Architecting the Cloud, part of the On Cloud Podcast

Mike Kavis, Managing Director, Deloitte Consulting LLP

Title: Utilize observability for more effective DevOps
Duration: 0:24:05

Operator:
The views, thoughts, and opinions expressed by speakers or guests on this podcast belong solely to them and do not necessarily reflect those of the hosts, the moderators, or Deloitte.

Welcome to Architecting the Cloud, part of the On Cloud Podcast, where we get real about Cloud Technology, what works, what doesn't and why. Now here is your host Mike Kavis.

Mike Kavis:
Welcome to the Deloitte's "Architecting the Cloud" podcast, I'm your Host Mike Kavis, and I'm here with Honeycomb's CEO and Co-Founder Charity Majors. Charity, welcome to the show, tell us a little bit about your background and about Honeycomb's.

Charity Majors:
Hey, yes, thanks, yes thanks for having me, I am a (Inaudible 0:00:18) Infrastructure Engineer who accidentally became a CEO of a company based on a lot of trauma honestly – trauma trying to debug and understand modern distributed systems using the tools that we developed over the past 20 years for systems that were not so distributed.

Mike Kavis:
Yes, it's great to have you on this show I know a lot of companies are founded based on pain, right? People experience pain and there is no solution.

Charity Majors:
Terrifying, yes.

Mike Kavis:
Yes, so I'm sure you have a similar story.

Charity Majors:
Yes, absolutely. Yes I joined Parse, the mobile backend as a service, as their very first Infrastructure Engineer, which is my sweet spot right? It's really great. I'd love to say it's right at the nexus of where computing meets reality. It is how I think of operations, and I really like to pattern as building the infrastructure, then building the team to replace me. So, I did that for Parse, and around the time that we got acquired by Facebook, I was coming to the horrifying realization that we had built a system with some of the best engineers in the world that was basically un-debuggable. It was not comprehensible in any reasonable length of time.

You know this isn't right, it was like, Parse is down, sometimes a few times a day, and I'd just be like, Parse is not down, like, look at my wall, my whole dashboard, like they're all green, like everything is fine because maybe one user will do like four requests per second mobile traffic is huge, in other words doing 20 requests per second, and I'm doing a 100,000 requests per second in overall. So, their problems will just never even show up in my radar, and the process is, like, I tried every tool out there, and the one thing that finally helped us get a handle all our problems was this obscure, neglected little tool at Facebook, but did one thing and one thing well, which was that let you slice and dice on dimensions of arbitrarily high cardinality in basically real time, and that's like you applied that to our problems. We built some scaffolding, and we dropped the amount of time it took to understand (Inaudible at 2:33) systems from a day, an unbounded amount of time, where it's an engineering problem every time, down to a few seconds, or maybe a minute, something that was not an engineering problem, but it was a support problem, and that made a huge impact on me so much so when I was leaving Facebook I was just – I suddenly realized I could not go back to engineering without it. You know it wasn't even just a thing I needed during outages, or would need to like solve problems. Now it's becoming so important to how they understand and experience the world it's like glasses for driving right? I could not imagine driving without my glasses, and I feel that way about using this kind of tooling for these large distributed systems because it's like the idea of going back to just not knowing what's ever happening and that's just unacceptable.

Mike Kavis:
Yes, that's a cool story – I remember my start-up days. There is no one tool, right; it's always a collection tools and…

Charity Majors:
Yes.

Mike Kavis:
We had it like six or seven different tools, you know we had the ones for Infrastructure we had the EPM tools, we had one for the database and one day the login for the app didn't work in the customer calls and says don't you guys monitor? I just shook my head, well next you know, we are doing syndicated users, and all their stuff there was…It was just never an end. We always, it was always another gap and always another tool that, you know, Ops?

Charity Majors:
Well that's the approach of chasing the problems right? Every time there is an outage you're like uh, well I'm going to create dashboard, so I could find this immediately the next time, monitoring, check for this case so that I can find immediately the next time, and the thing is that this worked really, really well for a long time because we had a more or less known set of ways that the systems would break. The problem is that nowadays we – it's more like, we have this infinitely long thin tail of things that almost never happen, except that one time they do, or it takes like five rare or impossible things to intersect at once before it sets off this bug right? And so, if you spend all

your time just chasing after them and playing catch-up and building monitoring and checks and dashboards, for these very specific problems, each of which is probably never going to happen again. you kind of lost the battle before you even start right? We need a kind of take a step back and invest in kind of tooling that lets you ask these questions and get back answers like very, very, it's almost like a state-of-flow, right, you're debugging you're understanding you're not monitoring for known. It's different between known unknowns you could monitor for and unknown unknowns you just have to prepare for.

Mike Kavis:
And the term, I don't know if you coined it, but since I've been studying the term "observability" always seems to tie back to you, but talk about what is "observability" and why is that so important?

Charity Majors:
Yes totally, I definitely did not create the term "observability," but I borrowed it, because when we started Honeycomb everyone was telling us that is a (Inaudible at 05:32)-problem, no more in the space. Everything is already fixable, and we knew and that this wasn't true because of our experience, but every time you try to talk about that, it was so much harder to figure how to talk about the problems and solutions than was to build them honestly. And I was just going round and round looking for words that would differentiate between what I had seen/what I had experienced, and observability is actually a term that's borrowed from control theory right? People have been using it as a synonym for telemetry for a long time (Inaudible 06:00), but when we started looking on I borrowed that very specific technical reason from control theory, and what it means in that mathematical branch, whatever it means.

How well can you tell what's happening inside the system, just by observing its outputs, and I read that and I went holy shit this is what we're talking about. This is the experience that I've had right? Like you can understand (Inaudible at 06:20) and run (Inaudible at 06:23) and your binaries all that stuff, but that's not what we need to be doing, we need to have tools that are gathering the right level of detail, but they're like composable bits right, so that you can ask any new question of your systems, when we get into these weird unpredictable states, without having to (Inaudible at 06:40) new code right?

So it's like gathering the right level of detail or information, and then also do the tooling that lets you not have to login, not have to just find – little bit but just answer any question and when I'm giving tags I can pull up a list it's like some of the problems that we had at Parse and Instagram like with this one router in Eastern Europe, it wouldn't fail over from UDP to TCP. It only added capacity to this round-robin DNS record, it seemed like pushed it out for all Eastern Europe like what are you going to monitor for? You can't, you really just have to like embrace the fact that your designing system for resiliency – your designing systems and you know that your systems are never fully up, right? They're always existing in the state of partial-degradation and as long as it's not impacting our users that's okay, right. So I would define observability very much as being able to ask any question of your systems to understand any edge case to understand anyway they can degrade or fail, without having to ship new code without having to write anything customized for that scenario.

Mike Kavis:
So the next question I ask you was probably a term that's going to make you cringe the NoOps term alright, yes so but what NoOps really means to me I call it less ops, and it doesn't mean there is less to do. What's happening is people are moving up the stack and they're going to.

Charity Majors:
Moving up to stack.

Mike Kavis:
More abstract and they're going to PaaS, they're going to serverless, so you have less operations with physical infrastructure which we always have operations so now serverless is a big buzz…

Charity Majors:
Yes.

Mike Kavis:
How does observability help when you no longer worry about infrastructure, but now you're managing functions and higher-level abstractions?

Charity Majors:
Totally, so I like to pointing to Ops. You might think that I have a chip on my shoulder and you'd be right. Like, I define operations as

basically the comfort of your organization's practices and habits around shipping software safely to users right? And when you define it this way it becomes clear that everyone cares about Ops and everyone has to participate in it right. It's a shared culture and it means that we care about quality, and we care about our users' experience, and I think there is no doubt that the shape of operations is changing radically. it's becoming less and less optional. Every software engineer now has to care about what happens when their code hits users, hits real infrastructure, hits real like user patterns. That's just not optional anymore. For a long time, we pretended like it was; we pretended like you could just write your code, like ship it over the wall. Ops would deal with the consequences, but what we realized was those consequences are terrible right?

It's bad for users. It's best when the person who writes the code has the ability to ship the code, and you can then look at it and see is it doing what I thought it would? Did I write what I thought I wrote right? Does anything else look weird? Because that's where you catch like 80%, 90% of all problems before users ever see them. So, I don't think of it as being less ops. Honestly I think of it as being more ops, but it's just differently shaped; it's broader instead of deeper right? Instead of having a few people who do nothing but Ops, in fact Ops is now everyone's job (10:00) and if you're lucky enough to have people on your team who are labeled Ops Engineers, well they're probably going to be great expert consultants who can help you figure out how to hone your code really well.

So serverless, yes, the way that this plays out with serverless, I feel like is when people get too anxious to shed the word, or the role, of operations, they shed the most important job that they have which is, you are responsible for your up time. I don't care how many providers you are using, how many levels of abstractions, it doesn't really matter. That is your one job is to get your code to your users and that is operations in a nutshell. So, I wouldn't shift topic too much right? Serverless is amazing. It's so cheap to run your functions as a service. It scales really well because you don't have to deal with provisioning, but it is still operations because your users are still going to complain to you if your stuff doesn't work. So, what this means to me is engineers need to be proactive in learning more about things like reliability, things about how distributed systems feel, how they work, the level of risk that they can tolerate, the level of risk that they can't and how to (Inaudible at 11:14) for it.

On call is everyone's job. Two years ago when we started talking about this, there was a lot of pushback. People would go, ooh no we can't put software engineers on call. They'll quit. And now the conversation has really changed, to, you know, we accept that this is the only way to build software that's high-quality. Now how do we make this a job that doesn't suck, right? So, I think it's really a fascinating and interesting time to be working in software, especially if you're coming from the operations point of view. A lot of what we used to see that was technical is now human centered, right, it tells about (Inaudible at 11:49) servers – sorry to interrupt. It's much more about consulting and telling people, telling engineers how they can actually do this in a way that isn't painful.

Mike Kavis:
Yes. And there's couple of factors at play. I grew up in the mainframe days. I'm kind of aging myself. There's this black box, and I ran code and something happened. I had no clue and then we moved to client server, and they're still like three tiers and now infrastructures code, and I can really build some great distributed systems that use like from a database perspective, like how do you use this one database well now, because database is a service, to me, I can use three different database technologies on the same or different problems, so operations is very complex now right?

Charity Majors:
Yes.

Mike Kavis:
But hopefully the companies are doing all right. They can simplify the operational processes like throwing it over the fence, and filling out forms and tickets, but the actual, to observe a system is, the systems are getting quite complex distributed…

Charity Majors:
Very, very complex and your levels of visibility into them differ widely right? This is one of the reasons observability is, I think, so important so quickly, is that people are realizing that suddenly, they depend on systems that they can't see into, right? Or, almost more interestingly, they depend on systems that they can't look into as a third party observer, but they can see through the lens of their instrumentation.

And this to me is one of the fascinating divides between monitoring and observability, as well as between software engineers and operations engineers, because, as a software engineer you should not actually ever have to give a shit about monitoring, or capacity, or any those things, like monitoring, is like one party looking at another party and saying how's it going right are you up? Which is actually about right. The point is that the software engineer cares about it only from the perspective of your code? Can your request execute from end-to-end, right? If every event executes, successfully your job is done. Your job is not actually to pay attention to capacity management and planning and enforcement. Your job is to make sure that your code can execute.

That is the bright line that I see, and that's where monitoring is going to continue to be important, because it's very important to monitor your systems to see that you have capacity and see that the aggregate stats are all correct and (Inaudible at 14:12) databases are great for those things. They're terrible for anything that revolves around the concept of the event, right. And the perspective of the software engineer who cares about the experience of their code as it travels its way through your system, all the slots matching directly to the way to your user experience as a system, so it lines up that empathy very nicely. Does that make sense?

Mike Kavis:
Yes it does and question for you. Do you see a lot of companies using observability early in the lifecycle, before prod, to make sure that when they go to prod, they are not blowing things up or those types of things? So is it something, are there tools or technology that have been built into the pipeline to get a early read on the impacts of.. (Inaudible at 14:58)

Charity Majors:
So here is the thing, you don't want them to be different tools you want to use the same tool as you use in prod throughout the lifetime of your software right? Because that's how you…I could talk about this all day…you really don't want it to feel differently. You want people to be – to not be too afraid of doing things in production you want it to feel familiar. You want it to be a hard to do bad things, and easy to do good things. Like, you want to be using the same observability stuff all the way from the moment you write the code and check it in, until it is decommissioned and every part in between. You really want to maintain the same toolset all the way through that lifecycle.

And yes, people that I see like pushing this way earlier in the set are the experienced engineers. The ones who have been doing battle with prod for a long time are the ones who have done time in a big time in a big company like Google or Facebook, and they've seen how powerful it is to catch these things early. Ideally, you want to really be getting your code out as quickly as possible, right? While you've written it, while it's still…while you've just written it, while it's still really fresh in your head you want to get it out into prod, but you don't necessarily want it to be in front of users yet, right, which is why things like "feature flags" come in. There's a lot of stuff around how to test in production, safely without your users ever noticing, that can really help make the difference here, and observability is just…people don't realize how blind they are flying right? People don't realize that they're shipping code and crossing their fingers and just waiting for something to break when they could be looking directly at the impact of what happens when your code meets users in production, and it's a very…once you've experienced that sense of comfort and certainty, it's really hard to ever go back to flying blind.

Mike Kavis:
Yes, and I work a lot in large enterprises, mostly in the cloud computing space, and a lot of times the cloud guardrails, I'll call them. So, like say before I let developers put the guardrails on top, put in policies, and then a lot of times there is dependencies, legacy tooling, right, so I'm a developer, I'm writing code, and I have dependencies on stuff outside of my control, but if it breaks, breaks my app, right, so that's what a lot of people in large organizations…It doesn't matter how good or bad they write their code. They have dependencies and things out of their control. How do you deal with that?

Charity Majors:
Yes, well, I will say that companies in my experience systematically under invest in tooling around production, like whether that's deploy scripts, or like you said guardrails, when in fact this has probably been the most important code, or they put interns on it right? They treat it like it's not important when, in fact this is the code that probably matters the most, right? Your deploy scripts and your guardrails around running things in prod are going to touched many times a day by every single person in your organization, and yet we don't put our best engineers on it, and we don't allocate actual engineering time to them.

Often, it just kind of gets, it has to fit in the cracks of real work, right? And this is a lesson that I think that every engineering manager wish would really take to heart and start doing battle with their overlords. We really have to invest in these tools. There are so many best practices. There are huge (Inaudible at 18:36) and articles that are waiting to be written with the best practice that we will develop and start using, but I can't think of a single company that I've ever seen that actually spends enough time and energy prioritizing that stuff.

Mike Kavis:
Sure, and I think one of the reasons…this is something I push up to clients, is typically you have a product owner and their job in life, I'm talking about larger enterprises here, your job in life is to get the next feature out? They don't own the (Inaudible at 19:07).

Charity Majors:
It doesn't have any owner exactly. It has to have…

Mike Kavis:

So, I could be someone under the receiving end of lot of chaos and say hey, we need this tool now. We've got to get these five features. And one thing I push is that, you know, we need change goals, incentives. Everybody needs to own the (Inaudible at 19:20) right? The security team doesn't own security; everybody owns security.

Charity Majors:
Exactly.

Mike Kavis:
Product owners

Charity Majors:
Yes but…

Mike Kavis:
As a product they – if it's not secure that's their problem as well, right, because they're prioritized…that's just…

Charity Majors:
It does need to have an owner, but it is still actionable, though, right? And that owner can move around between teams, but it has to be an owner, and it has to accountability, right? And it has to be at the same level as shipping product features, because otherwise, what you end up with is that janky system that is held together with duct tape that cost an hour a day of engineering time just doing battle with it, and maybe an outage a week of just somebody doing the wrong thing because it wasn't designed…oh…and how designers work on this with you too. I cannot recommend that enough, right? When it could be something that eats up like 30 seconds a day, amortize that engineering time in pain and outages over time, and I think we can all see what a huge burden it is when we don't invest in it.

Mike Kavis:
Yes, so you talked a little bit about testing and production, and I've been seeing a lot of action in enterprises looking at chaos engineering, and it's funny, a lot of people will say, well we have chaos, we just don't have the engineers but, you had a good quote recently that says chaos engineering without observability is just chaos, so explain?

Charity Majors:
You have to close the loop, right? I know people who have started doing it for fun—chaos engineering stuff – injecting chaos in your systems, and they ended up injecting faults that weren't detected for like a week or two, right? And so that, to me, is a clear signal you must be this tall to ride this ride. If you can't tell what's going on in your system at a low, per request level, then you have no business injecting more chaos into them until you can actually tell what you've done right? Like just having the ability to see the consequences of an action that you've taken is a pretty fair prerequisite in my mind. But most people don't have it. Most people just have the aggregate graph that, that doesn't actually them the kind of low-level, fine-grain visibility that you need in order to just see what happened when I pushed on this button.

Mike Kavis:
Yes. So last question for you, and I follow you on Twitter a lot, and recently there's been a lot of chatter with you and your followers about heat maps, so tell us about heat maps and why they are important and how they're helping people observe and understand your system state better?

Charity Majors:
Heat maps are so great because we humans don't look at lists numbers and understand really what they mean, but we do understand patterns and shapes really well, and there is a big difference between if your latency is, like 95th percentile, you know, your latency is spread out across multiple percentages, versus tightly clustered. And these shapes, we wrote a couple of really awesome blog posts recently that just talk about how to interpret some of these shapes and the color distributions and it's a kind of thing that once you've educated yourself on what those things mean you can just tell at a glance what's happening deep your system and that's really exciting.

We just shipped another thing called BubbleUp, which helps to answer the question that everybody asks, us, is can you tell me what to look at right? Can you tell me where to look at it if I install your stuff, and if it's a longest time, I would just argue with them I'll be like ah! This is the wrong question blah, blah, blah. And finally I stopped fighting. And I was like, you know what I think we can, I think we can, and so we wrote this feature called BubbleUp, and what it does, it and there is a demo on our webpage at honeycomb.io/play, with live data from our last outage. We even play this a little bit, but what it does is, that for any field, that you can draw a map on, you can just draw a box around the timeframe that you want explained to you, and it pops out and does like a birds eye view and gives you like a

heatmap of every single thing like you could possibly have feedback for that, right?

So, sort of iterate through this dimension, that dimension, this dimension, to try and find out where the outliers are, you just draw a square around it and, bloop, it does it all for you. And it's really cool because your eye is immediately drawn to the outliers, right. Your eyes are such great pattern matchers, and so maybe, you know, you had a cache problem, whatever, your eye will be immediately drawn like a two out of 20 or 30 graphs that there show to that problem is or it's centered or the export endpoint is great for this one user you – his eye is immediately drawn to like the endpoint outlier and the user outlier and maybe some other outlier for the query or something it's really, like, a cool, you've really got to play with it to see how powerful it is.

Mike Kavis:
The thing I like about it is, a lot of times you'll have an SLA of 99.99 or something, and you look up out there your dashboard and it's green, but what you don't see in that, like we have some like a heat map even though it's green you could see something looking ugly, or trending red and you could head it off before your screen turn red instantly right?

Charity Majors:
Averages cover over so many sins, like so much pain. They just completely obscure that. I could not agree more.

Mike Kavis:
Well Charity it's always a pleasure talking to you, tell us where we can find you on Twitter and where we could find all the good blogs and presentations, YouTubes I know you speak – so where can we find all this stuff?

Charity Majors:
Yes twitter.com/mipsytipsy, my personal blog is https://charity.wtf/ and the Honeycomb Blog is www.honeycomb.io/blog and I think that's' about it.

Mike Kavis:
Oh! Cool thanks again, so that's our show for today, you can find more podcasts like this from me and my colleague Dave Linthicum just by searching Deloitte On-Cloud podcast, we'll see you next time on "Architecting the Cloud".

Operator:
Thank you for listening to Architecting the Cloud, part of the On Cloud Podcast with Mike Kavis. Connect with Mike on Twitter, LinkedIn and visit the Deloitte On Cloud blog at www.deloitte.com/us/deloitte-on-cloud-blog. Be sure to rate and review the show on your favorite podcast app.

Visit the On Cloud library

www.deloitte.com/us/cloud-podcast