EPISODE 575

[INTRODUCTION]

[0:00:00.3] JM: Alexis Lê-Quôc started Datadog in 2010 after living through the internet boom and bust cycle of the late 90s and early 2000s. In 2010, cloud was just starting to become popular. There was a gap in the market for infrastructure monitoring tools which Alexi helped fill with the first version of Datadog. Since 2010, the number of different cloud infrastructure products has proliferated. There were new databases, queuing systems, virtualization and containerization tools, web 2.0 took off and thousands of new internet businesses got started. Many of these businesses used Datadog to monitor their increasingly wide range of infrastructure configurations, and Datadog began to scale.

On today's show, Alexi tells the story of how Datadog grew from its first product into a variety of tools; infrastructure monitoring, logging, and application performance monitoring. Monitoring is a unique challenge. There's a ton of data. The data is latency sensitive and the data is operationally important. These engineering constraints provide for a great conversation.

Alexi is the CTO of Datadog and we talked about cloud providers, building a business, infrastructure and how to scale engineering management. Full disclosure; Datadog is a sponsor of Software Engineering Daily.

[SPONSOR MESSAGE]

[0:01:29.9] JM: Today's podcast is sponsored by Datadog, a cloud scale monitoring platform for infrastructure and applications. In Datadog's new container orchestration report, Kubernetes holds a 41% share of Docker environments, a number that's rising fast. As more companies adopt containers and turn to Kubernetes to manage those containers, they need a comprehensive monitoring platform that is built for dynamic, modern infrastructure.

Datadog integrates seamlessly with more than 200 technologies, including Kubernetes and Docker so that you can monitor your entire container infrastructure in one place. With Datadog's new live container view, you can see every container’s health, resource consumption and

running processes in real time. See for yourself by starting a free trial and get a free Datadog T-shirt at softwareengineeringdaily.com/datadog. That's software engineering daily.com/datadog.

Thank you, Datadog.

[INTERVIEW]

[0:02:38.1] JM: Alexis Lê-Quôc is the CTO and founder of Datadog. Alexi, welcome to Software Engineering Daily.

[0:02:43.1] AL: Thank you very much. Thanks for having me.

[0:02:45.3] JM: Yeah, it's great to have. You used started Datadog in 2010 and that was after some experience going through the late 90s and the early 2000's internet. How did your early engineering experiences inform what you built with Datadog?

[0:03:01.0] AL: So in around 2003, 2004, I joined — With my cofounder, actually we joined a SaaS company, so it's the early days of SaaS, I guess, in education. So something that was kind of a new feel for me, but what that exposes to is the early days of SasS, how to deliver software as a service to an audience that was decidedly not necessarily very technical, a.k.a teachers in K-12 throughout the country. So that was formative, because really we're trying to figure out how does one run? How does one write? How does one design? How does one run software at a reasonable scale, because we were serving a lot of schools across the country. So that was formative, and I think figuring out what are the tools available?

Back then, I think we still had this classical developer operations divide where there was kind of a release management and you throw something over the proverbial wall and it was now working very well. I mean, we're sort of grappling with how do we make better? That I think was some of the background that informed what Datadog would become when we started in 2010 and what it is now.

[0:04:10.7] JM: Were you exposed to any of the insanity of the.com boom and bust, or did your career get started a after that .com boom and bust?

[0:04:20.9] AL: I got started before that, but I'm sort of more of a late comer. So I decided — I was working at IBM at the time, 97 through about 2000, just upstate New York, and at some point I was living in New York City and I was — There was all these startups in the city. I was like, "Wow! That must be cool. Let me try."

So I, of course, pick sort of the tail end of the boom in 2000 to join and I live through and I survived the crash. So that was really also some good lessons there, but it was a fun ride while it lasted, and then it was a little bit of a trough, the trough of, the trough of despair for a while and then finally sort of got back up.

[0:05:02.7] JM: There's a lot of people, myself included to some degree, who are looking at this cryptocurrency I CO boom and sort of wondering, "Should we be hopping on the bandwagon and trying to make our riches?" Like looking back, it seems like maybe this is sort of like the 1999, 2000 era where you've got all these people like sort of jumped on at the end and are just going to have to ride out the bust if they want to stay on. Does it look similar to you?

[0:05:34.4] AL: I'll say a little bit. Cryptocurrencies is something I haven't really delved deep into. So this is really the opinion of somebody who's not terribly informed, but seeing the wall swing in value is not unlike what we saw early 2001 when, yeah, valuation of companies with no product, nothing to show, sore through the sky and it came crushing really hard. There's obviously a parallel. That doesn't mean that that's the outcome we'll see. I think that's — If I knew what the answer would be, I'd probably be somewhere on the beach, retired.

[0:06:12.5] JM: Back in 2010 when you started the company, what kinds of monitoring and logging issues were you encountering?

[0:06:18.8] AL: Back in 2010, on the monitoring side, it was a very fractured environment, and by that I mean this is largely for the vast majority of the population pre-AWS. I remember still going to physical data centers, opening up boxes of servers, raking them. Doing the whole 9 yards to get the service up and running. The environment was very fractured. By that I mean we were running an environment that was probably typical of a lot of companies back then. It's kind of oracle databases, [inaudible 0:06:55.8], network gear, servers and so on, and kind of each

part of the stack had its own monitoring tool, and that made it really difficult when you are trying to troubleshoot something that obviously started at the application, but could really find its root cause anywhere from storage, network — I don't know, DNS application, OS. You had to go to jump from one tool to another tool to another tool, and it made it extremely tedious and inefficient, and I think that as a result, it was really hard to pinpoint where are the issues.

In addition to that, there was also a fracture in terms of who had access to monitoring tools. So you'd have — Again, I think, back then in a more traditional sort of pre-devops environment, the ops crew had a plethora, probably too many tools that at its disposal while developers had nothing, really had limited access. Maybe there was [inaudible 0:07:52.7] assistance that was out there, but even interacting with it was particularly difficult. So that's the landscape I think we found ourselves in both as petitioners initially and then when we started out, I guess, as a way of sort of seeing what the problem was and trying to get out of it.

[0:08:09.2] JM: That was a great answer. We're talking 2010. This was near the beginning of this ever-increasing sprawl of backend tools, and it also sounds like the tooling was overly weighted towards the operational side versus the developer side, and this was back when there was a clearer partition between these developer and operations teams. Today, at many companies, those roles are merging or overlapping or becoming more harmonious. Back then there was a clearer divide. This was also — I believe 2010 was two years after AWS was really taking off and there's really just this high volume of tools and disparate data sources. How did that affect the early strategy for the company?

[0:08:55.0] AL: That's right, I think 2010 — I remember, I think going in 2010, and Graphite was — I think this was the sort of first tool really merging, sort of post — Probably post [inaudible 0:09:05.2] were all on the open source side. I think on the one hand, for sure, it informed I think how we thought about time series processing, what kind of interface people would want to look at. We took at some point different — We made some different choices. For instance, from a tagging, metric-naming perspective, querying perspective. Generally speaking, I think we were seeing the emergence of probably both developers and ops crews getting more sophisticated about what they need to see to understand what's going on. So I think I want to say we started probably early about the same time or just a year after Graphite took off, then there were a lot

more tools coming down the pipe, [inaudible 0:09:49.5] and further down, Influx, Prometheus and so on.

What we saw though, and this may be tying it back to your comment about the fractured environment, was there's — So first of all, as sort of a new company with not much back then to offer, we're not going to approach people and say, "Hey, we have this great new tool rip. What do you have out?" Then we'll come in and then you'll see the world will be much better. I think we wanted to have a much more sort of gentle approach by saying, "Look. There's valuable data [inaudible 0:10:20.4], in a Graphite and a number tools that use right now that collect the data that maybe you struggle to be able to use, because it's not joint. It's in disparate environments. Let us bring everything together in sort of one central repository where you can then look at everything in context." I think that led us coexist with — And to this day still, coexist peacefully with a number of tools.

[0:10:46.7] JM: What was the earliest software architecture for Datadog?

[0:10:49.6] AL: I'm probably too ashamed to say, but initially we started with something extremely simple with — So storing time series in a SQL database. I think we didn't make the very first beginner's mistake of storing additional data points as rows in database, but it was just one mistake beyond that. It was very simple. It ran on a laptop. It was PostgreS, a bunch of Python, sort of back then, not even streaming processes, Cassandra data store. So something really, really simple.

The goal was really to prove we were not concerned about scaled up, we're just concern, "Does this company have a futures? Does the product have a future? Does it answer a problem that people have?" and that's why we went really for the quickest way out the door. Then gradually as usage started to pick up, then we introduced Redis. I think that must be probably your zero or your one, and then we just introduced Kafka at some point, and then we start replacing pieces and pieces with, by now, custom-made stuff that sort of works at the scale we're at.

[0:11:58.7] JM: Okay. You mentioned this architecture with — It started out, I believe, you're just storing monitoring data in, you said, a MySQL database.

[0:12:07.8] AL: It was a PostgreS database initially.

[0:12:09.5] JM: PostgreS database. Why is that problematic?

[0:12:12.1] AL: From a volume or from a sort of read-write usage patterns perspective, it just doesn't scale really well. The SQL databases where particularly in transactional mode are really good at doing sort of very targeted reads and writes, frequent updates to the data, which a time series database doesn't have that need at all. The past never really changes. I mean, you can all have for some exceptions, but fundamentally this is a right ones, read many times pattern. So for that, the traditional SQL databases don't work particularly well. You can get a little bit more mileage if you use, for instance, a columnar database, because obviously the way time series are coined, is never one points in isolation of any other point in time. You usually look at a span for a given time series. So that makes the sort of general-purpose OLTP engine not specifically well-suited.

That being said, this is really about sort of the storage backend, which I know people like TimescaleDB, I think they've kept the PostgreS frontend and then sort of replaced the backend without their own stuff to score time series. So it's definitely possible. Back then, I think with the time we had, really the choice was what are the best tools available which we can use now to prove that the product is something valuable in the problem it's s trying to solve, and we had early on limited time to sort of tinker and rebuild the engine. We're just too worried about surviving as a startup.

[0:13:48.4] JM: I guess this is a kind of problem that pushed the development of these time series databases, because now we've got a number of them in the market and the type of problem that we're describing monitoring data, this append only data type that is going to be probably queried in a columnar fashion, is that accurate? Also, I guess, in the pre-cloud days, it was harder to have the budget to store all of these data for many companies. Was this just an unsolved problem, this type of database, or was it only solved in a closed source fashion? Why were there not time series databases that were widely accessible prior to 2010 or whenever — I guess influx was started in like 2013 or 2014, somewhere around then?

[0:14:38.6] AL: I think there has existed in specific niches time series database. I think finances is one I can think where prior to 2010, this is the bread-and-butter of sort of finance, the treatment of time series database. Though they're typically — Sometimes either their timescale is different, either super fine granularity if we're talking about high-frequency trading or core [inaudible 0:15:06.3] when we're talking about sort of text for pricing and exchanges.

So I think that technology, because finances as they — I think as an industry was not necessarily as connected us as our own and didn't have the open source ethos necessarily from that start. Those solutions stayed there, and there was super sort of very focused, very performance solutions that just never crossed across the chasm and remained adopted by just a handful.

For me, you could argue, you could maybe trace it back to development of mobile maybe, because that's really driving SaaS and delivery over awhile, if you will. That in itself is driving obviously sort of large backend distributed infrastructures which themselves become too complex to understand by just looking at the architecture diagram to predict how they're going to behave in production, and that in itself causes us to want to generate data, capture the data and then sort of be able to process it and analyze it so we can understand while how is my infrastructure actually doing? It's not something that I can reason about just from pen and paper. Whereas if you thought about — I think if you think about a more traditional sort of three-tier infrastructure where back then you had a couple of databases server, tens, maybe the high tens, maybe low hundreds of application servers. It's something that all do the same thing. It's a lot more sort of synchronous processing or that was a very separate batch world. It was something that you could reason on I think with less data to be able to predict or to sort of look back on particular performance behavior and say, "Well, it was slowed because A, B, C."

I think now, just the distributed architectures we're seeing is making a lot harder to do that if you don't ton of data to look at to make hypotheses about certain things and discard hypotheses that don't play out.

[SPONSOR MESSAGE]

[0:17:19.6] JM: Kubernetes allows you to automate, and thus increase the speed of deployment. But as you rapidly deliver, are you aware and prepared for issues in production? VictorOps Incident Management empowers progressive teams to ship to prod without worrying about a nightmare firefight. With VictorOps, your team has context to fully understand application and system health. Coordinate on-call teams, collaborate when incidents occur and monitor Kubernetes through a large number of monitoring integrations.

Datadog, Prometheus and Grafana and hundreds of other tools integrate directly with VictorOps to help you give deeper visibility into application health.

Visit victorops.com/sedaily to see how VictorOps can help you manage incidents and improve system observability. That's victorops.com/sedaily, victorops.com/sedaily. See how VictorOps helps you build the future faster. Thank you, VictorOps.

[INTERVIEW CONTINUED]

[0:18:34.0] JM: So now we've talked a little bit about the storage in the early days of all that monitoring data that people were sending to you. There's a whole lot more to that story where to get data on to the servers at your monitoring platform, the customer has to do one of two things. They can send it over HTTP, or they can have an agent running on their host machine, a Datadog, agent and that agent can communicate the monitoring data to Datadog's servers. Can you describe the communication pattern between the host and the Data dog servers? How has that involved over time? Was it the same sort of pattern back then as it is today?

[0:19:18.1] AL: Fundamentally at some level it has not necessarily evolved. The pattern remains, either the agent or if it's a custom code setting [inaudible 0:19:27.8] pushes code to us. We never reach back in as it were, and that's really for security reasons. So we have sort of some — We and more importantly our customers have peace of mind about, "I'm in control of what data really leaves. It's never data that's asking me for data." That, for me, hasn't changed.

The transfer really hasn't change either. We still use HTTPS because we know it's going to traverse pretty much everything, any proxy, any firewall at this point. The format that we use internally for how we encode the payloads, that changed has been become more sophisticated

with the different applications. The frequency at which we collect data, how we formatted that change, but these are some of not where, I'd say, the core value resides. It's something where — As you can expect, we compress the data. Sometimes some pieces are still sort of output in JSON. I think the earliest pieces are still like that. Some pieces are proto buffed. It really depends. But fundamentally, the agent that sits on the customer's infrastructure just will push data at a frequent interval, and it could be every second, every 5, every 10 seconds or every 20 seconds depending on what application we're looking at.

[0:20:45.7] JM: When I think of monitoring SasS platform that's aggregating all this monitoring data from lots of different companies, I think of a lot of different bursty workloads where you could end up with — At the tales, you could end up with some serious load, and at the other tales you could end up with situations or people are not monitoring that much data. So the scale up and scale down story seems like it could be interesting. What did the early load-balancing infrastructure look like and how has that changed over time?

[0:21:20.5] AL: Yeah, that's actually I would say a problem that has become easier to deal with over time, but just simply by virtue of statistics. For sure, in the early days when we had just a handful of customers, a burst of one would basically require us to over provision our intake capacity. So the name of the game in monitoring is really controlling for latency and having really predictable latency throughout the pipeline. Otherwise you sort of throw your alerting engine off-track and you have false positives, which is sort of scourge of monitoring.

In the early days, it was really about providing or spinning up extra capacity to absorb extra load or burst your load. As the number of customers increases, you do have a sort of a reversion to the mean in terms of while some of them may be bursting, others maybe particularly quiet, and the burst also could happen towards to smaller customers or the midsize or the larger ones. Overall, we're not so susceptible to burst, definitely now and even a couple of years ago.

This is about time series processing where essentially emit metrics so that the size of the data doesn't really fundamentally change if you're, let's say, if you're in a crisis mode. You as a customer is fighting an outage. Generally speaking, sending potentially roughly the same amount of data where we see spikes as you're provisioning a lot more capacity at once. The will typically reduce as a spike. From your perspective, from our perspective, not anymore.

In terms of log, absorbing logs, that's a different story and that's something where — It's particularly a different problem to solve where you're in sort of steady state, you emit a volume, X-amount logs, log lines or megabytes or gigabytes per second. You're encountering a problem. Typically, your volume starts to grow and that's sort of when you need access to that data the most when you have a problem. Instead he said usually you're not so pressed for answers, and that makes it a particular challenge for that product in our platform to think about how we absorb the extra capacity without over provisioning 10-X or 100-X, then we start getting into having different sort of lanes of processing to be able to provide the this quality of service for latency in terms of how recent the data you're looking at is.

[0:23:58.3] JM: As we said, this was in the early days of cloud. Did you have faith in the cloud enough to put Datadog's architecture on the cloud or were you in some colo? What was the hardware story there?

[0:24:13.4] AL: The hardware story was — It was zero. I had enough faith in the cloud and I think I have this taste in colors and sort of mashing that whole part that from the get go was let's go to the cloud.

The thinking was when we're really small is, "Look. We need to move fast." When we have very few customers, you're not pressed for needing to have a ton of capacity at your disposal or necessarily even the fastest hardware. Back then, I remember comparing the kind of workloads we can run on-prem versus in AWS, and it's just in a cloud and the cloud was there. You could run decent workloads, but nothing really serious. It really pushes you towards heavily distributed architectures from the start, because just individual nodes weren't that powerful. Now, that's obviously changed, but back then I think that's also I think what informed a lot of the choices we made where, loo, at the end of the day, I think it is easier to manage just one supercomputer that can do everything in one enclosure? Probably, as opposed to managing 10,000 nodes that will interact and fail and be coupled it in weird ways that you don't fully understand.

The reality is there is no — None of that single supercomputer that exists. So you have to — You sort of force down the distributed path, I think, naturally. AWS, because that was our

provider, forces there naturally, and I think that's for the better, because that's — I'm sort of glad we went down that path.

[0:25:44.9] JM: And over time there's been the increase in virtualization technology, container technology. I am curious both about how this affected your architectural strategy internally and how it affected customers, because I imagine the cardinality of virtual servers increases the cardinality of data points that people are logging.

[0:26:13.5] AL: Absolutely, and this I would say — The first step I think was the first multiplier was indeed going from a physical hardware to virtualization, where now you could effectively run 10 to 1,500 nodes per physical hardware. The containers sort of added an extra multiplier there, and what we're seeing is not only the count, the share counts, which adds up in terms of number of data points to process and so on. But also the speed at which things get — Spring into existence and then get recycled.

For VM's, if you think about cloud instances or VMWare instances, we moved from a three-year amortization depreciation cycle for physical hardware. So basically your CFO was saying, "Well, this things is got to run for three years. Ideally, uninterrupted, but we want to squeeze the most out of it," to I can run an instance for a couple of days, a couple of hours, not a couple of seconds, because startup costs are too high. So that already changed the game a little bit in terms of velocity, and containers, then it processes on a machine. So you definitely — We see in our customers, sort of recycling containers after a few seconds, after a few minutes, maybe a few hours, and then that's it.

So that's changed fundamentally how you have to know deep down architects your backend, support that kind of time series, the query system, but also how you name thing. I think this is a transformation that's not necessarily very apparent, but beyond those sort of pets versus cattle analogy. This is really, back in the day, if you have, 100, 200 physical servers, obviously you can name them and so on, but you can refer to them. Their name is enough to refer to them.

In a cloud environment, and let alone in a containerized environment, the name is completely meaningless. You refer to things by the properties they have. You essentially express what

you're looking for in terms of a predicate, “I want all the containers that run in this cloud provider that run this particular image and that are this particular type.”

So it's a shift in how you think about it, but I think importantly it's a shift in how the software to support that has to be built, and that's why we've seen this re-tooling of all the sort of management layer from on-prem to the cloud and sort of, now, to containers. It's because the old stuff wasn’t built with that in mind, so it just became really difficult to kind of shoehorn that into the new reality. That's why, I think, we’re seeing this profound retooling and monitoring among the other things but it’s not I think the only place where that happens.

[0:29:05.6] JM: What about internally? Obviously, this is something that you're not just having to re-architect your product for, because all the customers are going in that direction, but it's something that as a technology company, you have an opportunity to leverage yourself.

[0:29:21.3] AL: Right. So when I look at our growth from a — Let's say, from a volume of data perspective, it's been surprisingly predictable. So that's, in a sense grateful, for that. We haven’t it, because I think we have this this acclamation, because the SaaS is one instance and you have very different behaviors of different customers that kind of cancel each other out and sort of gets you into a sort of ever-growing mean that you can actually forecast with a fair amount of accuracy.

Now, that being said, we have of course — As we keep the train running with replaced parts, we’ve laid down tracks just in front of it, if I use that analogy, which is really ripping out our own internal pieces when they couldn't scale, whether it makes sense anymore. I think that what I mentioned about the — Obviously, the SQL databases of early days, that’s long gone. We were using sort of no SQL for a while for times series storage. That’s long gone as well. Some parts remain. Obviously, things like Kafka. I don't see necessarily a point now where that has to be replaced. This is really about sharding it properly. Some other pieces remain about sort of storing blobs or presenting time series, historical time series. There’s also no point in replacing that. That scales nicely. But otherwise, containerizing, for instance, that's something that we’ve been doing for some time and it’s sort of continuing. I can’t say that we’re done in that journey.

I'd say customers have generally been — Because we have enough of a swath of different customers, we've seen sort of very early adoption of, for instance, Docker or around the Docker's 9 days. We already had customers using it and in production. There were few, but they existed. So that's also, for us, the beauty of SasS is to be able to also understand what are our customers using in terms of data store, runtime environment. How can we think about the scale at which they're running and so on and so forth?

So we produce these anonymize dataset that kind of let us also study, if you will, how, for instance, we had a couple of container adoption. So that came from anonymize datasets we extracted from our own backend.

[0:31:40.8] JM: Do you see anything interesting that you can comment on? Obviously, containers have taken off, but anything like, "Whoa! People are using Google Big Query a lot," or "Wow! AWS Lambda really is taking off even more than we anticipated."

[0:31:55.4] AL: At this point — I mean, Lambda is definitely something that I had not fully anticipated in terms of the adoption. I think it still makes sense for certain use cases, of course, no not all. I wouldn't necessarily want to build — Our own, for instance, backend would not make sense in Lambda, because we're really streaming data, so the process have to be running all the time. The data has got to be loading memory essentially, I think if I want to simplify it. So there's no point in starting a new context of the execution or ripping it out, and again, and out, and again, and out.

But there a number of cases where that makes total sense, sort of I think, even-driven, maybe you are at the edge, or you're even-driven, but these are not sort of continuous events being fired at you, it's certainly a lot easier to deal with than even to spin up the machinery you need to operate a container fleet for instance. So that makes sense.

I think containers is really still — There's kind of a trend that shows that it's here to stay. I think, Lambda, I can't say that right now. I'm seeing that it's — Look. Everybody's transitioning to it. This is not true. It's simply not true. But we'll remain alert, and that's why I'm always super interested in these new ways of solving problems, because that's our mission. I mean, as a company, it's monitoring. We're not container monitoring or cloud monitoring. Monitoring your

application, and your obligation is there to serve your customers. You're not monitoring application for the pleasure of monitoring or just for the sake of it. We have to be super attuned into where things are going, where we're seeing traction and make sure that our own backend, our own product really serves that. Otherwise, we'll be just like the oligarch at some point. You're sort of innovated out into oblivion by somebody who is more attuned to what's happening.

[0:33:45.5] JM: That volume of data that you're seeing over time that allows you to provision somewhat predictably. How does that graph look? Is it just linearly increasing over time? Is there some slope or is it exponential looking? What does that looks like?

[0:34:02.9] AL: It's mostly geometric growth. So it's exponential, if you will. What makes it difficult, I think, and challenging and interesting for us is, as we add new products, the amount of data we receive per customer increases sort of faster in, and when we release logs, for instance, that's when we start to see, "Wow! This is an order of magnitude or two orders of magnitude more, data coming our way. Now it goes to different pipelines," and so on and so forth, and that's what internally we tell ourselves is we have to engineer our systems for 10-X, 20-X over a relatively short time of horizon.

There's no point in engineering for 100-X or 1000-X. There's a point, but it's particularly hard and I think it's difficult and expensive, but we can't also be thinking, "Well, we'll just make it handle twice as much traffic and it will be set," because if we do that, we'll be set for six months, nine months and that's that. So that wouldn't make a lot of sense for us.

[SPONSOR MESSAGE]

[0:35:10.9] JM: We are running an experiment to find out if software engineering daily listeners are above average engineers. At triplebyte.com/sedaily, you can take a quiz to help us gather data. I took the quiz and it covered a wide range of topics; general programming ability, a little security, a little system design. It was a nice short test to measure how my practical engineering skills have changed since I started this podcast. I will admit, although I've gotten better at talking about software engineering, I have definitely gotten worse at actually writing code and doing software engineering myself.

But if you want to check out that quiz yourself and help us gather data, you can take that quiz at triplebyte.com/sedaily and in a few weeks we're going to take a look at the results and we're to find out if SE Daily listeners are above average. And if you're looking for a job, Triplebyte is a great place to start your search, fast tracking you at hundreds of top tech companies. Triplebyte takes engineers seriously and does not waste their time. I recommend checking it out at tripledbyte.com/sedaily. That's triplebyte.com/sedaily.

Thank you, Triplebyte, for being a sponsor of Software Engineering Daily. We appreciate it.

[INTERVIEW CONTINUED]

[0:37:09.1] JM: You mentioned Kafka, and I can imagine Kafka being useful for slicing up the incoming data by topics of the companies that are sending you data and perhaps buy the applications within those companies. Is that why Kafka is useful? Is it for building this hierarchical topical buffer of data so that you can process it in a more sane fashion?

[0:37:37.7] AL: We use Kafka mostly for, I'd say, couple of properties. The way we will inject data in Kafka will actually change if you think about the processing pipeline where you receive a payload that contains a snapshot of everything that happened, all the time series that existed between A and B on a particularly host, and you're having to pivot it into — Instead of having a snapshot of a bunch of time series, you sort of take each time series and you sort of collate it with the points that existed before.

Kafka is particularly suited at that buffering. In terms of how we address the topics, it tends to be very granular because it's really per — It could be per metric per customer, for instance, because we see customers have wildly different profiles in terms of how many time series that generate, for instance, or even a given customer may have a very different profile from time to time when they generate a lot more data and less data depending on what it is they're doing.

What Kafka buys us, I think, fundamentally is sort of very predictable performance, and that's — To me, it's going to the simplicity of the design, which is really a huge quality in my eyes, and it also — From a pattern perspective, allows us to have sort of loosely coupled components. This

is, in comparison, if you think about — Let's say, a pipeline processing [inaudible 0:39:03.8] storm or things like that where you have a very sort of strong semantics that are encoded in the topology itself. If you use Kafka and a bunch of, obviously, consumer-producer, you have a much more implicit topologies.

The producer produce. They only care about who's consuming are not, and sort of Kafka will also not prevent the producer to produce if no consumer is there, and that's been great for us, because it also allows us to have teams that build things at different timescales or that they don't have, the lose coupling that exists on software perspective. I think that gets reflected at the team level. So this last synchronization that needs to happen between team A team B if there's a — Team A produces, team B consumes. For sure, they have to talk to each other to make sure there's compatibility, but that's about it. They can release at different times as long as they respect that basic contract.

[0:40:00.3] JM: Have you built out a multi-cloud architecture where you've got some failover into multiple clouds? What's your multi-cloud story?

[0:40:08.9] AL: So multi-cloud is something that it's actually for us born out of business necessity, not so much out of technical necessity. By that, I mean, really what's been pushing multi-cloud for us has been privacy and data sovereignty regulations, or at least an environment where people pay a lot more attention to them.

So, for instance, we have some capacity in Europe for — And that's relatively new, but the goal is to address European customers who want their data t to reside in Europe, right? Historically, they have shipped that data over to the US, and that's obviously redefined, but we are starting to see a lot more demand for local data sovereignty, or locality of data in the jurisdiction that they feel comfortable with.

In terms of multi-cloud, as in multiple providers, the need there is really — It becomes more a business need, as in like, "Do you want to have just one vendor or do you want to have multiple vendors?" And you see a thing across the industry. The Netflix, a sort of traditional, sort of trailblazer [inaudible 0:41:18.8] or for AWS. Investing in other class. We do the same, and this is really, for us, now that clouds become — Okay, public cloud are reality and it's something that

business grapple with every day, and now you sort of revert back into, “Okay. I have multiple vendors that kind of offer the same thing, not quite, but how should I think about diversification?” From a risk perspective, if one goes down and so on, but also from a pricing perspective, such that you can, as a business, extract the best cost possible from your providers.

[0:41:51.8] JM: So this conversation about infrastructure and technical engineering could go on for a very long time. It's interesting to hear your trajectory. I'd like to zoom out a little bit. How has your approach to engineering management changed in the last seven or eight years?

[0:42:09.6] AL: It's changing. I’ll say, it's changing. It’s changing. Interestingly enough, I think my philosophy if you will, we trace that back to the years prior. It’s not to like, I think, I started the company and then — During management came to me. I think I just really observed, was taught by a number people, leaders from the day I started working as an intern upstate New York. That’s kind of showed me how to do the work that made me who I am, I guess, as a professional.

So I think — Obviously, there's been some key or moments I can think back to, and I think Netflix and its culture deck. That was, I think, in our industrial, a little bit of sort of pivotal moment for a company thinking about how it defines itself. I think Google’s definitely had a similar influence in terms of how they think about hiring process and the 20% time and so on. Of course, there's a lot of gradation between what diverse companies out there have put out as this is our motto or this is our model. This is how we think about and how we operate, and the reality in which they operate. But, still, I think that that's transformed a lot.

I think, though, the things that haven’t changed is everybody wants to do like their best work regardless of which company they’re in, here at Datadog or somewhere else and that's — I feel really grateful that to be in this industry at this time when there's so much interesting stuff to do, so much freedom and autonomy. Compared to other industries, my wife's is an architect. It’s a very, very different vibe there, and so that's why I feel grateful, really, every day. That’s great.

[0:44:00.1] JM: What about product development? Have you learned anything important about product development that stands out in the last seven years?

[0:44:07.7] AL: Yes, undoubtedly. First thing I would say is, look, SasS is great. It's an forgiving, because you literally have to earn the business of your customers every single minute, hour, day, week, if you will. The positive about that is you can be pretty introspective. You can really understand. You can ask a question, “Well, customer-X, are they adopting their product or not? If they're adopting, what do they respond to? If they’re not adopting conversely, what's going on? What did we miss?” This is something where compared to a sort of a shrink wrap, cellular license, perpetual license. I’ll just collect the support costs of the support — you know, send a support invoice once a year for the next 20 years. It's a very different dynamic, and this is something that help, or lets us and helps us basically instil that through every single software engineer. It’s like, “Look. Yes, we’re building a product, but ultimately what we’re selling is a service. We’re not selling the product. The product actually remains ours. We’re just selling the service.” That attention to it, and this is not a [inaudible 0:45:12.0] service. Look. It helps you stay honest, is like when you release something, is it working or not? Is it as successful or not? Is this valuable or not? Then you can look the reality squarely in the eyes and say yes or no.

For me, the other big lesson is early on we got our first few customers. I started doing support, because I actually enjoy that, chatting with customer and say, “Hey, how is it going? What’s working? What's not working?” Have this closeness. I think that's been great and have been a lot of intercom and so on, have been super helpful in mediating that conversation in ways that that makes sense.

When I came to supporting customers, I came with the mindset of you’re calling you cable company and, look, your support is a dirty word. You don't want to have to deal with support. So you come to that with that mindset, you're wasting a huge opportunity, because what happens really is when somebody is interacting with you, even when they have an issue, it means they care, and so this is a very strong signal, that what you're doing may not be perfect. It definitely needs improvement, but they care enough. It makes a difference in their lives that they want to spend 5 minutes, 10 minutes out of their busy lives to talk to you and say, “Here's what working. Here’s what not working.”

So, for me, the transformation was to see support not as a sort of traditional, “Oh, it’s a cost center. Let's find the cheapest way you can deliver support at an acceptable level to our customers.” It turned into, “this is a great opportunity to actually close more business, because

we have — If you're in trial with Datadog and you encounter a problem, you're contacting us means, look, you care." So chances are there's a higher chance that you will become a customer.

What that means then is if I'm a customer and I have a problem and I want a quick resolution, who should I talk to? What kind a profile should I talk to or put in touch with? The answer is another of my peer, and that's how we thought about support not as a sort of traditional cost center, but rather let's staff it with software engineers who want to solve problems, and this — It's a different mindset. A slightly different mindset from the traditional sort of product builder wanting to build things over a longer timescale. In this case, it can be, "I want to solve problem, but I want to see you a bit more immediacy and I want to have the pleasure of interacting with someone else. Which software engineering may or may not afford, depending on how you go about it.

[0:47:44.8] JM: One of the hardest problems that I have seen companies encounter, and I've encountered this myself trying to build things beyond the scope of a podcast, is that when you try to build a second product or you try to have something that's an expansion to your current product. The shift in focus away from the core competency can just destroy you. It can really erode your productivity. it can potentially disrupt your entire business and not disrupt in a good way. Disrupt in a very bad way.

You've expanded into a couple of additional products over the years, the APM work and the logging work. I think both of those were additional products that may be you had in mind at the beginning, but you didn't necessarily have. I think of the first core product was really infrastructure monitoring. Did you learn anything valuable in the product expansion phases of the company?

[0:48:42.7] AL: Absolutely. You're right. When we started, we only had infrastructure monitoring. It took us some time to, A, of course, realize that, "Hey, we need more," or pinpoint exactly where we needed. It's sort of costs and customer interaction that helped us with realizing that. Then we went at it in two different ways so far I think that I have been reasonably successful. One is in building it internally and the other one is building it to an acquisition. Not of them are easy. Both of them involve some degree of anywhere between discomfort and pain.

Internally, how we set out about it is we basically formed a new team to — So in the case of APM. We formed a new team to build APM, and that was painful because we had productive people contributing to whatever it is that they're working on and we say — We told them, "In a month from now you'll be doing something completely different. So you have more or less a month to figure out how to transition what you're doing out to other people. They'll be extra — Those other people will feel the load. It's not going to be fun," but we figured that the only way internally to go about it, is we have to peel off a number of people, form a new team and then push, and then kind of give them that sole focus. They should be focused on that. You should not be focused. Not necessarily. That's like an effort.

The way we've been, I'm think, fortunate enough that the core product has had enough momentum to keep us busy, and so it was really, "Look. We can't do — We have to do A and B. We cannot say just B. Let's forget about A," because that just wouldn't work.

The second way we went about it was an acquisition, and there we spent a lot of time not only surveying what the market was in terms of log management, but who it was, who was at the core of the companies? Were we compatible from just even an individual chemistry perspective?" We thought hard about what's the right size? We should bring someone in. Somebody too small and they feel their lost into Datadog. Somebody too big, and then you can't really make it work, because it's just a lot of — The two cultures are struggling to integrate with one another.

But I think both of them require a ton of work. You're right. The reality is you have to do more things. It's just not. You can't just do one thing and then another thing. I think, for us, where we have been fortunately lucky, is we've seen momentum in in all cases so that you're sort of faced with, "Well, we have — Yes, we have to make hard choices, but it has — All three things have to work. How do we go about that?

[0:51:17.8] JM: Alexis Lê-Quôc, thanks for coming on Software Engineering Daily. It's been really great talking to you.

[0:51:21.8] AL: Absolutely. Thank you very much for having me.

[END OF INTERVIEW]

[0:51:25.9] JM: Failure is unpredictable. You don't know when your system will break, but you know what will happen. Gremlin prepares for these outages. Gremlin provides resilience as a service using chaos engineering techniques pioneered at Netflix and Amazon. Prepare your team for disaster by proactively testing failure scenarios. Max out CPU, black hole or slow down network traffic to a dependency. Terminate processes and hosts. Each of these shows how your system reacts allowing you to harden things before a production incident. Check out Gremlin and get a free demo by going to gremlin.com/sedaily. That's gremlin.com/sedaily to get free demo of how Gremlin can help you prepare with resilience as a service.

[END]