EPISODE 1391

[INTROEDUCTION]

[00:00:00] KP: Microservice architecture has become a ubiquitous design choice. Application Developers typically have neither the training nor the interest in implementing low-level security features into their software, microservice or otherwise. For this and many other reasons, the notion of a service mesh has been introduced to provide a framework for service-to-service communication. Today's guest is Zach Butcher.

While working at Google, he was one of the earliest engineers contributing to Istio, an open platform independent service mesh that provides traffic management, policy enforcement and telemetry collection. Today, he's the Head of Product at Tetrate. Working on products like the Tetrate Service Bridge and Tetrate Cloud. We discussed the need and implementation of a service mesh and how companies are leveraging tools like their service bridge.

[INTERVIEW]

[00:00:51] KP: Zack, welcome to Software Engineering Daily.

[00:00:55] ZB: Hey! Thanks, Kyle, for having me.

[00:00:58] KP: Or maybe I should say welcome back, actually.

[00:00:59] ZB: Yep, yep, we were just talking about. It's been a couple years, I think. But it's good to be back.

[00:01:05] KP: Can you remind listeners a little bit about your technical background?

[00:01:08] ZB: Yeah. So I'm Zack Butcher. I'm one of the founding engineers here at Tetrate, which is a company that's all kind of all about taking the service mesh to enterprise. Before that – And there, at Tetrate, I work as our Head of Product day-to-day now. So kind of bridging engineering and customers. But previously, I've been a software engineer my entire career. So

before being founding engineer here at Tetrate, I also worked in Google Cloud. The last project in Google that I worked on with Issto. I was one of the earliest members of the project there at Google on Istio.

And before that, I worked across a whole bunch of different teams in GCP. So the central resource hierarchy, if you've ever gone and made a project in Google Cloud, project creation was my baby for a while. I worked on their identity and access management system, the service management system, and eventually some of the stuff that became kind of the service mesh inside of Google as we developed and rolled out that architecture. And in that same team that I kind of worked with, then pivoted and said, “Hey, let's go in and solve this kind of painful networking problem in Kubernetes.” And that became then the Istio project, which really has taken on kind of a life of its own, right.

And at Tetrate, one of the things we're excited about is kind of making it bigger than just Kubernetes and kind of bringing the service mesh everywhere, including back into legacy and into non-Kubernetes workloads. So it's pretty cool to kind of see it start from that angle and grow.

[00:02:38] KP: Could you talk about any lessons learned in building Istio that were informative when starting Tetrate?

[00:02:44] ZB: Oh, gosh, so many. Let me take one that's maybe more general than just Istio, which is listen to your customers. I think this is something that if you look at the different cloud providers, the different hyper scalers, they do this to varying degrees, right? I think Amazon is well-known for their kind of ruthless customer focus. Google, less so, right?

And so one of the big things that we did as we were kind of before starting Tetrate was go round and try and drive a lot of the initial adoption of Istio by a whole bunch of different companies. And we heard pretty consistently a common set of problems that Istio wasn't addressing, but that they wanted it to address. In particular, how do I start to bridge legacy infrastructure and modern infrastructure, right? How do I affect this move into Kubernetes and cloud?

And we heard that from a ton of different customers. And we couldn't really convince the Google leadership to follow that. It's a very engineering-driven organization. They don't do a very good job of listening to customers. And so, eventually, Varun, who's the product manager for Istio, who's our CEO now at Tetrate, left. And I joined him shortly thereafter to kind of solve these problems that we had heard customers describe.

And so that is maybe the number one lesson I would give, and it's maybe not technical, but that is listen to the people that are trying to use your software. Listen to people that are using it, to listen and hear how they talk about the problems. Hear what are the problems that they think it's solving, because they may be different than the set of problems that you think it's solving. And really key in and listen to them, and they're going to guide you in the right way. So that's one of the big things.

There's a huge litany of different kinds of technical things that I learned as well. One of the – Just to touch on one or two of those, and you can stop me at any time. I worked on some of the API design pieces in Istio as well, some of the networking API's, which are maybe notoriously hard to use, because it's a very flexible system. It's almost like Legos to put together and build the Istio configuration and the envoy configuration underneath.

And there were a ton of different API design lessons that I learned in the process of kind of helping build those with some of the other senior engineers on the project. I got to really sit down and learn from folks like Louis Ryan, for example, who's the lead engineer on the project, and also worked directly on that API design.

Maybe the single biggest thing that I learned from an API design perspective is this idea of unit of edit versus unit of ownership. This one is a very nice technical one and applies to any API, not just a configuration API. Not just an API hosted over the network. The same is true for API's we expose in SDKs or libraries, right? This idea of what am I going to make a user change? And who are the different users that may want to change the different pieces?

And it's really important to align the two. So the concerns of one user, for example, an application developer, can be addressed in a very small set of configuration. Ideally, like one document. And maybe not one because it gets too big. And that's a design trade off, right?

But then we have maybe security concerns. And it's pretty common in an organization that we're going to want to set security controls centrally. But we're going to want to let individual developers go and change traffic-oriented things at their own rate, as long as they're conforming to our security standards. And so one of the simple examples that we made in Istio's APIs where we got this wrong, where we mixed one unit of edit, the destination rule object. With two different owners, the application developer and the security team, because that destination rule controlled how traffic got to you. What is the load balancing strategy? What are the subsets? Is there a new version of the service that I want to canary? That kind of stuff. But it also controlled do you use TLS to talk to the surface or not?

Often, the security team is going to want to mandate, "You have to use TLS everywhere." You have to use mutual TLS everywhere. That's one of the big reasons to adopt the service mesh. And so you had this problem where teams want to allow application developers to edit their destination rule object so that they can, for example, create a new version of a service and canary traffic to that new version. But they don't want to let them edit the TLS parts.

And unfortunately, because they're in the same document, that destination rule API object, you can't really very easily control changing one and not the other. And this has subsequently been fixed with the authentication policy and some other pieces in Istio. So we split this out now. So that it's been addressed. But that was a key idea that I've taken away and that I use all the time in API design, this idea of unit of edit versus unit of ownership.

[00:07:31] KP: I like that. I think I'd like to take that as a takeaway as well. That makes total intuitive sense. But I also feel like I'm going to make that mistake a few more times before I learned the lesson.

[00:07:41] ZB: Always. Always. And it's a hard design trade off, right? Because you have to really think about what is the thing. And you have to think about it from multiple angles and from a perspective of multiple users, right?

[00:07:51] KP: Is there anything besides experience that can help a person figure out good ways to do that?

[00:07:56] ZB: Yeah, it is challenging. Certainly, experiences is a big piece of it. The other thing is to look at – Is to understand the concerns of the different users of your system. And certainly, that's going to come from experience. It can also just come from talking to them, right? The experience gives you maybe an intuition, but it's not going to be talking. And so if you go and sit down and talk to the an organization trying to adopt a service mesh, in this example, we heard very clearly, "Hey, there's actually maybe three or four different personas that are at play here." There's a central platform team. There's an application development team. There's a central networking team, usually, maybe a load balancing team. There's a security team. And they're all kind of grappling with this new technology that's right in the middle. And we have to figure out how they get to play together or how they actually start to interact. And that's really cool.

But it also means that there's, again, a whole lot of concerns that float around. And so there's a few different angles to consider. So my advice would be beyond just experience, talk to the users and uncover who are the personas that are going to be interacting with the system? What do they care about? Does this thing that I'm building line up for them? Do we need kind of a different view of the world for that persona, because they care about things differently? And that's pretty common that you'll need that.

That was another kind of – Actually, I'll touch on that. That's another deep technical takeaway that we learned building the Google Cloud resource hierarchy, that organizations folders and projects, is we built the resource hierarchy. And I keep saying the resource hierarchy, because we immediately turned around and figured out that we actually needed multiple. Because, for example, that organizational structure doesn't necessarily work if I want to do a billing roll up. The billing people in an organization maybe have a slightly different view than how I want to manage developer access to cloud resources. So those are two different views. And so that's one other kind of interesting – And so we learned in GCP, "Hey, there needs to be different ways to slice it." That's still an active and ongoing set of work across those teams, right? But that's another kind of key idea. And so it all ties into who's using the system? What are the ways that they need to look at the system? Because there's an underlying set of things that the system describes.

And then there's going to be a bunch of different users that come along and want to use those things, right? And they're going to care about different views. Whether that's the service mesh, and service mesh functionality. And those things are stuff like encryption and transit, and traffic control, and load balancing. Or whether it's higher level stuff like how we model resources in a cloud provider, in the different personas that are at stake there. Same idea.

[00:10:37] KP: Well, I'd love to zoom in on service mesh, but maybe starting from the high level, what is a service mesh? And why do technology groups adopt it?

[00:10:45] ZB: Yeah. So the service mesh itself is really kind of two pieces coming together. It's the idea that we want to pull a bunch of networking concerns out of the application. And we pull them out of the application by putting a proxy. A network, what we would think of as a traditional network proxy, actually right beside the application deployed next to it. And we're going to deploy one next to every instance of the application. We call that a sidecar. And that sidecar proxy, and I'll refer to it as a sidecar. The particular implementation that Istio uses, and that is most widely used across service meshes, is Envoy. So if you hear me talk about Envoy, thinks sidecar proxy. You'll hear me say sidecar. You'll hear me say proxy. I will try and be consistent.

So we have this sidecar beside every application. It's intercepting all the network traffic in and out. And really importantly, it can actually understand it at the application layer of L7. So for example, we can look at an HTTP request, not just a TCP connection. But we can do TCP too if we don't understand the protocol.

So with that proxy that's intercepting all the traffic in and out of the application, that gives us a control point that we can do all kinds of different stuff now with the traffic. So for example, we can produce a consistent set of metrics and logs for all of the traffic because we have this one consistent piece of software, this proxy, that sitting there intercepting it all. So we can say, “Hey, look, every service now automatically has a consistent set of metrics that describe the rate of requests, the rate of errors, and the latency, the kind of key operational metrics.”

We can start to enforce traffic control. We can start to do traffic routing. Because we have a proxy there, we can do client-side load balancing. And that gives us a whole bunch of tools and techniques for building a reliable set of applications against modern distributed failure.

And then finally, it gives us a policy enforcement point to be able to do security. This is something we'll talk about a little bit later in the call with NIST and some of the things there. But this idea of Envoy as a policy enforcement point is key. And when I talk about policy in this context, we can think of security policy. So, “Hey, I want to look at that HTTP request that's coming through and I want to apply WAF,” web application firewall. I want to just apply basic policy to the header. I want to do authentication and authorization. You can do all of those pieces as well.

So we have this sidecar proxy that's beside every application that gives us those capabilities around security, traffic management and observability. Then we need to drive those sidecars. We need to program them, we need to configure them to be able to affect the policy we want to so that a developer can say, “Do a traffic shift,” right? So a security team can say, “Make sure there's encryption in transit.” And that piece that configures those proxies at runtime is called the control plane.

And so that, when you see different service mesh implementations, a lot of them use Envoy for the proxy. We call that data plane. Then the control plane is what's responsible for programming a fleet of envoys and really kind of turning it from an individual proxy into a mesh, if you will. And some of the key ideas there are a central control point, the ability to change rapidly at the speed of configuration update. I don't need to go and redeploy my application to affect that change.

Finally, one of the other key ideas that a service mesh gives you is a notion of identity. So we want to give an identity to every application, so that then at runtime, we can apply policy based on that application’s identity. So I mentioned that we can do encryption in transit with the service mesh. One of the key things that we do is assign a runtime identity to every service and we encode that in a set of certificates that the service mesh distributes to those sidecar proxies. And when they communicate with other services, they can present that certificate. We can get an authenticated identity out of that. And so we can do service-to-service access control as well kind of with that identity. And that's a capability the service brings, that that control plane brings, on top of the basic primitive of encryption and transit that Envoy provides.

So again, just to summarize real quick, service mesh is these two moving pieces, the data plane and the control plane. For Istio and for most service meshes, that data plane is Envoy, the proxy, CNCF project. You can check it out at envoyproxy.io. And then for Istio, it's a control plane, is the one that is the particular service mesh control plane that I worked on that is maybe the most widely adopted today in terms of production footprint.

[00:15:35] KP: I'm wondering if we can imagine a small team. They've got a Kubernetes cluster going. Maybe 9 or 10 services they've stood up that all kind of orchestrate together. But they've done everything through load balancers and config files, environment variables. They didn't know about service mesh. And now they think it's time to adopt it. What are the steps to get something like that going?

[00:15:56] ZB: Yeah. So this is something we see very regularly, right? And so the idea is what you're going to want to do is, first, identify the specific endpoint that you want to use the service mesh to address, right? So it's really often that I see folks that are kind of, “Hey, this is a cool idea.” We know we want to sprinkle some service mesh in here.” And I go, “Great. What are you going to do with it? What's the value that it provides you?” And if you can't articulate a clear answer to that question, you're probably not going to have a successful service mesh adoption. And that's based on the experience I've seen firsthand talking with folks, right? The folks who are able to successfully adopt a service mesh in an organization, a large organization, a small organization, doesn't matter. They have a very clear and specific set of value that they want out of it, right?

So what is it in the small team? What is it that they need to get out? Maybe it's encryption in transit. Maybe it's consistent operational metrics. And maybe it's the reliability and resiliency. So assuming that they met that minimum bar, that they've identified what is the value point that we need from the service mesh? Then that should guide you into, as you start to do an implementation, what is the first set of things that you're going to implement, right? Because the service mesh provides a huge – It's a really featureful thing. There's a ton of stuff there. And the hard part is getting it into your system and using that initial set of features.

And then the incremental complexity and the incremental difficulty of adding new features, or using new feature, using features that exist that you haven't used yet, is very small, relative to

the cost of actually just getting the system in. So identify that first use case. Only do that first use case. Then mechanically, when it comes to the actual migration, the way that it works is go into your lower environments, your test and development environments. Identify an initial set of applications that are maybe your happy path that communicate primarily over HTTP REST, for example. And start to roll out the sidecar service-by-service there.

So you said in your example, maybe they have 10 services. Probably, in that kind of a world, maybe six or eight of them are happy HTTP services. The service mesh just fits in immediately, because we just put in the sidecar proxy, and it doesn't break anything. Then it tends to follow an 80/20 principle, right? And so then that 20% of applications, there tend to be a little bit more work. There might be an incantation or two that we need as far as how the service mesh delivers traffic to the application, and how it intercepts the application's traffic to make it totally work. And that might require a little bit of massaging, and that tends to be kind of a one-time activity for that application. It might be, in your world, if you use a common set of libraries, it might be, "Hey, get the incantation right for how we configure these little libraries to talk to the network." And that will be kind of a one-time thing for our company.

And so then we roll it out application by application in these lower environments. We see the proof points in those lower environments of the specific thing that we wanted to get value from, right? Encryption in transit, traffic control, durability, whatever it was. And we also use that to gain confidence in the operation of the system. And once it's been there and deployed, then we can start to promote it up into production as well, right? But it's a gradual, it's an incremental process. I would not try and do it as a big bang. Just turn on the sidebar everywhere, because you're going to have breakages. You need to do it service-by-service. There's a lot of tools to now roll out – Istio, specifically, has built a lot here to roll out the service mesh incrementally to parts of the application and not be disruptive in the world where you have parts in the service mesh and parts not.

[00:19:44] KP: As I start prioritizing my services, it's possible I could encounter a lot of diversity. Maybe the machine learning team has a Python service. The frontend is some express server. There's some serverless involved. Do I have to go into each and figure out how to do Istio and Python, then Node.js, and so on and so forth?

[00:20:03] ZB: Fortunately, no. So that's been a problem historically. The service mesh and the set of capabilities it provides is not novel, right? For any distributed system, traffic management, you've needed networking and things like that. And so historically, it has been done, for example, in libraries. And so you would have framework code for your Python, and framework code for your Java applications or for you C# applications, right? And there'd be a lot of work to make sure that they lined up. And it was pretty often, at Google, that we would see outages relating to – Until we built very robust testing that was expensive to maintain. But you would see outages relating to kind of the matrix of different versions of the libraries across different languages deployed and communicating.

And so that is one of the key ideas of the service mesh, is that because we're doing it in a sidecar proxy, not inside of the application itself, we're really interacting at the layer of the network. And so as long as that Python, or that Node.js, or that Ruby, or that Java application is communicating with external dependencies over the network, and ideally doing it with HTTP, although not only HTTP, then the service mesh will be able to plug in and work, right? And we're not going to necessarily need to change the application. The only thing we may need to change is some of the service mesh configuration itself kind of outside of the application code. But again, I want to stress that that usually follows an 80/20 rule where we see 80% of the time, it tends to just work in an organization regardless of the set of frameworks or languages.

[00:21:39] KP: When companies come to Tetrate for help getting this started, what are their typical motivations?

[00:21:45] ZB: Yeah. So I think one of the things we opened up with is the idea that we worked very closely with NIST. One of the areas that we worked very closely is around security, right? And so what we've seen is – Like I described, the service mesh is still a relatively expensive thing to adopt, right? Especially within a large organization. And Tetrate tends to work with very large organizations, usually about fortune 2000 or so. It's a hard effort to do this adoption. And, therefore, the pain that it addresses when a customer comes in, needs to be large, right? Just like I mentioned, you need to have one specific pain point in mind. That's true, whether you're talking to Tetrate. That's true, whether you're doing it yourself.

And so for us, we've seen the pain that is sufficient to really drive adoption in the largest organizations is around security and compliance. This is one of the reasons that we work very closely with NIST on not just service mesh security, but actually the set of papers that I write with NIST, the SB800204 series, is the authoritative set of recommendations for microservice security from the federal government, right? Not service mesh security, but microservice security.

And so that's the primary use case that we see folks come in with. Mechanically or tactically, maybe I shouldn't say that manifests as I need encryption and transit everywhere as maybe a first in specific tactical item. But immediately after we deliver encryption in transit everywhere, then it becomes, “Great! Now I want to do access control with that. I want to start to take advantage of the metrics everywhere to understand how traffic is flowing and where. And then I want to start to put it into developer hands so that they can do things like canaries and build more resilient fault tolerant systems.” But for us, it usually starts with security. And then once it's in, it's very cheap to start to give all the other capabilities to the rest of the organization.

[00:23:38] KP: How did the collaboration with NIST get started?

[00:23:41] ZB: Yeah. So that one turned out almost a little bit by chance, as these things kind of do pretty often. So JJ, our CTO here at Tetrate, had been working on access control a little bit. I worked in the access control space as well, I mentioned just a little bit, inside of GCP. And David, Dr. Ferraiolo, who's leads the team over at NIST, the secure application computing team, has been working on access control his entire career. That's really what he loves. It's his favorite thing. He is great. If you ever sit down and talk with him, he will talk your ear off about access control. And he happened to be in San Francisco literally the week that we actually started Tetrate. And we were in some offices on Spear Street there on the Embarcadero in San Francisco there that first week. He happened to be in San Francisco as well. And we had just read a set of papers that he had written a year earlier. He had just written them at the end of 2017. This was early 2018. And he was actually in San Francisco receiving the IEEE Lifetime Achievement Award for the standardization of RBAC. So everybody's using Kubernetes RBAC. Everybody's using some flavor of RBAC in cloud permissions, for example. He's actually the guy that standardized RBAC and wrote the RBAC spec years and years ago. And so he's worked in access control his entire career.

We read these papers. We saw he was in town. And we actually just sent an email over and said, “Hey, look, we know you're here. We're actually a few blocks over from you. We would love to sit down and have a drink, talk about some of the access control work you're doing. Because we think that it might be interesting, because we have this whole service mesh thing. And we have that whole idea of Envoy as the policy enforcement point,” right?

And so he was intrigued. And so we sat down, and we started to talk. We thought, “Hey, we can have a really useful collaboration together.” And so we actually started to collaborate around access control systems. And we actually still have an active collaborative research agreement with NIST around a set of technologies called Next Generation Access Control System, which David and his team have been developing for a couple years now. And that's actually in our product, which sits on top of service mesh. We can get into that in just a second.

But that's how we started working with NIST, is this collaborative research around access control, which was an area that JJ had been interested in, that I was interested in as well, and that we knew could be important for the surface mesh. And then, because that research was going well together, we were working well together and enjoying it. When it came time, there's been a clear need in government for better security standards for modern service deployments. David and his team were the group that are responsible for producing those. And so they asked if we would work together and to help author those, right? Because we wanted these to be really practical and hands-on compared to a lot of the other guidance that comes out, right?

And so because we were kind of in the field helping some very large organizations, including folks within the DoD, do this whole service mesh secure, zero-trust, this idea of this zero-trust architecture, that service meshes is a piece of the runtime for that. We were helping them. We were helping execute on that in practice. And so the folks over at NIST said, “Hey, great. That's awesome. Why don't we work together to write these standards so that they're based in that practice?” And so that's how then the collaboration on the SPs started.

And then that's even grown further. So one of the things I'll plug here is that we actually have done two annual conferences with NIST so far. And we have our third one coming up. And that Tetrate exclusively cohosts them when it's safe to travel. We actually do those in camp on the

[inaudible 00:27:26] campus in Gaithersburg. But this year, it'll be totally virtual in February. It'll be a zero-trust and cloud conference that NIST will be putting on. So do look out for that.

[00:27:37] KP: Very cool. I'm wondering if we can expand on the zero-trust architecture? What are some of the core principles?

[00:27:44] ZB: Yeah. So there's a whole lot of fud in the space around zero-trust today. It seems like everywhere turn around, everybody's claiming to have zero-trust, right? So when I like to explain that, I try and break it down into four key pieces. The first one being the pre-runtime activities. So what does it take to produce an artifact that can run in your production system, right? What are the code review processes? What are the CI/CD processes? What are the image scanning, the vulnerability scanning? What are the image signing? All of those things, the pipelines and processes that it takes to produce an artifact that can get run in production.

And there's a whole lot of stuff there. And there's a whole lot of players there in that space doing things like GitOps, doing things like software supply chain, software security, software bill of work, those kinds of things, right? So that's the first pillar. Then the second pillar is the runtime security. And this is the piece where encryption in transit is a key piece of this. But there's other stuff as well. You need to be authenticating and authorizing the services that are communicating. You need to be authenticating and authorizing the users of the services that are communicating. So there's a few different moving pieces in this runtime security bucket. This is where the service mesh really provides a whole lot of tools to be able to provide encryption and transit to be able, to provide a service level identity that you can be authenticated and authorized, and to provide a point to do consistent authentication and authorization of the end user credentials as well. So that's the runtime security elements.

And this is the piece, the lens to think through, or the question to ask yourself is, “If I took this workload, this application, and I exposed it to the Internet, what would I have to change?” That's the key motivation for the runtime piece. And if the answer is nothing, then you've achieved a zero-trust boundary, right? Or a zero-trust a system.

The third pillar – So we have the pre runtime. We have the runtime controls. Then we need consistent and continuous visibility into the system. We need to be able to continuously assert

that the system is in a healthy state, and that it is enforcing the policies that we want, right? We don't want to just configure policy one time and pray. We want to configure a policy enacted at runtime and then observe that it's being enforced continuously every single request, every single network interaction through the system. And the service mesh provides the raw data to be able to view and build that understanding that there is, in fact, that continuous assertion that the system is in the correct state. So that's the third pillar for zero-trust.

And then finally, the fourth pillar is the administration of all of that stuff. So we have those three. How do we get runtime artifacts? How do we deploy and control them at one time? And then how do we assert that the system is doing what we want it to? We then need to manage all of that. Who can change it? What are the changes that are made? What is the audit record of those changes? All of that stuff. So that we can then prove to auditors, prove to ourselves, that the system is in fact secure. So those are the four pillars that I think of when it really comes to zero-trust and what it means.

And so the service mesh doesn't really play at all in the pre-runtime pillar. Tetrate doesn't really play at on every runtime pillar, right? But the service mesh has a huge piece that it can provide in the runtime and the continuous assertion of the system in those two pillars. And that's exactly where Tetrate sits and provides help in achieving a zero-trust architecture. And then third, we actually sit over top, and we provide a lot of those management features that you described. So we talked a lot about the service mesh and the capabilities that it brings. But it tends to be specific to one cluster. And it's a control plane, it's a runtime thing, right?

So in an organization, we need to weave together a consistent and a coherent service mesh across all the clusters across all the sites that a place is running. And you need to map that to your existing organizational structure, right? Who are your teams? Who are your users in your organization? And who's allowed to change what? Can the frontend team go and change the backend team’s mesh configuration? Hopefully not. That's the kind of management administration that we lean over top. And so that's where Tetrate Service Bridge, our product, sits. As a layer on top of Istio weaving together a whole bunch of different sites that are deployed across a whole bunch of different sites within a single organization. And providing that centralized management, visibility, audit logs, controls, that kind of piece.

[00:32:38] KP: When you describe the sidecar, one of the important takeaways for me was that it's then at the networking level. So its programming language, application independent. Do we have the same goals in rolling out zero-trust policies? Do I have to involve developers? Or is it just at the networking level?

[00:32:56] ZB: So, eventually, you are going to have to involve developers, right? But there's a lot that we can roll out at the networking level. So to start with, we can get, for example, encryption in transit everywhere as a baseline without having to change application code. We can get very clear metrics to be able to understand who is talking to whom in our system so that we can even start to figure out what a policy that we would write would even be, right? Bootstrapping a runtime authorization policy is very hard. And so being able to look at the system and see the observed traffic flow to be able to bootstrap that policy is incredibly helpful as well. And so we see that as a second kind of key piece.

[00:33:39] KP: And when you do a deployment, are you like full service? Or is there an operator on your clients? And who's going to be at the dashboard that you give them? That sort of thing? How does everything run?

[00:33:52] ZB: Yeah, yeah. So there's a couple different ways. So we started off. I think I mentioned we work with the DoD. We work with a couple large financial institutions. And they want a totally on-prem installation, right? And so what we have is kind of a system that split into two parts that I kind of maybe alluded to, which is the first part is that the control planes, the Istio, the service mesh runtime itself. And that gets deployed in every place that the customer has compute, right? So every cluster, they're going to have to stand up that.

That been registers into that management piece, that central piece that I described. And that stood up once. For on-prem customers, that's going to be on-prem. For other folks, we actually host that as well. And so they don't actually have to do any stand up. Basically, all that they do on their side is, in the compute cluster where you're going to run the service mesh, you deploy an operator, a Kubernetes operator, and that deploys effectively just Istio. And then we have a second control plane piece that sits next to Istio that programs is fueled by pushing Kubernetes CRDs to it so you can have this installed, you can go and do kubectl get service entries and see

the materialized surface entrees trees that we're programming Istio with to do, for example, cross-cluster traffic failover.

[00:35:06] KP: And you'd mentioned a lot of fortune 2000 customers. Banking is an obvious industry that comes to mind, as well as military and defense. How low does it get? Are there certain industries that even startups are interested in your products? Or is this really an enterprise-grade solution?

[00:35:21] ZB: Yeah, definitely. So we've talked with a bunch of startups. For us, I think I mentioned earlier, it's a pretty hands-on thing to get a service mesh into an organization. So for us, that's not the primary place that we're spending our energy today. But I would say, in general, those organizations that are looking at a service mesh, go for it, right? The service mesh itself is not inherently an enterprise or a largescale thing. Instead, I would say, the service mesh, you should look at it on the axis of what is it's going to cost to implement these cross-cutting features? What's it going to cost for you to implement MTLS everywhere in my system? What's it going to cost me to get consistent metrics everywhere? That's the axis that you should use to evaluate service mesh cost or not. And then, hey, you want to come talk to Tetrate about doing that. Probably, unless you're a pretty large company, we're not going to be able to help you today. But there are certainly a bunch of resources that we can point you at and things like that to kind of aid you on the journey.

[00:36:18] KP: I know your solution supports multi-cloud systems. How common of a deployment is that?

[00:36:24] ZB: All the time. So especially when you're talking about large organizations, you have things like acquisitions. That mean that you're spread across all the different infrastructure providers that exist, right? And especially when you're talking about large financial services companies, for example, a lot of them have acquisition-based models. So they have super heterogeneous infrastructure across a bunch of different organizations that they've acquired over the past 30 years. So that's one part of the reality, things like acquisition-driven infrastructure.

Second key one is the idea of wanting to be multi-cloud. I would say that there's this false idea of wanting to be multi-cloud purely for portability, right? So you're never going to really win on compute arbitrage, right? So compute arbitrage, the idea that I'm going to like just migrate my workloads to wherever the compute is cheaper and run them there. That's not really the key driver for multi-cloud that we've seen an industry. And so, again, like I said, first is acquisitions and that kind of thing. The second key use case is allowing different teams within the organization to consume cloud specific capabilities for their domain, right?

So maybe one of the clouds is differentiated on AI and ML, right? Maybe one of the other clouds is differentiated on the cost or development speed for Lambda or compute, for things like functions as a service, that kind of stuff, right? The practical reality that we've seen in these large organizations is that they need to enable their development teams to consume the best in breed capabilities wherever it is that they run. And so it's not that one team is running across Azure, and Google, and AWS. Instead, what we see is that one organization has three teams. And one in the ML team is in Google, right? And one of the legacy teams is running in Azure, because they need some of the Windows server things, right? And some of the new, cooler, younger parts of the organization are doing stuff over in AWS, right?

This is a pretty common setup that we see. And of course, don't forget that the stuff that's actually making money is back on-prem in the data center, right? And so you have these kind of four different environments or more. And so it's pretty common that we see this divide across these environments. And so you need to bring consistency. You need to bring visibility. You need to bring control into those environments. And that's where the service mesh becomes very attractive, not just within one cloud, but cutting across and giving you that consistent control and visibility across all the infrastructure.

[]00:39:05 KP: Where do you see Tetrate five years out?

[00:39:09] ZB: Taking over the world, hopefully. No. We hope that – I think in that timeframe, we'll see the service mesh become ubiquitous, right? We certainly hope that, as part of that, we'll have a very large rise as well, right? Our goal is to deliver all the world's application traffic, right? It'll take a lot longer than five years to get there. But we plan on in five years being a really

key part of the runtime infrastructure and the runtime security of a lot of the different applications and a lot of the different companies that folks use day-to-day.

[00:39:45] KP: Well, I guess to wrap up, are there any interesting technology challenges you're preparing for along that path?

[00:39:52] ZB: Oh, so many. There's a ton. We could have probably an hour just to do a review of some of the technology challenges there, right? So just to give some of the tastes, how do you start to make sure that security policy is consistent across these different cloud providers given that they have different security models, right? The service mesh gives you some baseline to build on top of. You still need to do a lot of rationalization with the different models to build patterns to have the correct security posture as an organization that spans. So that's one area that there's both technical work as well as thought work that needs to be put into how to do that.

Things like actually distributing policy, you need to make policy decisions quickly and accurately at runtime with up to date information. That becomes incredibly challenging as the environments that we are running in get more and more distributed, and as edge computing becomes more and more relevant. Of course, edge means something different for everybody. Everybody's edge is somebody else's data center. But what we are seeing is information get pushed closer to users. And in that environment, it becomes challenging to do things like keep policy consistent up to date and accurate in enforcement. So that's one other area that there's a lot of technical challenges.

Things like performance of the sidecar proxy for every use case is a big area, right? So today, for example, Envoy tends to add about two and a half milliseconds of latency at the 90th percentile. If you just Google Istio performance, you'll see we actually have a dedicated page that gets updated every single release of Istio with performance testing and evaluation there. So two and a half milliseconds for a large class of applications is nothing. If I told you, “Hey, your developers don't have to write code that handles certificates. And they don't have to write a lot of the code to generate metrics. And all the traffic control stuff can be declarative, not in the application and configuration-driven. And you only have to pay two and a half milliseconds of latency at runtime to do that,” most people jump with that, right? That's a phenomenal trade off.

However, it's not the right tradeoff for every set of applications for every team, right? For example, that may or may not be an acceptable tradeoff in front of the database. So one of the other kind of long running technical challenges is that data plane, that Envoy sidecar performance. And making sure that that is as performant as possible. That's where folks like Intel are actively working with the Envoy community to help drive some of that performance optimization, for example. So that's maybe a third large area. There's a ton more that we can go into. But maybe that gives a little bit of a taste of some of the areas that there still going to need to be large amounts of technical work done.

[00:42:37] KP: Absolutely. Where's the best place listeners can follow the project online?

[00:42:43] ZB: So the best place that folks can go check out Istio and Envoy online or in their respective communities, Istio.io, Istio mesh on Twitter, and on a lot of social medias. Go to the join the Slack or discuss.istio.io to be able to ask questions and interact with the community there. Envoy similarly has a large community, envoyproxy.io to get you started there. It additionally has Slack and discussion channels in all to be able to dive in. And of course, both projects are on GitHub as well. That is where they live, github.com/istio, github.com/envoyproxy for those two respective code bases, projects and communities. And then finally, Tetrate is at Tetrate.io You can email info at Tetrate.io to talk with folks. You can email me at Zack@tetrate.io But that's with a CK.

[00:43:37] KP: Well, Zack, thank you so much for coming on Software Engineering Daily.

[00:43:40] ZB: Thank you for having me. It's been a fun discussion.

[END]