EPISODE 1320

[INTRODUCTION]

[00:00:00] ANNOUNCER: Whether sending messages, shopping in an app, or watching videos, modern consumers expect information and responsiveness to be near instant in their apps and devices. From a developer's perspective, this means clean code and a fast database. Apache Druid is a database built to power real time analytic workloads for event-driven data, like user-facing applications, streaming, and anything else that requires instant data visibility. Druid offers low-latency for OLAP-style queries, time-based partitioning, fast search and filtering, and out of the box integration with Apache Kafka, AWS kinesis, HDFS, AWS S3, and many more.

In this episode, we talk with Eric Tschetter, Field CTO at Imply, fellow at Splunk, an experienced developer who's worked in a large swath of backend infrastructure projects largely focusing on Druid. We discussed the use cases and power of Apache Druid.

[INTERVIEW]

[00:01:05] JM: Eric, welcome to the show.

[00:01:06] ET: Thank you for having me.

[00:01:07] JM: You were one of the founding members of the Druid project, or at least one of the early members of the Druid project. And the Druid project came out of Metamarkets. What were the evolutionary pressures that Metamarkets was undergoing that created Druid?

[00:01:28] ET: Yeah. Basically, start starting back from the beginning when I joined Metalmarkets, if I get the full origin story almost, the founders were going after – They were trying to build effectively a derivative market on top of advertising auctions and basically analyze at like Internet advertising auctions, understand the inventory, the pricing, and all of that, such that they could actually create a derivative market on top of them and, say, offer advertisers protections and stuff from things.

And when I was talking with them about that, I was like, “That's an interesting idea.” I know nothing about it. But they showed me this kind of dashboardy thing that they were giving back to the advertising auction houses in order to like try and effectively be like, “Well, give us your data. We'll make these derivatives. And we're going to give you this dashboard thing in return type of a thing.” When I saw that, I was like, “Oh, that's something I know about. I'm going to join you guys and work with you.”

When I joined, the data was initially in a single instance Greenplum. So basically, effectively Postgres, where the data was in there, we had it powered off – It was powering the UI. All of that was great. This was 2011 or so. That was right at the heart of or at the very beginning of everyone saying big data, big data.

So the next thing we went to when we were like, “Oh, if we want to scale up Greenplum, we're going to have to pay them a bunch of money in order to go distributed.” So that's kind of hard.

[00:03:20] JM: Remind me, was Greenplum just productized Postgres? Is that what it was at the time?

[00:03:25] ET: Well, it's distributed Postgres. It's basically Postgres that they adjusted to basically be able to run on multiple threads. And so one query can go across multiple processors, but they also took it out in a distributed fashion and kind of handled it. I'm not an expert in in Greenplum, but that's my understanding of it. Anyway, after that, we jumped into kind of big data, went with some Hadoop jobs, pre-processing numbers, throwing them into Hbase, powered the UI off of that. These jobs kept growing and growing in how long they took to do stuff.

At one point, basically, every time we would add a new dimension, it would take longer and longer. One of our customers, we added like a few new dimensions, and all of a sudden something that was taking 12 hours started taking over a day. And when I saw that, I was like, “Okay, that's not good. What do we do next?”

I started thinking back to some people I worked with at LinkedIn who worked on some search infrastructure. There were some people I worked with at NING before that in kind of data stuff

who had often said like – They had made statements by like – I mean, talking about data at that time. But they were always like, “This isn't large data. I can just throw this in-memory and do stuff with it.”

Anyway, from those statements, I started thinking about, “Oh, I wonder what happens if we just throw this in-memory.” So I then grabbed the data, just threw it in-memory and was like, “Well, how fast can I add? These numbers if it's just in-memory?” Found out the answer was superfast. And I was like, “Oh, okay. I think this is what we build.” I think there's an email, a blog post was written about recently that I sent out saying, “Hey, I'm going to try this thing as a background task. And that's when I started Druid.

[00:05:22] JM: So is the key innovation of Druid sort of just the timing where nobody was doing in-memory open source data warehousing like systems at the time?

[00:05:34] ET: Honestly, the question of the key innovation is hard for me to like believe like I'm honestly answering, because if I look at Druid itself, what it has inside of it – Like I don't know that there's actually new ideas in it. And you can probably go back to the old technology, especially now, is that there's not really that many new ideas. It's all old ideas kind of recombined differently. Or you approach a problem with a different set of assumptions. You combine the old ideas in a different way. And all of a sudden something comes out.

And so if I could say that there's some new innovation, it would be that. I came at the problem with a different set of assumptions. A set of assumptions that, oh, maybe the data can all just be in-memory. Maybe we can just run with it. Apply a bunch of old ideas to that. And it turns out it works out okay.

[00:06:36] JM: Okay, so if I'm getting the context right, you're at Metamarkets. You're building an ad tech –

[00:06:44] ET: Dashboard.

[00:06:44] JM: A high-volume ad tech product, like ad tech is super high-volume, bidding and markets-based information. And the application was let's dashboard this stuff, which is actually a

different application than what we would consider data warehousing, right? It's kind of different, right? It's the operational analytics database idea.

[00:07:13] ET: Yes. If you were to categorize it with the categories that exist today, yes, you would definitely go into the operational analytics world. Definitely, it did not start in the BI world except for perhaps – The fact that it's dashboarding, a lot of times if you say dashboards, people will jump to the likes of Tableau, or Looker, or something like that. But yeah, operational analytics is definitely the category that I would place it in given the words that exist today. At the time, there wasn't really an operational analytics space that people much talked about, at least not that I'm aware of anyway.

[00:07:55] JM: Okay. Let's talk real quick. Architecturally speaking, what's the difference between an operational analytics database and a data warehouse like Snowflake?

[00:08:06] ET: Well, so the difference between an operational analytics data warehouse, or operational analytics infrastructure and data warehouse, I'd almost say that like at the heart of it, when you step back from the technology, you get to slightly different requirements, or kind of different people who are actually using the infrastructure and trying to do things with it. And coming out of that is probably where differences in infrastructure come. But at the end of the day, in the fullness of time and fullness of investment, to a certain degree, infrastructure tends to converge towards being able to be used in kind of multiple areas.

But if we compare the kind of personas of, say, operational analytics versus business intelligence. In the business intelligence side, your user base tends to be, one, it's all employees of your own company usually, business analysts and that type of person. There's actually been many, many years of training in this language called SQL that everyone knows about how to query data, how to interact with it, how to work with it. And so like SQL as a language, it's just solidified and exists in the business intelligence space. And if you don't speak SQL, you cannot play in the business intelligence space, because all of the people who use it, all of the people who operate in that world expect SQL, all of the tools that they use expect SQL. And so like that's one difference to a certain degree between the infrastructures. And so if you want to be in business intelligence, one, you have to speak SQL.

Another thing is that in kind of traditional business intelligence data warehousing data lakes, latency is not as big of a concern. Of course, everyone wants results faster. You always want results faster. No one is going to tell you I want slow results. I remember driving on 101 and seeing a big billboard for Reddit saying, “No one ever says they want a slow database.” That's true. No one's going to ever tell you that they want something to be slow. But at the same time, in a business analytics, in a business intelligence world, you can usually kind of deal with extra latency. You can deal with a 32nd query. You can deal with a minute query and kind of still get your job done.

Switching over to the more operational analytics side of things where you have vendors like Splunk, I'd say Druid at least started in that space. There's a number of other kind of options as well. But when Druid started, the way to query it was all JSON. It was JSON objects over HTTP. There was no such thing as SQL on top of Druid. Splunk as well has its own language called SPL, which is its own language for interacting with the data.

And in this operational analytics space, there's not the base assumption that thou must speak SQL. There's more the question of how are you exposing the data or the answers? And are those answers accessible to the person who needs them? And so if the person who needs an answer is someone operating an ad auction house, then great. They look at the UI. They interact with that. They get their answer through that. If the person who needs an answer is someone in IT or a cybersecurity professional, they just need to be able to ask an answer the question. They don't necessarily care about it being SQL. A lot of times, you also actually want it to be UI-driven rather than actually query-driven, because you want to be able to scale out the team of people that's interacting with this data. At any time, you need to scale that out. You need to get it into something that's easy to train and kind of UI-driven. And so I'd say those are some of the difference is in thinking when you're approaching something in the operational analytics space versus the business intelligence space.

[00:12:14] JM: Again, to summarize, the query pattern is simply different, right? Because on average, the operational analytics database has to refresh a lot, I guess, right? Because you're kind of middleware for – Or you’re backend technology for, again, this dashboarding use case, this real time analytics use case, this real time aggregation use case. You really want data

freshness, whereas a data warehouses fulfilling a lot of applications where data freshness is less of an idea.

[00:12:47] ET: Yes, there's data freshness. I think data warehouses very rarely show up in terms of kind of UIs that you give to, say, a call center. Like you're not going to expose a Tableau dashboard or a Looker dashboard to all of the people receiving calls in your call center. Or at least that's not something I've heard of someone doing. There might be someone out there doing that. Where an operational analytics database, you would actually take it, throw a UI on top of it and give that UI to people in your call center so that they can look up some issue that might have happened.

And so like, because of that, there is this data freshness side of it where something just happened, I need to be able to see it. But there's also like if that call center has to sit there for a minute, two minutes waiting on results, and you've got 500 people, a thousand people answering calls all doing that, you're going to slow down your call center significantly. And call centers are already a fairly expensive item for any company. That's kind of operating one. And so there's also the kind of quickness. And the fact that it's an actual application that you're giving to somebody to use, rather than kind of something that's given to an analyst to basically crunch some numbers and then come back with an answer at some point in time in the future. I'm not sure if that helps clarify or not, but –

[00:14:16] JM: No, it definitely makes sense. It's very interesting how sometimes open source projects come from unexpected circumstances. But I look at the ad tech space, and the ad tech space is one of those areas where, really, like it is kind of a zone where there's so much high-frequency stuff going on. The amount of money that you can make is really spectacular. So it does create a kind of a petri dish pressure, set of pressures on building new technology. What was the closest thing to a Druid before Druid?

[00:14:58] ET: The closest thing to Druid before Druid is – Well, I can say the answer by talking about the things that we explored. So right around the time, open TSDB was a thing. It was just kind of starting up. And we explored that some. That's based on Hbase. There was Greenplum, Vertica, kind of the all of those. The distributed database vendors I think were other options for what we would have needed. But like, honestly, that was the landscape of things available for

what we needed, was there was the big data, the Hbase-based things that relied on a good chunk of pre-processing bore the distributed database vendors.

So I don't know that they're – Yeah. And each of them has different options. The distributed database vendors have been thinking the BI world a lot more than they've been thinking the kind of operational analytics world as we know it today. The kind of Hbase-y open TSDB world, given that it's a TSDB, it tend to be written under the assumption that one time series is dense. Meaning that it receives lots of data points over the course of time, where data for ad tech auctions or data for kind of this operational observability. Once you start including enough dimensions, you get very disjoint time series. You get very sparse time series. A time series might exist for just one data point over the lifetime of even the auction house.

And you start to see this some in infrastructure monitoring as well, where people have tended to use TSDB's. But now with the advent of containers and more ephemeral infrastructure, they're starting to push on the limits of the TSDBs because they're exploding cardinality. They're growing up in cardinality, because a container might come up, be around for a minute to five minutes and then go away. That creates a whole bunch of time series in the TSDB, adds cardinality, pushes the limits of cardinality. And so we see – Anyway, those were some of the limitations even back then when we were looking at open TSDB for this use case as well.

And part of the reason why I think even today there's a bit of a shift as people want kind of more business looking answers into their data. They tend to start looking at the things like Druid and other things in the strong operational analytics space.

[00:17:43] JM: When you have this key innovation of let's do it all in-memory, what does that mean? Does that mean that you literally had to take a database and fork it to use memory? Or you wrote a database from scratch and had an access memory? What does it mean to just –

[00:18:00] ET: Yeah. What it meant was – I mentioned, the first thing I did was said, “Well, what happens if I add up all these numbers in in-memory?” Because effectively, what we were doing for the dashboard was just adding up a bunch of numbers, right? And so the first thing I did was I took all the numbers. I load it. I created a long array in-memory. I put all the numbers in the long array, and I did a for loop over them and added them up and was like, “How long does that

take?" And it turns out computers are really, really fast at just adding up numbers in a for loop. And I was like, "Oh, that's fast."

And then from there, it was basically Druid was all new code. It wasn't taking something and adjusting it. It was like, "Oh, okay, let's take all the data that I've got." And the very first version actually – So it's written in Java, right? The very first version was pure primitive arrays. It was arrays of floats, arrays of ints, arrays of those objects purely in the JVM heap, loaded up, and effectively for loops written on top of them.

Over time, it's evolved greatly to where, rather than being primitive arrays in-memory, it's now effectively files that get memory mapped in, and then native memory access is used to access the data in those. There's also stuff where we do decompression on the fly, compression and decompression on the fly in order to shrink the data and various things like that. But the very, very first version was basically int arrays, long arrays, float arrays sitting there on the JVM heap loaded up and answering queries.

[00:19:43] JM: So meaning, you're doing memory management in the JVM, and you're just managing bits and bytes in the JVM. That's your system of record for serving these operational analytics systems.

[00:19:57] ET: I mean, that's where it started. I wouldn't say that the memory management in the JVM was the system of record. The fact that we were loading it up into the JVM and the fact that we needed to be able to lose nodes and all of that and be able to recover meant that, basically, the data was first placed into what we call segment files. And so it was organized and placed in segment files. And then that code in the JVM knew how to read the segment files into the in-memory arrays and then serve the queries on them. And so if I were to call something system of record, it would be the segment files, because any node could always pull down a new segment file, load it up and serve queries off of it.

[00:20:43] JM: What was the schema of the segment file?

[00:20:47] ET: Homegrown made-up me writing integers into a file. It's not something that has a name.

[00:20:56] JM: Nothing fancy? It's not like a Parquet kind of thing.

[00:20:59] ET: No. Nothing fancy enough with a name. It was like, “Oh, I've got an array of integers. Let me put this array of integers as bytes into a file. Flash file. Close file. Done.”

[00:21:08] JM: You know what's funny is I think a lot of people haven't experienced Java as a programming language to do that kind of stuff in. Java is really good for doing that kind of stuff. I worked at a trading company that built its own low-level data transfer systems. And Java is great for doing that kind of stuff.

[00:21:28] ET: Yes, it is nice. I like it's very good. As kind of Druid evolved and the development evolved for it, the one thing that I realized – So I mentioned earlier that we moved a lot of the data out of in-memory data, like pure in-memory arrays, and into memory mapped files that we're then doing native access for. One thing that we've run into as we've done that is that Java's interactions with native memory and just its rigidity. And how it needs to be able to apply object structure to native memory actually makes us do contortions that like if we had something in C or C++ where you can just be like, “Hey, here's a pointer. By the way, this pointer is pointing to this type of data. So just read it like that, and don't think about it.” There could be some nicer things.

And so like Java is definitely nice for doing this sort of stuff, especially when you're keeping things in the JVM. Once you start going across the JVM native boundary, you get into a little bit of some kind of contextual mismatch or some mismatch between what you want to be able to do and what Java allows you to do. Even though over the kind of the evolution of Java, like with the promotion of unsafe to – Like with the separation of unsafe into the things you really shouldn't use, and okay, these things are okay to use. And some of that like I see movement in the Java space that will probably make it easier to work with native memory from the JVM. But right now, even today, I'd say that once you start crossing that boundary, you run into interesting – And I'm not sure if they're fun, but you run into interesting problems.

[00:23:25] JM: And so that brings us to what – Was the moment when Fangjin started Imply? Fangjin left Metamarkets to start Imply or started that like after Metamarkets is acquired. I don't remember the lineage there?

[00:23:43] ET: Yeah. So the way things worked – So at Metamarkets, started building Druid. We hired. So hired Fangjin, hired Yang. Vadim was there from very early stages. We all worked together kind of building up Druid doing more and more stuff with it. So at one point I moved on from Metamarkets. Worked with a nonprofit for a little while. Kind of Fangjin, and then they stayed at Metamarkets, were working there for a bit. And then probably about a year or two after I had moved on, they were like, "Hey, we could start a thing around this." They went, started a thing. Had various conversations with Metamarkets to try to work out something amicable. Did stuff, eventually started. This was before Metamarkets got acquired by Snap. I forget how many years. I think they existed as an entity for multiple years before Metamarkets got acquired by Snap. But I forget exactly how many as well. But, yeah.

[00:25:02] JM: And the market opportunity for Imply is, essentially, we're going to take the operational analytics database known as Druid that was built at Metamarkets and we're going to generalize it as a service and make operational analytics as good as it can be.

[00:25:18] ET: So I actually do work at Imply now. I started fairly recently. But I did not work at Imply at that point in time. So I can't speak too much to some of the internals of that conversation. What I can speak to is what Imply did in the community. Because even though I wasn't at Imply, I've been a user of Druid at all the companies that I've been working at, Yahoo, Splunk, all of those companies. And Imply, effectively, they have been driving the community for a long time. Most of the development, especially the development that helped push the Druid closer to the business intelligence space. So like the development of SQL, the kind of SQL operational layer, all of the connectivity with SQL, the querying, and all of that, were primarily developments that came out of Imply.

And so if I look at that, I would say that like the thing that Imply provided through the development of Druid was really helping it get solidified as something that can be used in the business intelligence space. Helping get it connected to the other tools and the kind of language of choice of business intelligence SQL. And I think that that adoption of SQL, definitely, it helped

grow and gain adoption for the project, because people just like SQL. It's so much easier. Everyone knows SQL. It's so much easier when you can speak SQL to throw data in and get it worked with.

There are benefits to not SQL. But at the end of the day, like from an adoption perspective, speaking SQL is worth so much more. I'd say that like that's a major contribution that I saw coming out of Imply kind of from the community angle.

[00:27:17] JM: Yeah. As far as a platform for building operational analytics systems, to the extent that operational analytics is a real category, which I believe it is. I think it's very good marketing, but I also think it's a real category. What's the alternative? When I think about the competitor, the primary competitor to people using Imply a service to me seems like custom systems. Basically, custom dashboarding systems, right? Or is it Tableau? What's the competitor?

[00:27:49] ET: If you're going into – So from the infrastructure perspective, the other similar – So somewhat similar systems. There's is system called Pinot, which –

[00:28:00] JM: Oh, yeah. Pinot. Pinot. Pinot. Pinot.

[00:28:01] ET: Which, came out of LinkedIn and Uber. And there's an interesting story. Well, anyway, back when we were open sourcing Druid out of Metamarkets, I reached out to people I knew at Netflix, LinkedIn, and Facebook saying, "Hey, we want to open sourced this thing. We want to work together with some people to run it so that there's actually more than just us open sourcing this thing. Are you interested?" Did kind of tech talks at the various places, had conversations. Netflix was interested. They picked it up and deployed it and ran with it. And we basically gave them access to the code ahead of actually open sourcing so that they would work with us to help us open it up.

LinkedIn didn't take the code and run with it. I forget if we gave them access to the code or not. And I'm fuzzy on that detail. But a year or two later, there was this project called Apache Pinot that, at least from the architecture diagram, they talked about historical nodes and real time nodes and stuff like that, which is kind of the architecture diagram that we used. And I was like,

“Oh, okay. So maybe they didn't borrow the code, but at least like the idea of how things work was able to kind of take them in a place that they felt good about.”

And so, anyway, that's another one. And then there's something else that people look at called ClickHouse, which is something – People look at it together, at least from the infrastructure perspective when they're comparing Druid, Pinot and ClickHouse. They tend to kind of compare these. At the same time in terms of kind of operational model and the way in which the infrastructure scales, Druid kind of adopts a separation of ingestion from querying that allows you to scale out your ingestion independently from the actual querying. And so as you scale up larger and larger, this is a significant benefit. Because a lot of times your query load won't change, but your ingestion will, and you need to add nodes there, but you don't want to impact your query load in order to add those nodes. Or vice versa, your ingestion load stays the same, but you need to add more nodes in order to handle more queries. And you don't want adding more nodes to force you to then re shard your data and do all of that sort of stuff. And so Druid adopts this model that actually separates them to allow you to think of each one independently and scale it independently. Where ClickHouse adopts the, “We’ll put data and query it all from the exact same node,” which has some other benefits in terms of just like firing it up and running it and getting started because there's only one node to run. But as you like step up, if you want to scale out, you have to re-shard data and reallocate it and stuff like that. And so there're architectural differences there. But if I had to pick infrastructure pieces that people compare, I would say those are the three.

[00:31:19] JM: Gotcha. Tell me some architectural challenges, or engineering challenges, or software – I mean, you're an infra guy. And you're intrigued by the infrastructure challenges. Can you tell me what are the canonical tradeoffs of the Druid database?

[00:31:35] ET: Yeah.

[00:31:35] JM: By the way, Druid is effectively like – Mostly like a read only database, right? It's like you're not really writing to it, right? Or am I wrong about that?

[00:31:48] ET: So those words do not correctly capture the idea, I’d say. It's built under the assumption that you have a stream of data flowing in that is primarily always being appended to

your data set. And so like it's built under the assumption that that's the primary way in which data comes in. Now, being born in ad tech, in ad tech, even if you collect all of your impressions, and ad auctions, and all of that, it turns out there's a certain amount of bots and other fraudulent activity out there. And so people always want to go. They want to detect the fraudulent impressions, do that, correct it, restate the data, and rerun. And so like even an ad tech from the – Like even from the early days, we had a need to be able to rewrite data.

And so we had to be able to mutate it in some fashion. Now, Druid doesn't enable you to say, “Okay, find this one row and rewrite just that one row, but keep everything else the same. And so from that angle, it's not the same kind of mutability as a traditional relational database. It's more, “Okay, take this chunk of data and replace it with this new chunk of data.” And the new chunk of data might have some things removed. It might have some things restated. It might have some new things added to it. But, effectively, take this chunk. Like take this days' worth of data and replace it with this other days' worth of data, which effectively allows you to rewrite things in a way that is actually useful for a significant number of use cases, one of them being fraud and ad tech. But there're a number of other cases where that's sufficient enough rewrite capability for what you need.

And so it does allow for writes. But the primary mechanism of ingestion tends to be a stream of data. It can also do batch ingestion. And actually, Druid started with only batch ingestion. But over time, we eventually connected it up to Kafka and Streams. And like once we once we really got that solidified, I'd say that stream-based ingestion has kind of taken over in terms of the primary use case and the primary way of getting data in.

[00:34:10] JM: Okay. Anyway, tell me a little bit more about the canonical problems. Like what are the canonical challenges? Is cost an issue?

[00:34:17] ET: Cost is definitely – Cost is an issue. One thing that was very much top of mind for me, as well as cascading failures, having worked with other infrastructure at other points in time and carrying a pager, I've grown allergic of cascading failures to where I don't like them. Like a failure somewhere happens. It always happens. And if the system can't degrade gracefully in the face of that, then you end up with larger problems and the inability to self-heal.

So like that's one of the things that was top of mind in kind of building this up, especially as a small startup, because I don't want to be woken up every night at 2am, right?

And so like some of those stuff to do there. So I was mentioning earlier the separation of ingestion from query workload and the ability to scale those axes independently. That also lends itself to making the cascading failure problem easier. Where if each kind of zone is its own world, then a failure in one zone, say, the ingestion zone, doesn't necessarily have to impact the query side. The query side can keep going along. It can keep happening even while ingestion is having some problems. You then fix those problems. Data starts showing up. Great. But at the same time, query, you could have problems on the query side without impacting ingestion.

So in some other systems, sometimes if you get too much query load, you end up falling behind an ingestion, and all of a sudden your message bus or the data, the system that's handling the data coming in, starts to hit retention limits, and you get like backups all the way up the system, which doesn't end well. And kind of the separation of ingestion and query protects from that sort of stuff. Where even if something bad happens on your query side, your ingestion system, you can be relatively confident that it'll keep going.

There's another couple of things where when we ingest data and generate segments, there's something called a handoff, where basically, the data is handed off from the servers that deal with the stream of data to the servers that deal with effectively the querying of historical data. And these are the differences between what we call real time nodes and historical nodes. And that handoff is something that's managed by a coordinator, a coordination node that kind of takes data from one, hands it to another. And in some of the logic there, there's various throttling mechanisms that tries to ensure that when a node goes down, you don't suddenly get a thundering herd of like, “Oh, my God, we must all load brand new data and get it showing up and everything.” It's a much more metered problem. Because without that, I've run into issues before with other systems where a node goes down. All of a sudden, all of the other nodes are trying to re-replicate data. That starts saturating network interfaces. You get cascading failures. Badness happens, and you're like, “Okay, nothing's working at all.” And so like there's a number of things, like there were – I'd say cascading failures are one of those things that I definitely thought heavily about in the development.

There's also the separation of compute and storage. Where Druid actually started with separation of compute and storage, there was no data pre-allocated or existing on any node. And the data would always get loaded in response to queries coming in, which is something lots of people tout right now is, “Oh, my God, this is awesome.” We actually started with that. But for our use case, where we were really trying to get high-latency interaction, what that led to was a very slow start in the first query that came about, where we would have to download, say, for that first query, we would have to download 100 gigabytes, 200 gigabytes of data. And that just takes time.

And so the evolution of the actual software, given our use case, actually required us to pre-allocate segments to nodes so that they're sitting on the node and available for query ahead of time. Because what was happening before we did that is, every time I would release new software, I would also run a script that would basically ran a query for every one of our tenants in order to pre-load the data so that their data would be hot and ready and pre-loaded when they actually came around. And so there're another couple of things there that where kind of we start with one way, and then because of our use case, or because of things that we ran into, we adjusted and changed to the way things are today.

[00:39:11] JM: Gotcha. I wish we could talk longer, because I have a ton of questions remaining. But I think my last question is, basically, I want to know in like, I don't know, three to five minutes, what this thing looks like as a distributed system. Like can you just give it Kubernetes operators and make it run like that? Or do you have to do a lot of complex operational work?

[00:39:33] ET: So at Splunk, actually, we develop a Kubernetes operator and open sourced it. And it's been in the community now for about two years. And the community has kind of taken it and run forward with it. And so there is a Kubernetes operator for it right now where you can just kind of grab that and deploy for the open source. All of the companies that are doing anything with Druid tend to have their own method of deployment as well to try to effectively remove the cost and burden of starting up the infrastructure from someone who's trying to adopt it.

At the end of the day though, it started with just Java processes on independent machines. There was no Kubernetes in 2011. There wasn't like the idea that you needed to manage

multiple different services wasn't a common thing that people dealt with. And so there wasn't really a strong infrastructure for that. And so it was built as just a bunch of independent Java processes. And then how you choose to deploy them happens to be how you choose to deploy them. What we have now, Kubernetes operator or doing whatever the commercial offerings are probably the simplest ways to get going, especially if you're not comfortable or familiar with the JVM.

[00:41:01] JM: Gotcha. Anything else we should close on? Actually, are you working with Jad? Are you working with Jad much at Imply?

[00:41:12] ET: I am working with Jed. Yes.

[00:41:13] JM: I really like I really like Jad. I'm friends with them.

[00:41:16] ET: Nice. Nice. I am also a fan.

[00:41:19] JM: He's pretty funny.

[00:41:21] ET: The sense of humor I ran into is somewhat dry.

[00:41:24] JM: He's really dry. That's I was about to say. That’s the word to describe him. He's dry.

[00:41:29] ET: Yes, yes. Great guy. I'm super excited to be with the Imply team now. I'm enjoying all the people I'm working with, Jad included, and kind of doing that stuff. It's fun days.

[00:41:41] JM: One weird question. Aare you guys going to do anything with WebAssembly?

[00:41:47] ET: WebAssembly?

[00:41:48] JM: Do you know what that is?

[00:41:49] ET: I'm wanting to believe it's assembly language running in the browser to make it faster to do things, but I'm half making stuff up right now.

[00:41:59] JM: It's language agnostic, language agnosticism in the browser. You can run Rust modules in the browser.

[00:42:09] ET: Wow! Interesting. Given the degree to which I'm aware of what it is, I have not thought about using it for anything. So I'd say that. But that is an interesting idea.

[00:42:21] JM: So Dropbox uses it to compress files before they send files over the wire.

[00:42:28] ET: Oh, okay. So like a better web worker type of a thing?

[00:42:33] JM: Yeah.

[00:42:34] ET: Like we can do that. I think that's actually an interesting area of development. I think, in terms of just computer science development, there tends to be a pendulum that swings between centralized run everything in central servers and shove things out to fat clients, and do things on the edge. And kind of, if you look over history, I think you see the pendulum swinging between client-side processing, to centralized server processing, to client-side. And to me, most recently, the pendulum had been swinging to more centralized stuff. And I see it starting to move back to, "No, no, no, we can do more stuff on the client." And that it just kind of strikes me as another interesting thing to do there, where I think that there's going to be a number of innovations probably that come out of bundling a lot more complex logic, say, on all the browsers. Because if you think of it, all the browser's, to a certain degree, former botnet. And you can have a whole bunch of stuff done on a lot of different machines by just pushing it in the browser. Anyway, it's interesting just from a philosophical standpoint of that pendulum swinging between centralized and client server. But for me, personally, I haven't thought any more than that about how useful it would be or where it could be applied.

[00:43:57] JM: Well, I look forward to hearing any perspectives you have in that regard. It seems like a pretty transformative technology. I don't really know what it's going to do though. I just think you can build fatter clients with it. You can build more interactive richer clients with it.

Aad it feels like you should be sending compressed data over the wire and decompressing it on the client-side.

[00:44:18] ET: Right. I mean, because browsers have supported ZIN compression and stuff like that for a while. But it's not just that compression. It's actually like building in schemas to the data and truly getting it into a binary format before sending it in which to –

[00:44:33] JM: Do you all use Protobufs?

[00:44:35] ET: You can ingest data from Protobuf. You can ingest it from Protobuf. You can ingest it from – I'm blanking on the one that Google has. Or wait, is that Protobuf? Avro. Not Google.

[00:44:46] JM: Avro is the Facebook one I think.

[00:44:47] ET: Right. Yeah, you could do it from Avro. You can do it from Protobuf. Protobuf is the Google one. Thrift, I've even seen connectors for pulling stuff in from Thrift formatted data as well. But yeah.

[00:44:59] JM: I mean, with Protobuf – In this context, does it make sense to, in an ideal world, send everything over a Protobuf-like interface? Because Protobuf, like it's both compressed and schema'd, right? That's pretty useful, right? Isn't that exactly what you want?

[00:45:16] ET: Protobuf, I'd say for a lot of the serialization strategies, the big thing is actually – The big benefit that really comes out of it is standardization across an enterprise. And it's knowing that all of the services and all of the things in whatever ecosystem that are enterprise has are speaking the same format. That's the really big benefit that comes from these standardized formats. Like once you have that standardization, you can add optimizations around compression. You can add optimizations around binary, storage, and all of that. And so that's some of the stuff that you get from Protobuf.

When you're thinking about it from the pure kind of tech head space, you look at the binary, and compression, and storage size and all of that. But the real benefit of adopting one of these

things is the standardization, like driving that standardization across your organization so that you can know that the services are all kind of speaking the same thing.

And then as a side effect of that, of course, if you standardize on Protobuf, having your data also be on protobuf makes tons of sense. If you standardize on Thrift, having your data also be on Thrift makes tons of sense. Like, to me, it's actually a question of standardization. And then the optimization that you can drive by standardizing on one thing, rather than like, “Oh, this thing is better than that thing, or that thing is better than this thing for these reasons.” Does that make sense?

[00:46:49] JM: It does that make sense. Pretty interesting. A lot interesting architecture stuff to talk about. We've come a long way since the early days of Hadoop, haven't we?

[00:46:57] ET: Yes, it's a very different world. One thing I'm happy about is the word big data doesn't get used very much. It's kind of lost its place in the –

[00:47:06] JM: I heard data is the new oil.

[00:47:08] ET: Yes, I have heard that one. I ever heard that data is the new oil. I don't know. There're a lot of fun sayings.

[00:47:15] JM: Cool. Well, this has been great. Congrats on joining Imply, and bright future for the company. I mean, embedded analytics? I want it everywhere. I want across all my things.

[00:47:26] ET: Yeah. Thank you. Thank you for taking the time to chat.

[00:47:29] JM: For sure. Wait. So you built this thing, right? Like you were the first person to build it, right? You were the founding engineer on the Druid project, right?

[00:47:38] ET: The way I say it is I wrote the first line of code. I have the honor of writing the first line of code. Many people have written many lines of code inside of the project. And so like the project that exists today, it's not my one line of code that built it. But that first line of code was authored by me.

[00:47:59] JM: That's great, man. Congrats. Congrats on having the boldness to do that. Great talking. Let's do it again sometime.

[00:48:04] ET: You too. Bye-bye.

[END]