EPISODE 1194

[INTRODUCTION]

[00:00:00] JM: Microservices are built to scale. But as a microservices-based system grows, so does the operational overhead to manage it. Even the most senior engineers can't be familiar with every detail of dozens, perhaps hundreds of services. While smaller teams may track information about their microservices via spreadsheets and wikis and other more traditional documentation, these methods often prove unsuitable for the unique demands of a growing microservices ecosystem. A microservices catalog is a solution to this problem. A microservices catalog seeks to centralize information about the services in your software architecture including the purpose of its owner, its service, and the instructions for using it.

A microservices catalog can also provide a centralized source of knowledge about a system, which can help on-call engineers diagnose issues and provide resources for onboarding new team members. Larger companies can sometimes devote significant internal resources towards developing in-house microservice catalogs while smaller organizations may not have the resources at their disposal to do so.

OpsLevel's founders recognized that many teams were reinventing the wheel building internal microservice catalogs and set out to design a tool set that could meet the needs of users of all sizes. OpsLevel's team has a lot of experience with industry leaders in DevOps to create a large tool set for managing microservices infrastructure. OpsLevel integrates with Slack, CICd tools, Git, incident management and deployment systems.

John Laban and Kenneth Rose are the founders of OpsLevel. Before John and Kenneth founded OpsLevel, they worked together at PagerDuty where John was the first engineer on the team. Kenneth, who is OpsLevel CTO, was also a senior developer at Shopify. John and Kenneth joined the show today to talk about OpsLevel and how it can help developers manage their microservices better and even transform how their team does DevOps.

[INTERVIEW]

[00:02:03] JM: Guys, welcome to the show.

[00:02:05] KR: Thanks for having us.

[00:02:05] JL: Thanks. Great to be here.

[00:02:06] JM: You both work on OpsLevel, which is a service catalog and tracking and dashboarding system for overseeing microservices and other aspects of your infrastructure. We've done a few shows on products that are in this category. I'd like to talk about the category as a whole. But first maybe have you guys just explained what OpsLevel is and what it does for people?

[00:02:30] JL: It's a very new kind of category. And so I wouldn't say like all these different products and companies and things all fit within it exactly the same. But what OpsLevel is, we're calling it a service ownership platform. And really it comes in these two big parts. The one big part is being able to just catalog everything that you have running in your architecture. Maybe taking a step back, we can talk about the problem that these companies are facing when they see the need for something like this.

As architectures have become more and more complicated over the last couple decades, it's gotten to this point where because of the birth of things like cloud computing and virtualization and containerization and even like serverless, you have all of these different smaller and smaller pieces of software all kind of running in a distributed way in these companies architectures. It's becoming more and more complicated and there's more and more stuff. And being able to create like a microservices-based architecture or distributed architecture, it helps these companies grow and scale, but at the same time it causes all sorts of new problems, these new organizational problems that crop up.

Just being able to understand what's happening and what's running in your architecture in terms of what's out there? What services and systems exist and what do they do? Who owns these systems? So if you need to make a change to a system or I need to talk to someone about one of these services, who do I talk to? And then if I need to operate some of these systems, like where do I find the logs? Where do I find the runbook or the metrics dashboard? Have we moved from Splunk to Sumo Logic yet like for logs? Where is everything that I need in order to be able to operate these systems effectively? And how do I know how to like own these systems.

And so the first part that we're building is this microservice catalog. It's just being able to answer all those questions easily. You can think of it kind of like service discovery. How a lot of service discovery mechanisms are meant for like machines to be able to talk to other machines. We're kind of building that, but for humans. So you can understand what machines are out there. What systems exist? What services exist and who owns them and how do I operate them?

And then the second part of the service ownership platform that we're building is helping companies and helping teams understand how to operate their services. How to build and how to operate their services so that they're following a lot of the best practices for these companies? So for part of that, there's often these concepts of production readiness checklists or sets of best practices that you should be following in terms of how – Like when you're launching a new service, how to operate your service effectively? We help you codify a lot of those. So we can actually monitor them on a continuous basis and automate a lot of that measurement of like how well your service is doing towards these best practices and do you have the right observability in place? Have you set up your CI and your Kubernetes in the way that we recommend you have enough redundancy? Have you done kind of the right things that should be true when you're launching a new service especially when the service might be like one of your mission critical services? So anyway ,that's the big kind of answer about like what it is, but it's a service ownership platform.

[00:05:28] KR: Yeah. And, Jeff, to your point about like being a nascent space. You're absolutely right. I think what we've seen over the last 10 years is we've seen kind of CI being like one of the first kind of tools to become generically available. Then you have tools like PagerDuty for on-call management. Metrics tools, there's a diamond dozen. It used to be that you would roll your own, but now we have like tools like Datadog that you can just go ahead and buy. But this class of tooling around like service ownership I think has been emerging because you have all these individual tools that teams have to use, and there's kind of this tool fatigue that goes on. And just having I think of obstacles like a Metatool that kind of like helps you understand information about what's happening in Datadog, what's happening in PagerDuty, what's happening in your source code, what's happening in terms of your service ownership. And I think we're seeing that now because after all these other SaaS tools have been out there in the market, there's now this new problem of like, “Okay, we have all these microservices. We have all these tools to solve like individual parts of ownership, but nothing that's kind of like tying that all together.” And I think that's kind of been the impetus for, A, a lot of companies building their own internal service catalogs and then for companies like ours to come on and try to make an actual product out of this as well.

[00:06:27] JM: One pain point I could articulate that i think a product like OpsLevel can help with is the onboarding experience for a new employee. Like if I'm a new employee – I remember, I worked at amazon briefly for about eight months, and even at the eight month mark you're still onboarding. You have no idea how anything connects to anything else. Trying to interact with other teams and understanding how their infrastructure is laid out is really hard. Trying to understand how failover structures work is really hard, and just trying to understand what services are available. What kinds of things you could just ping to get what you need from the company. All that stuff is so hard and often requires meetings. It often requires email chains. And just having more, I guess, you could say mapping to the infrastructure, like maps that are laid out for like here's how things work is potentially so useful.

[00:07:18] JL: Yeah, absolutely. It's hugely useful during that onboarding situation when you're new to the company and like the names of the services often don't make any sense, right? Either they're following some like naming scheme where it's completely esoteric where it's

Futurama characters or Greek gods or something like that, or it gets to the point like at Amazon where everything was a little more literal but it got to the point where everybody was just using three-letter acronyms because it was so literal that it’s a multi-word. And so everything's like three-letter acronyms. So it's really hard to understand anything.

That problem happens not just as a new employee, but it can also happen when reorganizations happen, when you're moving from one team to another, there's transfers between teams or just like teams or reorg in general. You have that same problem, but it's much more acute because it's happening across like a whole bunch of teams all at once. And yeah, it's absolutely one of the problems that we help with.

[00:08:05] JM: So one of the problems with building a product like this is you have to actually have a way of getting all these, whatever company is adopting it, to plug their infrastructure into it. So you have this GUI that is going to just display the service health and different services that are available, the different repositories, the different teams, but it has to be configured. So what is the onboarding experience for a new company that's adopting OpsLevel?

[00:08:33] JL: Yeah, the onboarding. First step is to integrate all of your systems. So we'll plug into your deployment system. We'll plug into your repositories. We'll integrate with GitHub or GitLab or Bitbucket or whatever you use. And so we pull in like all of your repos. You send us all of your deploys. And that gives us kind of the first steps of understanding what's out there and we can try to start auto-discovering stuff.

We also will integrate into – If you have a process for creating new services, sometimes that comes into the form of a template that you clone for new services or sometimes you actually might have a – Some companies have like an entire workflow for creating new services. We have two different mechanisms. One of them is for that first cloning a template. We have config as code like YAML files that you can define within your repo and we'll pull those in and we'll use that to synchronize a lot of metadata about your services. But another is we have a full set of – We have a GraphQL API where you can call to us from different places. So whether it's that new service creation workflow or if it's your CI pipeline, you can push information to us from both of those places or from many other places. It's just like adding all of those integration points. And then we can start like threading the needle and giving you this more holistic view for what a service looks like and what it is and from all these different angles.

[00:09:45] JM: Does the entire company need to adopt a product like OpsLevel in order for it to be useful? Is it useful even just if a single team adopts it?

[00:09:54] JL: So it's usually better if you can get the entire company to adopt it. A single team, it can still be useful. Like that onboarding use case is really the one where you have like new members of the team. Being able to show them where everything is quickly and getting them ready to go on-call. If you're moving towards like a DevOps culture where you have your actual engineers both building product, but then going on-call for that product and having to own it end-to-end. Getting new members of the team comfortable and understanding like how to operate your systems quickly and giving them the confidence to be able to do that is valuable just on one team. But then you know there's a lot of additional value you can get if you can get all of your services and all your systems on ops level. That's where you start, being able to get paint a bigger picture of understanding what's going on so you can understand things like – Say you're doing a migration from one thing to another. Maybe you're trying to upgrade the version of the programming language that you're using, all of your Ruby services you're trying to upgrade to a new version of Ruby, or if you're trying to migrate all your services to Kubernetes for instance. Being able to see, “Okay, what are all the services that are in scope of this change? And which services have migrated so far? Which ones haven't?” Being able to actually understand, “Okay, which teams are responsible for getting those final services over the finish line?” You get a lot of that once you can get the full picture of all your services on ops level.

Before, like companies, to solve those same problems, they would often have to put together spreadsheets maybe for each initiative or something. They would put together like a set of spreadsheets and have a bunch of meetings and things with all these teams to understand, "Are they the ones that own this service? Can you commit to upgrading it? Did you do it yet?" and then trying to manage this all in spreadsheets before something like OpsLevel.

So I would say you can still definitely get value but it's usually just like the catalog part to some degree if you just have like one or two teams on OpsLevel, but you really get a lot more once you can get everything in OpsLevel.

[00:11:43] JM: I want to talk about the competitive landscape eventually, but one angle on that is there are a couple other companies at least that have built a product like OpsLevel. Like I've interviewed Effx, which is ex-Airbnb founder. I've interviewed the Cortex team. Both of those products are sort of similar to what you're doing. Why now? This is a problem that has existed for a long time. Is there something about the current infrastructure environment that makes it easier to build a service catalog? A communication path product like this? Because it's clearly a category unto itself, otherwise you wouldn't have these three or maybe more companies coming up all at once. Why now?

[00:12:24] JL: I think the problem has just come to a head. Like I said, are just becoming more and more complex and companies are moving to microservices architectures so that they can help unblock each individual team. Prior to something like a microservices architecture, companies often start with just like a monolith. Even at Amazon where you worked. They started with just a monolith. I think was called the Obidos back in the day. But eventually you get to the point where you have so many people working on this monolith. They're stepping in each other's toes. They're making, breaking changes. They're making conflicts against each other's codes, code commits. They're introducing bugs in each other's code. And they're often blocking each other. They have to kind of like streamline releases and things.

So to unblock all that, they start moving to this distributed style of architecture whether it's microservices or even like serverless or even just a service-oriented architecture. But that helps them scale so they can get a lot bigger. They have all these teams that are suddenly independent that are not blocked by other teams and they can work completely independently of each other. But then they introduce these new problems.

And so I think it's just as these architectures grow and these companies get to the point where they have hundreds and thousands of services, the need has become stronger and stronger to the point where many, many of these companies end up building their own internal tools. So we've talked to hundreds of companies, and there're so many out there that have like built internal tools. That's really our main competitor when we think of competitors, is like all of these internal tools that get built eventually to solve this problem after they get to like usually about one or 200 services. You get to the point where the pain is high enough that they decide, “Okay, we're going to dedicate a team for maybe the better part of a year to sit down and build something so we can just help manage the chaos out there.”

And so I think it's just we saw the writing on the wall. I think was over two years ago, Ken and I, we started building OpsLevel. And just based on all of these systems that kept getting built over and over again, we just saw that there’s definitely an opportunity here. And there're been other companies that have started since then. I think it just helps validate that this is really a problem that a lot of companies are facing.

[00:14:24] KR: Yeah, to John's point, I had the opportunity when I was at Shopify to use ServicesDB. If I can infer some of our more direct competitors, they've probably also seen the same kind of power and value that internal tooling like this gives, and it's just about how do we make that tooling available for the rest of the market. And that was for john and I, why we decided to go build OpsLevel, because we've really seen how tooling like this can just help level up internal teams and help drive reliability initiatives, help drive campaigns and just improve service ownership across the board.

[00:14:50] JM: So let's say I've got a service and that service is deployed. It's sitting in the wild. How does OpsLevel communicate with it? because if I've got this single pane of glass I'm looking at that maps all my services. It tells me what's going on with each of these individual services. I need to get the runbook associated with that service. I need to get the logs associated with. I need to know the P99 latency of that service. How is all that information getting piped into OpsLevel?

[00:15:18] KR: Yeah. So one of the things that we do is, as part of the onboarding, we integrate with all of your deployment tools. We kind of have a good understanding of everything that's being shipped out to production and trying to map that back to what teams actually own it. The second part is around kind of we have one specific feature around scorecards or checks that allow you to kind of drive campaigns. And so as part of service ownership, if it's important that you have a P99 defined or you have logs with a certain index defined, we can actually enforce that or measure how teams are doing and make it really easy to understand which teams haven't done that and kind of like gently nudge them to go ahead and get that information and put it because it's part of service ownership for an organization.

[00:15:55] JM: Is it hard to manage all the necessary integration points to build this kind of single pane of glass?

[00:16:00] KR: I wish I could say no, but there are a ton of integrations that have to be written. We currently integrate with all the major Git providers. We integrate with your deployment systems. We have APIs to help integrate with other tools that we don't have deep integrations with yet. But there are just a ton of third party tools that are out there that we are continually building integrations for. And it's a closed set, but it's a closed set with high tens to low hundreds of things to integrate with.

[00:16:24] JL: I think it speaks to just the complexity. Like the fact that you have the same view of your services and systems and so many other tools. Like I said, we try to thread that needle so that you can get this single starting point in this single picture of what your service is where do I find all the different pieces of it? But yeah, the fact that there is so many integrations just speaks to the complexity over the tooling landscape out there now.

[00:16:46] JM: Tell me a bit about each of your backgrounds and how your work on infrastructure has led you to building OpsLevel.

[00:16:54] JL: When I started my career, I started at Amazon. So I worked there as well. Amazon was really ahead of its time in terms of moving towards a service-oriented architecture. I started there in 2006. And they were already like heavily, like had a huge, massive sprawling service-oriented architecture. They had moved away from their monolith years before. And they had really adopted – They were very early in this adoption of you build it you own it type culture. They didn't have the separation between operations and development where the developers would like build the service, hand it off to operations to deploy.

Before the term DevOps had been coined years before, Amazon had very much moved in that direction where like the teams had end-to-end ownership of the services and systems that they're building. They would write the feature. They would write the service. They would deploy it to production and they would go on-call. They would get paged if it broke. And if there was a SEV1, they would be in there fixing it. And so Amazon moved to that early on, and I really saw like how a very large scale service-oriented architecture could work but then a lot of the complexity that came with it. And Amazon built a ton of internal tooling around managing that mainly because they were really early and there was nothing out there.

But then after Amazon in around end of 2010, early 2011, I started at this very small company known as PagerDuty. I was actually their first employee. It had been born from a few friends of mine that had been working at Amazon with me prior to them breaking off and starting PagerDuty. PagerDuty had started with a similar story to OpsLevel, and that they saw companies like Amazon building this internal tool to manage on-call rotations and manage paging and alerting when you're going on-call. And so they thought, "Why did Amazon have to build this? Why did Google built an internal system like that? Let's make it so that we can build it for everyone and have it be a product that instead of focusing your time and effort on building something that's outside your core competency like PagerDuty, let's build PagerDuty and have that be a product."

I started on as their first employee back in 2011, and it grew quickly along with the DevOps movement with this trend towards engineers going on-call for the systems and software that they were owning and operating. And so PagerDuty grew quickly. I spent a lot of time there on a whole bunch of different roles there. As an early stage employee at an early stage company, you have to often wear tons of. So I was a senior engineer at the start and eventually I moved into management and I was managing teams of teams. And for a while I was like VP of engineering. At the end I was more of an architect type role at the end. But during all that time, PagerDuty in a lot of ways was helping a lot of companies move towards a you build it, you own it type culture or a DevOps type culture, which is really important for DevOps and getting engineers on-call. But it was very reactive in nature. It was all about like fixing issues as fast as possible. When something breaks, go fix it.

But at OpsLevel, so we started to think about can I – This was after we had left. We started to think there are all of these issues that companies would rather be more proactive about. They want to start taking a more proactive stance. They don't want to have that downtime or that security issue in the first place, right? They want to start preventing them if they can. And so we started to build this product to help with that problem, to help with the ownership problem of like how do you make sure that your services are being built in such a way that they can be operated effectively when an issue does happen? Or even better yet, how do you set them up so that you have a less of a chance of having reliability issue in the first place or less of a chance of having a security breach in the first place? So that's kind of how OpsLevel came about. Ken, maybe we can cover your background.

[00:20:31] KR: Yeah, for sure. I was a kind of a math geek in high school and university. So I gravitated a lot towards pure mathematics and ended up doing my master's degree in geometry and worked for many years in like 3D graphics at Autodesk. And then kind of around like 2010 there was this – 2010, 2011, there was like startups and YC was becoming a thing. I was like, "Oh, like that sounds really cool. That sounds really interesting," like high growth. And something I decided to start exploring.

So I kind of [inaudible 00:20:53] it around for a few years with like different Toronto area startups. I'm from the Toronto area. And then I noticed, I was in this one startup and we're in this space with like a bunch of other startups and they were all using this tool called PagerDuty. And I was like that, "Oh, that seems really interesting." And an old colleague of mine I went to at University of Waterloo was actually working at this company called PagerDuty, and I found out that there's this guy John that's setting up a Canadian office and they were trying to figure out what city. And so John and I met. This was like late 2012, and it was like, “Oh, I tried to make the cut. Like you should set up in Toronto.” In Canada, like Toronto I think. And I still think it is, but at the time was like this is the hub for startups. Like this whole corridor between Toronto and Kitchener, Waterloo, it's very similar like the Bay Area in San Francisco, Palo Alto, the Canadian equivalent.

And so low and behold, I don't know if it was because of what I'd said, but PagerDuty and John ultimately decided, “Yes, Toronto does seem like a good city.” And they built the Canadian site here. And I was actually their very first kind of hire to join. John went more on the management side and I stayed more on the technical side. So I was a principal engineer focusing a lot on – I think the teams changed names many times over the years, but their real time platform like how PagerDuty kind of deals with this plethora of events coming from different monitoring systems and going through the ringer and turning that into notifications and actually like ringing somebody's telephone.

And so I did that for many years and a great experience kind of working with John. Ended up going after PagerDuty to a small Canadian e-commerce company called Shopify and also just a lot of challenges with service ownership there as well with the teams I worked with there. And so kind of when John and I were kind of talking and thinking about like, “Oh, what are problems that people have?” This problem around service ownership and proactivity and getting in front of things just kept coming up and we kept seeing companies building internal tools like this. And for us that was the spark that there's a market deficiency here that we need to go solve this.

[00:22:31] JM: How did the experience at PagerDuty inspire and inform what you've built with OpsLevel?

[00:22:37] JL: Yeah, it's a good question. I mean in a lot of ways we saw the DevOps movement kind of take hold at a lot of different companies. And we saw this movement towards you build it, you own it, and PagerDuty was helping enable that, right? But while we were there, we saw the kind of rude issue of DevOps, which is around service ownership. It's around like owning your services and being able to own them end-to-end. We saw a lot of companies struggle with service ownership. As they move to these distributed architectures like we had mentioned whether microservices or serverless or whatever, there's this explosion of stuff, and all that stuff needed ownership. And without ownership you would get orphan services rotting in production. They would build up these security issues or they would increase their reliability risk or you get an overall slowdown in engineering activity because the other end of the spectrum is where nobody wants to break anything. And so they're all walking on eggshells or everyone's busy fighting fires because everything is broken. And so everything is slower.

And so we saw these kinds of problems happening. PagerDuty was helping enable like fixing problems as quickly as possible and helping fight those fires quickly, but we saw this kind of like overall need for the service ownership platform or at least the problem that service ownership needed solving. And so that's kind of where OpsLevel came about, or it's through that reflecting on our experiences there.

[00:23:56] KR: Yeah. The other thing I'll add to that is like PagerDuty is just a fantastic kind of dev tool. It solves a really core pain point about like my server is broken. How do I get a human to go fix it? And we've got to learn a lot just about having empathy for people who are going on call and having to be service owners. A lot of what we did at PagerDuty was through the lens of kind of alerting and incident management and on-call. But to John's point, we saw kind of this, again, bigger story around the end-to-end life cycle of what it means to being a service owner. Like what are the requirements that an organization has put on you as a team to kind of own your service in production and what can you demand from your organization in terms of like training and support and coaching to be able to better be a service owner? But I definitely think I learned a lot about how to build good developer tools and just having a lot of strong empathy with developers and really understanding the pains that they go through when I was there.

[00:24:38] JL: And PagerDuty themselves, like as a software development company, they face these same problems internally, right? They started with the monolith. It was a Ruby on Rails monolith and they had moved eventually to creating services. I actually created the very first microservice there for better or worse and I helped design the second. And I think Ken worked on the third or something. But we had moved to microservices architecture. And over time there, that architecture both grew rapidly, but also started evolving, right? Like the very first services we had built were I think Scala-based and they were using like certain data stores. I think it was Cassandra, Zookeeper type things.

So over time we had started building services in other languages including I think –

[00:25:20] KR: Elixir.

[00:25:21] JL: Well, eventually, Elixir, but before that even Ruby and Go services and then Elixir as well. And frontend, all sorts of different technologies on the frontend like in terms of JavaScript frameworks. Yeah, and just things kept evolving both – And then in terms of all the tooling, even we started moving to containers, I think we used three different container orchestrators that kind of complexity would kind of grow over time as you have all these teams that are working independently of each other. They have that autonomy to be able to move independently of each other with having ownership over their systems, but with that became like this large – It was somewhat chaotic in terms of how it would evolve over time. And we eventually had to sit down and say, “Okay, we're going to try to put more focus and put more investment on a smaller subset of things so that we can both reduce the cognitive load on all of our engineers.” And also if we can get a certain critical mass behind a certain smaller subset of technologies, then we can move a lot quicker.

And so just seeing those kinds of problems and experiencing the first hand and then also trying to like answer broader questions around what's going on out there in our architecture. And when I was kind of taking on that architect type role at the end, I was kind of that glue in between all the systems, and it was really hard to answer a lot of these high-level questions.

Both PagerDuty as a company building a DevOps product helped inform OpsLevel, but also just the experiences of PagerDuty as a growing technology company helped inform as well.

[00:26:38] JM: We've talked about OpsLevel in the sense of this thing that sits on your desktop as a dashboard, a window into the world of different microservices and infrastructure in your company. It also has value as an incident response tool. Basically something that provides you the insights necessary to troubleshoot an outage. Tell me about how managing outages what's difficult about it and how you've tried to design a product that averts the kind of crises that can develop in an incident.

[00:27:12] JL: Yeah. You're right. It's not just about the proactiveness. It's also about being able to fix incidents faster when they do happen. And so OpsLevel, because we help track all of your tooling and you can list kind of out like where do you find the runbook for this service if something goes wrong? Where are the logs? Where is the metrics dashboard? Where is it living in Kubernetes? Where is it living over here and there? Where are all the different observability tools and infrastructure tools and everything that I might need when something goes wrong? We help you find that information really quickly.

We also have a slack integration so that a lot of the initial triage for these kinds of things often happen in Slack. And so if you do get paged about a system that you've never heard about, you can just search for some part of that word and we'll search across any kind of aspect of your service whether it's in the name or the description or the team or whatever and just return anything that seems like it matches. Help you find what you're looking for. And then from there you can pull in all those tools. And so you can just quickly find the runbook. Find the logs for this service. You don't have to worry if we migrated from one logging system to another yet. You know what it is. And you can also share this information with your team. Like if you have multiple people working on the problem together, you can share it within Slack quickly and easily. So we try to make it easier to find what you're looking for when things go wrong.

The other kind of thing that we do to help with this is we also have a deployment kind of tracking system where you can hook in your deploys like I mentioned earlier, and it helps you understand something break recently because of potentially a change. Was there a change that just happened very, very recently and could that be correlated with this issue? And being able to just see that quickly without ambiguity and then knowing whether or not you should just attempt to roll back as a first step can be really helpful.

[00:28:48] JM: Talking more about the day-to-day improvements that this kind of tool can help with, the ability to have reliable microservices to have reliable failovers in place, to have proper documentation, to have services that just sit there silently and do what you need them to do. That's another thing that this kind of tool can help with. Can you tell me more about how you help with just day-to-day micro-services operations?

[00:29:18] JL: Sure. Yeah. With the DevOps movement, there's a lot of know-how that kind of has to move two independent development teams so that, before, if you might have had like an independent operations team that can centralize all the knowledge about like all the different things that you need to do, but now a lot of that know-how has to move to the development teams. So this has to do, like you had mentioned, do you have a reliable failover setup already? Is your system in CI or potentially continuous deployment? Do you have an on-call rotation set up? Do you set up exception monitoring and continuous integration and vulnerability scanning? And do you have a runbook set up? And are you in the container orchestrator? All these things that might have been easy for one team to kind of know in depth now has to be kind of distributed across all your teams. And so that could be really hard for these engineers that many of them are already full stack engineers and need to know like a lot about the frontend and backend development. And now they also have to know a lot about operations and potentially also security and how to build into place systems so that they don't have easy security vulnerabilities within them. And so like that can be really difficult.

And so what we do with OpsLevel is we try to help empower these teams to fully own their services, which includes knowing how to set them up so that they can have all of these best practices. So we let you define the scorecards or these checklists basically, but these checklists are filled with checks. These checks are continuously monitored. Rather than someone having to go in and say like, “Yes, we did this. Yes, we did this.” There's something that we try to like help you automate a lot of that. So we can introspect, look at your code. We can look at your operational tool chain. And you can even set up fully custom checks as well using our APIs. And we have like various custom check APIs. We're going to build your own checks.

And so these are things that you can continuously monitor against all your services and you can highlight some of the biggest issues that things that you really should be doing but are not doing for your service. And so your SRE team or your security team can create these checklists and help distribute a lot of that know-how to individual engineering teams and help measure and show them directly what problems might be existing within their services and what kind of technical debt might be kind of building up quickly.

[00:31:16] KR: That's actually one observation I wanted to touch on as well, John. Like there are a variety of stakeholders that have input into what it means to be a service owner. You have your [inaudible 00:31:23] called the platform team or your SRE team that cares about like reliability and operability initiative. So, "Hey, this service is plugged into the Internet. Is it going to fall over? What happens if there's a failover?" You have concerns around security and like just are there high criticality vulnerabilities that are enabled? What version of certain libraries are we using? What version of certain packages are we using? You can have issues coming from FinOps as well like how much is it costing to actually run the service and how do we like surface that information? So there's kind of a variety of concerns that can go into it into service ownership and it's just about how can we as a product surface all that information to those teams? Because ultimately the responsibility of handling those issues should be on the team, right? That is the idea of you build it, you own it. That's part of that ownership part. But if you don't know what that set of responsibilities are, how do you actually begin to accomplish them? And so to John's point earlier, like one of the things we do with OpsLevel is we can have these checks that help provide governance and provide insight into what are the responsibilities a team has and across the organization helping measure how your teams are doing across these different dimensions.

[00:32:19] JM: You just mentioned FinOps there. I'd like to deviate our conversation a little bit and go in that direction. FinOps, as I understand it, is this newer term. I don't think I've even heard FinOps used before, but I understand what it is because I've done enough shows on this topic. It's the idea that no matter how profitable your company is, even if you're Airbnb, you're consuming so much cloud resources that it's starting to cost a lot. And there are some teams that are on the hook for a lot of money. And you can have some real, real tough problems, some hard conversations. If you have a team that is on a critical path of a piece of an application and that team is using so much infrastructure that it's costing more than the bottom line of the value that they produce, what do you do? That's a really tough question. And it's a question that more and more companies are having to grapple with as they're realizing, "Hey, look. We started off with move fast and spend money and that got us into this position where we can raise like a series B. But unfortunately our infrastructure costs more than the profits that we make. What do we do?"

And so you have this other tier of new tools that has come around, this set of FinOps tools, like I just had Kubecost on the show. They help you estimate the cost of your Kubernetes infrastructure. There're these companies that help you buy reserved instances more intelligently or by spot instances more intelligently. So this FinOps thing is a huge idea. Tell me a little bit about your respective experiences with FinOps people and how big of a problem that is.

[00:33:50] KR: So I think – I mean there's a bunch of stuff that's there. I think part of it is, okay, obviously if your cost of goods sold is higher than your margins, you're going to have a problem as a business. I think for a lot of companies, especially as you mentioned the growth stage, when you look at your P&L, you kind of don't care about your infrastructure cost. It's just you have a certain business problem you're going after. You're trying to capture the market. The infrastructure costs what it costs. And you'll kind of deal with that problem down the line. You can kick the can down the line.

Pretty far, a lot of the major cloud providers offer different startup credits. And if you're kind of a BC-backed company and you're growing, like the infrastructure cost is also generally not a huge amount compared on your P&L compared to salary and head count. That said, I think there's kind of like different classes of tooling. One of them is the, “Okay, we're spending X-millions of dollars on our database bill. What are the things I can do? Like how do I –” You stop using reserved instances and start using spot instances. Or what workloads can I migrate to something like Lambda where I have more elasticity on my pricing? And there's just a variety of tools that do that. They're useful, right? But the challenge still I think from a service ownership perspective just comes back to attribution. How do you know what teams are responsible for what services? Because it might be that, yeah, you're spending a lot of money on the infrastructure, but that team is powering the line share of your business. It is expected that you're going to have higher infrastructure costs because you're having higher growth.

The team ultimately should be empowered to be able to make decisions about what their costs kind of should be or have insight and input into that. And the attribution problem is actually really hard. Because you have teams spinning up infrastructure all the time trying out new experiments, and if you don't kind of know what teams own what things or if you have shared underlying infrastructure, it can be really hard to map that cost.

Again, for me it's still I think there's a ton of value in the tooling that's out there for the tools that will help you optimize your spend or reduce the number of reserved instances or use Lambda. Like that helps you kind of with your top line almost of just reducing your overall bill. But if you want to get really much more fine grained, it's about attribution. I remember one anecdote, we started to make cost data available when we were at PagerDuty and we had a hack day and it was an intern that lowered the annual spend by like $200,000 a year. And that's a lot of money, right? And it's just because like, “Hey, it turned out just by making this data available the engineers that kind of have the most context about a service can start to make decisions about, “Yeah, actually do we need this capacity? Are we using all this in the best way or the smartest way that we could?”

[00:35:54] JL: Yeah. I mean those FinOps tools, they often focus on the consumer of this information will be like a single person who has been kind of like tasked to reduce costs. And so they'll help that person make small but broad changes, changing certain type of instances with others almost like invisible changes such that they can reduce costs. And they do work in that way. But we found that, like Ken mentioned, a lot of powerful instances where if you move the cost information to the teams, the teams that are responsible for owning and operating these services and often are the ones who increase costs through provisioning more infrastructure or just in the way that they're building the service in the first place. If you move that information to the edge where they can see the cost, a lot of teams are like, “We're spending how much on this?” We can make a small change, a very small change to reduce our cost by a hundred thousand a year, two hundred thousand dollars a year, even more, and these can be very small like minor architectural changes but ones that are not that central cost person is not going to be able to make or be able to even like think about how to do that because they don't know the system in the same depth as the team does. So yeah, moving that cost information to the edge can have a lot of powerful effects.

[00:36:59] JM: Have you guys encountered any companies that have gotten themselves buried and cost structures that are too expensive?

[00:37:04] KR: I think for our customers, it's not that the infrastructure is too expensive. They're not like going out of business because their infra costs are too high. It's always something especially in large organizations that they found product market fit, they're still growing, but they want to keep a handle on. So it's an area of importance, but not of like urgency in a sense.

[00:37:22] JL: The bigger the company gets and the farther along it is in its life cycle, the more focus that they'll have on that.

[00:37:29] JM: Let's talk some about competition. You don't have to address your competitors specifically, but this is an interesting market that's unfolding. And I know I've talked to some of the VCs, the infrastructure VCs, and it's a hard one to bet on. Because just thinking of it from a venture capitalist perspective, if you had to choose between investing in OpsLevel, or Effx, or Cortex, it's pretty hard to pick. Like all three of these companies – I don't know if there's more in the space. All three of these companies has a really strong team.

They're kind of building something similar. The UI looks a little bit different. The go-to-market strategy is a little bit different. It's hard to know who to pick. And I think it's one of these spaces where it's the logging and monitoring space. Who's going to be Datadog? I'm sure that was a question at a certain point. I'm sure there were a lot of companies that looked like Datadog and then Datadog kind of became Datadog. So if you're thinking about the competitive differentiation, what are your opportunities for that competitive differentiation? How can you get ahead of your competitors?

[00:38:29] JL: I think at the end of the day it just comes down to building a really great product. And you had mentioned we're not really focused on our competitors and what they're building, but we're really focused on working closely with our customers. Understanding their needs really well and then satisfying those needs. So like we have a set of great customers currently that we work with on a regular basis very closely and we try to have frequent meetings with them where we get their feedback on both recent changes that we've made but also upcoming changes. Show them what we're thinking. Dig in deeper with them in terms of their problems in the space and make sure that we're going to solve them properly. That's the best that we can do. We haven't really like dug into our competitors' products in terms of using them very much, but we're just focused on our customers and being able to solve their problems well.

[00:39:10] KR: I also want to touch on Datadog. Like I've been using Datadog since like 2013, and Datadog started out as this great metrics product and they just really nailed metrics and dashboards. It was just such an intuitive UI. It was really easy to understand. And they've also just been able to execute extremely well. And I think that's the secret for any company's success, and I think it'll be the secret for OpsLevel as well.

Currently we're focused on the service ownership problem, and I want to be able to nail that problem. And to John's point, we work really closely with our customers to be able to understand the problems they have. And then in terms of expansion, Datadog was able to evolve from monitoring into logging and APM and just having a great execution try to do that because they built such a robust underlying platform. And strategically, I think we have a pretty similar playbook that we're following as well. We want to make sure that we build an extensible service ownership platform that allows for a future where like you can integrate new tools, where you can easily integrate new workflows. And as an early stage company, it's hard to balance between like building a platform for the future and building a product for right now. But I don't I hope we're doing a good job with that and I hope our customers agree as well.

[00:40:08] JL: It's good how – There is some competition in the space as well. Just it's kind of like the rising tide lifts all boats thing where it's bringing increased visibility into it and some degree of validation that like it's a problem worth doubling down on and tripling down on. We've all got great investors like you mentioned, and we're all going to be working hard to solve this problem. I think OpsLevel has just had a head start and we've been building out our product a little more for a longer period of time and working closely with the larger set of customers. But I think it's been a movement that's kind of helping all of us.

[00:40:37] JM: As far as the ongoing uptime of OpsLevel, like I'd like to understand that a little bit more sort of how OpsLevel sits there and continually gathers information and a little bit about how you've architected the product. I'd like to know more about what databases you use. How OpsLevel is sort of continuously screening the infrastructure for updates? How it's pulling updates? Or is it getting updates pushed to it. Let's just talk about the infrastructure of what you've actually built.

[00:41:03] KR: For sure. I wish I could say that we had like some super sophisticated, crazy, hard understand infrastructure. We actually don't though. Like, architecturally, because we're a small company ourselves, our app is fairly simple. We are a monolith. It's a Ruby on Rails monolith. We use MySQL as our primary like transactional data store. For things like job management, we use Sidekick and Redis. We integrate with a bunch of Git providers and Slack and PagerDuty and a bunch of other tools and that all just is handled by our Rails WebApp.

In terms of like deployment information, we have APIs to receive all that and we just receive this kind of push of events that come to us and we basically store a lot of it in MySQL. And then in terms of our frontend, we use Vue.js, which has been phenomenal. We use ES6. And ultimately we have to allow users to be able to efficiently kind of search and sort and filter and query the data that we've collected from all these different sources. And that's the secret sauce, but it's not that there're tremendous, deep engineering problems. The problems are much more around kind of user experience and organizational, right? Like how do you keep this data up-to-date of what's running in production? What teams own it, and tracking those? On the engineering side though, it's a pretty straightforward app so far.

[00:42:09] JL: It's probably also worth mentioning, it's a bit ironic, but we don't have a microservices architecture. We have a monolith, right? And I would really recommend any small company when they're beginning their first system and their service to start it as a monolith. You just kind of move a lot quite a bit faster and you have a lot more flexibility. Microservices, they come with a whole bunch of overhead. They definitely help you scale after you hit a certain tipping point. But before that, they just add like more overhead than they're worth.

[00:42:35] KR: I was going to add, in 2021, we're probably going to spin out our first microservice. We have some tables that are kind of like audit tables and logging tables. They could stay in the monolith, but like it might make sense to break out. But to John's point, we've been biasing towards moving faster as a team than just using microservices just because it's a thing to do. We use our own product internally though and use that as a way of gaining empathy. But it's at a much smaller scale compared to our customers that have like hundreds or thousands of services.

[00:43:00] JM: Tell me about the most difficult engineering problem you've had to solve thus far.

[00:43:04] KR: In a funny way, we have this config as code thing. Basically like we allow our users to drop in a YAML file in the – Or somewhere in the repo and we can automatically like pull that information and synchronize it. And that sounds on the surface really easy, but there's actually it turns out just a bunch of niggly corner cases that come up with – We have this notion of locking a service in the UI. So like if it's backed by YAML file, that YAML file is sort of the authoritative truth. So we don't want to let you edit that. And there's a bunch of corner cases that crop up around like, “Well, what happens if there's a service? How do we pull in this information?” We try to merge information as best as possible.

One field, for example, that we have on a service is just like a set of the operational tools. And if you have some list of tools in the UI and some list of tools in YAML, we will merge those. But if they're the same tool, then we'll overwrite the one in the UI with – It doesn't sound hard, but there just ends up being a ton of corner cases and the inference and the resolution of how these different entities need to merge together. There's kind of data in source A and data in source B and we want to automatically like merge those together. It turned out there was some sophistication that went into doing that correctly.

[00:44:09] KR: Yeah, we try to keep things really consistent there. So in a way, instead of a single database, you have a distributed database across hundreds or thousands of repos. And trying to coordinate all that can be kind of more difficult than you think.

[00:44:19] KR: Yeah. The other kind of fun challenge is just dealing with API rate limits from a lot of the third party providers and just thinking about how we can spread out our job life cycles and job runtimes so that we don't like, “Oh, a service got updated. Let's make 10,000 requests to GitHub.” No. We're going to be 429 if that happens. So how do we kind of spread those out but kind of know what services should be affected first? And there's some fun – We call it spreading the butter, but spreading out the kind of pace of API calls that we make.

[00:44:44] JM: As we begin to draw to a close, I would love to just get some more perspective on each of your macro pulse check on what is going on in the world of infrastructure. It's kind of a beautiful time to be building software because the abstractions are getting so good and so high leverage. I just love to get any sort of pulse checks or nice little sound bites that kind of put in context are current times.

[00:45:08] KR: For sure. I think, for me, one of the biggest things that just is really cool is a lot of the JamStack stuff that's coming out. I think the software of yesteryear, like late 90s, early 2000s, even late auths was like, “Okay, you got to spin up a host and install some software on it." And there's just been this focus now on like, "No. No. How do we actually like tease apart the kind of static UI parts and then the actual like dynamic parts that are data and have a very strict separation?"

As an example, for our homepage, which is entirely static. We use Netlify. And it's great. It's magic. We run a static site generator and off we go and like all of a sudden all this data is replicated in a bunch of pops around the world. And then any dynamic parts we can just have as an API with Lambdas. I think that in terms of the off-the-shelf tools that exist, there's just like a newer, faster way to build a lot of stuff that's just, I don't know, super exciting.

[00:45:53] JL: Yeah, I agree. Like the JamStack is really neat. Also kind of similar vein, the growth of serverless kind of applications and how it can be really easy now to just instead of building out a whole service to solve some problem, you can just kind of put up some functions that can all together they kind of form this what you might call before a service. But you don't have to deal with quite a bit of the underlying underpinnings of building a service and you really can just focus more on the business logic and have a lot of the rest of it done easily for you whether it's Lambda or other platforms. So yeah, that's also been really interesting too. And there're a lot of needs and gaps that have kind to light because of the growth of serverless, but I think it's an interesting time.

[00:46:33] JM: And what's in the future for OpsLevel? What do you guys want to build that you have not yet built?

[00:46:39] JL: Ken, I'll let you take that one.

[00:46:40] KR: Oh, thanks John.

[00:46:42] JL: We talked through kind of our upcoming road map, our vision for the product.

[00:46:46] KR: Yeah. I'd say that in terms of our vision, a lot of it really comes down to extensibility, right? I mentioned earlier the domain of service ownership is services and teams and various parts of infrastructure. So databases and queues and consumers and object stores. That domain is pretty well codified and works for like 99% of companies. Where the challenge comes in is just really around extensibility. Like what a service means to company A might be a little bit different than what it means to company B. And so having kind of a better customization and better extensibility just in terms of the data model for service ownership.

So this is what I was talking about earlier about OpsLevel as a platform. We want to be able to build a lot of that underlying plumbing so that just becomes really easy to integrate and extend OpsLevel so we can really cater to whatever use cases our customers have in terms of like the things they want to model around ownership.

[00:47:33] JM: Cool. Well, anything else you guys want to add before we draw to a close?

[00:47:37] JL: I think we covered a good amount. Thanks for having us on here. This has been a blast. It's been different. I think you got a great show.

[00:47:42] KR: Yeah, Jeff. Thanks a lot.

[00:47:43] JM: Okay. Absolutely. Thank you both for coming on the show.

[END]