EPISODE 1011

[INTRODUCTION]

[00:00:00] JM: At Airbnb, infrastructure management is standardized across the organization. Platform engineering teams build tools that allow the other teams throughout the organization to work more effectively. A platform engineering team handles problems such as continuous integration, observability and service discovery.

Other teams throughout the company use the tools that the platform engineering teams build. For example, there's a team of Airbnb that builds the search and discovery system that's used by customers who are looking for a place to stay. That team does not want to have to worry about how they're deploying software and how their service is being law logged and how to scale up or down. All of that should be taken care of by the platform engineering team.

At a large company like Airbnb, there is so much happening across the infrastructure. Services are being deployed. Services are having outages. Databases are being resharded. With all of this change occurring, it can be difficult for a team to pinpoint the cause of a service outage. Digging through logs and dashboards is often insufficient. That's not what you want to be doing during an outage.

Joey Parsons is the founder of Effx, a company that is building a platform for observing and managing the changes across infrastructure. Effx is like a newsfeed for a service. An application instrumented with Effx gives the engineers a single endpoint that they can navigate to for understanding the history of their service. Joey joins the show to talk about his experiences in infrastructure engineering at Airbnb and how that experience informs the work of his new company, Effx.

[SPONSOR MESSAGE]

[00:01:41] JM: This episode of Software Engineering Daily is sponsored by Datadog. Datadog integrates seamlessly with more than 200 technologies including Kubernetes and Docker so you can monitor your entire container cluster in one place. Datadog's new live container view

provides insights into your container's health, resource consumption and deployment in real-time. Filter to a specific Docker image or drill down by Kubernetes service to get fine-grained visibility into your container infrastructure. Start monitoring your container workload today with a 14-day free trial and Datadog will send you a free t-shirt. Go to softwareengineeringdaily.com/datadog to try it out. That's softwareengineeringdaily.com/datadog to try it out and get a free t-shirt.

Thank you, Datadog.

[INTERVIEW]

[00:02:37] JM: Joey Parsons, welcome to Software Engineering Daily.

[00:02:38] JP: Thanks for having me, Jeff. I'm excited to be here.

[00:02:41] JM: You've a lot of experience, Rackspace, SugarCRM, Cloud, Flipboard, Airbnb. Give me your condensed history for how infrastructure has evolved over the course of your career.

[00:02:51] JP: Sure. Yeah. I guess I've been at this for almost 20 years now, which is actually pretty crazy to think about. I just came to that realization over like the last year, and kind of like looking back at my career, it's kind of been almost two different decades of different types of work, right? When I first started my career, it was back in the very early like managed hosting days of Rackspace.

When I joined, we were about 100 employees. We had two data centers and we simply had a bunch of different like server sitting on bread racks that we would rent out to customers. From there, we ran basically a support team where we were kind of like the systems administrators for all of the customer. They would call in, have a question about something that was going on with like their Apache LAMP stack or something with like a sendmail or not being able to like send emails to their customers. On kind of like a dime, we would pick up a phone, immediately began kind of troubleshooting the issues and solve questions for them.

It was kind of like a really unique environment where you're kind of thrown into the fire every day and you literally had no idea what you are going to be working on. That was probably kind of the working in kind of like data centers and physical servers running in data centers is probably like the first 10 years of my career.

At SugarCRM, we rented out co-located cages from different providers in the Bay Area and literally ran kind of like both the data center operations, the software that ran on the servers and the business that rental on to their servers as well. Fast-forward 10 years after that to kind of like the next kind of evolution of my career where I worked at Cloud, Flipboard, and Airbnb. That was much more kind of like cloud-oriented primarily running on top of like AWS infrastructure. Cloud was pretty much in our migration from running primarily in our own data center to then like running some software in AWS. That was my first kind of experience working in that world. But then Flipboard and Airbnb were like cloud-native from the start running 100% on AWS.

[00:04:53] JM: Right. As you said, Airbnb was one of the first very big cloud-native companies to really, really scale to something gigantic. Do you remember any problems that there was no roadmap for how to solve being at a hyper scale cloud-native company where there's not really a roadmap to follow?

[00:05:15] JP: In some sense. I think one of the things that we ran up in Airbnb is we would constantly push the limits of like what AWS could provide. One of the fun scale challenges there was that, simply, every Sunday was our biggest day of traffic ever, and then every Monday was our biggest day of traffic ever. Not just from kind of like via our own infrastructure and being able to scale our own applications, but we would run into like the maximum size of like a database that we could get from RDS.

Then after, we'd really have to kind of figure out how are we going to scale beyond like the top level services that Amazon provided. It wasn't necessarily a ton of kind of like features that we're missing on the cloud, but it was really just kind of about like how do we continue to scale at the scale that we were going at that point.

[00:06:03] JM: You were at Airbnb for three years, from 2015 to 2018. Describe the infrastructure when you joined the company.

[00:06:09] JP: Yeah. When I first joined Airbnb, we were already kind of like a powerhouse company at that point. Had already kind of like obtained unicorn status and we're already kind of like a force in the industry. At that point, the application was still primarily kind of like a monolithic Rails app, right? The same monolithic Rails app that soon will be on Rails app that our CCO, Nate, had created many years before.

While there had been kind of like a move to service-oriented architecture at that point, it still was primarily the lion's share of like all the engineers were working on this one Rails app. We had our own kind of like deploy tool that a few engineers have spoken about before in other conferences, but our own deploy tool to basically take that Ruby on Rails app, run a bunch of test against it and then basically run it across a bunch of EC2 servers, right? That was like the primary like mechanism for most of like the Airbnb functionality to be delivered to our end user, and all of it was running on top of AWS at that point.

[00:07:13] JM: The team that you worked on, I know this is around infrastructure, would you describe it as a platform engineering team?

[00:07:19] JP: Yeah. We went through different evolutions of kind of like what the actual like teams kind of like were called, but essentially we were basically building the foundation for how all of the other engineers interacted with the infrastructure. Part of our charter was to make it so that every single engineering team could operate effectively without really needing to understand kind of like what was happening underneath the hood. You want most of your engineers in a company basically building leverage for the business by building products. Shipping new code that's going to no impact our end users, and our team was basically building kind of like all the underlying infrastructure that made that go.

[00:08:02] JM: What was your process for figuring out what those business service teams actually needed?

[00:08:10] JP: A lot of it was just talking to them, right? We would look at how they were able to basically ship code every day and look at how long did that took. Then also look at how reliable the services were that we were building, right? Some of it was reactive in terms of we would

have different types of downtime or we would run into particular issues with like getting code out in kind of like a timely way. Then we would prioritize that kind of stuff pretty heavily. Then on the other side, we would talk to them about what they needed and what they needed to have built and we would end up going and building that for them.

[00:08:45] JM: How much did you try to standardize? Did you try to standardize everything they used? The CI pipeline? The deployment method? Are you using containers? Are you using VM's? How strict did you want to be on the standardization?

[00:08:58] JP: I think what we wanted to do was basically create like a golden path. If you used kind of like the tooling that we provided, then we would provide support for you. There would be assistance that everybody could really understand what was happening in the incident or when something went wrong. If you kind of went down the golden path and use the languages that we build kind of like hooks for, then you were ended up in a better state than if you chose to kind of like use something off that golden path. But for the most part, most teams chose to kind of like go down the golden path just because of the support that they would get.

[00:09:33] JM: What was within that golden path? Could you tell me about what composes it?

[00:09:37] JP: Yeah. A lot of it was just kind of like choosing kind of like the right languages, choosing the databases that we use. Also choosing the observability tools that we provided. While we did use third-party providers, we had kind of like – We had particular use cases in terms of how we use them. It was basically a path of kind of like everything that was around the operability of your service. If you chose to kind of go down the golden path, then it was well supported by us.

[00:10:04] JM: Can you tell me of the blessed languages and the blessed databases or is that stuff that you can't talk out it?

[00:10:09] JP: It was pretty simple stuff. There's nothing like quiet too interesting there. We had particular types of relational databases obviously that we used and then a little bit of like NoSQL databases that we use. But for the most part, it's nothing quite atypical.

[00:10:23] JM: Do you support the data science and data infrastructure teams also?

[00:10:28] JP: We had our own kind of like foundational teams within the data world and they supported kind of like the data infrastructure environment, machine learning, data science on top of that, and that wasn't kind of like within the purview of like what we did on production.

[00:10:41] JM: Why are those needs different?

[00:10:43] JP: There are different use cases, right? In some cases, they overlapped, but for the most part you have kind of like your production serving environment that impacts – Like directly impacts your end users. While on the data warehouse side, some of those artifacts may end up in data stores that end up going to the end-users. For the most part, there's very different kind of like use cases between kind of like your almost back-office data systems with the production serving data.

[00:11:09] JM: When you talk to people that – I'm trying to think of companies in your – Like your contemporaries, like Netflix, Uber, Lyft, Stripe, Instacart. Did the methodologies of platform engineering, were they the same from company to company or do you feel like is there something distinct about Airbnb? Is there something distinct about every company about how you should do platform engineering?

[00:11:36] JP: I think that there's definitely consistency across the different platform teams across these kind of like hyper growth companies that you mentioned. But then there's also obviously like differences. A lot of it is just kind of when you end up doing things a particular way for a few years, that kind of ingrained its way into your culture and your development practices. But for the most part, a lot of infrastructure leaders across those companies talk and they get ideas from each other. Obviously, people move companies and a lot of like that mindshare spreads pretty quickly.

If you look at kind the evolution of Airbnb from a monolithic Rails app to then using microservices, we had senior engineers coming to our team from the likes of Google, and Facebook, and Twitter, and Lyft that were heavily influencing kind of like our move from a monolithic Rails app to then kind of like the snout world of microservices.

[00:12:32] JM: In my conversation with some of these different companies, there are these high-level business concerns that end up translating to how certain platform engineering team constraints work. For example, if you talk to people at Netflix, like building infrastructure for doing streaming and bitrate ladders and stuff has all these downstream impacts on like how you're surveying and what are the things they want to spend time on to optimize.

If you look at Uber or Lyft, the end user has this flaky network connection and the sensitivity of having information relayed as quickly as possible and being super reliable is really important, because these people are in these kind of like situations where you're like in a car with somebody. That has downstream effects to the infrastructure. I'm trying to think of what are the canonical business problems that feed down into how you would do infrastructure management.

[00:13:32] JP: Gotcha. I think in going back to your earlier question around consistency, I think for the most parts, every company wants to build a reliable environment that is cost-effective, depending upon your business, and also has no performance characteristics that meet the requirements of their particular types of end-users, right?

For the Uber and Lyft thing that you mentioned, they have much more of a real-time component than a company like Airbnb, but one of the most interesting things about Airbnb is that you never know where your users are around the world. Not only is there this component of like not being able to like geographically distribute your data because a person might fly from Japan, to the East Coast, and then to South America all within a day and stay at different Airbnb's.

Some of the most unique Airbnb's are going to be – And probably some of the most interesting areas in the world where there's not going to be a lot of great connectivity. We focus a lot on kind of like edge reliability and edge performance in a way that most people probably wouldn't think that Airbnb would simply because of where people were around the world and kind of like the greatness or the weakness of the networks they were actually using at that point. That was a big kind of effort there.

[00:14:50] JM: You would have to like find CDN infrastructure in Pakistan or something.

[00:14:54] JP: Yeah. It was kind of like finding some of like the best-of-breed CDNs that we could work with globally and figuring out a way to kind of really ensure that that CDN was the best use at that given moment based on kind of like real user metrics that we had seen at the point.

[00:15:07] JM: By the way, speaking of CDN's, your time frame, 2015 to 2018, and I think even maybe subsequent to that, the CDN's have really become like full-on cloud providers with all this like processing functionality. Was Airbnb taking advantage of that?

[00:15:24] JP: Oh, yeah. Yeah. I think there's a lot of interesting things that are kind of moving towards the edge these days, like whether it's on the Fastly side. Being able to ride Varnish at the edge there, or even with like AWS CloudFront. The fact that you can do Lambda at the edge and process data before it actually makes its way all the way back to your data center or have a little bit more intelligence around how cached objects work at the edge is actually quite a powerful mechanism that obviously wasn't available 7, 8, 10 years ago in kind of like a really easy way for an engineer to grok. Yeah.

[00:16:00] JM: How did Airbnb handle build verse buy?

[00:16:02] JP: Again, when you're a growing business like Airbnb, you really want all of your engineers, like your human capital, working towards solving business problems that can help that the business can leverage. We leaned pretty heavily on the buy side of things if there was a vendor that really made sense. In terms of like, obviously, on the cloud side, we primarily used AWS and leveraged as much as we could there from like the data store side to the compute side to really kind of not have to think too much about what we were doing on those aspects.

Then from kind of like an observability standpoint, we were heavy users of Datadog from the start and really leveraged what we could from kind of like both the vendor and open source world to not have to spend or not have to have so many engineers basically working on building those tools for us internally.

[00:16:58] JM: When you think about the word multi-cloud as it applies to a company like Airbnb, there's a number of ways you could practically approach it. You could say, "Look, we need full-on failover. We need to be able to failover all of our services. We need to be able failover all of our databases. We need to be able to failover all of our data lake. We need to basically replicate everything over to Google Cloud." That's basically unrealistic today.

You could say there are some key pieces of our infrastructure that we need to replicate over such that in the case that AWS somehow magically goes down, we're going to try to accommodate that. You could just say, "Okay. Look, AWS is on the critical path. There's no getting around it. If AWS goes down, we go down." How do you navigate that spectrum of choices when it comes to heavily rely on AWS?

[00:17:46] JP: Yeah. I think that it's not even so much a multi-cloud question or going all-in on AWS question, it really kind of just comes down to kind of like the needs of the business. It really just kind of boils down to kind of like what like your recovery point objective is and kind of like the time objective. A lot of folks will kind of equate this to kind of like am I active-passive, or am I active-active? Td those are really simplistic terms to describe the world of failover. But as your point objective and hour ago and you need to be able to get there in 30 minutes or is it 10 seconds ago and you need to get there in like one second? And a lot of that just really pertains on the type of business that you're running and really what you're able to provide and how much effort you want to put in that way.

I think like if you can get there by going multi-cloud and that's something that from kind of like a failover standpoint is important to your business, then obviously like the investment is worth it. But if you don't have like a point objective or a time objective that's so incredibly tight, if you can get by with coming back at that tail hater or something like that and taking a day to restore and that's how you invest your time, then that that could make sense for a business. But it's really just a kind of around what do you actually need to have happen if in case of kind of like a total failover.

[00:19:10] JM: Tell me about the observability tooling at Airbnb.

[00:19:12] JP: Yeah. For the most parts, we really kind of like leveraged external vendors. It wasn't something that we necessarily wanted to have a bunch of core expertise in at the company. It was obviously like we were predominantly like a Rails app. We used New Relic for APM, which was actually fairly great. It's like you just drop a Gem in and you get all of this observability like built-in for pretty much free. Then for kind of like custom metrics and your typical kind of like how well was this actual server doing and kind of the metrics that would come out of the application, we use Datadog primarily for that.

There's been some kind of like new tooling that's been built over kind of like the past couple years in terms of kind of like how to do distributed tracing within the system especially as they've moved from kind of like primarily like the monorail into microservices, but a little of that's been developed past my time.

[SPONSOR MESSAGE]

[00:20:09] JM: Logi Analytics believes that any product manager should be successful. Every developer should be proud of their creation and they believe that every application should provide users with value. Applications are the face of your business today and it's how people interact with you. Logi can help you provide your users the best experience possible. Logi's embedded analytics platform makes it possible to create, update and brand your analytics so that they seamlessly integrate with your application.

Visit logianalytics.com/sedaily and see what's possible with Logi today. You can use Logi to create dashboards that fit into your application and give your users the analytics that they're looking for. Go to logianalytics.com/sedaily.

[INTERVIEW CONTINUED]

[00:21:07] JM: Speaking from just an industry analyst standpoint, Datadog is a sponsor of the show. So I have to be like I guess disclosing of that. But so many logging companies, so many observability companies. Why was Datadog able to be so successful?

[00:21:22] JP: I won't necessarily give like the full like Airbnb viewpoint on this. It's a little bit more of my thoughts around this, is that I think they really killed it on the user experience.

[00:21:32] JM: Yeah.

[00:21:33] JP: I think when you're coming from the world at least prior to those, even before Grafana existed, most companies were just running Graphites and the default kind of like Graphite UI or Cacti and these other companies war was almost these not so much like real-time kind of like static images that would update that would you'd have on a monitor that you could see. There wasn't much kind of like ability to interact with kind of like your data in a really like great way for like the user experience.

When Datadog took kind of like the concept of like all of these different time series metrics and being able to visually build a chart in a really clean way with a great kind of like user experience there, I think that that was the ultimately kind of like one of the great reasons they got a lot of really early adaption. Then beyond that, they were able to really scale as like whatever they're using on their backend to be able to handle the amount of metrics that were coming their way ended up being kind of like what took that user experience to the next level.

[00:22:37] JM: Yeah. They're an interesting company, because like it does seem that they really ground out the rest of the market just by sheer persistence tactical decisions. There's no one weird trick, because I think when they entered the market, there's a bunch of logging providers, a bunch of metrics things and they kind of pulled away. Not totally speaking from an unbiased standpoint, but like it's interesting, because it's like this is kind of a commodity. I don't know. They are a case study I try to understand. I just don't really know how they were able to pull away from the market.

Did you have an observability stack that was deployed with every service? When the service team goes down the golden path, is there some out-of-the-box experience they have where they get observability across all their tools?

[00:23:28] JP: Oh, for sure. Yeah. Basically, when we were in the virtualized world of deploying the Rails app and our job apps on top of EC2 machines, by default, you would have kind of like

the observability stack agents installed on those machines and it would automatically pick up the service that was running, the environment that was running on, and then basically pull that into like the Datadog's and New Relics of the world and for logging as well. Yeah. It would basically feel like those services logs into our Elastic cluster.

[00:24:00] JM: Okay. That experience for the service owner, was it like a library? Was it a sidecar container? How did they get that out-of-the-box experience?

[00:24:09] JP: In the virtualized world, it was basically like an agent that ran on the machines, but in the Kubernetes world, either like a daemon side or a sidecar that's running in that environment where the agent is running and then pulling the relevant information from that environment. Other than in the Datadog world, having to go forth and instrument custom metrics that you wanted to have as part of like your application. Let's say you have – Let's say you're writing the search service and you want to be able to track kind of like your search queries per second. By default, it wouldn't automatically like increment a counter in Datadog every single valid search request that came through.

An engineer would need to use kind of like the Datadog libraries within kind of like the application within the frameworks that we provided to then write something that would send kind of like either the counter or the gauge or the timing metric over to Datadog.

[00:25:04] JM: I talked to Lyft a couple weeks ago and they were saying that – Or Vicky was saying they deploy like five sidecars per service. Did you know how many like – Every service you deploy, was there like an overhead of a bunch of sidecars, like logging, monitoring? I don't know. A bunch of different things?

[00:25:25] JP: Yeah. I don't know exactly kind of like what's running in that environment, but for the most parts, and in most companies, you're going to have some sort of – You'll have like your monitoring agent, your logging agent. This could be the same thing. In some places, you may have like a sidecar that's running for security reasons. Then you'll likely have some sort of proxy sidecar, right? Whether it's something like Envoy that Lyft uses, or Airbnb famously ran SmartStack, which was a kind of like service discovery system built on top of HA proxy for a long time. I'm not sure where the state of that is today, but that would also be a sidecar that

could actually run and in this world. You're typically going to have 3 to 4 of these at least depending on what kind of golden path your team is provided.

[00:26:11] JM: Okay. Well, let's go deeper on the service discovery front, because that pertains to what you're actually doing today. Tell me what problem service discovery solves, because I'm sure there're people listening who don't really know the problem of service discovery.

[00:26:25] JP: Yeah. Service discovery really kind of helps engineers basically build the frameworks for how services talk to one another and kind of like how the services like register with service discovery to then begin receiving traffic.

I'll take you back a few years to kind of like the SmartStack implementation at Airbnb, and it's an open source project that's out there on Airbnb's GitHub. But essentially what SmartStack is is two different components. There's a nerve component and that a synapse component, and the backend for SmartStack in those days was basically Zookeeper.

Whenever let's say you have the pricing service and the pricing service is running on three different containers or three different machines, when that pricing service becomes available, it uses nerve to then register that like, "Okay, this host or this container is now ready to receive traffic," and then publishes that status into Zookeeper. Then all of the different clients that would then talk to the pricing service are watching that particular like node in Zookeeper and saying, "Okay. Now there is an additional node to then send traffic to."

The way this manifested in SmartStack is that that synapse client that would run on the machine that would be talking to the pricing service would then update its local HA proxy config to say, "Now, this set of two hosts is now three hosts for this backend that talks to the pricing service," and then it would then route traffic that's intended to the pricing service locally into its HA proxy and then to those backend services.

[00:28:05] JM: The problem that is being solved there is that if I have a user request, that user request probably is going to need to hit multiple services. Services are going to need to hand-off requests to one another. Therefore the services need to know the locations of each other. Is that right?

[00:28:25] JP: Exactly, yeah. It's basically kind of like – You're basically building a routing map within your infrastructure dynamically. Many years before, if you think about like why the concept of like service discovery and service mesh has become so important is that prior to this, you'd have something that would need to basically run regularly to then update a list of hosts that you would stick in like a load balancer and then someone would be riding that config, applying it to the load balancer and then going from there, right?

Now, with like the advent of cloud and containers and like short-lived kind of instances, you really need something that's dynamic that doesn't require like human intervention or someone to kind of like know what is up and what is down so that things can automatically join kind of like the mesh as they become healthy or as they become like spun up for the first time and then become healthy.

[00:29:22] JM: The implementation the you describe at Airbnb, the Zookeeper era implementation by modern standards sounds pretty complicated. Correct me if I'm wrong, but relative to what we do today – Or in the post- Kubernetes world, what is the standard by which people are doing service discovery?

[00:29:42] JP: Yeah. I wouldn't say that there's necessarily kind of like a standard I guess at this point, but through new service meshes out there, most famously, like Istio and like Linkerd and things like that, a lot of this functionality just kind of like get from the start and not just kind of like the discovery piece of it, but how you kind of like handle things like circuit breaking and failover and kind of graceful degradation. These are kind of like first-class things with these environments that really make it a lot easier on kind of like the end-user developer to build highly available applications without having to go through the effort of setting something up as even in modern standards as involved as something like SmartStack.

I think that the – In kind of like the container world where things are even more short-lived than the environments before in like the post- Kubernetes world, you really need something that is really quick and dynamic and can deal with kind like the daily kind of like machinations of really kind of like fast-moving environments like your container is running in pods.

[00:30:51] JM: The company that you're building is Effx, and the way that I see Effx might be categorized as service catalog or maybe service discovery plus service catalog. Explain what you're building.

[00:31:05] JP: Yeah. The kind of like primary way to describe what we're building at Effx is that it's really helping engineering teams kind of overcome some of the organizational challenges of using microservices. What we've spent a lot of time talking about is a lot of the really cool kind of like modern underlying technology that companies are using to build kind of like these new infrastructures and new ways of doing things.

While on the technology side, a lot of things are really moving along greatly and you can build a great company on top of Kubernetes. You can go all-in on something like Amazon ECS. You can go all-in on something like AWS Lambda. There's a lot of kind of like cognitive load challenges that come with moving from that world where you have just like this one application that you need to worry about and suddenly you need to be able to worry about tens, if not hundreds, if not a thousand different components and really kind of like just keeping track of what anything is, right?

A lot of kind of like what we've been building at Effx actually comes from challenges that we experienced while moving from a monolith to microservices at Airbnb. The first primarily is just kind of like naming things and knowing what something is based on its name. There's no naming scheme that any engineer can come up with in the world that's probably going to scale beyond kind of like 30 things, much less 100 or a thousand different things.

Let's say you decide to name all of your services off like Game of Thrones, or Star Wars. Those universes still aren't large enough to really kind of like fully give you a unique name for everything. If they are, who's to say that all of your engineers are so well-versed in that world to where they can intuitively know what something is?

I'll give you kind of like the example of like a pricing service, for example. You can't name the pricing service the pricing service, because if you go to like your wiki or you're like looking in your email and you search for pricing, you're not going to get a lot of really unique hits that are relative to that service. You might get a lot of just random things about pricing that have

happened in kind of like your organization, like what if something, your biz dev, to persons talking about pricing. What companies typically do is they end up giving it sort of like a unique name, but imagine being a new hire in a company and looking at all of the different things that have been named in joining a team where you're working on 15 to 20 services and just not even being to like even grok what something is.

At Airbnb, there wasn't like a particular naming scheme. We allowed kind of the team that owned a particular microservice to be able to name it whatever they wanted to. The naming scheme for things that may have happened in search were not related to anything that was happening on kind of like the pricing side. Then even something like the core components within the application were named completely differently. There are services that we interacted with on a daily basis that were involved in like incidents and downtime that I still quite don't understand why they were named that, because they made absolutely no sense.

Imagine a world where you're coming in as a new engineer on a team or you're dealing with an incident where it's a couple dependencies away from something that you worked on and not even knowing what that service is and not being able to find a good place to actually describe what that service is. Not only is like the name important, but all of the metadata around what that particular microservice is. What is it written in? How is it deployed? Is it a tier zero service? Does it actually impact end users or is it a more kind of like a backend services that could go down without end use impact? Do you group your services in any particular way? Is this part of like the edge infrastructure? Is this part of kind of like the service infrastructure?

Then like, most importantly, who owns that service? Which team is ultimately responsible for its health? For its documentation? If you have questions about that service, what’s the team to go talk to? All of these things actually become like really important. Once you get above like 10 to 25 engineers, like 10 to 25 microservices, like the cognitive load issues just become really difficult to be able to kind of track these sorts of things.

One of the key kind of like tenants of Effx is to really kind of be the system of record for all of the microservices in your environment and all of the metadata around them so that engineers aren’t left hanging and don't have to like run around in Slack or search across 10 different tools to

actually figure out what this particular microservice is and be able to really kind of grok that in a really elegant way.

[00:35:54] JM: There is this term I've heard before, service catalog. Is that what you're building?

[00:36:01] JP: In some sense. I think there's definitely kind of a – Service catalog sounds very kind of like ITIL, like an IT ops days from yesteryear. That's obviously still really important. But I think that what we're building is a little bit more built for kind of like modern cloud native companies that have adapted no new technologies to move towards kind of microservices and it's really just kind of like one big component of what we're doing.

The second large component of what we're doing at Effx affects is that from like a catalog standpoint of like knowing what all of your microservices are, the state of those microservices is also quite important. Going back to kind of like a monolithic application, when you want to understand the state of like that monolithic application, there's typically like one or two different places to look. You're going to go look at whatever you're using for deploys. You might go have a feed of feature flags or experiments that are being turned on or off and then you can go visit like the AWS console to understand its capacity.

Imagine having to do this or understand the state of a particular microservice when there's like a thousand of them and they could be using different deploy mechanisms. You could have some that are running on AWS Lambda that are being deployed via like serverless.com or Amazon Sam. You could have containers that are being deployed through Kubernetes, and you have all of these different places to look for the state of a particular application.

One of the things that we've built that I think is really important is basically a reverse chronological feed of all of the changes that have happened to that microservice that have been initiated by a human or through automation. This kind of like low frequency data, but actually like high fidelity and really important to understand kind of the state of an application.

Before a product like Effx, what would happen would be you get paged at 3 o'clock in the morning. Your pricing service is down. You hop on line. You open up a bunch of different tabs.

You might open up you know your observability tool. You'll open up your CICD tool. You'll open up feature flags. You'll open up experiments. You'll open up that capacity tool, and you're trying to like correlate time in like different time zones in difference different visualizations to figure out, "Okay, if anything happen in like the last 10 minutes, that's the first place I should start." Was there a deploy to the feature it turned on? Was there like a capacity change that suddenly go from like 10 host to 3 host? Those were the typical signals that you're going to look for before you start digging for something even more serious.

Right now, you might see like a vertical line on your graph in a particular observability tool maybe signifying that there was a change, but you don't have a lot of context around like who did it, like what exactly was the change? Which merge in GitHub or which version is this related to? Through our feed, you'd basically be able to say, "Okay. Five minutes ago, Joey did a deploy, and that's probably the reason why we could be in this state that we are right now."

Really kind of beyond kind of like the metadata around what a service is, we pull in as much information as we can about like what has changed in that service so that we give you kind of like a reverse chronological view almost like Twitter of all of those different changes that have happened so that you can quickly kind of like narrow down if anything was related to kind of like a human change.

[00:39:27] JM: Okay. If you get all your services instrumented with this thing, you can have a change set of updates that have been applied to or potential, like critical outage, or anomalies that are being surfaced. You can go into Effx and just see a newsfeed of things that are changing around your infrastructure.

[00:39:52] JP: Exactly. Yeah.

[00:39:55] JM: Did you build this thing at Airbnb? Was there an internal thing that was similar to this?

[00:39:59] JP: Yeah. We actually built two components of this that we've combined in Effx but were separate projects at Airbnb. In one sense, we had a very naïve version of like service ownership, like tracking at Airbnb where we'd basically have like a YAML file that would exist

like in applications like Git-Repo that would basically say like, “Okay, this is the owner's file. Here is the actual owner owning team,” and that worked fairly effectively.

Then on the like operational change alongside, we basically built a similar tool to this. It was much more kind of like logging oriented and that it was basically logs of particular types that would end up in the system, but it was actually really effective to be able to – We built a listener in Slack that if a PagerDuty notification triggered, it would immediately like notice that PagerDuty notification triggered and then list out the five changes that recently happened on that service below it so that the engineer would have a lot of context to basically say, “Oh! This could be related to one of these feature flags, or deploys, or experiments that was turned on in like the last few minutes.”

[00:41:14] JM: For a user of Effx, what is required to instrument their application with it?

[00:41:21] JP: Yeah. From getting your services into our platform, which is kind of like the first big component. Right now, we've really focused again on kind of cloud-native technologies and more modern ways of doing microservices. From a Kubernetes perspective, we essentially drop a pod into your environment. It will discover all of the services that you’re running and then send that information back up to our platform.

Then from an Amazon ECS Lambda perspective, we basically have you create like an IM user in your environment and delegate access to us and then we use the AWS APIs to grab a lot of like the relevant service information that you're using in lambda and ECS and then pull that into our environment.

After that, from the Kubernetes side, we basically have a UI in the app where you can define kind of like the description of this microservice. Here’s the owning team and other metadata around that particular microservice and then click generate config and it basically generates Kubernetes annotations to when you would then add to like your service file, your service YAML for that particular like Kubernetes service. Then from then on, that information would be sucked into our environment.

Yeah, from getting your services into the platform, it's fairly easy for kind of like that first step of building that in. Then from the feed perspective, we've built a lot of different integrations with different tools to be able to kind of pull in from a common CD systems, like monitoring systems to pull in alert data, things like PagerDuty for now incident data to be able to build kind of like a rich feed, and a lot of those are just simple API integrations to where within those applications you create an API key of read-only API key and delegate access to us and then we use – We basically build something that then kind of like pulls down from those environments.

Then we also have an API. So let's say you have a custom system, a custom deploy system, or you're using something that doesn't have kind of like an API that we can pull this information from. At the end of an event, you can basically post like a JSON object up to us and we'll ingest it into a system and then show it as part of like the – Show it as part of the feed.

[00:43:33] JM: From what I've heard, Airbnb is not the only company that has wanted something like this or has built something like this internally. Have you talked to other companies that have built this kind of thing internally?

[00:43:45] JP: Oh! For sure. I mean, most famously, like on the feed side, Facebook has a project called Opslog internally that I think a lot of inspiration for tools like this that other people have built have kind of come from. I think that from the research that we did before starting this company, I spent a bit of time with other infrastructure leaders in this world, people that were similar, like peers while I was at Airbnb.

For the most part, everybody has these challenges, right? Everybody that's gone from like a monolithic to microservices or have started on microservices from the start, it's a pretty quick getting to anywhere between like 10 to 25 microservices depending on how complex they are. Once you get to that point, you begin having these issues of like keeping track of things, having proper owners and just kind of like the organizational complexity of moving to an infrastructure like this.

It was a pretty cool thing from a business perspective to hear that a lot of folks have these problems and have either built tools internally that solve some of these issues that they'd rather

have a vendor come in and kind of like build and continue to evolve for them, or they've been looking for something like this and haven't been able to find it in the market.

[00:45:00] JM: Yeah, because when you're troubleshooting, today it's like – I mean, the most recent innovation – Well, I don't know if it's the most recent, but was significant innovation that comes to mind is now I can look through my Slack channels. Now I can look at Slack notifications. We're drowning in Slack notifications.

[00:45:20] JP: Even still, right? You may not know the right Slack channel to go look at, right?

[00:45:25] JM: Right. That's right.

[00:45:27] JP: Yeah.

[00:45:28] JM: Yeah. I think you've written about this a little bit, like you also don't know what dashboard to look at. Like single service might have like 12 dashboards and you're like, "Okay. I'm going to look through these," and like, "Oh, actually, now I'm looking at my distributor tracing and I noticed that actually there's something that causes me to go look at this other service, because like there's upstream latency and like that's something."

If you had a first port of call that was a newsfeed the you could look through to see stuff that's related to infrastructure that you work on, that could be highly useful for just at least setting the table for how are you going to triage this given scenario.

[00:46:07] JP: Exactly. Yeah. Going back to that analogy of like it's 3 o'clock in the morning. You get paged. The pricing service is down, right?

[00:46:13] JM: Time to read the newsfeed.

[00:46:14] JP: Yeah. Before that, what would happen is let's say you're actually working on search and the pricing service is three kind of like degrees of separation away from you as far as like a dependency goes or – Sorry. Bubbling up on the search service as a result of something strange happening on the pricing service. Typically, what you would do is like, "Okay,

what observability tools do we use for search? Let me open up all of those in my tabs. Then I'm going to search for like whatever rename the pricing service and all of those tabs. Maybe they use the same tools. Maybe they don't. Maybe you're using Datadog for observability and there's five different dashboards or whatever you call the pricing service and you're not quite sure which one to look at because you don't look at this service all day long."

What a tool like Effx gives you is like you find whatever that pricing service page is, and on the left-hand side you have the links to all of like the best dashboards that have been set up by that team that owns the pricing service and you can just click directly through on those and really kind of be able to dive in quickly and understand what's happening without having to ping that savant engineer that knows all of this information in their head or go digging around to find this information across a bunch of different tools and you may not even realize that you're looking at the wrong thing.

[00:47:30] JM: This is what the investor people called devsumer, right?

[00:47:35] JP: That's right.

[00:47:36] JM: It's the developer tool that feels like a consumer product. Developer tool where you open it up and you're reading a newsfeed.

[00:47:45] JP: In some sense, yeah. I hadn't quite thought about it that way.

[00:47:46] JM: You haven't heard that one?

[00:47:47] JP: There was a while where we were kind of talking about this is almost like Twitter for your infrastructure.

[00:47:54] JM: Yeah.

[00:47:54] JP: Right? Where kind of you have the metadata around like your profile and what the service is. On the right-hand side you have like the reverse chronological feed like Twitter used to be of just kind of like that reverse chron feed of changes that have happened. I think

that there is definitely a reason why a lot of us are building tools that are very similar to the tools that you might just end up using every day. There's user interfaces that you're familiar with, and I guess devsumer is a good way of putting it.

[00:48:24] JM: Yeah. Buoyant is working on a product that is very similar to this. Did you see Dive?

[00:48:31] JP: I'm familiar with Dive. Yeah.

[00:48:33] JM: At what design decisions that they've made differently?

[00:48:35] JP: I actually haven't gone kind of like too fully deep in it yet. A lot of it is just really built around like the cataloging piece. I think that the cataloging piece is just one big component of it. I think the timeline events of changes ends up making the applicability of what a tool like Effx could be fully used for.

I think we're just kind of kind of focused on making that experience as solid as we can and really listening to our users that are using this for improving times or kind of like understand what's happening in incidents and get to the right tools faster and really kind of help them pinpoint a potential cause that could be happening when they're doing these sorts of things. I think that there's definitely kind of like the coordination component that really comes with just the catalog piece, but I think that there's a lot of real power in kind of like that event feed of changes that have happened. What's what we're focusing on.

[SPONSOR MESSAGE]

[00:49:40] JM: DigitalOcean makes infrastructure simple. I continue to use DigitalOcean because of the low friction and attention to user experience. DigitalOcean has kept the experience simple and I can spin up a server in less than a minute and get high quality performance for a low price. For an application that needs to scale, DigitalOcean has CPU optimized droplets, memory optimized droplets, managed databases, managed Kubernetes and many more products. DigitalOcean has the flexibility to choose the right instance for the right workload and he could mix-and-match different configurations of CPU and RAM.

If you get stuck, DigitalOcean has thousands of high-quality tutorials, responsive Q&A forums and a customer team who treats customers respectfully. DigitalOcean lets developers focus on what they are building. Visit do.co/sedaily and receive $100 in credit over 60 days. That $100 can be put towards hosting or infrastructure and that includes managed databases, a managed Kubernetes service and more.

If you want to get started with Kubernetes, DigitalOcean is a great place to go. You can use your $100 to start building your distributed system and you can get that $100 in credit for free at do.co/sedaily.

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

[INTERVIEW CONTINUED]

[00:51:15] JM: A thing about the market, if a company is entirely on Kubernetes, this thing sounds pretty easy to instrument. It sounds pretty easy to adapt or they're entirely on AWS. Your integration there seems pretty cool too. I don't know how tough it would be to integrate, but it sounds like reasonable at least.

What I wonder, from the business building standpoint, a lot of companies, the big enterprises, the big, juicy enterprise contracts, they have heterogeneous infrastructure, right? It's like the cloud native superstar companies have already built something like this internally. Maybe they would adapt Effx, because it would be like better than whatever they've built internally.

But as far as really getting to the enterprise customer, the bigger enterprise customers, they're going to have really heterogeneous infrastructure which is going to be harder to integrate with. Maybe you can integrate over some subset of their services and it can be useful enough for them to use it, but do you worry about that? The fact that a partial integration is maybe kind of tough, or do you just feel like you're just early to the market and – I don't know. Tell me how you think about.

[00:52:22] JP: Yeah. I think what you mentioned is definitely true in terms of they clearly – I mean, even in some of like the kind of like hyper-growth companies that you'll hear that are using things like Kubernetes. They're not all the way there yet, right? They're either partially way through kind of like a long migration and things like that.

I think that while initially we've really focused on a few technologies for customers that are really kind of like understand this problem, like we found that through kind of like some of these modern technologies, the folks that are using those things really kind of grasp what we're building easily. That's why we focused on like those particular things and kind of like the ability to kind of go from zero to one in those worlds is a bit easier for us, but we do recognize that – And we've begun building tools to work in any environment to be able to ingest data simply because not everybody is going to be like a nice, perfect cloud native example of a company. Going to like the enterprise, like heterogeneous concept that you mentioned, it's actually quite interesting use case for our product.

One thing that ends up happening in large enterprises, you have a ton of acquisitions, and those acquisitions all have their own infrastructure and it's just incredibly difficult to kind of keep track off, or you're going from you're in this world where you're moving these 1,000 on-prem things into the cloud and converting them to lambda or converting them to ECS, and our platform with the ability to add arbitrary tags to an environment or to a service, you could essentially use those tags to track the migration. You could pull up and see like, "Okay, these are running in VMware locally, these 10, 15 services. Our plan is to move those 10 to 15 services to ECS over the next three months and you could use a tag to kind of keep track of that status as it moves across the different platforms within our application.

Then going back to the acquisition example, use our platform to kind of like tag services that have come in through a particular acquisition so that you could pull all of them up into different one simple view. We're working on our way to basically support any kind of compute environment so that it makes sense for any size company in terms of like where they are in their move towards like cloud-native tech.

[00:54:52] JM: Yeah, the sprawl problem. Even a company like Airbnb where they're all in the cloud, like my understanding is that these companies all have sprawl problems. You cannot

keep track of everything. Was that an issue at Airbnb? Is there some dashboard that's like, "Here is everything. Here is the MapReduce of all the costs, or all the number of EC2, whatevers," or is it really just complete madness?

[00:55:17] JP: That one place where everything exists was like the bill.

[00:55:21] JM: The bill. Okay. All right. All right.

[00:55:22] JP: That was like the single spot where you can actually see everything in kind of like a retrospective manner. I'm not sure how similar how Airbnb was to other companies, but we really kind of – We didn't want to get in the way of innovation. We allowed engineers to spin up whatever they needed, whether it was like a data store or more hosts and things like that. It worked out pretty well. We believed everybody was like well-intentioned and it was going to do the right thing. But it would manifest itself in kind of like the billity end, right?

We've built some tools obviously to track this stuff through kind of like service ownership, because if you had a question, you needed to be able to know who to kind of like reach out to and talk to them about it. That was obviously important, but it's hard. It's hard to really keep track of everything, and I think what a company like Effx can do in terms of like cataloging, at least just like your microservices, and building a really kind of great culture around kind of like the maintenance of that I think is a really good first step.

[00:56:26] JM: Yeah. Can you tell me, did Airbnb use cloud cost? Any cloud cost management software or was that – Did you build anything in-house? I mean, that's a huge issue also, right?

[00:56:35] JP: Yeah. We use some of like the main vendors out there to kind of like look at our kind of like general overall costs and we built some tools internally to track that stuff better, but it's hard to kind of like really get a full purview of everything that's happening and really kind of like understand it, right?

I think one of the challenges that happens with any sort of like external cost tool is that it's really hard to understand kind of like internal attribution, which is really what you want. You really kind

of want to be able to like assign those to kind of like the cost structure within how the company does like financial reporting internally.

[00:57:13] JM: Sure, like what's the expected value of an additional server or something.

[00:57:18] JP: Exactly, right? A lot of – Most of the tools out there, at least the ones that I had the opportunity to kind of like evaluate and look at, they're really great from just like how do we manage our reserved instances? Where are places that we can save money? What are the opportunities here in terms of like reducing our bill? In terms of any sort of financial forecasting or attribution to kind of call centers internally, that kind of stuff is like really hard to come by and you basically need to build a translation between either that tool or the AWS bill into a structure that makes sense for you internally. It's something that I thought about a lot when I was at Airbnb. There's not really anything out there right now that I think really fully knocks it out of the park.

[00:58:00] JM: This is like the next Datadog category. I'm convinced. It's like one of these cloud – Because none of these cloud – Because none of these cloud cost managers – I mean, there's a lot of them that are doing great, but like what's going to be the one that like everybody centralizes on that's the trendy one to use? It will be interesting to follow. It certainly seems like it's – I mean, if you can be the cloud cost management company that everybody goes to, it's a great place to be.

[00:58:21] JP: Yeah, it really depends on your angle there, right? I feel like what I've seen is a lot of folks focusing on kind of performance and efficiency rather than kind of any sense of what would make sense to an executive. I think that's the challenge when it comes with like definitely kind of money and representing it away.

Are you building something for your engineers to understand the cost of the services that they're running so that they could then make them more performant and make better informed decisions there, or are you building like a financial reporting team for like your finance team and your CFO and the executive team to fully grok? There's probably a market for both of those companies, but I think the marriage of that is probably what we really need and it just doesn't fully exist right now.

[00:59:09] JM: Yeah. Yeah, I need a newsfeed for my AWS expenses, maybe.

[00:59:14] JP: Then like you throw on new technologies that add additional cost suckers on there, like AWS Lambda and Fargate and even like running Kubernetes in the cloud, right? It suddenly becomes a different equation than even just running kind of EC2 5 years ago. That was even harder to grok at this point, but now you have all of these kind of different levers as supposed to just kind of like paying for a host for like an hourly charge. Now it's much more – There's much more variance in terms of how does a lambda function charge, right? What are the unit economics of that and figuring out should we move more to lambda? Does it make sense for us? Make informed decisions there is actually pretty difficult right now.

[01:00:05] JM: I also feel like there's not a whole lot of great information out there. I feel like I have never heard of a company – I'm sure there's some out there and maybe people can send me a message about this if they know. I feel like I have not heard of any companies that have shut down because of too expensive infrastructure. I mean, it's always because they over-hire or pays people too much or something. Have you heard of companies shutting down because AWS bills too high?

[01:00:29] JP: Not really. Not that I know of. It definitely could affect your margins, right? Like if you're an advertising business and your AWS kind of costs were out of control, but it's not usually the leading indicator of why a company would fail, right? Because there's enough that you can do there to kind of – Again, that tradeoff of going back to what we were talking about earlier in terms of your high-availability, right? You could run everything in one availability zone and not be charged for like data transfer across AZs and run at a much leaner cost than you would if you're running across multiple environments and paying for that sort of transfer.

I think there's a lot of levers that you can use as you're figuring things out to kind of reduce your cost, but then that also has like the detriment of not necessarily being as highly available. It's always just a tradeoff.

[01:01:16] JM: I want to get more perspective on your strategy for building this company and just generally speaking how to build an infrastructure startup in 2020. You went to KubCon, right? The most recent KubCon.

[01:01:29] JP: We did go to KubCon. We had a booth. Yeah.

[01:01:31] JM: Yeah. Was that booth – Were you just doing that kind of to show people what's going on here or were you trying to close some deals or just POCs? What was your goal there?

[01:01:40] JP: Yeah. We went to KubCon in November and we had just really began talking about what we were building in kind of like the middle of October. That's when we've launched out website and tweeted about what we were doing and wrote a few blog post about why we think it's important. We hit up KubCon a few weeks later. It was really just kind of a really great experience for us, because all of companies, large and small that are using Kubernetes that can really kind of grok this problem were coming by the booth. We were able to give a lot of demos. I was actually able to kind of like AB test the way that I talk about the product to see to what really resonates with people.

From us, it was really just kind of build some brand awareness, get the Effx name out there so that if people begin kind of having some of the organizational challenges that come with microservices, they might remember that that company that they ran in at KubCon or has followed up with them. Then from kind of just like product marketing perspective, just kind of like better understanding kind of like the users and being able to show them the app and get a lot of feedback and work on messaging on-the-fly was really invaluable for us.

Obviously, like getting leads is great and getting people to follow up to come back on, but it was really just about like building our brand and being able to talk about the product, do 100+ demos a day and really take that information back and inform some of our future decisions about how we're building and messaging the product.

[01:03:12] JM: What was the response? Take my money?

[01:03:14] JP: Pretty great. Pretty great. I think one of the biggest worries I had going in is that people necessarily wouldn't understand why we were building what we were building, but the second that you big in kind of talking about like tell me about your microservices infrastructure that you're running at your particular company and some of the challenges that come with it, most everybody will kind of go to that it's really hard to keep track of what anything is, right? It was validating for us from that perspective, and that was a lot of the original premise for building what we're building and it was really great to kind of hear why, why that was a challenge to a lot of companies.

[01:03:47] JM: Yeah. It'd be interesting like how you convince people that it's a pain killer instead of a vitamin.

[01:03:56] JP: Yeah.

[01:03:56] JM: Right? Because I think a lot of people are going to look at them and be like, "That's nice. I have no time to implement that." But I think you'll get there. By the way, to the point of the heterogeneity earlier, it seems like this is useful even if you only have it over a subsidy or services.

[01:04:12] JP: Exactly. I think there's definitely a team use case where you can keep track through that feed of the changes that have happened that can help you in incident response and help you when things go wrong, but I think there's also just kind of the tracking the links and being able to get to the right place at the right time when you need it the most is important even when you have two to three services. It's definitely kind of like a valuable thing.

We've built our own platform on top of microservices as well. We're in the teams of microservices of this world, and it's actually useful for our own team internally to understand kind of like changes that are happening cross all those services as well as getting to the right dashboard when you need to. We're eating our own dog food in terms of like using our own product to track our own microservices to make it actually easier on our engineers to do a good job every day.

[01:05:10] JM: Going from infrastructure platform developer engineer at all these different companies you've worked at to building a product, what aspect of the product building, the product engineering process, what has been harder than you anticipated?

[01:05:29] JP: Huh! It's been harder than anticipated. I think coming from the infrastructure background where the majority of my teams have been in kind of like the site reliability world or building observability tools or working on performance and traffic and building like the backend systems that help kind of like engineers really go. I think a lot of that world can sometimes be quite reactive. You're dealing with ways to make improvements to your infrastructure based on things that have recently happened and some toil around that kind of work and basically doing a lot of things to kind of keep the lights on.

In some organizations, you're able to get over that hump to where you can start thinking about the future and really kind of like build products to make things better. But that doesn't always happen, right? I think one of the I think really exciting things at least for me which could be looked at as a challenge is that coming from like a greenfield space where there's not like a fire to go put out beyond kind of the problem that we're trying to solve and instead really being able to like think deeply about the product and build and experience that is helpful is useful has been a different experience for me personally in terms of like not having to worry about like the thing blowing up all the time at this point and really being able to focus on kind of like the true kind of like product development path and iterating with feedback. It's been different, but also a really fun and refreshing and exciting for me.

[01:06:58] JM: Is it an electron app? Is it a desktop web? Give me a little bit description of the stack or like what the tool actually looks like.

[01:07:05] JP: Yeah. It's a desktop app, desktop web app right now. Just runs completely in your browser. Again, on the backend, it's built on top of Kubernetes. Most of our backend infrastructure is written in Go. We're using GRPC to communicate between those services and using Envoy as kind of like the – Envoy as the proxy between all those services to talk to one another. For some like our ingestment pipelines, we're running on those on top of AWS Lambda. So any kind of like API request that are coming from the outside or going from a data pipeline before it make its way back into our backend.

On the frontend, it's basically a React web app that we're using TypeScript for. Going with like a strongly typed language like Go on the backend. We wanted to have a lot of – With GRPC as like the communication layer. We wanted to have a well-typed language on the frontend as well. We're using TypeScript and then most of the queries that go to our backend or going over our GraphQL interface to the backend. Yeah, that's primarily it at this point.

[01:08:08] JM: Compare your present self to the intuitions of Joey Parsons five years ago.

[01:08:14] JP: Let's see. I think like from myself five years ago, I think one of the cool things going back to like what we talked about really early on is that I've really been able to kind of see the evolution of how things have really changed overtime, right? I remember in kind of like the late auths, when AWS was becoming a real-kind of like powerhouse when it had gotten beyond just kind of like SQS and S3 and maybe EC2 at that point, I wasn't necessarily kind of like excited about that kind of move for infrastructure that I was running just from like a cost and security perspective without really thinking about all the benefits and agility and the ability to move fast and build a business there.

I always kind of like looked back on kind of like that timeframe and think about like what my mindset was there in terms of like what I believed was possible with like new technology and kind of going away from potential chromogens ways back then. I think now I wouldn't say that I'm too kind of like closed off around the potential of new tools and the abilities that those tools can provide to make the engineering experience of running infrastructure better.

There's always a lot of stuff on Twitter around kind of like you don't need Kubernetes. You don't need to use things like Lambda or ECS. Just a good old EC2 box will do the trick for most web apps.

[01:09:44] JM: Chromogens.

[01:09:45] JP: Yeah. Not necessarily chromgens. I think that they definitely have like valid points, but I think from what I've seen and the types of companies that we've talked to that are kind of like in the early stage, they're building new startups, is people are going with this stuff

from the start. I think that the Airbns of the world, the Lyfts of the world, these kind of more forward-thinking enterprises at this point have kind of proven and will continue to prove, and every single time one of them goes on stage and talks about the infrastructure that they've built even outlaying the challenges that plants the seed in the head of a lot of engineers to go out there and then begin playing with this stuff and figuring how it fits even a really small company.

I think that it's going to continue to evolve and I think the more folks that are using this at different stages, it's actually great for the community and great for the ecosystem because the user experience of getting started will just continue to improve and make it even easier for the early adapter of the future to jump on board. I think maybe my contrarian thing isn't necessarily contrarian and that I believe something completely different, but I think that a lot of this new cloud-native technology is here to stay and I think that the more people that use it early on, the better off like the community and the onboarding experience is going to be and it's just great for everybody in the future.

[01:11:12] JM: Yeah. I mean, the lambda stuff is the same thing that you didn't anticipate with AWS. It just makes it really fast and really productive and it makes it really easy to wire your stuff together, and yeah, you're surgically stitching yourself to AWS, but who cares? You're going to move fast. You're going to build a business. AWS is never going to squeeze you like a sponge. If they do, then Google Cloud is right there ready to stand up replacement services for the things that they raise prices on. Yeah. I mean, all-in on serverless, right?

[01:11:45] JP: Yeah. Even kind of like the advancements in like Google Cloud Run where you can just kind of drop a container and you're essentially running it without much stress or like EKS on Fargate. I think those are both really powerful tools that early adapters or folks that are just kind of getting started can use without actually really needing to fully grok the full platform underneath because they don't need to, because the providers are obfuscating that stuff from them. I think it's a pretty cool time from this space and I'm really excited what the community is going to come up with and let's see what we can build on top of it.

[01:12:20] JM: Joey, thanks for coming on the show.

[01:12:20] JP: Appreciate it. Thank you.

[END INTERVIEW]

[01:12:31] JM: As a company grows, the software infrastructure becomes a large complex distributed system. Without standardized applications or security policies, it can become difficult to oversee all the vulnerabilities that might exist across all of your physical machines, virtual machines, containers and cloud services. ExtraHop is a cloud-native security company that detects threats across your hybrid infrastructure. ExtraHop has vulnerability detection running up and down your networking stock from L2 to L7 and it helps you spot, investigate and respond to anomalous behavior using more than 100 machine learning models.

At extrahop.com/cloud, you can learn about how ExtraHop delivers cloud-native network detection and response. ExtraHop will help you find misconfigurations and blind spots in your infrastructure and stay in compliance. Understand your identity and access management payloads to look for credential harvesting and brute force attacks and automate the security settings of your cloud provider integrations. Visit extrahop.com/cloud to find out how ExtraHop can help you secure your enterprise.

Thank you to ExtraHop for being a sponsor of Software Engineering Daily, if you want to check out ExtraHop and support the show, go to extrahop.com/cloud.

[END]