EPISODE 981

[INTRODUCTION]

[00:00:00] JM: Every large company generates large amounts of data. Data engineering is the process of storing, transforming and leveraging that data. Data infrastructure companies provide tools and platforms for performing data engineering. The last 15 years has seen a rise in modern data management companies built in a time of decreasing storage costs, and increased volume of data, and the prevalence of cloud computing. Modern data companies include Hadoop vendors, cloud providers and a wide variety of individual software companies offering products such as databases, ETL tools and open source tooling.

The go-to-market strategy for a data infrastructure company requires a deep understanding of the data engineering landscape. A company must build something useful, sell it to customers and eventually build a replicable strategy.

Sean Knapp is the CEO of Ascend, a company that builds Apache Spark-based data pipelines that connect to APIs, data lakes and data warehouses together to enable data applications. Sean joins the show to talk about the process of building a data infrastructure company as well as his lessons building Ascend.

If you're planning a hackathon, checkout FindCollabs Hackathons. Whether you're running an internal hackathon for your company or you're running an open hackathon so that users can try out your product, FindCollabs Hackathons or a tool for people to build projects and collaborate with each other.

FindCollabs is a company I started to allow people to find collaborators for their software projects, and our new hackathon product allows you to organize your hackathon participants to make your hackathon as productive as possible. If you're thinking about organizing a hackathon, check it out at findcollabs.com.

[SPONSOR MESSAGE]

[00:01:58] JM: I love software architecture. Software architecture is the high-level perspective of how to build software systems. Much of Software Engineering Daily is about software architecture, and if you're interested in software architecture, there's no better place to go to discuss and learn about software architecture than the O'Reilly Software Architecture Conference, which is coming to New York February 23rd through 26th of 2020.

If you are interested in software architecture, you can go to oreillysacon.com/sedaily. That link is in the show notes, and you can get 20% off your ticket to the software architecture conference. The O'Reilly Software Architecture Conference is a great place to learn about the high-level perspectives and the implementation details of microservices, cloud computing, serverless and also systems like machine learning and analytics.

If you've been listening to Software Engineering Daily for a while, you know that these systems are hard to build and they take engineering details at both the high-level and at the low-level. Whether you're a seasoned architect or an engineer that is just curious about software architecture and maybe you want to become a software architect, you can check out the O'Reilly Software Architecture Conference at oreillysacon.com/sedaily. Use the discount code SE20 and get 20% off your ticket.

There are lots of reasons to go to the software architecture conference. There's networking opportunities. There are plenty of talks and training opportunities, and you can get 20% off by going to oreillysacon.com/sedaily and entering discount code SE20. I've been going to O'Reilly conferences for years and I don't see myself stopping anytime soon, because they're just a great way to learn and meet people. So check it out, and thanks to O'Reilly for being a longtime sponsor of Software Engineering Daily.

[INTERVIEW]

[00:04:16] JM: Sean Knapp, welcome to Software Engineering Daily.

[00:04:18] SK: Thanks for having me.

[00:04:19] JM: The conversation we're going to have is about data platforms, data warehousing, modern data systems and every company – Well, most companies, start off basically with a transaction data store, and the typical acronym for this is OLTP, online transaction processing. This is typically like a Mongo database, or a MySQL database, and it handles all of the basic transactions, like the user transactions or if you're talking about Uber, it handles all the rides. You call a car, the car comes. The driver information is handled by the transaction. The ride ends.

All these things are logged in transactions, but then a company reaches a point where they have so much data in that OLTP database and then they also have other kinds of data, like telemetry, and logging data and all kinds of other stuff where they start to have all these different data stores and different data systems. They need to start taking advantage of all those different sources of data.

I want to understand what happens from your point of view between that point of just having an OLTP system for the startup or the company that just got started or the new project and the point where it has so much data where they need to do "big data".

[00:05:43] SK: It's really interesting, and I think you described it really well, which is most companies start with a lot of their transactional data, and this data tends to be better structured. It tends to really clear around the size, the volume and the velocity. As you start to get all this telemetric data, what starts to happen is you add one or two, sometimes even three orders of magnitude and you also start to introduce a lot more complexity, and that really pushes most companies into this big data world where we start have to look at entirely different systems for how do we store that data, how do we process that data? Most importantly, how do we really start to refine that data and extract the value from it?

It tends to be just RAR. It tends to actually have more needles than those haystacks and you really have to actually process a ton of that data to create the derived insights that are of similar or incomparable value to the actual transactional data itself.

Most companies usually go from this point of having their OLTP system to then really starting to have data pipelines and it's usually tied to systems that you have data collection systems, then

you have your date lake, and you're building a lot of these complex pipelines to collect, refine and process all of this really raw telemetric data to create these incrementally derived insights around your users or the systems that you actually operate.

[00:07:12] JM: Tell me more about the data problems that the company develops as its builds all these transactional systems and builds telemetry systems and wants to do analytics over them.

[00:07:24] SK: We see a lot of problems start to be introduced when you're building data pipelines and when you're really starting to refine data, and they come from really just a handful of sort of root issues, which is the first thing is you end up starting to have a lot more data and you're going to have it across different systems. You're going to have some in databases. You're going to have in some data lakes. You're going to have some sitting behind other cloud services and APIs. Keeping your entire data ecosystem connected and in sync becomes the first real big challenge.

The second thing that really becomes challenging for folks is how you actually then refine that data and ensure that it's accurate? We find a lot of people start to build pretty complex data pipelines, but you end up introducing this really interesting challenge, where in that database, I actually have a database engine. It has a quarry optimizer. Well, a query planner and a query optimizer.

When I actually start to move around larger volumes of data, I have really powerful storage engines. I have really powerful processing engines, things like Hadoop, or Spark, or Karka. But what I don't have are actually pipeline engines themselves. We find data engineers end up running huge amounts of code to construct and design these pipelines really to try and re-implement a lot of these heuristics and efficiencies you see with a query optimizer in a database to actually manage their data pipelines themselves.

[00:08:47] JM: Explain what you mean by a data pipeline.

[00:08:50] SK: A data pipeline in a really simple sense usually ends up being some form of ETL. I'm extracting data from somewhere, often times a data lake. I'm transforming it. I'm doing

everything from just cleaning the data and normalizing it to even doing complex transformations, like aggregates, roll ups, machine learning, etc., and then I'm usually loading that somewhere else. I'm putting it into a data warehouse, another database, or back into my data lake.

As part of these pipelines, I'm having to actually keep track of the work that I've done before. That's really where a lot of the challenges get introduced is in a databases, in the simplest construct. If I keep running the same inquiry as my data itself changes, if I run the same query over and over again, I get a new updated result. That's where databases have become so powerful.

In a data pipeline world, I am moving data through and I'm copying replicas of data or derived datasets, and this becomes really hard from an engineering perspective, because if my underlying data changes or if update values, I as an engineer have to then really implement an intelligent system to figure out what I have calculated before. If the new data or the change data affects those previous calculations, and where did all that data go?

This is one of the really both interesting and challenging aspects of data engineering, is what happens when there's new data, when there's change data, when there's lake data that introduces a whole host of challenges that just never had before in databases and data warehouses.

[00:10:19] JM: I think what you're saying is company gets these transactional systems. They get these telemetry systems, and let's say they want to do offline data science. Maybe every night they want to aggregates the sum of the amount of money that they've made from all the rides that have taken place across Uber or something like that and then they want to do some transformation on it, like transform like the sum to yen and euros and for whatever reason. Then they want to write all those different things to an Elasticsearch cluster. I don't know. I'm just making up an example.

That's like a three-step process, and that could arguably be a data pipeline where you've got step A, step B, and step C. At each step you have incremental data that gets created and you got to write that somewhere. You got to keep it somewhere, and then maybe you want to throw it out at the end. Maybe that's a fourth step, and there are existing systems for orchestrating

that, and you need to have some kind of system to orchestrate that because if something goes wrong along the way, maybe you need to retry. So, other things can go wrong.

Tell me what the systems are for doing that pipelining for managing those incremental steps of a data pipeline.

[00:11:39] SK: Yeah. I think you break it up into the really appropriate and interesting steps, which are ultimately I'm moving data from these various stages and the technologies and the tools that we use gently factor in to just a few different buckets. We start with where do I store my data? We have the systems that actually process the data. Then there's the tool on top, which actually orchestrates it. Is this system that's sending a lot of instructions to those underlying engines.

What's really interesting as we look at the domain itself and really where we spent a lot of time at Ascend focusing on is this orchestration layer? When we look at how a lot of this segment and where a lot of the technology investment has happened to-date, there's a lot of imperative-based systems. We see a lot of systems that are frameworks and tools that let us write a variety of different code and structures to process data, but what we have actually taken a really big approach on is how do we build a declarative system.

We've taken this approach as we've seen this trend happen a lot in software engineering in general. The move from imperative to declarative systems. SQL is a declarative language. We use Terraform to declare and define our infrastructure. We use Kubernetes to declare and define our container orchestration. The reason that we believe that there's this really huge shift from imperative to declarative systems, it's fundamentally based on the need to maintain and understand state.

When I'm writing all of these pipelines, I mean, we talk about the sample pipeline you talk where I'm processing a bunch of data. I'm converting it from dollars to yen and then I'm loading it into my data warehouse or I'm loading it into Elastic. One of the fundamental needs we have in those systems is to understand what have we done before? What already exists? Then be able to understand what is the delta between what is there today and what needs to happen. The

system itself can very quickly and efficiently generate the new work and accomplish the overall goal.

When we did a survey of orchestration across the industry and looked at all the various tools we had options for, it really kind of breaks into this category of mostly imperative-based systems and/or declarative-based systems. The fundamental difference is as an engineer, are we writing code that essentially in an imperative model inspects state? Defines all the steps to then take us from one state to another.

For example, an imperative system, I would inspect what's already in Elasticsearch. Inspect what's been maybe cached inside of S3. Look at the jobs that are run before. Do my integrity checks. Analyze the upstream data. See if there is any late arriving data, because a server was offline and came back online, and then figure out what that delta was to then process, or in a declarative system, you try and essentially push a lot of that down to the – Usually, it's a control layer or a control plane that then automates a bunch of that for you.

[00:14:34] JM: Tell me how the data platform that you're describing varies between a company that started in the pre-cloud days. We're at AWS Reinvent right now. There's a ton of enterprises that have been around for a very long time and they're going through this "digital transformation" or whatever you want to call it.

One of the things on the laundry list of that digital transformation is to build a data platform. But then you've got these other companies that are post-cloud, like Uber, or Lyft, and they're starting from a little bit more of a clean slate in terms of their data platform. Tell me how the infrastructure of these two worlds differs from one another and how that impacts you as a startup that's trying to build a "data platform" that serves these types of customers.

[00:15:23] SK: Yeah, you highlight something I think really, really important, which is the data ecosystem itself. As we look out across a plethora of different companies, if we look at the digital native companies, they tend to have simpler, more straightforward ecosystems. They tend to store a lot of their data inside of blob stores or a handful of warehousing or data basing technologies.

When we look at a lot of the more established companies and the technologies they use, we find that there's an order of magnitude more complexity. They have more data and more systems and it's really important to actually recognize the fact that while they may want to be modernizing a lot of that tech stack and moving a lot of that to cloud, there is really an incremental migration strategy that's really important.

What that means for technology companies like Ascend is you have to have a highly flexible and highly pluggable system that works with everything else that's in the ecosystem today. The ability to tap into whether it's your data sitting inside of a queue or on a lake, or your data that's actually sitting behind entirely proprietary systems with a totally custom API. You have to be able to plug into things that are talking both GRPC and SOAP. You have to be able to plug into things that may be sitting behind a special VPN tunnel in their on-prem data center, but also then be able to plug into their API partners who are running in Amazon and Google all at the same time.

Having that flexibility of a system that, frankly, can plug into everywhere is super important. I think the thing that stems from that is, frankly, the acceptance that you will have replicas of data trying to pull all of that data into a single canonical system is different than saying we are going to synchronize with these sources of truth and we're going to find really fast and efficient systems that can monitor the upstream data sources, synchronize their data in a way that we can guarantee and ensure that we have the right data in a unified system, but not actually have to require these tend to be more traditional enterprise companies to also then have to move their entire system over.

Finding that balance between how you integrate into all the various systems and then how you have a really intelligent system that can properly synchronize that data and make sure that whatever that your new centralized technology is, is properly in sync with those sources of truth is supercritical.

[SPONSOR MESSAGE]

[00:18:02] JM: Today's show is brought to you by Heroku, which has been my most frequently used cloud provider since I started as a software engineer. Heroku allows me to build and deploy my apps quickly without friction. Heroku's focus has always been on the developer

experience, and working with data on the platform brings that same great experience. Heroku knows that you need fast access to data and insights so you can bring the most compelling and relevant apps to market.

Heroku's fully managed Postgres, Redis and Kafka data services help you get started faster and be more productive. Whether you're working with Postgres, or Apache Kafka, or Redis, and that means you can focus on building data-driven apps, not data infrastructure.

Visit softwareengineeringdaily.com/herokudata to learn about Heroku's managed data services. We build our own site, softwaredaily.com on Heroku, and as we scale, we will eventually need access to data services. I'm looking forward to taking advantage of Heroku's managed data services because I'm confident that they will be as easy to use as Heroku's core deployment and application management systems.

Visit softwareengineeringdaily.com/herokudata to find out more, and thanks to Heroku for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:19:36] JM: How do you avoid becoming a services company in that problem? Because when I talked to companies that are building data platform solutions, what's really hard is that there is such a multiplicity of different systems that have to be plugged into to the "data platform", and there's always more features that people want. There's always more plug-ins that people want, and it really raises the question; is this like data platform thing actually feasible? Because if you go and talk to Uber, for example, Uber's data platform is actually like 50 different technologies that are strung together with duct tape and chicken wire.

The idea of – It would be great if we could just like sketch a rectangular box around like Huber's complicated systems and say, "Here's your solution. We've done it. We've packaged it up. It's a one click thing. Here it is." That seems impossible. It seems really, really hard. So, how do you avoid becoming a company where you go from customer to customer and just kind of have to provide a bespoke system?

[00:20:55] SK: Yeah. It's a really, really good question, and the short answers, one, it's hard. It takes a lot of conviction fortitude. Honestly, frankly, just a lot of compassion for your users and your customers to find that balance.

We certainly see this a lot, and I think IDC even released their report where consulting services and custom development around in the data ecosystem is a $20 billion a year industry and it's growing.

[00:21:18] JM: Okay. So maybe you could. Maybe it's not so bad. Maybe it's not something to avoid.

[00:21:21] SK: Yeah. Part of it is also being like super true to what you are as a company, which for Ascend, we're a technology platform company. How we've solved this, it's a few fold. One, obviously, we worked super closely with a lot of our customers. If they have deep product needs, we really work hard and prioritize those. Any enterprise company should of course do that.

I think the other thing we also did is we partnered with a bunch of other services company so that they can build their businesses with the Ascend platform. I think that's a really healthy balance because you have to know what your strengths are and you got to find the folks around the ecosystem whose strengths are not your strengths so you collectively can build something really cool together.

I'd say the third piece, which is the really interesting one from a technology perspective, is to be able to take a bit of a step back and really think hard around the architecture. One of the actual core values that we have internally inside of Ascend is this notion of built for 10X plan for 100X. You should always kind of know where that next horizon is just to make sure that, for example, you're not painting yourself into a corner, whether it's from a business perspective or a technology perspective.

We spend a lot of time really thinking through and architecting what are the variety of use cases we have. We spent a lot of time with our users actually working through the, "Hey, you want to integrate into X or Y or Z. You want to accomplish A, B or C," and we go adn survey a ton of

folks and then we try and actually boil that down to what are the re-factorable, like architectural components that we can provide that may not give you exactly A, B or C, or X, Y or Z, but if we give you this thing and that thing combined, you can accomplish this, but we can also meet the needs of a plethora of other use cases.

For example, when we think about like how we integrate into a variety of different systems, we've taken the approach where we have a ton of native integrations and a ton of different connectors, but we also designed an entire framework that said, “Look, if you can boil down whatever you want to connect into, into just a handful of concepts, a handful of snippets of Python or whichever language you want to write, we can simply then fit it into that same architecture.

We spend a lot of time thinking through what are these fundamental concepts, and then we find the what are the 90%, 95% of the use cases that we should go and really automate and make incredibly easy to go do? But then for the other 5% or 10%, let's make sure that is still possible. So we give the frameworks on top that people can then really – For all of us as software engineers, can still go accomplish those things too. You just kind of find the balance between those two.

[00:24:01] JM: My guess is that in many situations, you go and talk to a customer and they already have some usage of a combination of open source technologies and close source technologies. Let's say they've got Amazon RDS somewhere. They've got maybe Snowflake somewhere, or they've got Apache Sparks somewhere. Can you help me understand the canonical architecture or a typical architecture of data systems that when you start talking to a customer and that conversation proceeds eventually to a sale, what are the systems that they have in place and what problem are you solving for them?

[00:24:50] SK: Yeah. We see often times that sort of segments into a couple of different configurations, and a lot of it goes back to your question of are they a larger customer who has a lot of traditional systems, for example, came from the on-prem world or the more of a digital native. In the digital native world, we tend to see a lot of customers where they come in and they've already invested in collecting a ton of data. They have it coming in through Kafka or other technologies and they’re collecting that. Some of the data stream interim being processed.

A lot of the data is being collected inside of a data lake. There're putting into a blob store like Amazon S3, or the Google Cloud Storage, or Azure Blog Store.

What we find then is most companies are – They're processing a bunch of these. They're running a ton of Spark jobs or maybe Hadoop jobs on top of that. What they're doing to orchestrate all of that is they're using technologies. Usually nowadays it's Airflow. Basically it's really the dominant orchestration tool, and they're orchestrating a ton of these jobs to then move the data into usually Snowflake, or Red Shift, or Elasticsearch and then they're feeding their downstream systems from that.

Where we really come in and help our customers is by simplifying a lot of the code around that orchestration tool and system itself. We come in with the Ascend platform, which is an alternative for orchestration. We think of –

[00:26:11] JM: Alternative to Airflow.

[00:26:12] SK: Yes. Exactly. If you think of Airflow, it's a beautiful and incredibly powerful framework in technology, we come in with much more of a wholly success approach, and we allow our customers to shift that paradigm from an imperative-based system to a declarative-based system. We end up really helping folks cut out huge chunks of code that are just orchestration code, it's scaffolding, it's parameterization, it's things like all the code you had to write for lineage tracking and state management and data persistence and intelligent declarative system that, for example, with us has a backbone of a control plane. That maintains all of the state, maintains all of the lineage. Knows how to actually persist data at various stages.

We come in and can take that same logic that they're applying to the data, but greatly simplify the system. The equivalent would be, for example, our world as software engineers before and after use of Kubernetes, for example, where we have really advanced declarative container orchestration and we get the same thing with data pipelines.

Ascend really comes in and just introduces this layer that sits on top of Spark and on top of blob store that if we think about how the technologies work today, a lot of the investments have been in how do I store more data and how do I process more data. But the Ascent approach is how

do we actually make those underlying systems more efficient by being able to write more pipelines that are lower maintenance burden and our intelligence layer just simply sends smarter things to those systems to begin with.

That's really where we start to come in, is most folks who have worked with Airflow or other orchestration tools, we often times even find folks who are [inaudible 00:27:49] a bunch of scripts and doing a lot of this manually. The number one pain point they feel is these systems are brittle and I'm constantly having to triage and troubleshoot. I think the number one question that we ask folks, which is how confident are you that your systems are actually generating the right data?

If you looked at your warehouse, your database, your data lake, everything that sits downstream from your pipeline, how confident are you that it actually has the right data and that your system did the right work and you didn't have to triage it. You didn't have to troubleshoot it and that you actually know that the data that ends up there is correct.

The reality is most people can't actually guarantee that, because the system itself, they knew they ran some code on some set of data at some point in time, but they're writing all of these safeguards into their system trying to ensure that it's actually accurate and that is actually efficient. That's really been –Our focus has been how do we hope you architect that pipeline but can trust a smarter system that can maintain full state and guarantee that your pipelines are running as they should be.

[00:28:52] JM: Okay. If we zoom in on Airflow, many of these companies are, as you said, building systems with Airflow that allow them to have a series of data processing jobs that together compose a data pipeline. The before is some OLTP systems or just various data, data lakes, data platforms, databases, and the output is some aggregation or rollup or series of data transformations that have taken place.

If we zoom in on Airflow, the first thing you pointed out is that – As I understand, Airflow is imperative – Well, you're usually writing Airflow jobs in – What? Python? You're writing imperative programming using Python and you're suggesting that if you moved to a declarative

configuration-based approach to writing your Airflow jobs, that'll be superior, or at least superior for some subset of users.

In addition, it sounds like the fact that you are a SaaS solution means that you can do things like very easily wired together some storage such that these jobs can have – When they're doing their incremental processing, when they're doing their different stages along the data pipeline, they can save the intermediate data representations as the database is moving through the pipeline, right?

[00:30:24] SK: Exactly. When we introduced the notion of a declarative system, we're really defining our pipelines, and you pushed down into that control plane, really, the responsibility of executing. It's sort of the separation of logic definition and execution. You can then do really, really advanced things that we weren't able to do before in imperative systems, because frankly it just takes too much time and too much code that we have to repeat over and over and over again.

Really, at the core of, a sense, control plane, and the really exciting part about what it enables is imagine if you never had to run the same block of code on the same piece of data more than once. We have a really advanced – What's basically a recursive SHA algorithm on both the code that's operating on the data as well as OLAP to the data itself that ensures that it doesn't matter if you copy all of the code for a data pipeline. We won't rerun any of the work. It doesn't matter if you're moving from prod, to staging, to dev, and backup to staging to prod.

The system itself has a deep understanding of what is the logic that was applied to that data. You get a whole world of efficiencies. You get to do things like really proactive persistence, which from the other worlds that we've all been engineers in and think of it as like really smart caching, right? But we get to do things like never reprocess data, whereas in most data pipelines today and most data systems, think of how many times you run that same snippet of code on a same block of data or a same file or a same record, because we're reprocessing. We're re-pulling that data out of the lake. We didn't know how to safely persist it and cache it. So we're reprocessing it.

All of a sudden in a declarative system, they can really actually just keep tabs on every piece of data. How it was generated. What code ran on it? Why it was generated? Then most importantly, even where it went. We can do all these really cool things on top of it. We can make sure that we never reprocess data. We can de-duplicate everything that flows through our system. We can track lineage so that we have to go chase down automatically, of course, a piece of data because it was updated or it was retracted. A system that is declarative in nature can actually do all that in a fully automated fashion. Whereas in an imperative-based model, as an engineer, the simplest example is what do you do when you updated a file inside of your data lake and you had to retract some data?

In an imperative-based system, you actually don't have state in the same way. So you don't know where all of that data went. You have to just go rerun and reprocess huge volumes of data and hope that you actually caught all of the sort of trail of that data and got the database updated appropriately and so on. I think that's really the important part as we evolve out of these imperative-based systems where entire notion of state management and tracking of like historical calculations was pushed on the developer. In a declarative model, you can actually then push that burden back down to the system itself, that control plane, and let the developer frankly be far less of a plumber and far more of an architect.

[00:33:26] JM: The configuration-based approach, is that something you can open source?

[00:33:32] SK: Yeah. It's a really good question. We try and align with a lot of the standard methodologies and definitions how we actually define all of our pipelines. The definitions are open and can and frankly should align with any of the rest of the open source community. This is one of the things that we actually found was really interesting as we started to work on Ascend, is the definitions of the pipelines generally tend to not be that complex. You're defining data sources, which is here's the location of my data. Here's the schema I expect, and that usually is roughly sufficient, versus you have some conditions around what your error handling is and so on.

When we start to write transforms on data, frankly, you're just writing snippets of code. If I'm writing SQL, I don't even have to specify output schemas because it's all infer. If I'm writing Python or PiSpark even, we can actually infer a lot of the output schemas as well. So what you

start to actually find is the entire definition of pipelines when we remove the orchestration tool or the orchestration logic and we actually separate just the data logic from the data execution or the orchestration, the entire notion of how you define a pipeline becomes incredibly clean and efficient and elegant because it really is just simple transformations on data and logic itself. It becomes as easy as something would be ready on your local desktop or be crying inside of a database.

Going back to your point, I do think it would be phenomenal if the industry, and Ascend is part of that too, really start to standardize an open source. This is how we define data pipelines and it really does help for us as we started to invest in what we call our declarative pipeline workflows, is this really simple combination of YAML, Python and SQL to just define here is a data pipeline that is decoupled from its application to the data itself. I think that's what becomes superpowerful and elegant, is it really is simple and easy to understand these pipelines as you clearly define those boundaries.

[SPONSOR MESSAGE]

[00:35:48] JM: Apache Cassandra is an open source distributed database that was first created to meet the scalability and availability needs of Facebook, Amazon and Google. In previous episodes of Software Engineering Daily we have covered Cassandra's architecture and its benefits, and we're happy to have Datastax, the largest contributed to the Cassandra project since day one as a sponsor of Software Engineering Daily.

Datastax provides Datastax enterprise, a powerful distribution of Cassandra created by the team that has contributed the most to Cassandra. Datastax enterprise enables teams to develop faster, scale further, achieve operational simplicity, ensure enterprise security and run mixed workloads that work with the latest graph, search and analytics technology all running across hybrid and multi-cloud infrastructure.

More than 400 companies including Cisco, Capital One, and eBay run Datastax to modernize their database infrastructure, improve scalability and security, and deliver on projects such as customer analytics, IoT and e-commerce.

To learn more about Apache Cassandra and Datastax's enterprise, go to datastax.com/sedaily. That's Datastax with an X, D-A-T-A-S-T-A-X, @datastax.com/sedaily.

Thank you to Datastax for being a sponsor of Software Engineering Daily. It's a great honor to have Datastax as a sponsor, and you can go to datastax.com/sedaily to learn more.

[INTERVIEW CONTINUED]

[00:37:28] JM: If you go to a potential customer and they are using Airflow and you want to convince them to use the Ascend declarative-based platform, where do they start? Do they refactor an old workflow that's in Airflow already and they rewrite it in declarative syntax or do you have like an automated way of generating declarative syntax from their imperative syntax, or do you ask them to do a greenfield application? How do you get them to start using Ascend?

[00:38:06] SK: Yeah, it's a great question. One of the things we always do for our customers is we offer a free dev trial just like most SaaS offerings. We try and make it as fast and efficient as possible. People go to our website like here at Reinvent where we'd send everybody towards that trial, and literally within 60 seconds, they're logged into the platform. We give them a bunch of sample data so they at least can sort of familiarize themselves with how to build a pipeline on Ascend. That really helps them at least start to understand the ease and capabilities and power of a declarative system.

Then we really start to get to the, "Okay. I understand the concepts, and this seams really neat." Really, you then start to get into this. Do you want to go about a new greenfield pipeline? Do you want to migrate pipelines over?

What we find for most teams to be really frank is I've yet to find in the 15 years for myself as a data engineer and the four years for Ascend as a technology provider, I've yet to find a data engineering team that has just idle capacity. Every data engineering team in the world is basically at half the headcount in team size of what they actually can and should be.

If you go talk to data engineering managers, nobody's like, "Oh! I've totally hired my entire team and I have no open headcount. We're just great." Every data engineering team managers is like,

“Hey, I have headcount for 10 or 100 and I have half of that and I have no line of site to actually go hire anymore. We have a three-year long backlog of projects,” etc. etc.

The reality is like, “Look, most data engineering teams are pretty crushed with their workload and they're trying to actually stay afloat from all their existing pipelines and the technologies that like worked great at smaller scale. Not actually smaller scale from data, but smaller scale of company size and team and people were tapping into them and they're trying to figure out how do I rapidly retrofit this thing. How do I basically just like stop the pain and then figure out how I migrate and upgrade at the same time?

To your question, really what we find is there're a lot of really cool greenfield projects, the biggest impact for a lot of folks often times is, “Hey, what is the most painful thing that you're dealing with today? What is that pipeline? Let's dive in and actually hope you really rapidly migrate over that most painful thing.” We found is we've sat down with teams and we don't have a fully automated system yet. We've migrated hundreds of thousands of lines of existing pipelines in a matter of a couple of weeks to the Ascend platform and cut out 90+ percent of the code that’s required. This is the really cool part about it, is it becomes so fast and efficiently when you’re actually refactoring and separating your logic definition from your orchestration code and the execution layer. That is remarkably fast to move to that declarative model.

What we find with our users and our customers is, frankly, it's incredibly freeing to actually just carve out all that stuff around the actual logic that you're architecting and you’re really quickly start to find that if you go tackle the most painful pipelines that are breaking, that are actually creating a lot of pain for your downstream systems, you then actually start to free up enough time and enough capacity to go chip away at the other things and then get to your really cool greenfield projects.

It may not be as exciting and sort of glory filled to help people alleviate a lot of that pain, but gosh, it really does make their life's a heck of a lot easier to just upgrade and automate the most painful pipelines they’re working with.

[00:41:37] JM: Have you been able to get people to move their workflows entirely over from using Airflow to using Ascend?

[00:41:49] SK: Yeah, we have, and I think it depends really on how big the team, how the big company. One of the things that's really important to us is you certainly should be able to move everything over. But it's also really important that you don't have to. We're all software engineers, right? Different teams have different tools for different use cases, and it was really important for us as we designed and introduced Ascend into the market that, "Look, we can and should still be able to run side-by-side with Airflow and we should be able to automate and orchestrate a bunch of pipelines and still actually feed data in and out of Airflow orchestrated pipelines at the same time."

We find that certainly for a number of folks, we can move everything over. But you don't have to. I think that's really important, because if it's an all or nothing thing for your customers, then we have this notion of one-way and two-way doors. If it's all or nothing, that feels a lot more like a one-way door that you can't walk back through and you maybe more encumbered, whereas for us, we can hook straight into things that you're doing with Airflow. You can run Airflow pipelines on top of. [inaudible 00:42:49] platform tapping into our internal data stores. As a result, this gives you as an engineer a ton of flexibility. You just, frankly, get a smarter orchestration system for many of your pipelines.

[00:43:01] JM: How do you price a product like this?

[00:43:06] SK: Pricing is really interesting. It's a good question. We've spent a lot of time trying to figure out a good pricing model. For us, fairness and sort of clear transparent pricing is super important. When we priced, and what we've really settled on is an entirely variable Elastic-based pricing model. Same thing that we get when we're buying EC2 or Google Cloud Storage or any of the other sort of various cloud offerings.

We added a really important component to this, which is not only is it Elasticand Invariable, but the other thing we also added was you're actually paying for use as supposed to capacity. When we buy most big data products today, it doesn't matter if we're buying the data warehouse or a Spark cluster or anything else. Usually, what you're paying for is not use. You're paying for your capacity. You're reserving 18 nodes of X, Y or Z size and you're paying for that, whether or not they're actually processing anything.

The pricing model that we introduced was much more of a Lambda model or a serverless style pricing model, which is you get billed for the seconds that those core and memory are actually being used. Frankly, on the Ascend side, if we're popping a bunch of workup on a 16-core box, but you're only using two cores of that, we're only going to charge you for the two cores for the seconds that you're actually using it, even if there is a whole lot of inefficiency and waste around the side, because, frankly, in the appropriate model, we feel that optimizing that should be our burden.

For you as a developer, you should just be focused on when are your jobs running and how much data are they processing. It really frees you up from not having to worry about what is the size of my cluster? What are the instances that's running on? Do I have the right CPU to memory ratio? You just get to focus on the data itself. Frankly, for us, it feels better for engineers.

[00:44:54] JM: We're in this time where the major cloud providers – Well, specifically, AWS, is simultaneously beloved and also in some cases kind of resented basically because of how good and unified it is. You have the all-in on AWS people, but you also have the mostly in on AWS people. It's like who are 80% in on AWS. Then we have a collection of other providers that were willing to use. We use Snowflake. We use Databricks. We use Datastacks. We use maybe MongoDB, but 80% of our infrastructure spend goes to AWS and it's going to take a lot of work to convince us that we should be using your point solution that is not AWS because of a couple of reasons.

One, we don't trust infrastructure that's not as good as AWS and it's going to take a ton of work for you to convince us you're as reliable as AWS. Two, whatever you're doing, if it's good enough, AWS is going to build it next year. Why would we buy-in today? We've got enough stuff to work on. We'll just wait a year and we'll get an AWS fully integrated solution. Why on earth should we go with you point solution provider? So, how do you build enough market share in that environment?

[00:46:24] SK: Yeah. It's a really good question, and I think you hit on a lot of the right points. The approach we've taken is a few fold. The first thing is we've worked really hard to be really

tightly integrated inside of AWS. We're integrated in some of the marketplace. So you can just buy us like any other Amazon service, which we find a lot of folks enjoy just because it's easier to procure. It's actually even talked under a lot of the standard cloud provider enterprise agreements. Frankly, for really big customers, we even have an enterprise deployment where we'll deploy inside of their own infrastructures. Their infosec and ops teams actually have access to all the underlying engine and is totally isolated and so on, which is great.

I think when we think about why won't Amazon or, frankly, the other cloud providers too, Azure or Google just go introduce what you have over time, I think the answers is really a few fold. Simply put, at least for us, as we're one of many point solution providers, but for us, this technology is hard. Running highly autonomous big data systems, this is not new. Amazon has had the opportunity to create a lot of these stuff for many, many years and it has to has Google. It has to has Azure.

I mean, heck! I wrote my first MapReduce job in [inaudible 00:47:37] at Google in 2004 as an engineer there. We've all known that this is a thing for 15 years, at least for me personally. Yet people still haven't actually solved the problem, and I think because frankly it's really hard to solve. We spent four years SN creating this engine, and we knew that it was going to be a very, very problem, because what we're basically doing is we're creating a pipeline engine itself. It is the push equivalent of how a database engine and a query planner and a query optimizer works. This is hard technology. It's not grabbing something that is off-the-shelf open source and SaaSfying it. This is a very fundamentally new and hard technology to build.

It may take them more than a year, maybe two. Ultimately, over time, we do believe that the world will move from imperative to declarative. We know all three cloud vendors have their imperative-based orchestration tools. Let's assume that they all do deeply understand that they should be building declarative orchestration. They should be moving into far more advanced systems. Then we really get into the world of why would you use that point solution versus one of the cloud specific vendors?

Our approach has been actually very similar to what even Google does with Anthos and Kubernetes, which is you really to need to be able to run an entire multicloud strategy. One of the huge benefits of using somebody like Ascend is we can run on all three clouds. We can

connect into any of the various systems. The beauty of this is – And we've had customers who have literally gone from one cloud to the other, and migrating over is literally like outside of the data gravity is. If I actually copied the definition of a pipeline from one cloud to another, our system itself will start to execute and self-heal. We'll find the data. We'll move it over. Reprocess it if appropriate, and you get this level of cloud portability that we haven't really seen before, especially for data and pipelines.

Similar to why we observed that there is such excitement around Kubernetes and the freedom that gives you across cloud providers, we think that there's this same thing around data pipelines when using frankly a neutral third party that is not just cloud-agnostic, but really optimized to run across all three at the same time.

[00:49:53] JM: What's the biggest strategic challenge that you have in terms of go-to-market strategy today?

[00:49:59] SK: I think the biggest strategic challenge today is most of the industry today is really well-versed in imperative systems, and nobody to-date before Ascent had introduced declarative orchestration. So, frankly, it's a lot of for us really academic and philosophical conversations with people introducing this notion of what declarative is. I mean, think about how we used to process data before we had SQL.

The introduction of SQL as a declarative language for processing data introduced a fundamentally different paradigm. Today, engineers are really experience. We know that the benefits of declarative engineering. We used Terraform for infrastructure. We used React on the frontend. We used Kubernetes for container orchestration. But we have so many conversations with people who just haven't actually thought about that for pipelines. The industry is just been – To be really frankly, has just been beaten down to accept how painful these imperative-based systems are.

We love the opportunity, and this is why we offered tons of free trials and just want people to get their hands on the product, because that's really what starts to open up your mind around how much better the world can be when you move away from imperative-based systems into

declarative. It's just that education and socialization across the industries is really the biggest go-to-market both challenge and opportunity.

[00:51:26] JM: I've talked to a lot of people building data products in the last couple of years, and it's truly interesting market right now, I think, for many of the reasons that we've discussed. Do you have any advice for people who are building data products today? Whether we're talking about databases, data warehousing tools, data middleware, all of these things are really hard to build, because it's such key infrastructure. Its core infrastructure, and buying from startup is pretty concerning, because if there's a bug and it messes with your data integrity, then it can be catastrophic. There's that challenge out of the gate, and then there's all the typical challenges of a startup. But what advice would you give to data startups, the hard-earned advice over the last 4 or 4-1/2 half years?

[00:52:25] SK: For advice for data startups in particular is I think the industry for the longest time has I think been solving incremental problems. We've been trying to store a little bit more data or process a little bit more data. To be really frank, I think as a result, there are still huge inefficiencies for our lives as data engineers. Of course, I'm biased at Ascend as we followed and scribed to this, which is go solve the big, hard problems.

Why I created Ascent was, as I mentioned, I was writing data pipelines inside of Google 15 years ago. The last company I founded, we processed insane amounts of data at incredibly high-volumes and velocity. When I looked at the thing that actually made our lives hardest as data engineers, it wasn't like do I have a bigger, better, faster storage or processing system? It was I feel like every single day I'm reinventing the wheel and just writing the same kinds of code and heuristics every single day.

In the big daily ecosystem and just data engineering in general, we have all this excitement around we're applying data and AI and machine learning to solve all the rest of the world's problems, but we should really actually turn that intelligence back on to our own jobs for how we actually solve our challenges and pain points and data engineers and make our lives easier as well.

[00:53:52] JM: John, thanks for coming on the show.

[00:53:52] SK: Thanks for having me.

[END OF INTERVIEW]

[00:54:03] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[END]