and those kinds of things at scale. And so, we have some pretty good idea about where things break down, and why, and what to look for, and the leading indicators and stuff like that. We've gotten pretty good at running Flink both at Decodable and from some of the previous places that we've worked to see the stuff at scale.

[00:28:25] JM: SQL is great. But there are probably a lot of people that have defined their pipelines imperatively. Is there some complexity or communication difficulty in defining for your users kind of the spec for how to define everything in SQL?

[00:28:48] ES: Yeah, that's a really good question. I mean, I think it depends on the audience. I think you're right, in that a lot of the more sophisticated shops who have been dealing with streaming for a longer period of time may already be thinking in like Apache Beam, or Flink, or Spark, or K Streams. And honestly, I think of those things are working for people, and they don't have challenges. Maybe we just don't try and convert those people. I think if they see value, then they decide.

And I think, as a vendor, I don't know that it's my business to sort of convince somebody that they're doing it wrong if they built some code that works for them. And maybe this is just a function of like us being an earlier stage company. There are so many people who are not the people that you're describing. That we already sort of have our hands full with just making them successful. And these are people who sort of start thinking about the world from the SQL perspective.

I do think, as we grow, we will encounter more people who say, "Well, I have to do this in like imperative code," in whatever weapon of choice they have, Beam, or K Streams, or Flink, who say like, "I have to be able to do this." That we may sort of gently say. And in fact, I think we've had some of these conversations where we kind of go like, "What did that job do?" And they're like, "Well, I filtered this and I transformed these fields." And you're like, "Okay. Do you really need to do that?" And they go, "No, not really." And so, there might be some amount of coaching people to rethink how they think about these pipelines.

To that end, we've come up with like a catalog of patterns that we see most commonly. And they use the verbs that the engineers, or the DIV engineers, or the application engineers use. This catalog, I think it's 12 patterns. And they're things like filter, enrich, trigger, aggregate. And like they can be sort of composed in sort of different ways.

And so, we sort of mapped out this catalog, and we said to people, "Well, if you need to filter and aggregate, here's what that looks like in SQL and in Decodable." And in a lot of cases, people say, "Oh, that makes sense." And they go and they do it. There are occasionally cases where people have imperative pipelines mostly because one small part of that pipeline is really complicated and really domain-specific. And then the rest of that pipeline is like the filter, transform, enrich. Basically, the simple stuff that can be expressed easily in SQL.

And for those people, I think there is a little bit of education to say, "Well, what if we actually split this into two pipelines with a stream between them so that the simple stuff is simple and the place where you truly need to do something really, really complicated."

For instance, if you were a real-time delivery company doing sort of like location prediction or sort of delivery prediction or something like that in terms of time, that's probably a thing that's like hard to express in SQL. But a lot of the feature extraction for that could be SQL. And so, that's a case where we go like, "Well, maybe we split this in two," and we sort of like leverage Decodable for like the simple stuff that it's not differentiated. And then for the crazy stuff that is like specific to your business, we don't want to get in the way of that. And for that, you just drop down basically into Flink, or Spark, or K Streams or whatever is that you're using. And that has actually resonated with people. Resonated – Listen to me. I'm like a terrible CEO now. That makes sense to people. And I think that they kind of go, "Oh! So then like I actually don't have to deal with all the details."

We'll continue to see whether there are more cases that we can absorb. But ultimately, we want to be in a position to help people not try and convince them. Not start like a religious war about the right way philosophically to think about their pipelines.

We have found, like I said, enough people are sort of aligned to the way that we think the world works. And admittedly, what we think is based on our own experience sort of running these systems for like internal use cases and stuff like that at companies, like, consumer, businesses, and enterprise businesses and stuff. And so, to some degree, we're sort of building the kinds of systems that we would like to be able to use at some of those companies. And I think that lands well with people. Not kind of like being pushy about what they use the product for. But it's a really interesting question. And I think we'll have to continue to see how that develops, though. Good stuff.

[00:33:43] JM: When you look at the way that teams are structured these days, the breakdown between data engineers, and data scientists, and ML engineers, there's kind of, I guess, a shifting of responsibility that is happening as tools get higher level and easier to work with. And I'm wondering if you feel like Decodable reduces the workload on the data engineering teams.

And maybe you can just talk about how – Well, I mean, I think not just Decodable, but companies like Fivetran that reduce a lot of the effort that previously would have been the responsibility of the data engineer, how responsibilities shift in the data teams.

[00:34:34] ES: Yeah. I mean, our two communities that we think about are the application developers building like event-driven microservices. And like you're talking about the sort of like analytical data teams that are the data engineers or analytics engineers, data science, and sort of like increasingly a bunch of different specialized functions there.

My sense is that this isn't necessarily about putting people out of jobs. It's not that we think that like data engineers shouldn't exist. And in fact, the thing that, for instance, we do – And I think the thing that Fivetran does and companies like that, although I won't claim to be an expert on their business. The thing that I think they're doing is they're actually removing the burden on the central data platform team, the people who know the lowest level sort of details about database infrastructure and the state management and sort of like those kinds of things about like job recovery and those kinds of things.

I still think that tools like Decodable, Fivetran, sort of Airbyte, that whole ecosystem of things. Some of which are optimized for different use cases. I don't think we actually necessarily compete with like a Fivetran or an Airbyte. But I really think what we're doing is we're allowing the data engineers and that sort of like whole cluster of folks to be able to do more in the same amount of time. Because it's not like there's like a fixed number of pipelines.

I think the work of the data engineering team sort of like grows to fill the capacity of like what the business can throw at it. I would argue that for every data engineer you hire, you find like 10 new things that you want to do with data. Or sort of like all the customers that are on the other side of the data engineers, whether it's data science, domain experts like who aren't maybe even engineers who are sort of like the data engineers are fulfilling the building of different kinds of like data products for, I don't know, civil engineers or for whomever it is that they're sort of supporting, or the business users beyond that. My perspective is that what we are doing is we're making them more efficient.

Now, I think there is something interesting that sort of happens. One of the things that we had a conversation about just the other day at Decodable is there was a time where an application developer could not do anything with the database without a DBA. And like I don't know exactly when it happened. But I don't know a lot of DBAs anymore, except for like in sort of like the most high-demand sort of database systems.

I think, increasingly, we've actually changed the way applications use things like Postgres and MySQL, mostly the operational database systems, where like people spin those things up on their own and do relatively simple operations. But they do a lot of them. I don't think that data engineers are going to get squeezed the way that DBAs got squeezed by the development of systems like Decodable. Because I don't think data engineers should be or even are in the business of building these maintaining like Flink runtimes and like dealing with like class or nonsense. I think they prefer to spend more of their time building pipelines and thinking about how data is used, versus building data infrastructure.

Now, maybe I'm wrong about that. But my experience has been that, at many of the typical Fortune 500 businesses, they'd rather be thinking about banking, and insurance, and retail than thinking about messaging systems and stream processing engines. And so, I think our job is to make them more productive versus like limit the work that they do.

And I think we're actually doing that. I mean, I say that reluctantly, because vendors always think that they're helping people. But I think, practically, on the ground, data engineers can do more, which means that all of their customers downstream that the teams that they support can do more. And I think that that is – I mean, that's the goal, right? That's, to some extent, the effect that we're trying to achieve. And if they spend less time on what I would call undifferentiated scut work, right? Like cleaning up data sets is not the thing that makes Lyft or Uber Uber. It's really the marketplace of rider-driver pairing systems, and customer safety systems, and rider prediction systems and those kinds of things that make them who they are.

And so, I think, over time, we'd like to be in a world where people don't have to build at that level. If we can make the current work, which feels like assembly code more like C code or even more like Java or Python code, I think using that as an analogy. Of course, if we can make their jobs a little bit easier, then we see that as a net win. And I think that that's what's happening.

And I think we will see more interesting applications of data hopefully for the better of retail users or insurance applicants and so on and so forth. I think that we're on sort of a healthy path to be able to support them and make them more productive.

[00:40:31] JM: As we begin to close off, just to drive home the point of what you're building, can you walk me through the life of a SQL query executed on Decodable?

[00:40:42] ES: Yeah, absolutely. I mean, at Decodable, there are really two different things that people wind up doing. One is they configure connections, either source connections or sync connections to like a Kafka topic, or an S3 bucket or something like that. Once that's done, what people get is a stream. And that stream sort of feels like basically a Kafka topic that can be used by any number of connections or pipelines within Decodable. And then they write their chunk of SQL. And a pipeline is a single insert into select from SQL statement, which in turn compiles down to a Flink job. And I say compile those down to sort of in the loosest sense. There's quite a bit of like SQL parsing and planning and those kinds of things, some of which we leverage the pre-existing some Flink code, and Apache Calcite and those kinds of things.

But eventually, we have a catalog. Basically, a metadata catalog similar to like the information schema inside of like a Postgres, or a MySQL or something like that that tells us about the streams, and their schemas and all these other kinds of things. And so when you click save, what we're actually doing is we're loading that metadata. We're doing the parsing and planning. We are looking at the metadata we have about streams. We're doing schema mapping and compatibility checks to make sure that, for instance, we will not let you save a SQL query that references a column that doesn't exist in a stream, or thinks about it as the wrong data type, those kinds of checks.

And once we have made sure that all that is true, we actually store that SQL plan, that that query plan along with a whole bunch of like metadata. For instance, we extract the streams that you've referenced for dependency information and lineage information, those kinds of things. And either activate that pipeline. Or once if it's running already, you can deactivate it and stuff like that. And what deactivation or start stop is, is actually launching it.

Once it's saved and planned, we know that that's sane, and it's always going to work. And when you activate it, that's when we actually launched the job and actually start producing things. And that's where you get into the low-level detail of launching the underlying Kubernetes containers and doing any state restoration. If that job is being restarted, we'll load in all sort of the previous state information and resume from the last offset that we processed. And we do all the exactly once processing shenanigans. There's lots of idempotency that sort of like we have to handle there around job recovery and recovering from sort of the right places and bringing in sort of any state around aggregation functions, and enrichments, and window functions and those kinds of things.

But the net result for somebody is that they type some SQL, they hit activate and data starts flowing. And then once the pipeline's running, there's a whole bunch of infrastructure about getting metrics off of it to make sure that it's healthy and what the performance is. When you log into Decodable, you see things like events per second and bytes per second, in and out and like those kinds of details that tell you more about your pipeline. And you'll continue to see more, I'll just say, stuff from us to tell you about things like data quality and those kinds of things from pipelines in the future. We don't do enough of that just yet. But we'll do more of it. And then it's kind of off and running from there.

That's sort of the life cycle of building and deploying a pipeline with Decodable. There's all sorts of different ways that you can do it by the application, or via APIs, APIs or you know declaratively in some files and stuff like that that feels more like kubectl if you're a Kubernetes person. But we try and make it as simple and error-proof as humanly possible. But that's it.

[00:44:41] JM: Awesome. Well, that seems like a good place to close off. Anybody building streaming data pipelines should obviously come hit you up. And thanks for coming the show again.

[00:44:50] ES: Thanks it's a real pleasure. Long-time fan of the show. Really excited to get a chance to talk to you about it. Appreciate the time.

[END]

