EPISODE 1374

[INTRODUCTION]

[00:00:00] KP: It does not matter. And I really cannot stress this enough, it does not matter. If it runs on your machine, your code must run in the production environment. And it must do so performantly. For that, you need tooling to better understand your application's behavior under different circumstances. In the earliest days of software development, all we had were logs, which are still around and incredibly useful. Nowadays, you're likely to also consider an application performance monitoring tool, or APM. Observability is an evolving and important feature of any software system.

In this episode, I interview Christine Yen, cofounder and CEO of Honeycomb.io. We talk about her start in software, meeting her cofounder while working at Parse. And how some of the experiences shared their shape their vision for an idealized tool for figuring out what's happening with your system. Christine, welcome to Software Engineering Daily.

[00:01:04] CY: Hello, thank you for having me.

[00:01:06] KP: Well, I definitely want to get into Honeycomb and your work there. But I thought an interesting place to start this out would be to ask you about your journey and how you got started as a software engineer.

[00:01:16] CY: I got started when, I think, I broke my ankle in seventh grade. I was stuck at home when the entire class got to go on some like Great America theme park day. And I was so upset, and I spent that day poking around on my computer, or on the family computer, and kind of got pulled into creating a website. And I grew up in the Bay Area. And so I had the chance to go to a couple, "Oh, learn basic camps as a kid." But it was this exploring building websites as a kind of preteen that really opened my eyes to what is possible when you don't have teachers are assignments, but it's just you poking around on the computer seeing what you can build.

And in the early 2000s, there was a – Late 90s, early 2000s, there was quite a scene of all these personal websites, where young webmasters would have these sections that would teach

each other, “Oh, this is what cascading style sheets are. And this is PHP you can include if you paste it into this file and name it this way.” And it was really cool even then to be part of this community of random people on the Internet just teaching each other things and kind of showing off, but really showing off and being most helpful to other people.

That led to more website stuff and a little bit more PHP stuff. And I ended up getting involved in student run school website effort in high school where we, at peak, something like 80% of the students were signed up for it. And I actually remember debating with my friends I was working on it with, “Can we put profile photos up? Can we trust high school students to put profile photos up?” “No, we can't. They’d upload something ridiculous.” This is obviously before Facebook. And from there, I think I just knew I wanted to build software that I was addicted to that conversion ratio of the perceived effort of an individual putting lines of text into a computer and the number of people that it could impact.

[00:03:38] KP: And what's the first coding language you get serious about?

[00:03:41] CY: I will not say PHP, because I can't say I was terribly serious about it. It was just a means to an end. I really got into Ruby, I'll say, in like the later couple years of college and right when I came out. I was very much caught up in the, again, the friendliness of that ecosystem and the great learn on your own materials that were out at the time. And I liked the expressiveness and friendliness of the language.

[00:04:08] KP: And at what point did you cross into being a professional developer?

[00:04:12] CY: I guess internships in college kind of count as professional. It's an asterisk internship. I went to college for CS and had the privilege of being able to learn alongside my classwork from the internships what it's actually like to – What a code review really feels like, and what it is like to tinker on a huge existing codebase and just work down a list of bugs to fix. And I think that was – In a sense, that was almost more fun, right? It turned software from this thing that you build on your own out of lines in a text editor into a puzzle, someone else's puzzle that you get to solve and you get to make sense of and improve.

[00:04:54] KP: Well, let's fast forward a little bit and introduce Honeycomb. Tell me a little bit about the founding and why you started the company.

[00:05:02] CY: Honeycomb really got started – Like I met my cofounder when we worked together at a company called Parse. I joined Parse in 2012. I think she joined six, nine months before I did. And she was on the infrastructure and op side. This Charity Majors, for anyone listening who may know her. And I was very much on the product development side. So if you imagine the engineering team was literally laid out in the office from like backend to frontend. She was on one end of the spectrum. And I like sat next to our designer on the other end. And by spectrum, I mean, room.

And we didn't interacted a ton, honestly, at Parse professionally. We interacted socially. But the only times we interacted professionally, usually what was happening is I had released something, and something in production was breaking. And so Ops was literally and figuratively knocking on my desk being like, “Hey. Hey, help me fix this. What did you do?” This is part of the nature of the work and part of the nature of the platform that we’re building. There was just so much heterogeneity, and chaos, and kind of unexpected behavior of our users hitting our API in unexpected ways, that this level of unpredictability was becoming our norm. But what came out of it was this feeling of, “Oh, man, there's so much happening out there.” And at the time, we only had kind of the barest sliver of a view into what is impacting our customers. Why is this customer having a fine experience? What is that customer messaging us about how Parse is down? We learned a lot from having to make sense of the chaos in order to ensure our customers were having a better experience.

I said I joined in 2012. We were acquired by Facebook in 2013. Over the next couple of years, as we were slowly exposed, and in some cases very grudgingly exposed to Facebook's internal tools, many of them were not suited for a startup running a platform as a service. One of them threw out a number of the assumptions that we'd had around limitations of a logging or a monitoring tool because it was neither. It was neither aligned with the traditional metrics monitoring approach, nor the traditional grepping through logs approach, and made a set of tradeoffs that allowed us finally a couple years into the Parse journey to, within minutes, pinpoint, “Hey, why is this one customer complaining that things look down for them?” this one customer of tens of thousands, hundreds of thousands, millions.

And so when the two of us respectively were thinking about leaving, again, I'd left about six to nine months before she did, we both had this feeling of, “Well, it will be sad to leave this internal tool behind, because there's nothing quite like it out there.” And truth be told, shortly before I left, one of our – We tried to build this experience for Parse customers, actually through the Parse analytics product. And it just wasn't quite the same. And so I think we both had this feeling of, “Well, we've had this experience with this tool that allows you to track down unknown unknowns, that has this flexible schema. That believes that a fast answer that's mostly right is better than one that's 100% correct. We've had this experience with this tool, the rest of the world should also.”

And I mentioned I left for Charity. I was kind of exploring and taking my time feeling a little bit burned out. And when I was ready to start thinking about what was coming next, Charity tends to be the nexus of all things interesting. And so I messaged her being like, “Hey, do anyone working on something interesting back in San Francisco?” I am sort of poking my head around. And she jumped on me and she's like, “Hey, I'm actually talking to someone thinking about starting a company. That tool is called scuba. So I'd like to build Scuba as a service. Do you want to build this with me?” Again, at the time, the last several years I'd spent looking up to Charity as this badass infrastructure Ops engineer, and I had this feeling of like, “Really? Me? Are you sure?”

But when you step back, our skill sets were sort of perfectly complementary. I knew that with her we could handle scale, we could handle – We could make sure that we would be able to build this massively multitenant thing that would be in the critical path of a number of our customers. On my side, I knew that we would be able to build something that was usable, and friendly, and felt responsive, and would really marry the experience that we wanted to provide with sort of the technical housing of it.

And so, I often say if it had been her with any other idea, or someone else trying to work on this idea who wasn't her with me, I don't think it would have worked out. I don't think it would have come nearly as far. But it has been – And this whole journey has been recognizing that we represent two ends of a spectrum. Dev and Ops. And that in many things, including who the product is for, the right answer is somewhere between the two of us.

[00:10:09] KP: Well, at that time, it was very early on in the space of observability. I think the market shares continued to grow. I don't know if you would say you were first to market, but certainly early days. What were you looking to accomplish and deliver that was cheaper, faster, better than the other available options?

[00:10:26] CY: Well, when we looked at the available options out there, you had logging tools, you had monitoring tools, and you had APM. And when you think about why there are the three buckets, logging tools and monitoring tools are evolutions of the technology that was available to us when we first started writing software meant for another machine, right? You had grep, which then became distributed grep, which became distributed grep with a UI, which became a logging SaaS. And then you had counters, because you couldn't have both grep – You had to do either grep or counters back then. Your computers couldn't handle much more than that.

We had counters that became RDs or time series. And then fulltime series databases, and then time series sasses became these monitoring tools. And our point of view is that we very firmly did not want to build kind of the next iteration on either of these two product lines, because the two of them shouldn't have been – Weren't word separated for any good reason anymore, right? They were separated for good reason back then when we had to choose between grep and counters.

As of 2016, when we started out, we had SSDs. And we had the cloud. And we had approximation algorithms. And we had lambda. And we had all of these things, and we call them stores. And all these things that were kind of available building blocks now to build an experience where you could have the flexibility of logging tools while having the speed that people associated with time series. And sidebar, APM emerged as a way to kind of bridge the gap between logs, tools and monitoring tools, and then it became their own thing. And that's why they're their own bucket.

But we're looking at these three buckets and we're like, "They shouldn't have to be these three buckets." Each of them assumed a set of tradeoffs that aren't true anymore. And as a result, deliver a subpar experience, right? How many people do you know use a logging tool on their own? No. Everyone supplements it with an APM or a monitoring tool. How many people are using a monitoring tool on their own? They've got logged somewhere. And that sucks that relying on human joins, to go from one tool to another tool sucks.

And so when we started out, we had the benefit of having a very clear idea the experience that we wanted to provide because we'd interact with this internal tool. And honestly, the biggest challenge in the first year or so was not as much building the product or the technology, but instead introducing the term observability, which some companies have been using internally, but not really in this context.

And Charity spent, I'm pretty sure, 80% of our time the first couple years just going to conferences, writing blog posts, trying to wake people up from trying to just claim that observability is a new fancy word for monitoring. When instead, it expects a different set of human processes and cultures and how you interact with the data.

[00:13:30] KP: Oh, that's a great point. It's not just the tool. It's the operations as well. Do you see any best practices? Are there ways that organizations should adopt and leverage solutions like yours?

[00:13:41] CY: I'm going to answer a slightly different question and probably get to yours, which is what are – There are many signs that you have shaped your processes to a tool that is not meant for that use. A good example, many people talk about alert fatigue or dashboard blindness. This is where you just have too many alerts or too many dashboards and you can't find the right answer anymore.

And to keep our own Charity’s words, this is because in a world of monitoring tools and time series where you have to define ahead of time what matters, what results is building up your ability to debug based on known unknowns, right? “Oh, this thing went wrong that one time.” “Well, we better build a dashboard around it so we can find it again.”

Well, chances are, if your system is sufficiently complex or chaotic, that same thing isn't going to happen again. And building that dashboard isn't actually going to help you in the future. And so when folks talk about alert fatigue or dashboard blindness, we’re like, “Hey, check out this flow, where instead you can define instead of 100 alerts or 100 things that might go wrong that you want your team to be aware of, figure out the small handful of customer impacting signals and explore from there." Figure out, "Does this actually matter? Is it impacting customers? Or can I go back to bed?" Have that be automated. Enable your humans to slice, and dice, and explore, and start from that signal and instead, follow their nose, follow the data, rather than scrolling through dashboards where you're just scrolling through artifacts of past failures.

I can rattle off other best practices. But the theme here is the folks who wrote the software really taking part in real ownership of their software once it's in production, and tapping into the natural curiosity and inquisitive nature of engineers to explore and tease apart and reduce search space to find the subset of your requests, say, that are erroring or that are misbehaving. That's your task as an engineering team, as a top performing engineering team. Our task as a tool provider, build the technology to enable that is to just be fast enough and provide enough kind of clues and expose enough kind of interesting entry points for you, the human, to use your excellent judgment and figure out what in the world is happening with your system, right?

This is my beef with traditional APM. Traditional APM really rose at a time when enough software architectures were similar enough, right? You take 100 Rails apps, there's going to be enough similarities in them where you can say, "Hey, if your Rails app is doing this, you probably want to do that to resolve it." We're not in that world anymore. So that promise of an APM sort of being ready to go out of the box and giving you answers and not requiring you to customize or to explore or to stick your hands into a graph and take it apart. That world is in the past, and we need tools now that can flex with the complexity we've introduced into our software worlds.

[00:16:56] KP: Well, if we think about a company or a small team, maybe large team that decided to adopt Honeycomb, what's the onboarding process like?

[00:17:04] CY: It depends. Today, if you are using OpenTelemetry already. For anyone who isn't familiar, OpenTelemetry is an open standard that is contributed to by all the major vendors in the space so that you can instrument once and then send that instrumentation to whichever tool you choose. So if you happen to be using OpenTelemetry already, it's as simple as pointing that data at Honeycomb.

For other folks, if you're starting from, say, a kind of Greenfield. You've got this new service or application. You don't have it instrumented. Similarly, OpenTelemetry is a great place to start. Every vendor also has their own auto instrumentation library that you can drop in, get some basic telemetry to begin with.

I talked about the complexity and increasing heterogeneity of software. Ultimately, if you're running a web service, you have HTTP requests, and we can extract some kind of basic metadata from there. But I think of that auto instrumentation output as the baseline. That's where you start so that you can cover your basic bases.

But the next interesting step is engineering teams looking inward and saying, “Okay, what actually matters to our business?” What are the identifiers that, for example, I, as a developer would put into my tests to make sure the business logic works right? Say you work for an ecommerce site. Those pieces of metadata, probably things like user ID, SKU, shopping cart ID, price, things that will allow you to really understand, again, “Does this thing – I see this weird thing over here in a graph?” Is it actually weird? Is it impacting customers? Or is it just some artifact that like lower urgency to figure out?

[00:18:50] KP: Do you find that when a new team comes on board that there are any low-hanging fruit that they find just by adopting and starting to look?

[00:18:58] CY: Almost always, right? I think that this is true of any tool in the space. When you go from not being able to see what's happening in production to, “Oh, I've got some graphs. I've got some breakdown of errors or latency by endpoint. There's almost always some surprise, right? Either there's an expensive endpoint that is called more than you thought, or something that you thought would be expensive and heavy, but it's actually not interacted much by your customers. That first day, there's almost some, “Oh, interesting. Let me go adjust that. Let me go dig into that.”

The key is making sure that it's not just one and done. Software systems are constantly evolving. That's what makes it so interesting for the engineers who build it where we’re constantly adding new functionality, refactoring, taking things out, improving things. Asterisk, sometimes it's not always improving. And so, again, what you want beyond that first day, dopamine hit of learning something new, is making sure that observability becomes as much a part of shipping software, developing and shipping software as writing documentation or writing tests, right? Instrumentation is just commenting your code, but in production. It's how you make sense of what's happening and whether your expectations of what should happen are still lining up with reality.

[00:20:28] KP: And does observability end up changing the coding style of the teams? Or are you really looking to have an impact elsewhere in how they perform?

[00:20:36] CY: I think it does. I'm going to tell a story of some of our long term customers and friends over at Geckoboard. I think of them as using observability in an advanced way, where not only are they thinking about it using it to deal with unexpected issues in production. They will actually look at what's happening in production today to figure out how to – The right approach to take when writing their code or using that to validate an approach very early on.

An example I tend to give is a year or two, they were working on some new feature that boiled down to the bin packing problem behind the hood. And their engineering team was like, “Well, we could spend a week trying to debate about the ideal implementation of this MP complete problem and hope that it performs well in production. Or we could whip up three example implementations of solving this problem, put them into production behind a feature flag so customers don't actually see the kind of impact of these choices, but production traffic, or that the algorithms are each tested against production traffic. And then we can capture those results. And after a day, pick which one we go forward with and just move on with our lives.

And I love this example, because it's kind of a change in how you're writing code. It's almost more of a change of how you form your mental model of what reality is and what your customers want and what your system is going to do. And it is the way to ensure that your mental model is grounded in reality, because no matter how good of an engineer you are, the more complex your system is, the more likely your mental model is going to be flawed, or the more likely it's going to be out of date.

So if you’re doing what Geckoboard does, by pulling production into that early stage of the development process, you're not only shipping better code in the first place. You're empowering your engineers to have autonomy, make decisions and move quickly, which are all key. I don’t know. Those are all the things that I was looking for that were key to my happiness in my career.

[00:22:56] KP: Well, when you have a client like that, who's at a mature stage with the product, they've had the early wins and taking the low-hanging fruit. And now they've evolved into kind of an ongoing process, can you describe the user experience if I'm a developer in that setting? And what kind of role does Honeycomb play in my day or my weekly process?

[00:23:14] CY: Anytime we write software, we have a set of assumptions of what use case are we solving? What is the kind of common case? What are the edge cases that do or do not need to be incorporated? And I think the ideal case we're building for is one where engineers spend half their time in their IDE and half their time looking at something like Honeycomb, asking questions that allow their time in their IDE to be more productive and more effective than they would otherwise.

I can't speak to precisely how often they're using us or the frequency. But something that Charlie and I have always said is that if people only open up Honeycomb when something's going wrong, we will have failed. Because we know that the value here is in really ingraining observability and that sense of what's happening out there in the wild into the everyday development process.

And other customers, we'll hear stories of product managers putting Honeycomb graphs up in an all hands show progress of something, adoption, or performance or scale. Like that's awesome, right? Everyone should ultimately be sharing the same language and view into what's actually happening in production that’s killer.

[00:24:36] KP: Can we dive into the user experience? I know this is probably better served with screenshots and that sort of thing. But do I work in SQL? How do I query those graphs you mentioned? What’s the nature of the tool?

[00:24:48] CY: The main area where people spend their time, there's a query builder at the top and graphs and tables underneath that show the results of your query. And this is something that people have asked us for years, right? “Hey, Honeycomb, why don't you just – I'm an advanced user. Why don't you just open up a giant text box at the top of your screen so that I can type in SQL and you can do your thing? And I have a flip answer to that, and a technical, and a kind of personal technical bugaboo answer to that. The flip answer there is when you're woken up at 3am, especially if you're a new engineer on the team, I don't want to be messing with SQL. I want the interface to be as helpful and as kind of idiot proof as possible.

So we really pride ourselves in building as intuitive and experienced as we can. We're never done. We always have more to improve, but really making it something that you can repoint and click and try to navigate around to ask the question that you want to get the data back that you want. And we've gone a little bit of a tangent here before I get to that technical bugaboo answer.

Another thing that is key to our product experience that has been there since day one, and again, lots of room to improve, but I think is pretty special, is we try to recognize that no engineer exists in a vacuum. Most engineers don't exist in a vacuum. We all have teams that we share on call rotations with, that each of us are different domain experts in different parts of the system. On the right hand of the screen is – You can call it an activity feed. I don't like that. It makes it sound too consumery. But there's essentially a record of not only every query that you have run, we can go back and look at the results of that query no matter how old it is. Our permalinks are permanent. That's what they mean.

There's also a kind of history of everything any of your teammates have run on the dataset. And so the intent here is, again, if you're woken up at 3am and you're sort of disoriented and you're exhausted and brain isn't totally working, you should be able to go back and be like, "Okay. Well, crap. Well, Charity was on call last week. And I think she said something about MySQL errors. And I'm seeing MySQL errors. So let me go back and see if I can find the queries that she ran last week when she was on call and jump off from there. And rerun, and tweak, and iterate."

We don't want people to have to be geniuses at observability to use our tool. We don't want them to have to be geniuses at SQL. We don't want them to be geniuses of any sort really. We want them to be able to point and click and borrow other people's work, because that's how people work together naturally, right? Think back to when we were all in an office together. You had that feeling of looking over each other's shoulders being like, "How did you solve that?" "Oh, okay. I'm going to screenshot that or I'm going to write this down in my notes so I can improve on that." So that kind of acknowledgement of the team as a collective is also a key part of our user experience.

The technical bugaboo answer to why we don't have a SQL box, for anyone who's curious, I think the experience of every vendor defining their own SQL variant that looks SQL-ish, but then causes you to stub your toe when you write something that should work in SQL, but doesn't in their dialect, is awful. And is a painful, painful experience that seems like it should be more productive and always ends up more frustrating than otherwise. And never say never. So maybe there will be a future where Honeycomb or something like that. But I will fight it kicking and screaming.

[00:28:36] KP: When I think about Honeycomb. I'm seeing two groupings of use cases in my own mind. First being incident response. Something's happened in production, and I just need information. Second, being more of a forward-looking process where I'm exploring possible optimizations, learning about my application, things like that. Do you have a sense if that's fully covers use cases? And if so, what kind of breakdown you see across incident response versus being proactive?

[00:29:02] CY: Honestly, what you've defined are not just two use cases, but you've defined two ends of the spectrum. When you think about instant response, or performance optimization, right? "Hey, I've got an endpoint in mind. And I need to figure out how to make this endpoint less painful." All the way to the Geckoboard example of, "Hey, I'm trying to get ahead of myself and try to predict the future." Those are all realistically the same sort of emotions involved. You're exploring. You're starting at something high level. You're teasing apart. You're trying to understand and isolate certain behaviors or certain artifacts. The thing that changes as you go along that spectrum is urgency. It's severity. It's that time pressure. It's how stress is the operator?

And I think recognizing that enables us to work with customers and say, "Hey, that thing that you're doing when everything's falling apart, let's talk about how to do that a little more proactively," which will have the added benefit of ideally causing fewer things to fall apart. And it's, again, when we came into this, when Charity and I first started working together, we still sort of had this idea that like, “Okay, well Ops people do that thing over there. And devs do this over here. And they're just different skill sets.” They're not. They're not, right?

Incident response and this forward-looking kind of – We’ll call it performance optimization, but proactive work. They are same motions just with different – Often with different nouns, with different concepts, right? Instead of CPU, and memory, and disk space on the Ops side, it's user ID and shopping cart ID on the Dev side. Why do they – Like the separation that is coming out of my words doesn't have to exist in the tool, doesn't have to exist in our teams. There's so much opportunity for developers to get comfortable in these “traditional production tools” just by bringing some of their nouns that they're familiar with into their traditional view of production.

We first started talking about observability. We spent a lot of time talking about high-cardinality data and educating various conference audiences about what high-cardinality data is and why it matters. Because as we build this more complex world, when there are these nouns, again, to overuse this word, when there are these nouns that have potentially millions of possible values, like container ID, or Kubernetes pod ID, or user ID, or shopping cart ID, the support for being able to reason about this more complex world as quickly as possible is what's necessary to make sense of our own production software and be able to do that kind of investigative approach, either in the incident response workflow or proactive optimization workflow.

[00:32:02] KP: Well, I think about times when I was really doing a lot of development, and I was heavily focused on optimizing for something that mattered to me, getting out features. Maybe reducing a response time. I wasn't always thinking about stuff like the CPU load, or the memory pressure, things that someone in a more traditional DevOps position might have top of mind. Is there a story for collaboration in Honeycomb? How do people with these different concerns work together?

[00:32:30] CY: I think one of the key learnings that we as an industry are sort of have been wrapping our heads around over the last couple years is that the – We talked about DevOps where Dev and Ops blend. But even you just described DevOps as like, “Oh, yeah, Ops – CPU and memory pressure and these things. And yet, we know that if the Lyfts and Ubers of the world are going to go on stage bragging about how they have 100 services per engineer, man, you can't have an Ops team supporting that or a DevOps team supporting that. Your devs have to be aware of and be a little bit more involved in what it takes to support those 100 services.

And so I think that both Dev and Ops are having to move towards each other. Devs in being more familiar with what's going on with this Kubernetes deployment? And what is the impact of my architecting the services in this way? And Ops, in recognizing that they can't just always treat services as a blackbox anymore, that they have to get some visibility into that business logic layer to understand, again, whether something is user impacting, whether it's now. How big of a deal it is?

I know, I sound a little bit like a broken record here. But so much of this is sharing that same language to talk about production, to recognizing where and how Dev and Ops areas of kind of ownership and comfort, how they overlap, and how to increase that area so that DevOps or Ops folks can tease apart, “Okay, what is the business impact of this? How do I go figure out which engineers to bug about this?” So that the engineers can quickly then take symptoms of some production incident and translate that into their test cases, repro it and fix the issue.

[00:34:37] KP: Well, I'm thinking about some of the organizations I've worked with who weren't always at the forefront of investing in observability. Sometimes there's a perspective of, “Look, we have logs, we have traces, we have some metrics. Everybody has access to them. Why do we need to invest in tooling?” What are your thoughts to someone who's a little bit hesitant and curmudgeonly in that regard?

[00:34:59] CY: Chalk this up to am I being too much of an engineer? Not enough of a CEO business type. But I think that I'm not in the business of yanking people away from what they want to use. If they are genuinely happy with their logs, or metrics, or traces, or combination of those tools – Sorry. Logs, or metrics, or APM, tools, combination of those, great, we'll be here when they start to fall apart. We'll be here when they start to creek.

And it's not a knock on them. It's not a knock on their systems. It’s just its effect, right? When you have one of each tool, one of each of those old buckets, what you're doing is you're taking a moment and something that happened in your system. So, a moment in time. You're slicing it up into three different ways. You're sending it into three different tools. And then you're trying to, probably, as a human, piece together, what actually happened by piecing those like three data fragments together. And that's really hard. And it works for us sometimes. And when you start to go into this world of we used to have one monolithic app on five AP servers. And you can kind of reason about that. And you kind of do the thing where you're like, “Okay, well, in aggregate, I see this my monitoring tool, and I can look at the individual log lines and the logging tool, and I can kind of figure out what's happening.

Now when you have 50 microservices across 500 containers and cycled through however many Kubernetes pod names each day or each week, you can't do that anymore. And I think it's frankly inevitable for other technology trends that we're seeing today to drive changes in the sorts of tools that we use. There will be cracks. There will be frustration. I've never met an engineer, even with Honeycomb, right? I've never met an engineer who's like, “Yes, my observability, or monitoring, or logging tools are perfect. And I don't want anything to be improved about them.” There's always something. And there will be – To them, I just say your time will come. I'd rather talk to the people who were frustrated.

My favorite thing to do back in the day when we had tradeshows and conferences, I would love to be at booth and someone would walk up to me and be like, “Hey, what does Honeycomb do?” And I’d say, “Hey. Okay. Have you ever looked at a graph about something happening in production and seeing something funny and scratch your head and been like, “What is that? And why is that happening? Why is that spike here?” Have you ever wanted to stick your hands into that graph and kind of rip it apart to see what's underneath?” And I would always get a, “Yes, I've been there.” And I get to say, “Great. You can do that with Honeycomb.” And it takes some time. It takes some like battle scars to get to that point where you're like, “I've been there. It's frustrating. And I want more for my tools.” But I think that we as an industry are getting there. So I can be patient in that.

[00:38:00] KP: Is there or do you envision adding any sort of machine learning component?

[00:38:05] CY: I think machine learning is – Especially sitting well within the startup hype machine. I think machine learning is a loaded term that means many different things to many different people. I think that there is definitely a lot that can be done in thoughtfully identifying areas where machine analysis and kind of applying basic statistics and kind of surfacing outliers or interesting trends is possible.

I am very leery of – As a user, I'm leery of tools that liberally sprinkle AI and ML all over their product pages. Our approach is, again, grounded in the belief that you, as an engineer and engineering team, know your software best. You know what's interesting. You have the context of, "Oh, that was load test." Versus, "Oh, this is something that's very strange. I shouldn't look into it."

And what we would rather do with a tool is supercharge these instincts, supercharge your eyebrow. Like if you want to dig into something, you should be able to dig into it, and we should be able to help. The analogy that our Principal Developer Advocate, Lis Fong-Jones, uses is that we want to be your Mecca suit. We're not going to try to build a droid to do your job better. But we want to give you superpowers.

[00:39:33] KP: Yeah, yeah. I like that analogy. Well, I'm also curious if there're any KPIs you're aware of that your customers use once they've on-boarded? Are they trying to reduce errors per month or cycle time? What do organizations generally want to improve upon when they look to Honeycomb?

[00:39:51] CY: I talked about baselines. I think that the baseline KPIs for anyone who adopts a new APM tool, there are some that should come out of the box, error rate, latency kind of throughput. These are good places to start just to answer the questions of what's my system doing.

The next level approach that I'd like to see customers take is really explore service level objectives. Service level objectives or SLOs, frankly, could be their own hour on Software Engineering Daily. But the upshot of it is if there is an indicator that kind of you, as an engineering team, hold yourselves to that determines whether you're providing a good service. For us, it is something like API latency on our ingest endpoint. It should be very low, and it should be consistently available. And so we find a way to kind of wrap that English in a definition of a metric, just a thing that is calculated over our data. And we attach a threshold to it, right? We say, "Hey, 99.9% of the time, we want to be satisfying this service level indicator." And that allows engineering teams to actually have a conversation about what are the, again, small handful of standards that we hold ourselves to as an engineering team, right? What is our contract between ourselves and our customer? What actually matters? So that, for us, it's – The ingest, it's making sure our homepage, the first page that you see when you log in, loads quickly and doesn't throw terrible errors. These are more nuanced than just error counts or latency. They are things that are customized to us, to things that matter to us, because we are able to feed in some of that metadata around this particular status code or this particular error message. You can ignore that everything else is signal full and should be considered.

And honestly, the best part of these SLOs is that they tend to be the result of really meaningful conversations between the humans on your team, right? Again, this is the, “You know software best.” We just want to augment that philosophy and practice.

[00:42:11] KP: If there's a team, maybe they're already on OpenTelemetry, and they'd like to kick the tires, run a demo or jump in headfirst, how can somebody get started with Honeycomb?

[00:42:20] CY: You can just sign up for the product. We have quite a generous free tier where you can – It's forever free. No timer. The kind of threshold between free and paid is just whether you send us enough data to merit jumping up to that next level. So if you're curious, check us out. I think that we are quite different. We're going to be quite different from many tools that you've used. And it's worth taking us for a whirl.

[00:42:44] KP: And what's the best URL to check it out?

[00:42:46] CY: You can sign up honeycomb.io. Or we have some public playgrounds available play.honeycomb.io, where you can just play around with the tool. And there's a little bit of a guided tour so you can solve an issue and the example data set. But honeycomb.io is where you can sign up for an account of your own.

[00:43:05] KP: Well, Christine, thank you so much for coming on Software Engineering Daily.

[00:43:08] CY: Thank you so much for having me.

[END]