EPISODE 707

[INTRODUCTION]

[00:00:00] JM: Matt Klein has worked for three rapidly growing internet companies. At Amazon Web Services, he worked on EC2, the compute as a service product that powers a large percentage of the internet. At Twitter, he helped scale the infrastructure in the chaotic days before Twitter's IPO. Today, he worked at Lyft building systems to allow for ridesharing infrastructure to work more safely and reliably.

Hypergrowth internet companies are faced with quickly growing demands on their software. The demands on the software exposed problems with the core infrastructure. As the demand ramps up, the infrastructure gets strained and it can lead to problems. Simultaneously, the company is trying to ramp up its hiring process. More engineers get hired and the institutional knowledge within the company starts to weaken. Your documentation gets out of date. Senior engineers get overworked. They burnout and they leave the company.

When a company starts growing quickly, everything can break down. Communications can break down. Infrastructure can break down. It's a good problem to have, but that doesn't mean that it's easy to work through. A hypergrowth company can suffer from a lack of human scalability. Matt Klein has observed these challenges at Amazon Web Services, Twitter and Lyft.

In his article, the Human Scalability of DevOps, he explains why these problems manifest and what can be done to alleviate them. In a previous show, Matt discussed the engineering challenges at Lyft that led him to create Envoy, a service proxy. That show was highly technical. It was a great review of modern infrastructure.

This episode covers some broad technical topics, such as DevOps, and site reliability engineering, and platform engineering, but the episode is mostly about how a hypergrowth company can manage culture and hiring and engineering organization and just general operations.

Matt is a very fun guest to have on, because he questions some of the strange practices that have been widely adopted by successful companies. Internet companies are a very new phenomenon and the management tactics that they have adopted are not very well proven. We don't have a whole lot of data on how to run an internet company well. So it's great to have someone like Matt provide a fresh perspective on ways that companies that can scale their technology and the organization more effectively. We talked a lot about culture and how to make an engineering culture that is appealing to engineers and how to retain those engineers.

This is probably one of my favorite episodes that I've ever done, because Matt talks really honestly. He's a fantastic engineer and he's also just got a holistic view of how a startup, how a big organization, how any organization that's moving quickly with technology can orchestrate the operations and the technology infrastructure of the company in an intelligent fashion. I really hope you like this episode with Matt Klein as much as I did.

[SPONSOR MESSAGE]

[00:03:26] JM: OpenShift is a Kubernetes platform from Red Hat. OpenShift takes the Kubernetes container orchestration system and adds features that let you build software more quickly. OpenShift includes service discovery, CI/CD built-in monitoring and health management, and scalability. With OpenShift, you can avoid being locked into any of the particular large cloud providers. You can move your workloads easily between public and private cloud infrastructure as well as your own on-prim hardware.

OpenShift from Red Hat gives you Kubernetes without the complication. Security, log management, container networking, configuration management, you can focus on your application instead of complex Kubernetes issues.

OpenShift is open source technology built to enable everyone to launch their big ideas. Whether you're an engineer at a large enterprise, or a developer getting your startup off the ground, you can check out OpenShift from Red Hat by going to softwareengineeringdaily.com/redhat. That's softwareengineeringdaily.com/redhat.

I remember the earliest shows I did about Kubernetes and trying to understand its potential and what it was for, and I remember people saying that this is a platform for building platforms. So Kubernetes was not meant to be used from raw Kubernetes to have a platform as a service. It was meant as a lower level infrastructure piece to build platforms as a service on top of, which is why OpenShift came into manifestation.

So you could check it out by going to softwareengineeringdaily.com/redhat and find out about OpenShift.

[INTERVIEW]

[00:05:34] JM: Matt Klein, you are the creator of Envoy and an engineer at Lyft. Welcome back to Software Engineering Daily.

[00:05:39] MK: Thank you. Great to be here.

[00:05:40] JM: You are an infrastructure engineer and the last time you came on we talked about Envoy, and your recent post was more about humans. It was called the Human Scalability of DevOps and it was about the org structures and hiring processes of engineering companies. Actually, I guess this is not your most recent post anymore, because you've written about open source since then. We've got a lot to talk about. But I want to start by talking about this human scalability of DevOps article, because I thought it was quite interesting. Why did you write this article?

[00:06:13] MK: It's a topic that's been top of mind for me recently. I think we're at an interesting point in the industry right now where we are moving towards a model of operating software, which is mostly cloud-based or moving towards a bunch of serverless paradigms where we would like people to focus on business logic.

We're racing towards this theoretical world in which we don't necessarily need operations engineers. I think people, we can have a long conversation probably, an entire hour, on what DevOps actually means. But there's been a shift in the industry towards a general expectation that we have software engineers that will operate their own online services. That's obviously

figuring out how to have alarms? How to observe them? How to monitor them? What are the SLOs? What are the SLAs?

It's been top of mind for me to see this evolution all the way from 15 to 20 years ago where we would generally have operations engineers who would take software and maybe do some testing and then obviously shift that out. To now we're doing continuous integration, continuous deployment. We're expecting software engineers to write software as well as actually ship it and then own it in production.

From an organizational perspective, I don't necessarily think that as an industry, we have figured out how to support people and actually being fully successful in those roles. By support, I mean, how do we educate them? It's not like people learn how to operate reliable internet software in college. They'll typically learn that on the job.

What type of new hire education? What type of continuing education do we offer people to learn those skills? How do we do documentation? How do we expose to people what systems are available? How they use them to do networking and data stores and observability and those types of things? How do we get them help when they might not understand what are the right settings to use, or what system they should use to solve a particular business problem.

I'll stop there and let you ask follow ups, but I just think we're at a very interesting point where we're moving rapidly towards a world in which we want to enable people to move quickly and ship software. But I don't think we're necessarily fully supporting them and being successful doing that, and I think that that can create organizational strike sometimes.

[00:08:54] JM: Yeah. You've seen this play out in its most acute form, because the companies you've worked at have been under tremendous load. They've grown extremely fast at the times that you were at these companies. So just to give people some background, you worked at AWS on EC2, which is the server infrastructure that underlies everything. You worked on pre-IPO Twitter. You have now worked at Lyft through some of its busiest growth periods, and each of these were extremely fast-growing products.

You see the most acute form of what happens when the business pressures, the growth pressures, the user pressures end up impacting engineering teams in a really big way. I think companies that are growing in a slower fashion experience the same issues, but maybe in a less condensed time period. It's not like these are only issues that occur at the fast-growing startup internet companies, that they just occur in slower motion at other companies. How does the business pressure, that growth pressure at these kinds of high-velocity engineering environments, how does that affect an engineering team?

[00:10:11] MK: Yeah, and that's a great point, and I did point that out in my post where I think that my experience at working at these hypergrowth companies is definitely – It's an extreme example of some of the problems that I'm talking about. I think at companies that are growing less quickly and possibly have less human growth expansion in terms of the number of people that they're hiring or they're at the business pressures or the revenue growth or those types of things, I agree with you. I think these things are going to – They're going to exist, but they're not necessarily going to be quite as acute.

I think hypergrowth companies, particularly internet hypergrowth companies, we have increasingly moved into a world where everything is always on. Everything is 24/7. Everything is app-driven. People are iterating very quickly both in their lives, but also from a product perspective. There's a general perspective or a general feeling that people should be able to ship changes 24/7 and they should be able to iterate super, super fast and just adapt to the marketplace.

I think at some of these extreme hypergrowth companies, like whether that'd be Twitter, or Lyft, or early AWS or a bunch of other of the major unicorn type startups, what you're seeing is you're seeing extreme business growth. You're seeing the general feeling that, "Oh! We have lots to do, so let's hire more people as quickly as possible." Again, we have a long conversation about people have been writing and talking about this for 40 or 50 years now about whether adding more people to a software engineering project is going to make it go faster. Sometimes it does. Sometimes it doesn't.

In general, these hypergrowth companies are seeing major human growth. For example, I think Lyft, when I joined three and a half years ago, I want to say it had 80 engineers. Now we're

probably about a thousand. That's a 10X growth in like three, there and a half years, and I think that's not totally uncommon. I think Twitter maybe grew even faster than that.

I think what you see is that some of the softer things that are not related to writing code, but are really critical, and I'll come back to those. So, education. Again, how do we teach people when they're onboarding? What systems we have? How to use them? How to find documentation? Then if we teach them how to find documentation, we have to actually have documentation. Where's that? What format does it use? How do you search it? How will you support? Do you have a support queue? Do you have a clear understanding of where you are in that queue, or do you use Slack, and it's total chaos?

I think in general, again, I don't want to speak for the entire industry, because my experience is mostly in this hypergrowth space, but I have pretty good understanding of what many of the Lyft-like companies are doing and I know that this is a persistent problem at all of these companies. I think where that problem really rears its ugly hat, and that's what I was talking about primarily in the blog post, is I think you can get into a situation where you create a lot of organizational angst, and it's mostly through people that are frustrated that they don't feel like they can do their jobs well.

For example, if you're a product engineer and you're building business logic features, obviously you're being judged on making the business move faster, on shipping your particular feature, on generating revenue for that feature. You don't want to get held up on infrastructure, and I say this in most of my talks, that unless you're an infrastructure company, infrastructure is basically overhead. No one makes money on building infrastructure. Anytime that you have an application engineer that's building infrastructure or fighting with infrastructure, not building logic, that's time that's essentially being wasted.

There can be some angst that comes up where a product engineer rightfully so, if we don't teach that engineer what technologies to use, or where the documentation for that technology is, or how to get support when there's a problem, they can feel very frustrated. On the infrastructure side, if you're building infrastructure for a company like Lyft, you're not building low-level databases. Now, you're mostly leaving that towards a cloud provider, but you still have

a lot of complicated stuff to do around networking and databases and observability and stitching together a system that people can use and build their business logic on.

If you're an infrastructure engineer or you're trying to keep the lights on at a hypergrowth company that's having traffic growth and you're trying to do support, but the support isn't well-organized, somewhat chaotic. Like I said, there might not be a queue. It's unclear who's supposed to be doing this support. You can also have a bunch of angst and a bunch of anger on the infrastructure side when they feel like that they're not able to do the rest of their job or catch up or get on with their tech dev.

You get into this situation, which I've seen a bunch of times, where now you have angst on both sides. You have angst on the infra engineer side, you have angst on the product engineer side, and that can be pretty brutal from an organizational perspective. I think a lot of conversations around how do we deal with this? Do we do it with education? Do we do a documentation? Support systems? Do we have SREs? What's the right model here? There's no consistent answer right now within the industry. I think it's a really crucial problem that we have to deal with right now.

[00:16:09] JM: It almost sounds like a new flavor of the breakdown between developers and operations is product engineering trying to ship things faster and work with their PMs and so on, and then the product engineering team is interfacing with the infrastructure team where the "platform engineering team" and saying, "Hey, want to ship this new feature," and the platform engineering team is saying, "We've got this giant backlog of things around observability and just making the platform run properly, and our Slack channel is totally oversubscribed and has alerts out the Wazu and we've got alert fatigue and we just had three engineers go on leave last week because they're burnt out." I guess it's not as adversarial as the dev and ops breakdown, because at least we've learned to communicate with each other, but you do have couple classes of teams that are at odds.

[00:17:03] MK: I think you're exactly right, and I think you're also right that we've learned as an industry that the walls where you have dev, you have QA, you have ops, those walls are not – They're not the most sufficient way to ship modern software.

But at the same time, you're exactly right. I think we have new roles. We have this platform, or infra engineer role. We have the product engineer role. Some companies have this SRE type role. I think part of the problem if I were to really pinpoint it is I think as an infra, as a group of infra engineers, and I'm talking about companies like Lyft, but also from the cloud providers, we have this feeling now that the platform is available. The platform just works. You come and you have a database and you have all of those things and you can build your fantastic serverless infrastructures and it just gets monitored and everything is absolutely amazing. We have this vision where operations engineers are no longer necessary or even reliability engineers. Some companies might not think that they're totally necessary, or even infra engineers. Why would you have an infra engineer if the cloud provider or the serverless or the PaaS system is providing you with all of these primitives?

I actually think that vision is fantastic, and I think if you look out 10 years, I could totally buy that the majority of softwares being written on these magical serverless platforms where you show up and use these base primitives and things mostly work. Where that breaks down is it breaks down in two ways. The first way is that we – I don't think our infrastructure is as good as people think it is right now, and I think things don't quite work fully reliably well, particularly for operating high-scalability time systems for companies like Twitter, and Lyft, and Uber and companies like that.

I think we like to think that these platforms are plug and play and you just run it and they just work, but the reality is that the infrastructure and the tooling that we have today, it still requires a substantial amount of handholding and a substantial amount of wrapping, putting together a platform at a company through deployment, to testing, to, again, how people do networking, or databases, or caching, or observability.

We have to wrap these things up and give it to people in a way that makes sense even on top of something like AWS. That's the first part of how it breaks down. I just don't think that we have this vision. I don't think we're there yet. We're heading there, but I think we're getting ahead of ourselves.

The second point of where it breaks down is I don't think that you're ever going to be able to remove this concept of reliability engineering and operations focus. The reason that I say that is

that a big part of running a reliable system is understanding your customers, understanding what business metrics actually matter to your customer and figuring out how to measure that at the customer.

For Lyft, of course we monitor things, like what is our overall success rate? Is it 99.99% or something like that? These are valuable things that you can alarm on and get a general sense of what's going on. Ultimately, for Lyft, we can serving at the server 99.99% success rate all HTTP 200s, but it can all be JSON that's busted that's causing the client to crash and no one's taking any rides, right?

There's a much larger problem here of figuring out from a business perspective how do you reliable roll out software with your business? What feature flags do you use? How do you measure it at the client? What are the business metrics that matter? How do you measure week over week, hour over hour?

My larger point is that even as we move towards a more magical serverless type infrastructure, that doesn't preclude us from understanding these concepts. Even in 10 years, even 20 years, 30 years, there's still going to have to be an operations or a reliability focus to how we build software, because we have to know and understand how to measure things from the business perspective and how to safely roll out software that doesn't break the business, and that can be very business dependent.

I think that's where we're breaking down right now. I think we've really swung too far as an industry to thinking that infrastructure is magic and it is the solution to all of our problems with infrastructure as code. Don't get me wrong. I think we can do a lot from an infrastructure perspective and we can consolidate a lot of roles or a lot of functions that would have previously been there. I think that we've swung a little hard and we don't recognize how important all of these other things are in terms of reliability, operational agility, education, documentation, support. I just don't think we've focused on that enough and that's creating problems.

[SPONSOR MESSAGE]

[00:22:45] JM: Today's show is sponsored by Datadog, a monitoring and analytics platform that integrates with more 250 technologies, including AWS, Kubernetes and Lambda. Datadog unites metrics, traces and logs in one platform so that you can get full visibility into your infrastructure and your applications.

Check out new features like trace search and analytics for rapid insights into high-cardinality data, and Watchdog, an auto-detection engine that alerts you to performance anomalies across your applications. Datadog makes it easy for teams to monitor every layer of their stack in one place, but don't take our word for it, you can start a free trial today and Datadog will send you a t-shirt for free at softwareengineeringdaily.com/datadog. To get that t-shirt and your free Datadog trial, go to softwareengineeringdaily.com/datadog.

[INTERVIEW CONTINUED]

[00:23:52] JM: You have had experience at Twitter and Lyft, and these are products that were built in a certain timeframe – Well, they were built in two different timeframes. The original code for them were built in different timeframes along the cloud infrastructure timeline. I would say Lyft, which is the company you work at now, it was started before some of these serverless products came to market in a way that they would be digestible to a company like Lyft. It's hard for me to imagine Lyft running on Heroku or Firebase. I don't think it would be impossible, but these do seem like – I haven't heard of any giant companies, really giant companies, that are entirely on Heroku or entirely on Firebase. But I think we'll get there eventually. Maybe it looks like AWS Fargate and you have a collection of Fargates or the Azure Container Instances or the Google, whatever the Google equivalent of those. Maybe that's like the serverless backbone that you can start with. Even then, it's like not really "serverless", because these are servers basically. They just happen to be standalone containers instead of VMs. They're just cheaper, more discardable servers.

I'm of two minds when it comes to this question of do we have the things to build serverless at Lyft, at a company like Lyft today? I think we should probably talk more about how you are actually seeing things at Lyft, because you have a lot of wisdom to convey here. I think one place that we could touch on is this question of the human capital that you're bringing on to your team. So you have this term you call the fallacy of fungibility, and I think this really articulates

some of the issues around hiring that you have seen at Lyft, or maybe at previous companies. What is the fallacy of fungibility? What do you mean by that?

[00:25:54] MK: Well, just to start, it's not just Lyft. I've seen this at Amazon. I saw it at Twitter and I've seen it at Lyft and it definitely exists at a bunch of other companies. I just want to be clear that I don't think has abled that specific thing, though I have experience here. I think over the last – Again, this is more the last 10 or 15 year phenomenon, and it parallels this move towards infrastructure as code, or machine learning as code, or operations as code. It's essentially the idea that we can code or program our way around all problems, and therefore we are all software engineers.

Where I see that most acutely, and I've seen this at almost every company, is that – From a fungibility perspective, it's the additional idea that not only are all problems solvable with code or software engineers. It's that you can take one software engineer and you can take them from one project and you can move them to somewhere else and they'll be totally productive. There are companies that make it an explicit goal to actually hire this way, that software engineers – I don't want to go as far as say, cogs, but the idea is that they are replace units that can be slotted in and they will do the task at hand.

I really think that this cannot be further from the truth, and I think it's not true for a variety of different reasons. Just starting with personal interest, we have a large industry. There are all kinds of things that people work on, from infrastructure, to games, to machine learning, to firmware. The list goes on and on and on. Just from a personal interest standpoint and setting aside capability, there are types of problems that some people enjoy working on more than others. Some engineers love debugging. Some engineers love writing code. Some engineers like working on graphics. Some like working on client programming, or webpages, or operating systems problems.

I'm not even talking about capability. I'm just saying that people have preferences. In a competitive hiring environment, when people can move around, just because someone can work on something doesn't mean that they want to. I think we consistently see that when we have people work on things that they are not necessarily passionate about, their productivity is lower. They might have more interpersonal problems. I think that can become problematic.

Again, I'm not saying that everyone needs to be happy 100% of the time. This is work after all, but I take the general philosophy that given a competitive hiring environment, that work should generally be a net positive. It's not always going to be awesome, but if we can't make people happy, say, greater than 50% at a time, we're probably doing something wrong. That's part one.

Part two is I do think that there are capability differences. I'm not talking about intelligence. I'm just talking about that there are people that are just good at different things. Some people are amazing MD file programmers. They can just craft new code in an MD file and it's going to come out in a more maintainable way with less bugs.

There are people who are incredible debuggers or incredible problems solvers. They can dive in to a codebase that they haven't seen before and fix almost anything. There are people that have incredible operations or reliability intuition. They just know how to go in and figure out how to rollout software and ship software in a way that doesn't break the customer. These skills, it's not that people can't learn them or can't improve on them and, of course, that absolutely happens.

People are good at different things and often a good team is composed of people that are good at all of these different things. I think that sometimes we can have a whole talk around how we do interview loops within tech, and I think it's very problematic, because particularly in software engineering, we're going to try to jam people through a stereotypical tech screen interview with some coding and some design, like a bunch of other things. But it's not necessarily indicative of either what they would do on a day-by-day basis. Meaning, are they really going to be writing a bunch of code or are they going to be doing incredible operational excellence that is having absolutely tremendous impact on the overall business? Let's figure out how to interview them for that and actually hire them.

I've seen a lot of people either get pushed out of an org because they don't fit a particular mold or not get hired because they can't pass a stereotypical software engineering interview loop when they would have amazing impact. That is frustrating to me. That's the general concept of fungibility, and I wish as an industry, we would better recognize that people have things that they're better at, that people have things that they like more, and that we have to start thinking about software engineers and kind of the group of people that we build up together to work on

larger projects as not interchangeable cogs, but as people that have discreet things that they can add to a larger project. I think if we did that, we would potentially have less overall problems. We have people that would be able to focus more on reliability, or on operational excellence, or on education, or on all of those things, and we don't support that very well from my personal experience.

[00:32:07] JM: This is a really important topic to me, and it's basically the reason why I left the software engineering industry and started a podcast around software engineering, is because I was so tired of this mode of thinking, of this industrial age style thinking where, "Oh! You're trained to work on the assembly line. So we're going to put you on the assembly line."

Some people are okay with that. I think this is something that's going to come back to really bite the big tech companies if they're not proactive about it, because now as a giant tech company, if you've got this one size fits all, maximize the funnel of people who can reverse a length list of binary search trees that are formatted in JSON of these stupid problems. If that's your bar for hiring, sure, you're going to get people who can solve those and then the ones who are able to fool the process, because they're really good at memorizing this test, and I know because I was one of these people, they're going to enter the job, they're going to do your boring work that you give them and then they're going to get sick of it. This issue, it bothers me so much, and I don't understand why it's so persistent, other than literally nobody else wants to do the work. This is the only way to snooker people into maintaining your terrible legacy code.

[00:33:29] MK: Yeah, and this issue bothers me too. I guess I will say two things. One, I think it actually goes even beyond software engineers. I will give you a particular anecdote from Lyft, which I found particularly frustrating and I don't actually mind sharing, because it was so frustrating. I think that one of the biggest ways that we can help infrastructure software, and I've said this a lot in a lot of different talks, is that I think we underinvest in the UI and the user experience of the tools that we're building. We build lots of config files. We build lots of amazing systems, but we don't necessarily build portals. We don't build beautiful UIs to make it easier and more accessible for people to come in and interact with the overall system. I think if we did that, we could build systems that people – It would require less documentation. It would be easier for them to figure out what's going on.

I actually found a user experience designer who was passionate about this and wanted to work on it, and she had a great portfolio and was super awesome. I got her into Lyft and I had her meet with the design team and she did an interview loop and she actually got an offer. Then we went through this whole process of where the design team basically said, “Well, we don’t know how to support a designer in infrastructure. She would have to work on the app.” I won’t go into further details, but it’s very indicative of this larger problem where we have these rules, like we have a designer, and the designer works on the webpage, or on the customer-facing app without thinking about the fact that we need some of these skills in different roles. I guess I just wanted to point out that this is a system problem within the industry.

Coming back to your point of like why do people do this, I have to be honest, I think Amazon has actually had a large part in pushing this forward. Amazon is an incredible company, but I think historically Amazon has thought of engineering just as another part of its warehouse effectively. It’s like always trying to optimize how do we do different things with the same set of people. I think it goes along the same lines of the open plant offices. It’s just a general feeling of like we’re all the same. We’re all going to work in a line and get swapped out.

What I would say is that I do actually think that people are realizing that this is problematic. I think there is some light at the end of the tunnel, because 5 or 10 years ago, I remember being at Amazon and really having some extensive arguments about this in terms of because I think for Amazon, and I don’t know if it still is. I haven’t worked there in a while. Fungibility was actually one of their primary things that they actually hired for. I think Facebook historically has also been similar. I think people are starting to back off from that, because it’s becoming clear, well, that it’s a competitive space. People have different needs and desires. People are good at different things. Also, we need different skillsets to work on different things. I think people are starting to think about this, but I don’t think it is in a consistent enough way or I don’t know that we’ve really changed our overall thinking. I still but up against this and it’s frustrating.

[00:37:04] JM: Just to give more color what you’re saying about Amazon, because Amazon was the last place that I worked before I started this podcast, and I just want to discount everything I’m about to say with the fact that I love Amazon. I am inspired by Amazon. There is no leader who has inspired me more than Jeff Bezos. I think the company is also very introspective and it really does try to solve the problems among its workforce, and I hope that somebody at Amazon

who might be listening to this can understand where I'm coming from in saying this, and then I'm really saying this out of a place of love for Amazon. But when I was at the company, I expressed a desire and a capability to work on other areas of Amazon than the thing that I was fungibly assigned to. I just received no positive encouragement or receptivity from within the company. I've heard of some actually material changes that came into place just after I left, which perhaps make that more – Make an engineer at the company more mobile.

The thing is like I was there and I was like – I would just talk to people and I'd be like, "It's really amazing that AWS enables us as engineers. We could leave and build something for zero dollars," and this is like unprecedented, like, "Hey, are you guys seeing this? This is pretty cool. I can just go and leave and build a software company for zero dollars if I want to." Other people would be like, "Yeah, I'm fine working on this service that I don't really care about." I'd be like, "How do you think that way?"

I mean, people are waking up to that. They are waking up to the fact that they could even go on Upwork and just be a developer for things that they love on Upwork and make a decent living. That is the lower bar, and that's – Frankly, I think that sounds like a better existence than at a lot of tech companies. You get to work on stuff that you love and make money from home. Sign me up for that. I mean, obviously, you have the green card issues, which that's a whole other snookering in issue.

Yeah, anyway, I think these are really important points and I really, really hope they get addressed, because I think there is a way to address them. There is a way to say whether it's 20% time or 50% time or whatever. You work on the service that's not very interesting this percentage of time and you get to work on something that's interesting to you this other percentage of time, and it's allocated to you and you get to demand it, and that – How is that not win-win? That's what the situation should be.

[00:39:29] MK: Yeah, there's that, and I think there's also just as companies grow and scale, I think there just needs to be a recognition that there are more specialized roles available. Again, whether that'd be operations, or reliability, or debugging, or kernel programming, or like something. I think it's okay to allow people to specialize and to be happy in doing that if that's

what they want. I'm not saying that we should force people to do that. I think if people want to be general as engineers, that is absolutely fine and it's a great thing too.

Again, I do see this changing. I know I haven't again worked in Amazon a long time, and full disclaimer, I'm also a big fan of Amazon. I believe Amazon has improved in this area, and I think other companies have too. But like I was saying, we still but up against this in terms of a team needs a particular role. That role is not – Not only is the normal software engineer interview not even applicable. They're just not even going to do a normal software engineering role, and they're happy about that, and we're happy about that. Let's figure out a way to actually hire them.

Sometimes I feel like as companies scale, just there becomes enough bureaucracy around HR and like what does the job rec look like and how do we do this, and then people just stall. I wish companies would facilitate more customer roles, and I feel like more people would end up being successful and happy.

[00:41:05] JM: That story of the designer, the designer who wanted to build internal UIs for infrastructure teams. I was talking to one of the VP engineering, or director of engineering at Stripe, Raylene Yung, and I was asking her, I was like, "Stripe seems like they're really trying to figure out how to do the nomenclature of engineering or off teams. How do you name designers? You want designers to feel free to work on different areas of the product, but you also want to give them KPIs and stuff," and this applies to these – It just is a challenge to figure out how you restructure a company where you can have these. Maybe you optimize for workforce satisfaction or something like that. I mean, how you solving this at Lyft? How are you – Are you solving it? It sounds like you're dealing with these issues right now.

[00:41:57] MK: I mean, it's a continuous discussion. I was in a meeting yesterday where this came up. It's a thing that people are aware of. Again, this is not less specific. I don't want to sit here and seem like I am bashing on Lyft. I think this is an industry-wide problem.

[00:42:15] JM: Right. For the record, even Stripe. They were like, "We don't know how to do this."

[00:42:20] MK: Right. I don't think anyone knows how to do this, and I think as in all things management, or tech management, there are no easy answers here and people have good points on both sides. You want to have discreet job roles so that you can have a clear set of levels and career progression. If you don't have that, then how do people know what they're doing or what they're tracking for?

I mean, there are real problems here. I'm not saying that this is simple. I'm not even saying that I know what the answer is, but I do know from being an engineer or a leader on the ground, that I just wish that we were more adaptable and allowing slightly customer roles. It's like call them a software engineer, but let's allow the interview to be a little customized, because we kind of know what they're actually going to be doing, right? I do think that there are gray areas here that we could enable, but it's hard, and there's no answer here and it's a continuous problem.

[SPONSOR MESSAGE]

[00:43:34] JM: Managed cloud services save developers time and effort. Why would you build your own logging platform, or CMS, or authentication service yourself when a managed tool or API can solve the problem for you? But how do you find the right services to integrate? How do you learn to stich them together? How do you manage credentials within your teams or your products?

Manifold makes your life easier by providing a single workflow to organize your services, connect your integrations and share them with your team. You can discover the best services for your projects in the manifold marketplace or bring your own and manage them all in one dashboard. With services covering authentication, messaging, monitoring, CMS and more, Manifold will keep you on the cutting-edge so you can focus on building your project rather than focusing on problems that have already been solved. I'm a fan of Manifold because it pushes the developer to a higher level of abstraction, which I think can be really productive for allowing you to build and leverage your creativity faster.

Once you have the services that you need, you can delivery your configuration to any environment, you can deploy on any cloud, and Manifold is completely free to use. If you head

over to manifold.co/sedaily, you will get a coupon code for $10, which you can use to try out any service on the Manifold marketplace.

Thanks to Manifold for being a sponsor of Software Engineering Daily, and check out manifold.co/sedaily. Get your $10 credit, shop around, look for cool services that you can use in your next product, or project. There is a lot of stuff there, and $10 can take you a long way to trying a lot of different services. Go to manifold.co/sedaily and shop around for tools to be creative.

Thanks again to Manifold.

[INTERVIEW CONTINUED]

[00:45:49] JM: To touch a little bit more on your article. It was about these collection – I mean, actually this does relate to what we've been talking about. Your article does relate to what we've been talking about, because it is this question of human scalability. It's almost like it doesn't feel like an engineering problem or a software engineering problem as much what are we changing about org structure? Because we know that the way that we change the org structure is going to reflect how our product evolves overtime. I think you've built out a platform engineering team at Lyft, which is one way of solving one area of this human scalability problem.

The last time that we talked, we were talking about Envoy, and Envoy was a solution to very serious technical problems, because it spanned these issues of we're afraid of deployment, because we don't have good instrumentation around things. This was a standardization of a certain aspect of instrumentation. That was 1.5 years ago. How have the scalability problems, if we're talking on a technical level. Have they changed today? Do you still have kind of outage issues and observability issues that are worth going into here?

[00:47:08] MK: Yeah, that's a very interesting topic. In general, I think the type of scalability problems that we're having at Lyft now are they are much more subtle, via Envoy, via a bunch of work that people have been doing on our database systems, a bunch of reliability on various other pieces of the system. In general, at Lyft, we don't have – It's extremely rare. I can't remember the last one when we had a more base level distributed systems outage, like we're

DDOSing ourselves, or some database is just down, or something like that. We don't really have those problems anymore.

Our problems now [inaudible 00:47:49] reliability perspective I think are a lot more directly related to the types of things that I wrote about in that article. Meaning, we are asking developers to operate their systems, but we don't necessarily educate them on the appropriate ways of doing that. How do you monitor very small things that are very business specific? I'll just make up some examples. From Lyft, how do we know that ETAs haven't regressed or how do we know that we're using a proper pricing model, or that primetime hasn't done something bad, or etc., etc., etc.? There are all of these things around payments and just very Lyft specific business things where a lot of our outages or a lot of our incidents, they're more subtle business level things. They're not based distributed system things.

I think a lot of these outages ultimately could be better solved by figuring out how to expand this reliability culture out to the entire engineering order? That's where you get into these larger problems? How do we educate people? Because we cannot expect them to just know this. That's not fair, right? That's just not fair.

What type of new hire education? What type of continuing education? What type of documentation? Again, how would you support? How do we do design reviews? Like production readiness reviews? How do we set SLOs? How do we set SLAs? These are things that we have not scaled very well at Lyft, because we still don't have an SRE type role. We are rectifying that, and that's something that's being worked on.

I think that's very common where companies, when they're in their initial hypergrowth stage, they're going to have a lot of really fundamental distributed systems problems, and then depending on their growth level, they'll continue to have them. Within the order of magnitude, those things typically get solved. Then I think you get into these more subtle issues.

I think that from an organizational perspective, my impression is that we're now at that next level where we're trying to figure out we are fundamentally reliable. Lyft doesn't just go down anymore, but we still have a bunch of different niggling issues which might cause issues. At that

point, let's figure out how to actually solve that. Let's figure out how to educate people and get the right systems in place to make that better.

[00:50:25] JM: Yeah. I think we can use this as an opportunity to once again applaud Amazon, because I think they arguable led this. This is what I see at most fast-growing startups these days. It's an amazing development. We no longer have outages. It's like, "What?" We don't have system-wide outages anymore. That is the norm. We have things that are like outages. We have serious problems with pricing, for example, because of a machine learning rollout mistake. I mean, that's a whole different class of problems. In some ways you could say, "Oh! An incomplete failure is worse than a complete failure." But generally that's not the case, because people just need to get their rides from point A to point B reliably. That's the most important thing of our service.

[00:51:11] MK: Yeah. I think that – This brings up the larger problem of, okay, it's great, that as industry or at least from the larger hypergrowth starter perspective or building technology, that will hopefully a lot of people to not have some of these fundamental outages. But I think it's our goal to build 100% reliable systems. For me, as a software and as a system engineer, really what I like about my job is I like building something that people will use and I want it to work 100% of the time.

Now, of course, nothing works 100% of the time, there's always bugs. But I'm of the opinion that if we're not striving to make it work 100% of the time, we're not doing well by our customers. Whether those customers are the customers of the product or internal customers of infrastructure. I think that it's great that we're making progress and we're making systems more reliable. The base level of reliability is definitely higher than it used to be. But now we're at that next level, which is let's take off and let's solve those other more nefarious problems around pricing or more subtle bugs, and that's harder. But I think that's where we're at right now.

[00:52:30] JM: Okay. I now we're up against time. I wanted to just ask you a little bit about open source, not to open a can of worms, but I want to open that can of worms. You started Enjoy, which is one of the most successful open source projects in recent memory and it's infrastructure piece. Monetization of open source infrastructure is a classically difficult problem. Some companies had made it work really well. Some companies have completely bombed

trying to make it work, and we had this recent announcement of the Redis license change to commons clause. We've covered that a little bit in the previous episode, we maybe you could just give your perspective on the issue and the ways in which you believe open source is broken and hope that it can be fixed.

[00:53:15] MK: That's a tough one to do in a couple of minutes. We should probably circle back at some point and do a larger one. I can just –

[00:53:23] JM: I'm going to have Kevin Wang in the near future. He seems like he's got to a really balanced position on it.

[00:53:30] MK: Yeah, and I can just talk about it super briefly. I think at a very high level, we are generally moving as an industry particularly for infrastructure software to essentially requiring software that we use to be open source. For a variety of reasons, which I don't have time to go into now, I think people are increasingly skeptical of proprietary or a closed-source software. Not in all cases, but for fundamental infrastructure [inaudible 00:54:00] for things like Enjoy. They just want to have the comfort of knowing that they can see source code, that if something really bad happen, maybe they could go take it and actually edit it.

I think that's one thing. I think startup by Linux and then a whole bunch of other projects have come up over the last 20 years or so. I think there's a general expectation that like a software should be open. I think at that time, you come into this really meaty problem of people want open software. They want it to be very high quality, which takes high quality engineers who need to make a living. Then there's infrastructure around building software. Who actually pays? That's become very complicated.

What we've seen over the last 5 or 10 years is open source is funded really one of two ways. It's the offshoot of a company that's making money some other way. So whether that'd be Google Search or Facebook with their ads or something else, or Lyft with their cars, open source is offshoot of that. Then you have people that are starting companies around open source software, and they are attempting to make it open source, because that's what's required. But then they need to fund their own development. So then we get into business models. Whether

that'd be open core. So making part of it free and part of it paid. Whether it'd be offering it as a service or something like that.

Right now, we're just in this difficult time. We want everything to be open, but there's always ulterior motives at play in terms of who's paying for it? Why? There's the interesting role of foundations. I would love to talk about this at length, but I just think that we're in a difficult time right now where it's unclear who's paying for the critical software that we actually rely on and how to keep that software open and in partial without having ulterior motives, without potentially depending on software that might get altered later because of ulterior motives. That's the complexity that we are in right now.

[00:56:18] JM: All right. We will certainly revisit this in the future. I'm sure we're going to have more developments in the coming months around this subject. I don't have we'll have any shortage of subjects to discuss next time.

Mat, thanks for coming back on the show. It's been really fun talking.

[00:56:30] MK: Thanks so much. This was awesome.

[END OF INTERVIEW]

[00:56:35] JM: Failure is unpredictable. You don't know when your system will break, but you know it 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, blackhole 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. Checkout Gremlin and get a free demo by going to gremlin.com/sedaily. That's gremlin.com/sedaily to get your free demo of how Gremlin can help you prepare with resilience as a service.

[END]