EPISODE 1167

[INTRODUCTION]

[00:00:00] JM: Micro-services route requests between each other. As the underlying infrastructure changes this routing becomes more complex and dynamic. The interaction patterns across this infrastructure requires operators to create rules around traffic management. Tobias Kunze Briseño is the founder of Glasnostic, a system for ensuring resilience of micro-service applications. Tobias joins the show to talk about micro service routing and traffic management, and what he has built with Glasnostic. If you'd like to become a more intimate supporter of software engineering daily, go to softwaredaily.com and become a paid subscriber. You can listen to all of our episodes without ads. That's softwaredaily.com.

[INTERVIEW]

[00:00:44] JM: Tobias, welcome to the show.

[00:00:46] TKB: Thanks for having me, Jeff.

[00:00:48] JM: We've got all these different services, they're running in containers across our infrastructure? How are we controlling traffic that is being shuttled between these different services?

[00:01:00] TKB: That's a really good question, right? Because today, the only thing you do you have two places where you can control how things work out. Number one is the old school setting timeouts in code, right, what we used to do in the Unix days, and number two is configuration files, right? You can utilize a service mesh, you can use other configuration mechanisms. You can use plain envoys in between and there, you can specify retries. You can specify timeouts, all these kind of things. But it's fairly limited, and it's very difficult to change over time.

[00:01:34] JM: Tell me more about the modern state of the art for traffic management.

[00:01:39] TKB: Well, there isn't really much right, because the industry is so focused on observability — the control aspects kind of fall by the wayside a little bit. Right, it’s all about how do I get more transactional visibility? How do I get more tracing done? How do I get when I trace? How do I get more data around — what I'm tracing. Whereas the traffic management aspects are almost orthogonal to these concerns, right? There's a little bit of a disconnect in the industry today, where the behavioral aspects of how my systems work together, are not in the center of the focus. And that's mostly because as engineers, we've grown up to write applications, write functions, think in terms of thread of execution, that's the happy path, right?

And then we, when we deployed, we maybe deployed five or 10 services is most, and most of the time, these were standalone, right? Those were standalone applications, or at least as far as we're concerned as engineers, right? But what happens in the real world is those are connected to 20 other systems, right? Really what's out there is a whole fabric of services — a fabric of systems, that's all connected. And at that stage, you get the success of your architecture is almost 100% dependent on how all these other systems work, right? And that's where traffic management comes in. That's where you need to be able to become the air traffic controller, right? You need to have a way to insulate yourself to control the pieces that you don't own. Right, but that you depend on.

[00:03:32] JM: You work on Glasnostic, explain what Glasnostic does.

[00:03:36] TKB: So we are an operations solution at the highest level. We take on those large scale, constantly evolving service landscapes. Think 20-50, to 200, teams deploying in parallel, all these services, the systems connected. We think about how to control and manage how these systems work together. So classical management task — with the goal that you don't have to think about this in code, or that you don't have to go and put these behavioral directives really in static configuration. So we have real time control of how systems interact.

[00:04:21] JM: Why is that useful?

[00:04:23] TKB: Because as we mentioned earlier, FIDE systems become connected systems, no application is an island right? Now you if you're connected to other systems, you depend on these other systems in some way, shape, or form. And that means your fate depends on how these systems perform. Now, if you connect more than three of these systems together, you get very quickly in a multi-body physics problem where things get chaotic, very quickly. And if you think about it even little bit more, all these systems are stitched together and everything is finite. So, as you stitch finite systems together, you're gonna hit all kinds of limits. If you listen to any Site Reliability Engineering talk on a, you know, industry conference these days, the number one cause is limit. Systems are limited, you are going to hit some kind of limit, whether it's a replica limit, whether it's a read count limit, whether it's some other weird limit — all of a sudden, your systems behave differently.

So you need to be in a position to detect this very rapidly, be able to see this very rapidly, and then do something about it. And our standard model of doing something about an outage, a failure, a degradation these days is really to diagnose, diagnose for five hours, eight hours, and then fixing something and maybe, you know, five hours, sorry maybe five minutes. So we change that model. In our model, you can detect in within 20 seconds and do something within 10 seconds.

[00:05:58] JM: So you see yourself more as an SRE tool?

[00:06:03] TKB: We are a, an operations tool, or even more an operations solution. We change the way you operate these large scale infrastructures, these large scale architectures, right? So I see ourselves as an enabler for operations to run 10 x more systems at the same time to interconnect them more quickly and to evolve them more rapidly.

[00:06:29] JM: Can you describe the usage of Glasnostic in more detail?

[00:06:35] TKB: Sure, it’s very simple, really, deploy into the network, we act as a bump in the wire in your network. We don't touch any of your workloads. We don't install agents or anything like that. We deploy it into the network, then essentially, we look at who's talking to who, how much and when. And that's what we then make visible and more importantly, controllable.

[00:07:03] JM: So give me an example of deploying a service and how Glasnostic would work with that.

[00:07:11] TKB: Yeah, very easily, right, you have a piece of code running down a CICD pipeline that gets deployed typically, in pre-production environment, a staging environment or something like that. That, of course, is not really representative of what's happening in the real world. So ultimately, what you want to do is deploy into production, because staging is expensive, and not really that useful anymore — in larger installations. So deploying in production, if you could just ring fence what you deploy or quarantine to some degree, right? And cannery deployments help, obviously, but you know, a more generalized way of ring fencing things, you can massively mitigate the deployment risk, right?

So it's a way of controlling at runtime, how your services interact. And if you have that ability, you're not going to have an outage. You may run in a degraded form, you may run it, let's, let's say 90% 95% capacity. But the upside is that you can avoid the outright outage, right? If you think back at the beginning of COVID, when Robin Hood went down for three days, right, very publicly, nobody could trade markets tanked by 25%. Nobody could trade their shares. The official cause was, "Oh, sorry, our DNS services got overwhelmed," right?

And now the question clearly is to ask yourself, why did this happen? Well, you're exposing yourself to that kind of load. And you have no way of exerting even like classic back pressure, right? It's almost negligent, right? So with us, we could solve this in like 20-30 seconds. all that's needed really is to see, well, there's a lot of traffic hitting our DNS services, let's exert back pressure. And the situation is resolved. At least it buys you enough time so you can think clearly about what you want to do. As opposed to picking up the pieces after everything fell down, and you've been down for hours and your phones are ringing, right? And your pages are going haywire. So it's it just fundamentally smarter approach to operations.

[00:09:28] JM: How does visualizing the the traffic management of different services? How is that advantageous?

[00:09:37] TKB: Yeah, think about what happens if you have a some form of failure in your architecture, right. And again, keep in mind, we're talking about service landscapes, but just constantly evolving, pretty complex environments. What happens is, all your failures are nonlinear right? Something may start out as a CPU issue, there becomes a latency issue upstream. So something starts as a CPU issue on one system becomes a latency issue on the next system then turns into retry storm, right? And then all of a sudden you got like, wide ranging, asymptomatic slowness. Now it touches maybe 20-50 systems, right? And all these systems throw arbitrary alerts, because they only throw alerts on the metrics that they see. Right?

So now the engineering teams are in headless chicken mode, they run around and try all kinds of things, Oh, we got this alert. You know, we're out of file descriptors here. And nobody knows what's going on. Everybody's looking at the trees, nobody's looking at the forest. Right? So the way we’re advantageous is we only focus on the big picture, right? We only look at how does everything work together and by looking at how these services interact with with each other, and how these interactions change over time, and how these individual golden signals we capture correlate to each other, you get a very good sense, what's wrong. And you can do something about it by treating it like managing and controlling how these systems behaves. From the outside.

Think about as an operator, you're really acting like an emergency medical team, right? There's a victim at the side of the road, you come in there, you look at golden signal, pulse, temperature, you know, pupils, whatever you do, the number one goal is to sterilize. It's not about diagnostics, it's not about root cause analysis, these kind of things. Most of the time, as you know, when you go to the doctor root cause doesn't really matter. Too many factors involved, not necessarily coming back, time wasted, more important to treat the patient, right?

That's what we're all about. That's why it's so effective. Quick visibility, where is something going? Not quite right, what doesn't look right, and then do something about it. And doing something about it doesn't have to mean to shut things off. It really only means shaping a little bit, but adjusting, allowing it to interact a little bit differently.

[00:12:18] JM: When a failure occurs in infrastructure, what kinds of tools do we want in place? What kinds of visualizations and SRE tools do we want in place to make that failure easier to deal with?

[00:12:30] TKB: Well, you're asking the right person. But of course, I would say Glasnostic. There's a place for observability for data dog for New Relic for app dynamics — these things, but they increasingly local concerns, right? If you think about it, you're a team of two pizza team five, six engineers working on four or five, maybe two handfuls of services, right? That's your horizon of responsibility. That's what you're really interested in. Right? That's where you're looking at how can I trace to these requests? I want to get more runtime debugging going. But you may be looking at specific profiling of these services.

All these tools are super helpful. As your services now become part of a larger whole, those tools are not that useful anymore, right? I don't want to trace into 40 dependencies that I have. Those are just that, they're just dependencies. I don't I don't even own the code, right. I don't need visibility, and you know how many sockets there are in close weight, right? It's just, I, it's a service, I need to depend on it. And that's all I care about. If these services misbehave, I need a way of insulating myself as much as far as I can from these failures. That may involve circuit breaking, that may involve inserting some kind of bulkhead, even segmentation. Any of these control primitives are helpful in that situation.

So that's exactly what we focus on fine. Ultimately, everything comes down in these systems to how systems interact. It does no longer matter really how what these systems do internally, whether there's a defect in code, what matters is how they interact with other systems. And that's why we focus exclusively on that aspect.

[00:14:19] JM: Tell me more about remediating a failure that might occur.

[00:14:25] TKB: Sure. Imagine you have, I mean, going back to the back pressure example, right? Very classically, you get overwhelmed. Some services get overwhelmed. Most of the time you don't notice this until it hits — spirals out of control and hits systems that are three-four hops upstream, or other downstream systems, you know, cousin systems. With us, if you have the ability to quickly see that you're running a little bit out of capacity here, and your auto scaling rules, whatever you have in place to deal with capacity are not quick enough to react —

If you see that nothing matters other than the ability to do something about it, right? Don't watch it, spiral out of control, do something about it quickly for a few minutes while your other systems spins up your — you know, replica gets, becomes a primary, you know, whatever you do. And meanwhile, take care of the situation. So a typical example right? Other example, you have things replicated in two different availability zones, of course, you want to separate them out, but at the same time, make sure these, you know, critical — business critical servers can failover to the other zone. So just insert of an operational bulkhead in between that that kind of takes everything that's going on in one zone, isolates it to some degree from the other zone, while at the same time allowing a failover for specific systems, right?

Those are the critical capabilities. If you run anything at complexity, right? You need to be able to manage very much like a manager in a large organization, if you have 50 teams, you restructure them to suit your goals for the next quarter. And then you're not going to lean back and not do anything, right? At that point. Your job is day to day to make sure your teams work together, and that they're not blocked on anything. Right? That's your job. We bring that capability to the operational side of running complex architectures.

[00:16:38] JM: Tell me about the deployment model for Glasnostic, that this is something that you would generally have a sidecar deployed for. But tell me about your deployment model.

[00:16:48] TKB: Yes, so we avoid the side cars, simply because they're invasive, they tend to become very costly in the lifecycle management stages. And most importantly, because we don't need them, right. We, as I mentioned earlier, we are a essentially a bump in the wire. I like to think of this as a brain in the wire, it's a little intelligent function in there. But essentially, we are bump in the wire as people in the networking space, call it. That means we get to see everything that flows across, you know, across the wire. We can resolve endpoints, and then we can do something about, you know, set of endpoints really arbitrary set of endpoints.

Now, that has a obviously a network specific part of how we insert ourselves. And it has our network function — our virtual network function, essentially, right? So the two parts now, we can insert ourselves in a number of different ways, right? In a standard VM based or on premises model that's host-based VM based, we are just going to spin up a new VM that becomes, you know, router in the network. Or all the way if you think about {inaudible 18:02} is running STO, we can plug into an STO proxy or an envoy right? And just simply plug in, just use those as our routing infrastructure, if you will, and then just plug in the virtual network function on top of it.

So very flexible, each environment is a little bit different, but plain {inaudible 18:20} with STO given eaters with like other service meshes VM based multi axis edge computing, but different environments. But the system is always the same. The idea is always the same. A little function, little brain that just regulates it's the traffic cop regulates how systems interact with each other for stability and security purposes.

[00:18:44] JM: So I thought this functionality was handled by envoy or was handled by STO.

[00:18:50] TKB: To some extent, it's not really well, manageable. Also, the policy piece is slowly — seems like it's slowly getting out of STO, we work in a slightly more general way. So we can control traffic between arbitrary sets of endpoints. It's not really a virtual destination that we need here. It's a much more general way of in much more rapidly manageable way of dealing with these policies, and other advantages, as your operating complexity, you cannot think about keeping your policies from not overlapping.

So it's very important that you can have, you know, a generic bulkhead somewhere and at the same time, have a policy that overlays it, and also gets, gets applied, right? It's actually a fairly complex piece of code and logic that allows large operators to layer these policies in any way any which way. So that's the big difference. We see the the policies in a service mesh as very, for lack of a better word, developer-focused, meaning I'm sitting in my ID, but I essentially just want to talk to some other service, I don't care which instance just hand me over to the best service. Yes, and throw in some metrics, though in some encryption. And that's really it's, it's, it's link me to that other dependency, right to the destination.

And the other way, it's difficult to express right? It's very difficult to express a policy that says, "Well, here I have a bunch of, you know, a federation of systems, and I want to shield them from an onslaught of a DOS," or something like that, right? So you'd have to go in and create YAMO. And then debug that YAMO. Make sure it works, right. There's no visual aspect of this really, it's difficult to readjust. And it's very easy to keep these YAMOS in the configuration. Now you're looking at something where this becomes the next cause for an outage. Right? So we just pull this all up centralizes make this a first class citizen. So you can operate with these capabilities.

[00:21:05] JM: So how would I make a like a policy change in Glasnostic?

[00:21:12] TKB: Yeah, very simple. So everything is UI driven. And everything that you can do on the UI, you can also do via an API, obviously. So other tools can integrate with it. But it's UI driven, if you, as an operator, are sitting there, you you put in, let's say, a rate limit between two services or sets of services, right? You discover that's not exactly the right number, you just go back and change it.

Or, you know, you pull it out, we show you a full overview of which policies are in effect at any given time. And it really allows you to work your infrastructure, your architecture, much more like an air traffic controller, right? So if you think about, there's really two different views on what you're running, right? How you operate things. And that, number one, if you end developer, you care about the happy path, this thread of execution. That's, as I mentioned earlier, that's where you want to have tracing full visibility, all that yeah? You're much more like the pilot and an airplane where that flight is the only thing that matters to you. And of course, there you need full visibility, right, you need to have a full cockpit of gauges and switches and controls. And you want to know what the oil pressure is unlike engine three, right?

But as an operator in a large organization, you have thousands of these flights, right? You You need air traffic control, you can have these flights computer generated these flight plans as much as you want to, there's always unpredictability, but you have to deal with these things. So air traffic control, on the other hand, doesn't go deep. It doesn't look at what the oil pressure is, right? It looks at very simple golden signals, it needs a callsign, and it needs precision, altitude, direction, speed, for us exactly the same way. We take care of the entire architecture, how everything works in that shared environment that shared and finite environment, and make sure nobody steps on each other's feet. Right?

So we need high level signals of everything that goes on there, that can't be a single service we don't see, right? And so first, we need the service name. How many requests are coming from a service? How long do the take, on average, how many pileup at the same time, right, the concurrency of things, and then bandwidth. So that's the kind of visibility we give our operators. And that enables them to work at that forest level of visibility.

[00:23:48] JM: How does Glasnostic integrate with monitoring infrastructure?

[00:23:54] TKB: So remember, monitoring is a local concern, right? Monitoring is, if you take care of like five or two handfuls of services, yes, you want to monitor them. Each team wants to monitor their services, right? We live in a stitched together world, but a lot of systems getting, you know, put together stitched together wired out very quickly today. At that level monitoring is pretty useless. There's too much detail — it's very difficult to see the big picture. So that's why we come into play.

Now, of course, as we discover some degradation — let's say latencies, blowing up a little bit between in some corner of your system. Then of course, we want to hand that over to the development teams, we want to be able to do something about it, right? Maybe circuit breakers, you know, tier three services. So shed some load or you know, whatever the, the, whatever the operational pattern is that you want to apply. But then we want to hand it off to the teams and say listen, this is what we see here, and you know, create a ticket, incident ticket, you know, whatever the case may be, maybe. And then teams look at that, and maybe, you know, look at the monitoring, see what happened here — why did we all of a sudden, like send way more requests than before? Right?

But now this can all progress in a very orderly fashion, as opposed to the headless chicken mode I mentioned earlier. Right? That's an important integration. But it is a an integration between a big picture layer, and the so the global big picture layer and the local tactical layer.

[00:25:40] JM: What about security? How does Glasnostic work with security problems?

[00:25:45] TKB: That's an excellent question — because we used to think about security as its own space. And it is not really its own space. I always encourage our customers to think about a government spectrum. And it's essentially this spectrum of things we depend on, but we don't own. So we need to be able to control. And that starts on one end of the spectrum, it's performance issues, right? You may hitting some performance limit, some other limit compounding effects there, but then that turns into stability issues, stability like my services, you know, fail period, or get run in a highly degraded fashion.

That turns into availability issues, that in turn, can turn into security issues — meaning now I'm being DOS'd, I am getting access violations, because some backup route is in the system that all of a sudden gets activated, then, of course, specialized version of security was really governance, right, compliance governance. So those need to be seen on a spectrum. And the interesting part is here, that if you talk to security professionals, the answer, or the desire, that you pick up on is that these days is that these people don't want a security solution anymore, right?

Because the experience has been very focused security solution that brings serial benefit to the operational side, is not very effective. So what you increasingly hear is that we want something that's essentially an operation solution. But it's also important for the security side of things. And here's an example. right — let's say you get a some system is breached. Or let's say you have a vulnerability scanning tool running, right, all of a sudden discovers Oh, there's there's a machine that hasn't been patched as a zero day on it. So that tool, typically, you cannot connect to any action. There's somebody needs to like look at it and do something about it, right?

And now with us, you can just simply tell this to a throw this alert, you know, make an integration call into this API right away and shut this machine down. So because it's an operational solution, it's highly relevant for security, mostly on the access control side. But also, you know, on other levels as well.

[00:28:27] JM: Now, if a failure occurs under bad circumstances that can create a cascading failure. Can you talk about how to avoid cascading failures and how to alleviate them?

[00:28:40] TKB: Yes, this is, in essence, what we are all about. Because as you scale architectures, as you compose more and more systems, these cascading issues actually are the normal case, right? Your system is always running in some kind of degradation under some kind of degradation. Now, most of these degradations balance themselves out after a while. Some of them spiral out of control, and then you have a massive outage, right? Going back to the Robin Hood example, but there's hundreds of others, you can just listen to any SRE talk these days, right.

So it's critically important to see these degradations and then be able to do something about it very quickly. And as I mentioned earlier, these compound cascading failure modes that essentially noisy neighbor, ripple effects, things like that, right — where it systems it's like a stack of dominoes, that each system affects the other system in in a very particular way, and highly unpredictable way. Right. And that is why the unpredictability is the key reason why you need to have runtime control. You can no longer operate these systems with the old DevOps cycle, where you ideas, I'm going to monitor something, I'm going to learn something, I'm going to go back to writing code and fixing and releasing that patch. But that takes too long. That is just your validated what you see. And then tell the team to write a write a fix, that takes too long things blow out of proportion, blow, you know, spiral out of control at a very short notice. So you need runtime control, real time control. That's what we're all about.

[00:30:31] JM: What was your inspiration for starting Glasnostic?

[00:30:36] TKB : Yeah so, I'm a pass guy platform as a Service guy, if I was the co founder of a company that became reddit openshift, spent a few years really exclusively only looking at like, how can we optimally support the building of applications, right? Essentially, just add code and, you know, run people's applications. What, then — what I discovered then is that nobody writes these applications anymore, right? If you think about it, as a developer, you think you write an application. but as you know, as you operate these things, we're connected to a load of other systems.

So the connectivity and the complexity only got accelerated by cloud. And because everything all of a sudden is like, you know, you just call an API, you get a bunch of new machines, you know, all these kind of things — containers, you know, exact {inaudible 31:35} to that, at the same time, the business needs to move faster, right. So the only way to move faster as a business and write more code and get to code to market faster, is to parallelize your development, right? Ten years ago, you still had like, large development teams, now it's very independent, two pizza teams. They all deploy something all the time, right? So you get 1000 deployments in your entire infrastructure every day.

Now, that means not just complexity, but also highly dynamic environments. And as you combine these things, it's very simple to deal with complex systems on their own. Right? If complexity — if this if it's static, I can deal with it, I can polish it, and at some point, nothing's gonna, nothing's gonna go bad. Similarly, I can deal with rapidly evolving systems if they're not complex, if they're fairly simple. But market forces truly conspire these days, and everybody gets pushed into the corner of that matrix, right? Where complexity and rapid evolution come together. And that causes unpredictability — exactly those cascading failures that I mentioned earlier, right?

So the only way to deal with unpredictability, and those are really the unknown unknowns that you're discovering there, right, is to be able to see to detect and respond to them in real time, right? You can no longer put things in code. So that's really the key idea behind Glasnostic, that with all these systems that we are stitching together today, in the enterprise, right, with all these systems that we are stitching together across clouds across on premises, edge computing, right? That all talk to each other. At the same time, you get this volatility of where workloads run, right?

In particular, with edge locations, you may migrate something from the internet data center to the edge location and back, right, or between edge locations. That means how these workloads how these systems interact, becomes the key behavior that you need to be able to control. If you can't do that, it's like running an organization with 50 to 100 teams, and you can't manage. Right? So each team does whatever they want. It's absolute chaos. So we are in the business of enabling operations to run the show.

[00:34:07] JM: Tell me what else you've learned in this PAZ journey over the last five or so years?

[00:34:14] TKB: Yeah, I believe the PAZ to beat ultimately, is an AWS, right? Essentially, the idea of PAZ in my mind comes down to I am a developer, I want to create a functionality that my business wants me to do. And how many Lego blocks can I use to accelerate my development? And it means — very simple, I'm using a hosted my-skills service because I don't need to run my own one, right. I'm using a Authentication Service because I don't have to roll my own one, right. There's 5000s of these blocks in AWS. There's like thousands of these Lego blocks on Azure. There's other blocks on GCP. Right? So essentially, to me, this is PAZ today, I think we are past the state where we are writing little two tier applications.

There are few companies, but they are very rare that still do that essentially run hundreds of these little two tier applications, totally independent of each other. Everybody else I see in the marketplace, runs essentially a service landscape. So what is a service landscape? It's a decentralized architecture, architectural style, right? It's not a single microservice application. It consists of many micro service applications that are interdependent. And it may include mainframes, it's not really just confined to microservice applications, right?

So that's what we call a service landscape. That's where you, as an operator, as a business really, deal with the complexities of, you know, a multi body physics problem. And the only way to disentangle that is by very quickly in real time, observing what's going on at the highest possible level, at the air traffic control level, and then being able to do something about it again, at the air traffic control level. And that gives everybody in the organization, the guidance they need to then go down and diagnose deeply if needed, right?

Most of the time, organizations discover it's not really needed, I'd say most of these issues, are flukes, they're random, and all that's needed is just that you can do something about it. But some of these issues, obviously you want to, you know, prevent for the next time around. So that's the learning really, from the PAZ side that the building of applications is increasingly less important than how you run it. So, maybe think about it as nature versus nurture, right? We grew up because that's the expensive thing to do was how can we write code, code writing was expensive, right? Developers were — capable developers were expensive, hard to get by not many around.

Now code, because the stack, you know, grows up all the time — the actual code is an increasingly smaller piece of the whole. So it's just an natural thing, that the importance of code itself goes down, and the composition of code becomes more important. So as developers, we still think that the value of everything is in writing the code, but it's actually not true, right? It's, we kind of think of it like cars were building the car. That's where I'm gonna pay 2040 $100,000. Right for the car. And then yeah, afterwards, yes, there's oil, there's tires, there's gas and all that insurance, but that's negligible compared to the value of a car. That's how we still think of it as developers, but in reality is much more like, “Hey, making the babies easy, raising it as hard.”

Because I'm writing code, I can write code in half a day, I can run it for a couple bucks a day. But the problem is, how is it going to behave next week, when two other services use it? All of a sudden, right? Now, does my scaling still work? Probably not. You need to raise that code. So the life cycle, the runtime behavior of everything you do as a developer, is increasingly the important piece. And that's the motivation behind Glasnostic, because there's nothing in the marketplace today.

[00:38:49] JM: I'd like to now get your predictions on the next five or 10 years of what PAZ and microservices looks like.

[00:38:57] TKB: My prediction is that the trend to stitch more things together is inescapable. It's going to be in every vertical, in every industry vertical, not just financial services, not just managed service providers, it's going to be everywhere. Because ultimately, it's the only way the business can go fast. Business wants to be agile, you know, working on an agile operating model. The slow piece today is how to get code to market. And it's not just a matter of deploying code, right? It's a matter of like, how can I then control the chaos that I'm creating.

But the fundamental belief I have is that we are in a post distributed systems world in the sense that distributed systems is really difficult — it’s a black belt of software engineering, right? That's where machine man versus machine is really coming in its own and it's — the important piece is that it's too slow. It's too brittle. It's too expensive. And most of the time primitives where I need distributed systems logic already solved, right? I can just use something I can use a dynamo I can use, you know what I need. But if you think about it, there is no distributed system in nature. But there is no {inaudible 40:20} in nature. It just doesn't exist. It's an old abstraction that we like, because it's acid compliance. And then how Yeah, it gets kind of difficult. How we do is do this with like, 2000 machines.

To — in my view, that's a niche problem, in the marketplace in real life, we're going to see way more stitching together, wiring up of systems — because whatever new functionality the business wants, I already have 90% running somewhere. So I just need to like, build on top of it. Everything has an API already. So it's inescapable. We're going to see these service landscapes become the default architectural style, not even microservices applications, that's the developer view of the five or 10 services I'm responsible for, but these are just one cog in a way larger system. And that needs a better way to operate.

Now, the other important prediction I have for the next five years is that I think culturally, we're still in that world where the developer's the kingmaker, right, we've been in it for like, 10 years, again, because writing code was expensive. I believe that model is going to flip around as the developer, the more systems come together, as a developer, I don't really know what is being delivered at the large scale at the global scale. So the business needs to start talking to the operations people. Today, the operations teams are, typically below in the organization below the engineering teams, the development teams. And that's kind of odd, because the operations people know how to deliver stuff, right? The business wants stuff delivered the operations, people know how to deliver it.

As a developer myself, I know I have 100 different definitions of “done,” right? And age old complain about developers, it's done on my laptop, it's done in testing, oh, it's done, this ticket is done. And you know, whatever, right? We don't know how to deliver things — operations people do. So my prediction is that the model of developer being the kingmaker is going to flip around and we're going to have a situation where the business reliably talks to operations for anything that’s delivery related. And then the operations teams know every single developer, they know the development teams, they know the intricacies of how something needs to be redeployed, all these things by changes made to a deployment and so forth. So that's really the best way of ensuring an agile organization, in my view. So that's my that's my prediction.

[00:42:51] JM: That sounds like a great place to close off. Anything else you want to add about micro services or Glasnostic?

[00:42:58] TKB: No, I think we touched on everything. It's a pretty wide field.

[00:43:03] JM: Okay, well, thanks for coming on the show Tobias.

[00:43:05] TKB: Absolutely. Thanks for having me.

[END]