EPISODE 1308

[INTRODUCTION]

[00:00:00] JM: Prophecy is a complete low-code data engineering platform for the enterprise. Prophecy enables all your teams on Apache Spark with a unique low-code designer. While you visually build your data flows, Prophecy generates high-quality Spark code on Git. Then you can schedule Spark workflows with Prophecy's low-code Airflow. Not only that, Prophecy provides end-to-end visibility into your data flows with metadata search and column level lineage. For enterprises, in addition to developing new workflows, data teams also need to migrate thousands of old proprietary ETL workflows to the cloud. For that, Prophecy has built a transpiler that automatically converts Ab Initio, Informatica, SSIS and Alteryx workflows to high-quality Spark code. Learn more at www.prophecy.io.

In this episode, we speak with Raj Bains, who is the founder and CEO of Prophecy. Previously, Raj was the product manager of Apache Hive at Hortonworks through the IPO. He also headed product management and marketing for a new SQL database startup. Raj continues to actively code and compiler and database technologies. His engineering roles include developing a new SQL database, building CUDA at NVIDIA as a founding engineer, and working as a compiler engineer for Microsoft Visual Studio. Full disclosure, Prophecy is a sponsor of Software Engineering Daily.

[INTERVIEW]

[00:01:22] JM: Raj, welcome to the show.

[00:01:24] RB: Thanks, Jeff. Super excited to be here, and I look forward to a fascinating conversation.

[00:01:29] JM: Let's start with a brief history of data engineering. So my perspective on modern data engineering is if you want to find a meaningful place to start the timeline, my opinion is the best place to start the timeline is the Hadoop MapReduce paper. I would love to get your perspective on why MapReduce, and perhaps to a lesser extent, the Google file system paper. I

think that's the other one that's relevant here. Why did those products, those papers have such a massive influence on the history of software?

[00:02:00] RB: Actually, I think to answer that, I will perhaps start a little bit sooner. So I'm coming from the old system space from before Google MapReduce paper. And I've seen the world before and after that. And if I were to look back, the ETL tools – And I've primarily worked with enterprises. The ETL tools have had Ab Initio and Informatica, where Ab Initio is something that worked on large amounts of data for large banks, fortune 500 companies. So it is really been the performance leader. And Ab Initio performance execution engine is very similar to Apache Spark. And that's what has been used for ETL by a lot of enterprises. And then Informatica was the volume leader. And that's a $10 billion industry in that.

Now, the second part is that then you have the enterprise data warehouse with Teradata as the leader. And Terdata started in ‘79. In ‘83, Christmas, Wells Fargo was running a lot of their data on Teradata. And by ‘86, it was company of the year. So at this point, in middle 80s, the enterprise data engineering space, ETL as that was called, was set up, right? And then, coming to your question, what has happened is that as you had Google and other people come in, they had higher volumes of data. They had higher volumes of data, where per unit data, there was lower value. So they wanted more commoditized solution. They wanted to run it on commodity hardware, which wasn't so expensive. They were not going to pay Ab Initio and Teradata millions of dollars. So they built their own thing.

I think the lasting impact of having Google file system is you keep multiple copies around, you get cheap storage. And the MapReduce is essentially – The long-term impact is that in a database, when you're processing a query, you're going through the filter, join, aggregate you're streaming data, and the long term impact of MapReduce is, in the middle of that pipeline, you store the data on file. So it's easier to recover. I think if you look at it in the long arc of history, if you look at what today Spark looks like, what Snowflake looks like, they've got a traditional database optimizer, a traditional database execution engine. Both of them have SQL optimizers. So the impact is it seems it has been more, but unfortunately, like I believe it has not been as much as people make it out to be. Hadoop companies today are nowhere, right? It's already a legacy technology.

[00:04:52] JM: That said, the maturation process of the Hadoop companies has been okay. I mean, they kind of moved their focus to Sparke a little bit. They kind of move their focus to developing internal technologies, or there's a number of open source projects that came out of the Hadoop companies. I mean, what is sort of the corporate legacy of the Hadoop companies? You worked at Hortonworks. How do you think those companies stand today? And what is their impact on the industry?

[00:05:18] RB: Yes. So I worked on Hadoop. I was the product manager of Apache Hive. And Hive was the most used product in the Hadoop stack. A lot of the revenue, or I'll say vast majority of revenue for Hortonworks –

[00:05:31] JM: Can you do a quick review on what Hive is?

[00:05:34] RB: Sure. So Hive is – So as people started to use Hadoop, MapReduce was quite hard to use. So Hive added a SQL layer on top of Hadoop. So it was designed for high-throughput. So it was primarily designed for ETL at really hard, large volumes, where traditional ETL tools didn't work. Right? And the entire sales motion for Cloudera and Hortonworks was we went to customers who had Teradata, and we said, "You have this very expensive product. 70% of the capacity of Teradata is being used for ETL. And for that, you don't need the Teradata architecture. You don't need the caches. You not need the indexing. So why don't you move that workload to Apache Hive, a product that is built for high-throughput?" And that was the entire sales motion. So Hortonworks, Cloudera, as billion dollar companies were built on offloading ETL from Teradata, which is not as sexy, right? But that's the truth of it.

Now, as we go forward – And the other products in the Hadoop stack were used somewhat. Now, the people who did build Hadoop did not read as much research. I remember when I went into Hortonworks, we had to throw away the query optimizer of Hive because it was written as a hack. And then it was like, "That's not how query optimizers are written. This is 20 year old research." So then Hive was improved. Significantly, MapReduce was thrown out and a new engine was put in place, and it has evolved to be more and more like an MPP database, and of course with slightly different characteristics of storing those intermediate results that are talked about. But then if you look at the impact today, there is a consensus as we talk to large enterprises that Hadoop is not the future. People want to move from Hadoop or Hive to spark

even though Hive is used significantly more. And in the overall scheme of things, today, Hadoop stands as a legacy product. Same as the other on-premise legacy products. These companies never invested enough in R&D. They never had the margins to do that. And they have fallen behind. And they don't have the engineering sense to come back. So they slowly withered away is my take on that.

[00:07:55] JM: All right, very interesting. So I have been inspecting the data engineering space since the beginning. And I have noticed, as I've said before in a few different episodes, the biggest surprise to me was the fact that Snowflake and Redshift became these sources have massive data gravity. And a lot of people have built their systems around Snowflake and Redshift. And that surprised me because I always thought that the streaming systems looked like they were better abstractions for doing data engineering, like Flinks of the world, the Apache Storms of the world, even Samza really. Like I see these things as very interesting glue for building a distributed system. Or you look even more nuanced, the Apache Beam project, I think. But really, the problem is, is that these things are hard to work with. And they're hard to reason about. And so ultimately, our fix to that as an industry has been to move towards Redshift and Snowflake simply because it's easier to understand like what is the materialized view you're getting in a given moment. Do you think that's accurate? Is that an accurate perspective?

[00:09:09] RB: Yeah. So first, I would say that I am surprised that everybody's surprised. Right? Because –

[00:09:16] JM: You agree everybody is surprised though, right?

[00:09:19] RB: I agree, everybody is surprised. And I am hugely surprised by that, because like I told you, right? So Ab Initio and Informatica, the ETL tools industry was $10 billion. And then the data warehouse was $20 billion, right? And you had Teradata and IBM Db2. And if you look at the on-premise footprint, it is one thing that looks like Spark and one thing that looks like Snowflake, and that's what every large enterprise looks like. And the large enterprises run massive workloads. They don't have like a hundred workflows that you can schedule with Airflow. They have tens of thousands, to hundred thousand plus. We are working with a fortune four bank. They have more than 100,000 workflows that run every day, right?

So if you look at that, basically what has happened in the cloud is an exact replica of that. You got Spark, which is very much like Ab Initio. And you got Snowflake, which is very much like Teradata. And now within these, if you look at Teradata, you would say, “Hey, enterprise data warehouse was a $20 billion industry on-premise. And it was one of the largest technology markets.” But 10 billion of that is data engineering or ETL, and 10 billion of the Ab Initio. So $20 billion is just data in engineering on-premise, right? So that's what people have used. So the first thing is there is no surprise enterprise data warehouse would become one of the primary technologies. So having ending up with Spark and enterprise data warehouses, the two technologies, that is normal and to be expected.

The other thing is about streaming, right? Streaming is much harder to do. Streaming also doesn't make sense unless you can do something with the results, right? When you're working, let's say you are a business, you are a bank, you are an insurance company, right? It's like, okay, you got your data for the last four years till. And that is, let's say, four hours old that you ran a batch workflow on. Now, if you add to that the data of the last four hours, what is the new decision that you are going to take on that? Hardly anything, right?

So streaming will not become big, because streaming does not have a real business use case, except for narrow use cases. Some, you're doing some machine learning real time, finding fraud or something. So there are those narrow use cases outside of that. If you don't need to do streaming, you should not do streaming, because streaming has out of order events. All these extra issues you have to deal with. It's also not as cost efficient, because for me to run a batch job which processes like yesterday's data in one go is much more cost efficient. It's simpler to do. And if something goes wrong, it's simpler to recover from.

So that said, Kafka is becoming more popular as for certain kinds of things. And I think we'll continue to see that. But streaming is this technology, which over 15 years has stayed on the fringe. People have thought it's going to become mainstream. No. No. No. It never does. I don't think it will in the next 10 years. I do think Kafka will become more important. But apart from that, it will stay on the fringes.

[00:12:36] JM: It feels it feels to me like it's really early days for streaming in kind of a powerful way. I just did two shows about Pulsar. They're going to air in the next few weeks, and it kind of blew my mind, because I had thrown away Pulsar in my brain as being second fiddle to Kafka. And it's actually not. So the thing about Pulsar is it's a distributed queue to the same extent that Kafka is a distributed queue, but it has configurable storage. So if you think about it, like Kafka is sort of the Java of the distributed queue. It's garbage collected, basically, I think. Garbage collector in the sense that you don't really think about storage in Kafka. The storage and the memory don't have a deep abstraction. In Pulsar, you have Apache Bookkeeper. And Bookkeeper is highly configurable storage abstraction. And because you have a highly configurable storage abstraction, you can build multi-data center replication more easily. So I just found it to be a very fascinating case study in what I would argue is data infrastructure. I think Kafka is kind of this data infrastructure middleware thing.

[00:13:35] RB: Yes, yes, it is. And I've had some other friends who have looked at Pulsar. One person by the name of Simba, who's written some blogs about it. So there has been a lot of – So I've heard from some people that it's very interesting. It's not something that I know too much about. But streaming is always exciting, but we don't deal with that much streaming in our thing. We deal with some Spark streaming, but not much.

[00:13:59] JM: It makes for great podcast episodes, I'll tell you that, because streaming is really interesting. Yeah. So anyway, we've been building up context. Like we've kind of given a bit of your history and sort of how you think about things. So I saw what you're building with Prophecy. And it's definitely taking an approach that makes sense, which is essentially low-code data engineering. You have a low code layer on top. You have a data engineering system underneath. And to me, this is almost an obvious problem. I have seen other companies attacking this in a similar fashion to me, but it’s almost, to me, this is sort of like it's like a Hadoop era kind of thing, except it's bigger, where we're basically saying, “Look, we know we all need to build data engineering systems. We know they need to be easier to build. We know we need to have people that are essentially like data analyst level talent doing data engineering. Like that would be the ideal if they had a data engineering operating system that they could play with. And that's what kind of low-code enables. So I think if you think about it connectively, we've solved data engineering from the standpoint of now we have the Unix level abstractions of data engineering. We have the Kafka, and Fivetran, and Hightouch. We have like reverse and

forward ETL. We have storage at all areas with robust abstractions. We have queueing with robust abstractions. We can essentially do distributed systems data engineering without Paxos level boredom. And Prophecy is able to also exploit what I call the React revolution. I assume you're built-in React, right? On the frontend?

[00:15:34] RB: Yes, built-in React, definitely. How I would characterize the space of data engineering is that, one, there is a stark difference between the startup ecosystem and the enterprise ecosystem. We happen to be focused on enterprise ecosystem. The use cases end up being quite different actually. That said, how I would put it is that for data engineering, the processing engines are there, right? So Spark is there, Snowflake is there, Kafka is there. These are all processing engines. And now you can write code on that. But on top of that, the data engineering product, if you look at somebody who's been on-premise, they have a visual drag and drop designer. They have metadata management, column level lineage, scheduling, CI/CD, sharing. Like none of that really exists, right? So where is the Snowflake of data engineering product on top? That you land your data into S3 from operational systems in an enterprise. Now you have to do all these transformations. Let's say you do them on Spark. Then you load the data into their data warehouse, and you manage your views and everything. And you can see lineage across all of them. You can search your data sets. Like if you look at all of that and you say, “How do I do that?” Nothing really exists, right?

Let me give you another example. We had a customer come in recently who said, “Hey, we are looking to move from Ab Initio to Databrix. And we have 22,000 workflows that we need to move in the first go.” And we’re like, “Okay.” And they're like, “How do we schedule?” They're like, “Oh, the simple ones use Cron and the more complex ones use Airflow.” But Airflow doesn't run. Astronomer is improving that. But Airflow doesn't run for more than a thousand workflows. And what are you going to do? Write a thousand schedules one by one, right? With error recovery in each one of them? Works for a startup, right? You have a hundred workflows? Great. You have a small setup? Great. You have 22,000 workflows? Absolutely does not work. It does not work at the engine and the processing level, because Airflow cannot run more than a thousand workflows.

The second thing is the tooling on top. Like there's just nothing there. Like how do you manage 22,000 workflows? You're going to manually write Airflow 22,000 workflows each with recovery

in this and manage dependencies between them? You can't do that. So to your point, the processing engines are there. On top of that, it's in its infancy. The kind of tools, the kinds of products you need for data management, for data engineering management just don't exist. And 90% plus of the data engineering is not in the cloud. It is sitting on-premise. And everything that you – The narrative is very different. Let me give you another interesting anecdote. I was reading through the Snowflake S1. The Snowflake S1 says they have somewhere like – If I recall the number correctly, somewhere like 50 to 60 companies that pay them more than a million in revenue. And Snowflake charges for the hardware as well. Now you tell me which enterprise can do data engineering in under a million dollars. Nobody can.

For us, it is normal to see a top four bank would spend something like $20, $30 million just on licensing or software for data engineering. They probably spent $40 million on data engineering a year. So if you look at that versus you look at – So majority of the data engineering today sits on-premise, because there is not enough product in the cloud. It's just the processing engines you need a lot more on top. And that's what we're building.

[00:19:14] JM: Yeah, it's really interesting. So I'll just be blunt, and say my key concern about this kind of company is essentially the N-by-N problem. I mean, you've got these abstractions that are so complicated. I mean, the API's are at least well-defined at this point. But to me, it seems like you have to build a shim over each of these data engineering things. So you have to unify them somehow, or have a common language for integrating them. And I'm just wondering, like, okay, let's say I build a low-code workflow in Prophecy. How is that materialized into infrastructure? What happens there?

[00:19:55] RB: Sure, definitely. So let's talk about that, right? First is that the NxN problem comes if you're trying to target every infrastructure, right? But the big thing that's happening in the cloud is consolidation. There is consolidation around Databrix and Snowflake. So as far as we are concerned for us as a startup, there are two ecosystems that we care about. If you're not one of them, we don't really care. I mean, okay, you could be Redshift or this, but that's pretty much the same as Snowflake for us. So first there is convergence. So we don't need to support 20 different things.

The next thing is, so we as a data engineering product sit as a layer on top of your existing systems. So what that means is you come in, you say, “I'm going to do drag and drop, and I'm going to build a workflow.” Data flows are very, very easy to build on a visual canvas. I say, “Here's my source. Here's my join. Here's my filter. Here's my this.” As you're doing that, Prophecy, side by side, is generating very high quality readable code on Git, and it is on your Git. You can take away Prophecy and run it, right? So it's code on your Git. As you're running it, when you're hitting play, you're running it on Spark. So you hit play. After every transform, you can see the data before and you can see the data after. And that is running against your live Spark cluster. That's running on your database, your EMR, whatever your Spark is, right? And then you still can now load this into Snowflake, and it will load it into Snowflake, right? So that's our common customer use case. So we are sitting on top of that. If you want Airflow for scheduling, we will say, “Here is – Just point us to your Airflow. And here is low-code development on that. We can deploy on that, monitor on that basically.” So you just have to go to one of your existing workflows that you just developed and say, “Schedule this one.” And you can wait for some data to show up in S3, then run this workflow, then this workflow, then this workflow. Again, you can visually draw that. And side by side, you're getting very high-quality Airflow code that's being generated on your Git, right? So everything we are doing, we're computing linear and storing it on your Git. So basically, what we are doing is helping you write code on your Git. The only thing is that to develop this code, you don't have to know coding. You can do visual drag and drop and SQL expressions. And that's all you need.

[00:22:13] JM: What's been the hardest engineering problem you’ve had to solve so far?

[00:22:17] RB: We solve some very interesting problems. So one thing that we have done is – So first thing is that as you do visual drag and drop development in Prophecy, we are generating high-quality readable code side by side. This code structure is where you can see the control flow very quickly. And for each of these, it's very readable. It usually ends up being better than the code person can write by hand. The second thing you can do is now you can – You have a code tab. You click the code tab. You go into the code. And now you can edit the code. And you go back to visual and we pass this code. So now, all the changes you made in the code are visible in the visual one. So now we have said, “Hey, this visual drag and drop development of data flow is encoding. We've made it the same thing,” right? So we have these parsers.

But now what we are saying is, “Wait, wait, wait, all our users want to extend this,” right? They are going to be like, “Here's my data quality library, and I want to standardize on that. And I want to roll it out to my customers.” To do that, they can define their own transforms. So when they're defining their transforms, we are generating the code generators and the parsers from their spec. So doing some really complicated stuff there, our metadata system now merges the metadata and code. And that has been quite an interesting thing to build, because it merges the world of code and data, and a lot of the metadata just sits on Git, right? When we are computing column level lineage, we are parsing the Spark code and computing lineage from the Spark code, then aggregating it, then serving it in a very fast fashion. We are doing search. So you can search the contents of this. And we'll point and say, “Hey, here is the expression where you use that particular column,” right? So we are searching the entire content of your Git. So we so we are doing that also.

There's another thing that we do. Another thing we do is we take the legacy ETL products, right? And we have transpilers that can convert them to high-quality Spark code. So for example, if you were to say, “Here is my workflow in Ab Initio. Can you convert it to Spark?” Yes, we can. Informatica? Yes, we can. Alteryx? Yes, we can.

So what we have done is we have written cross compilers, where we've reverse engineered the formats of these, including Ab Initio, which was a very hard one. They don't even have documentation in the public domain. And nobody in our team has ever used it. So we reverse engineered first their format. Then we reverse engineered their programming language, which is not even completely relational. Then we have compiler analyses and optimizations that converts it into high-quality readable Spark code. So lots and lots of sophisticated technology under the hood.

[00:25:02] JM: Why couldn't this have been built earlier? I feel like people have been trying to low-code data engineering for a while. It's been really hard. Why hasn't it been done earlier?

[00:25:13] RB: There is just not the skill set to do this kind of thing in the tooling layer. So there is two kinds of engineers typically. One is like the engineering engineers who are – Think of the engineering culture. Think of Google. Think of many other companies, like that who have a very, very engineering-focused culture, right? So what they do is they take a problem that is complex

and they come up with a solution that is equally complex, right? So think Kubernetes, think TensorFlow, right? It's like complex solution to a complex problem. The usability is really low. In tooling, what you need is more an often Apple-like approach, where the technology is complex, but the product is simple to use.

Now, the other thing is some people have tried to build low-code products, but there're just not enough technical chops. Like we are doing a whole bunch of stuff. So for example, when you're running a Spark workflow, we go inside the Spark workflow when you say, “Hey, I'm running in my ID,” and you hit play. After every step, you can see the intermediate data, few sample rows. You do not to get that? We go inside the Spark’s query planner, we insert our own operators, logical operators, physical operators, then we use the Spark stat mechanism over WebSocket to send the data back from executives to masters. It never touches disk, and you'll get the results back in three seconds.

So to build this product, I've had to know compilers in terms of the C compilers of the world. I've had to know database internals and how to build databases. And I had to know how to do design, which is basically saying that, “Can we take this really complex technology and wrap it up into a UI that is very simple for –” Somebody, a data analyst who uses Alteryx can do it. So it requires a lot of different skills to build this. And the other thing is who's going to fund it? Right, if you look at any big piece of technology, LinkedIn is going to fund it. They are going to fund building off a processing engine. They might fund a Kafka. Yahoo might fund a Hadoop. What is the usability of those products? Extremely low, because they're built for the engineers. There is no path today in the venture industry and in the software industry to put a good amount of funding into a design product, because you are going to need years of investment, right?

So for example, even a Fivetran took five years before series A. DBT probably took four or five years before series A. Why? Because nobody's funding it, right? So that's the second structural problem, is there is not enough funding. Because, you see, the processing engines get funded. They require a lot of R&D, but they get funded by the big tech companies, the LinkedIn, the Googles, the Yahoo's. Well, the tooling, design products, these companies are not funding it. Who will fund it?

[00:28:00] JM: I just looked at your investors. You got SignalFire as an investor. Was Elaine your lead?

[00:28:05] RB: Ilya is the lead.

[00:28:07] JM: Who is it? Ilya?

[00:28:08] RB: Ilya, yeah. Ilya Kirnos.

[00:28:10] JM: I don't know him.

[00:28:11] RB: And Ilya is one of the founders of SignalFire, right? And he's an ex-Googler. Amazing guy. And they funded us quite a lot actually. So yeah –

[00:28:20] JM: This is a great vision. I love your vision here. Nobody's doing data engineering Apple. Like Apple for data engineering is such a great idea. It's such a pure vision.

[00:28:31] RB: Yes. And we're super excited about that. Yeah, we think that usability is so super important. And part of it comes from the empathy, right? As a Hive product manager, I would go sit in front of the customer, and I can see the customer struggle to get the work done, right? And it's like, "No. It has to just work. It has to be super easy to use." It just works. Good design.

[00:28:54] JM: What stands between you and series A level traction?

[00:29:00] RB: We'll raise series A in a couple of months. We're just waiting for the – VCs will come back after August. So we are working. It's been a couple of years since we took a seed round. But we've taken like above 7 million. So it's a big seed round. And we have a big team. Now we have certain customers we are working with. So this way, like our first customer is, again, a fortune 500 company that you might associate with credit cards. One of the biggest, most prominent ones in the world. We are now working with top four bank, a top entertainment company, a top insurer. When I say top, means it's probably in fortune 50, or fortune 100. So it just takes six to nine months for these contracts to close. There are so many moving pieces. And because we started at the top end of enterprises, it took us a while. Now the first customers

are becoming successful. We're also starting to – We are like, “Okay, now that we can get these 250K to 750K deals, let us come down market and try to get these 50K deals, right? 25, 50K deals,” which is basically trying to show faster, or more repeatability, which is what typically people look for series A. So I think we will be there in eight weeks. We've launched the SaaS version, which actually became usable kind of at Spark Summit, which is just this end of May. So we have some customers closing now. So we're just doing paperwork with a few customers. I think in series A, we'd be there in eight weeks or so. We'll start raising.

And one last thing is like we’ll have named like nobody else does, right? So we would have like five fortune 500 companies as the first five customers, and then some slightly smaller ones, but no startup. Not Databrix. Not Snowflake. Nobody gets those customers as the first five customers.

[00:30:46] JM: Are you feeling any roadblocks? Are you feeling – Because I look at your business. Actually, I look at – The timing is epic. Your timing is so good, because people are starting to realize the whole power of React. But I don't think people realized it. So I just wrote this book about Facebook called Move Fast. There's a big part of that book about React. And I don't think people understand how important React is. It's extremely important.

[00:31:13] RB: Yes.

[00:31:13] JM: Right?

[00:31:14] RB: For us, we are building a large UI system. It's extremely complex, right? Because you're going and building these what we call gems, right? You're drawing on a canvas with gems, and these visual drag and drop gems are there. There’s complex state there. Then you click and you go into a code editor that's actually an embedded **[inaudible 00:31:32]**, which is the Visual Studio code. So we are running that. It's a complex application, right? And then we are generating from a spec these gems and the UI is for that, right? Because like I said, the users can come in put in their own library. So most of our UI is actually generated from this spec, which is – So some super exciting and hard stuff to do there. And then we're getting autocomplete where we are running a language server backend, which is saying, “Hey, it looks like you're trying to write this expression,” right? It understands the language. It understand

Spark. It understands the columns that are flowing through at this point in time. Our expression builder that helps you build expressions. So for example, if in the spec you said, “Hey, this thing is an expression and uses my incoming columns. When the user goes to type there, an expression builder will pop up, and it will say, “It looks like you're trying to write a concat function. Oh, in the concat function, do you want to use one of the columns that you already have?” So it understands all of those. And to have these components, like without React, we could not build such a complex application that we have been able to build. It's just like this is not like the regular webpage UI. Our UI is extremely complex. We have graphs. We have sub-graphs. And there’s lineage, which is showing you columns flowing through these. So some really, really sophisticated and hard stuff. So we've been using React a lot. We've moved to Typescript, just because like writing such complex UI is just not something that you can do without that kind of thing. So pre-React, we would have really, really struggled to build this.

[00:33:11] JM: Yeah, and that's why I see your timing is so pure here, because the data – I mean, look at Confluent, right? Confluent just IPO’d, right? Like when Confluent got started, Kafka was really, really tough, right? Like that's a hard product to operationalize.

[00:33:27] RB: That's a hard product, for sure.

[00:33:29] JM: What role does Kafka play in data engineering these days?

[00:33:33] RB: So these days, as we're talking to customers, many of them, as they're moving to the cloud, they're saying that, “I'm going to land data,” right? Some of them are landing data from their on-premise system to S3. But some are landing it to Kafka. Then you do a bunch of transforms. Then you land it to the next Kafka, right? So in that sense of transform, where you have multiple publishers and multiple subscribers, it's just very easy to say, “Hey, here is the next data. Every four hours, you post it to Kafka.” And then all the downstream consumers who want it can just subscribe to it, pull the data off. And then if one of those systems goes offline, you can just go replay from the Kafka commit log and rebuild the thing that you wanted to build, right? And Kafka has become so central. For example, for our own metadata system, right? The backbone of that is Kafka. So for example, metadata seems like a simple problem. But every time you change a workflow, there is a Kafka event that says, “Hey, this workflow changed.” Our lineage service is listening to that, and it says, “Oh, workflow changed? I must recompute

lineage." It recomputes the lineage. Puts the new lineage there. The third service will pick it up and say, "Hey, lineage changed. I must update my search indexes," right? The execution or scheduling service will look at this and say, "Hey, looks like the workflow changed. This workflow is scheduled to run at 9am tomorrow. I must re compute the JAR and redeploy it to my Airflow."

So Kafka is that backbone, right? So the two use cases I talked about is one is Kafka being used as the data backbone between workflows where you have multiple publishers and subscribers. And we're seeing that pattern used more and more. The second is Kafka as the central eventing system, where all your various – You can have this decoupled microservice architecture sitting on top of Kafka, as we do in our metadata systems. So we are seeing both use cases. Yeah, and Kafka is, I think, having an impact on both. One is more data. One is more coordination if it may.

[00:35:39] JM: Your company, do you have to play with Kubernetes in your company? Or are you more working with managed services?

[00:35:47] RB: Oh, so our entire software, right? So let's talk about our software stack. So our software stack at the bottom, we are written as a Kubernetes operator. We have some customers who are on-premise and some customers who are on the cloud. And most of our customers – Since a lot of them in the beginning have been banks and financial institutions, we have to run within their network, because we are dealing with a very sensitive data. So we need to install there.

So at the bottom, we are written as a Kubernetes operator that's written in Go. On top of that we have our microservices layered that is written in Scala. And that's because we have to do a lot of compiler work, a lot of – So it's just functional programming. And our character is Kafka, and Reactive program. Yeah, Scala and Reactive programming is like the core of our microservices. And then we have the heavy UI, or the really complex UI, which is built using React, right? And so we use Kubernetes a lot. We install in different customer environments. And if you want the – For a customer who was on AWS, we just say, "Hey, spin up your EKs, and then we'll install on top of that." So that's typically how we work. But we hope that Kubernetes would have more standardization. It is still tricky, because we have some system services like for monitoring and logging, etc. And then when you go to a customer, they're like, "Oh, no. Use us." And you're like,

“Oh, I want to use Elasticsearch.” And they're like, “Oh, use ours.” Or they'll say use your own. So there is a little bit of back and forth there. But yes, we have been using Kubernetes. Another interesting anecdote is like some companies are very comfortable with us having an operator. Some companies are like, No, no, no. We want the YAML files that we can inspect. We don't like this operator written in Go.” So we've had to kind of write to installers now.

[00:37:43] JM: If I look at the landing page of your website, your focus is on Apache Spark and Apache Airflow. Such great taste. Why those two pieces of technology? When you think about data engineering, why do you look so closely at Spark and Airflow?

[00:37:59] RB: Sure. So we have a very different opinion on Spark and Airflow. Spark, we love. And why look at Spark, is that if you look at their data processing, there has to be the essential complexity of data engineering requires you to do things that SQL cannot do. But SQL is still the most prominent thing there, right? The programming model of it, right? So for some things, you want to use SQL for some things, an abstraction above SQL for some things, and abstraction below SQL.

So what Spark allows us to do is to have a SQL productivity layer when we want it and write stuff in SQL. Then when I have something that does not fit in SQL and requires me to write code, I can just drop down and write the code. The second thing, it's open source. And so for our customers who have been paying millions of dollars to vendors over many years on-premise, as they come to the cloud, they don't want to get stuck again. So partly it is driven from what our customers want who wants Spark, and partly it is from the fact that it's open source. It's going to be free. For 20 cents on the dollar, I can get it from any cloud provider. It's ubiquitous, right? If I want that the really good Databrix version, I can get that. If I want the Amazon basics, yeah, I can go get DMR.

But that said, so if our customer writes their 50,000 workflows against Spark, they know forever that they're not going to get locked in. The second thing, like I said, the processing, it gives me the SQL layer that I definitely want, but then it gives me other things and programmability around it. That is also great. Now the SQL will become better over time. And now at least Databrix is adding the Photon engine. So they will be able to do BI. So Spark, we heavily believe in, is the right thing for transformation. And I've seen tons and tons of ETL, right? I've

seen it on-prem. I've seen like – I can't tell you how many workflows I have read the code off, especially when writing the compiler. That's number one.

For Airflow, we care less about Aairflow. It's like we had to put some scheduler there. If it were Airflow, that'd be fine. If it were one of the new ones, Prefect or Dagter, we don't really care. So right now we have added low-code on Airflow. We can do the same on Prefect. We can do the same on Dagster. For us, the hard part of scheduling, right? So like I told you, for our customers, they're like, "I have 20,000 workflows, and they have dependencies across them." And using lineage, we can see what the dependencies are. And they said, "Generate the schedule," right?

So a lot of values in generating the right schedule, a lot of values having monitoring and those things. And then the actual last mile physical scheduling of that is not that much of a value add to at least our customers. So we don't care. Airflow can do it. Fine. Right? The problem was that Airflow can't handle much in terms of scale. I don't know where Dagster and Prefect are with being able to handle like 10,000 workflows and 100,000 workflows. So yeah. So Airflow is like fine. It's the best product out there. Not super excited about it. But it's okay. So we'll go with whoever is most popular.

[00:41:16] JM: Do you think it's a multi-winner thing? Do you think Dagster and Prefect are direct competitors? Or is this a divergence?

[00:41:24] RB: I think Prefect seems to be a direct competitor with Airflow. And I haven't spent as much time looking at Dagster. But it seems like Perfect, and this is like a similar stuff. People use Airflow. People will use Prefect. And I don't think there is – It depends on the use case. For simple use cases, there might be much just something to choose from them. But for the more complex use cases that we get, they are too low-level anyway. So it doesn't really matter. As long as they can schedule a certain number of graphs and run that reliably, that's fine. But let me say, none of them is ready for an enterprise workload today.

[00:42:05] JM: I'm really interested in what are you doing. What do you need right now? Like are you trying to find more customers? Are you like trying to iron out the infrastructure? Are you trying to recruit? Like what's going on with the company?

[00:42:14] RB: So I think our biggest challenge is what I would call the thought makers, right? The podcasters, startup engineer, junior VC Nexus just doesn't understand data engineering very well. They don't have the historical context. And so I can talk about what the prophecy pillars are. What are the main things we are doing? And what's important to the customer. I can talk about the programming model of which one will win. But what we are looking for there is right now to say that – And we can talk about, of course, like what the future looks like for data engineering. And I know you've been super excited about DBT. So I was like, "He's going to ask me about DBT."

[00:42:54] JM: Oh, yeah. Oh, totally.

[00:42:55] RB: Which we never talk about it.

[00:42:57] JM: No, no, no, no, no, I mean –

[00:42:59] RB: It's going to be very negative.

[00:43:00] JM: Oh, wow. Okay, please, lightning rod. Give us the lightning rod.

[00:43:04] RB: Okay. So you want to ask a question about DBT or –

[00:43:08] JM: Okay, I'll give you a question about DBT. DBT, overrated, underrated?

[00:43:14] RB: So DBT is extremely overrated, right? So let me give you – Again, like I say, I've come in historical work working with large enterprises for many, many years. And if you look at DBT, one good thing that they do is that they're bringing software engineering practices to data engineering, right? And this is something that customers want and this is the future. We do it also in Prophecy, right? You're doing visual drag and drop. That's code on Git, tests on Git, CI/CD. You get everything, right? So that's the first thing.

The second thing is the question of the programming model, right? So if you look at – So what does DBT do, right? So for us, we can consider it as one of the backends, because as you're

doing visual development, Prophecy can generate Scala code. It can generate Python code. And we can generate SQL++. It's the same to us, right? So Scala has SBT? SQL++ has DBT, right? So in one sense, it's like a build system, like a Maven, or this, right? So that that's first part of DBT.

The second part of DBT is they have macros on SQL. So that is like Jinja templates, {}, variables, functions. So they've added a few constructs there. It's like, fine. If you're writing SQL small setup, that is great, right? But then if you look at an enterprise that we work with, they need 100 times more product than what DBT provides. And we've talked about a lot of that, right? And so the question really is what is the future of DBT? It's great some startups, some small setups will use it, and I think it will just wither away. This will be one of those shocking times like early 2000s where we would be like things got so crazy that a company that added Jinja templates on top of SQL got valued at one and a half billion. Like if you look at what their proprietary technology is, it's like – Okay, not even proprietary, right? They have an open source Jinja template on this. When did the Jinja ginger template company – Or SVT or Maven a billion dollar company, right? If you look at what they're building, there is just nothing much there. They make it convenient to add these variables and stuff. But then beyond that, there is not much to it. So if you look at the future of data engineering, if you say there is going to be tomorrow a $10 billion company, a $20 billion company, something as big as Snowflake, as big as Databrix that does data engineering on top, that thing probably won't be like SQL with {} in that, right? That's not what it's going to look like.

And then it's not just that they don't have enough product. It's also that the direction is wrong. It's like they're working with simpler ETL. So that they go to SQL, and then they add variables, then they add functions, then they add macros. And they'll keep adding this, right? If you look at the complex ETL we deal with, they will just go on and reinvent Python, slowly, one step at a time, right? And then who wants their code to be in some proprietary SQL++ language after a while? So it's good for data analysts. I think they have a niche. They don't compete with us. No enterprise would use them for their core data engineering. So I think it is one of the biggest – If we'll look back at this moment with a lot of amusement, that you had a product like that, which had no technology, and became a billion dollar company. So it's fascinating. It doesn't affect us.

[00:46:42] JM: All right.

[00:46:43] RB: [inaudible 00:46:43] the race. But it's just like there's nothing there.

[00:46:47] JM: Wow. Okay. Alright. It's kind of hilarious.

[00:46:50] RB: We can talk about what the – You can tell me what the technology is inside there, right? Or what is the –

[00:46:56] JM: Listen, man. I don't – Okay, look, DBT confuses me. I don't know a whole lot about it. I've done some interviews about it. I'm very confused.

[00:47:04] RB: Through all the documentation, I was like, "Okay, maybe we can use it as a backend just like we use Scala and SBT as a backend." And when we started looking at it in great detail, I went through the documentation and I'm like, "What is this? This is a SQL with a few templates in there." And it's like, "Okay, there's a good community, and they're asking each other questions." I mean, so they've given some things. But yeah, I am surprised to see that there is nothing. So we can talk about it after a few years. There was the NoSQL fad. And then there was the Hadoop fad. And so many companies, right? There were people who are like, "Hey, how do you do this in DBT?" And somebody's like, "Hey, why don't you use talent to do this in DBT? Because DBT doesn't do –" Like you have to go through the forums to see all the amazing stuff. But yeah, it's not going to be a big thing in a few years from now.

[00:47:52] JM: So I normally don't do this on air, but I am just very interested in your company. And I would love to be potentially involved. I think this is going to be a really interesting series A that you're pulling towards. I mean, have you gotten much investor interest yet for a series A?

[00:48:10] RB: We did talk to investors, a couple of investors. Not right now. Quite a few months back. Like we talked last year. And we had like one customer, and they were like we're looking for some more GTM maturity. And now we are probably going to have many, right? So now, this time, I think we'll be fine. We've just told most investors that we are not raising right now. So if I get a request, I just say, "Hey, we're raising in September." So not having any investor conversations right now. So it will be – Yeah, we're super excited to – Yeah, I don't

know what part of it makes it, but definitely super excited to have you get involved, right? We can talk about that offline perhaps.

[00:48:51] JM: Right. Okay, cool. Well, let's cut to the chase there. And maybe just to wrap up. Okay, here's my last question, little from far left field. So machine learning and data engineering, what's the opportunity there for you? Or what's the overlap there in that market? Do you think of machine learning as sort of a downstream thing you can attack after you solve data engineering?

[00:49:12] RB: So one of the main things to realize is that, first, a lot of the work, 80% to 90% of the work of machine learning is data engineering, right? And for example, I'll give you an example. We had a customer in one of the top telecom companies in Europe come to us and say, “My team is trying to buy these features stores. And these features stores are supposed to give us lineage and say, “This column, this is how it was derived,” and build these features. They’re like, “When we look at that, with a couple of abstractions, you can build a lot more than these feature stores give us,” right?

Basically, what is going to happen is that a lot of the feature development, or having the feature stores the metadata. So now you have this metadata system that has your entire data engineering system in it, all the information about it end-to-end, right? And now you have one more step to go, which is like I am computing these features, which is essentially columns for machine learning. And that entire feature engineering is going to move into this core data engineering. And then feature store is just a metadata system where you can query the features, right?

For example, the transformations for getting the features in, the transformations for managing the feature store, all of that belongs in your data engineering system. So everything outside of the core machine learning training algorithm very naturally belongs into our data engineering product. And even though some people might be like this Tecton and other features to our companies, they are going to get eaten, because we already –

[00:50:44] JM: Wait. You think the feature store companies get acquired, you're saying?

[00:50:47] RB: No. No. No. So the thing is the feature store companies, I see that their data engineering products will subsume them, because you see, they are building features store. Features store is just a slightly different packaging of ETL and metadata. So basically, right now, they're building it, but we have so much more build in data engineering, in lineage, in transform. Letting people transform, letting people manage them. That feature store as a separate category will probably get subsumed by the data engineering systems.

So like you said, what is an area to expand? Well, feature store will probably become a part of our product soon. And we have customers asking for it and saying you can do better than Tecton and these companies. So the data engineering will expand. To reach that point, I think except for the core machine learning training, apart from that, rest of it goes through data engineering tools, is my take on it. And again, a lot of controversial take. So that should be –

[00:51:44] JM: Great. Fantastic interview. Just fantastic. I love your vision. It's really interesting.

[00:51:50] RB: That sounds great.

[END]