EPISODE 841

[INTRODUCTION]

[0:00:00.3] JM: A service mesh is an abstraction that provides traffic routing, policy management and telemetry for a distributed application. A service mesh consists of a data plane and a control plane. In the data plane, a proxy runs alongside each service with every request from a service being routed through the proxy. In the control plane, an application owner can control the behavior of the proxies distributed throughout the application.

As the Kubernetes ecosystem has developed, the service mesh abstraction has become an increasingly desirable component of a cloud native application stack. As companies enthusiastically adopt Kubernetes, they eventually find themselves with a large distributed system that is difficult to operate.

A service mesh simplifies some of these operational difficulties, as we have explored in numerous previous episodes. The Kubernetes community has grown to include lots of enterprises. Those enterprises want to adopt service mesh. Today, many of them are afraid to adopt the technology, because there are multiple competing products. It is unclear which one of these products the community will centralize around, or if the community will end up supporting multiple projects.

Over the next few weeks, we will be airing interviews from KubeCon EU 2019 from Barcelona. These interviews are a window into the world of the Kubernetes and cloud native ecosystems, which are transforming the world of infrastructure software. The most prominent theme across these shows, these nine shows from KubeCon was that of service mesh.

Why was service mesh such an important topic? Because the battle for service mesh supremacy is a classic technology competition between a giant incumbent company and a startup with far fewer resources. The Kubernetes ecosystem is beautifully engineered to allow for a marketplace of warring ideas, where the most worthy competitor wins out. In some cases, there is room for multiple products to occupy different subsections of the market.

Across these episodes, one theme that we will explore is the governance and the diplomacy of these competing solutions and how the Kubernetes ecosystem is structured to allow for harmonious resolution to technology battles. It is tempting to look at this competition between service meshes as winner-take-all. As of late May 2019, we do not yet know if it will be winner take-all. In order to predict how the service mesh wars will play out, the best we can do is to look at historical examples.

Container orchestration wars was a winner-take-all market. Container orchestration was a problem of such depth, such technical complexity and integration that there had to be a single winner for the ecosystem to marshal around. During the container orchestration wars, as Mesos and Docker Swarm and HashiCorp Nomad and finally Kubernetes fought for supremacy, many large enterprises made bets on container orchestration systems, which were not Kubernetes.

When the dust settled, Kubernetes was the victor. These large enterprises who had adopted container orchestration systems other than Kubernetes, but grudgingly began thinking about how to migrate to Kubernetes.

During the container orchestration wars, many more enterprises were sitting out altogether. They did not choose Kubernetes, or Mesos, or Swarm. They chose to wait. Enterprise technologists are smart and they can tell when a technology is immature. Although, many enterprises wanted an orchestration system to manage their Docker containers, they did not want to insert a heavy abstraction that they would have to tear out later on.

Once Kubernetes won the orchestration wars, enterprise dollars piled into the space. The cloud native community has grown faster than anyone expected, because we solved the collective-action problem of centralizing on a particular container orchestrator; Kubernetes. From enterprises, to cloud providers, to independent software vendors, to podcasters, we all share the same vision for Kubernetes. It is the Linux for distributed systems.

Within the Kubernetes ecosystem, the thought leadership tries not to pick winners. It's better for everyone if the winners are decided through competition. In order to foster competition, interfaces into Kubernetes can provide a layer of standardization along which different products can compete. Enterprises can buy into an interface without buying into any particular product.

What does that mean? Well, examples include the container networking interface CNI and the container storage interface CSI.

Every Kubernetes application wants storage and every Kubernetes application wants networking. These Kubernetes applications might not want to be locked into a particular networking, or storage provider and that provider's particular proprietary APIs. Since there is a standardized interface for networking and storage within the ecosystem, these enterprise applications that want to adopt some particular technology that's selling to them, they can be insulated from lock-in. These applications can potentially swap out one storage provider for another, or one networking provider for another. How does this relate to service mesh? In the service mesh market, Buoyant was first to market with its open source project Linkerd.

Today's guest, William Morgan is the CEO of Buoyant. Over the last four years, Linkerd has slowly grown a following of dedicated users, who run the open source service mesh Linkerd in production. Over the last four years, Linkerd has changed from its initial technology of the embedded JVM service proxy developed at Twitter, to a Rust-based sidecar data plane and a go based control plane.

Buoyant's dedicated focus to the service mesh space has won over much of the community, as was evidenced by Linkerd becoming the predominant apparel brand at KubeCon EU 2019. Linkerd hats and t-shirts were everywhere at the conference. Why did Linkerd become trendy? Ironically, it's because of a competing service mesh whose launch strategy was widely seen as an affront to the spirit of the cloud native community.

Istio was created within Google and launched with a set of brittle partnerships with IBM and other companies. Istio careened into the Kubernetes ecosystem with violent fanfare, trumpeting itself as the cloud native service mesh du jour, through endless banner ads, marketing e-mail campaigns and KubeCon programming.

Any listener to this podcast knows that I am as gullible as any technologist. I'm an idealist and I wanted to believe that Istio represented the service mesh equivalent of Kubernetes. It's from Google, it launched with a bunch of impressive logos, it has an inspiring vision, looks cloud native, smells cloud native, must be cloud native, right?

Unfortunately, Istio's early marketing aggrandizements were disconnected from the nascent realities of the project. Istio was buggy and difficult to set up. It quickly developed a reputation as Google manufactured vaporware. Nice idea, not nearly ready for production.

For Linkerd, the timing could not have been better. Istio’s romantic vision of an operating plane for routing traffic and managing security policy and measuring network telemetry had seduced the enterprise masses. Finally, they were ready for the service mesh. With their cravings unmet by Istio, these enterprises surveyed the market and quickly found their way to Linkerd, the humble service mesh next-door, who had been waiting patiently all along.

The tide has turned against Istio and towards Linkerd. The service mesh wars have just begun. As easy as it is to criticize Istio, the project is not only vaporware. Istio has a vision for a detailed operating plane, that will evolve together with Envoy, a service proxy sidecar developed at Lyft. Perhaps, Istio’s early embers had too much marketing gasoline poured on them initially, but the project could still succeed. It's not easy to build a service mesh, so it's no surprise that it had issues at its early days. Just happen to not align with how much it was promoted.

Google is the most sophisticated, well-resourced company in the world. Judging from Google's adjacent strategic messaging around Anthos and other strategic initiatives, the company has already decided that Istio will be around for the long haul. As a community, we should be grateful to witness the folly of Istio’s carpet-bomb marketing strategy. It is validation for the earnest resilience of the cloud native community, that even under the omnipresent duress of Google marketing, the community was able to collectively reject the Istio Kool-aid.

This should come as no surprise. The cloud native computing foundation resides within the Linux foundation and the Kubernetes ecosystem has been ordained with the ardent technical purity of Linus Torvalds. The CNCF was formed under the looming shadow of AWS, Amazon Web Services. The CNCF was seated with the donation of Kubernetes by Google, much like the Linux community was positioned as a rebellious movement in reaction to Microsoft's dominance, the Kubernetes community represents a fervent desire to open up the market to cloud providers beyond the tight-lipped proprietary dominion of Amazon.

With such a deep spirit of insubordination, it is no surprise that the community has rejected Istio, like a set of loosely coupled organs rejecting a foreign skin, attempting to layer itself across them. Even though the CNCF was founded by Google, the community was formed in spite of big centralized clouds, not as a marketing vessel for their products, which may or may not be open source.

Microsoft seems to understand this fact better than Google, at least in the domain of service mesh. The day after this interview with William, Microsoft announced the service mesh interface SMI, which is a project it partnered with Buoyant and other companies on to create a minimal spec for what a service mesh should offer to a Kubernetes deployment.

The SMI represents a safe buy endpoint for enterprises, who want a service mesh, but do not want to get caught in the evangelistic crossfire between Istio and Linkerd. It is in this environment that we begin our next series of shows on the current cloud native ecosystem.

Thanks to the cloud native computing foundation for putting together an amazing podcasting zone at KubeCon. Thank you to Wendy and Natasha and Dan Khan and everybody else who put together KubeCon. This was a really an amazing event and it has just improved every different conference. The attention to the podcasters needs have really been met, so I love this conference. I'm definitely going to San Diego.

Just a few announcements that are unrelated to this episode; we have a new Software Daily app version for iOS in the App Store. This is the next iteration of our app and it's got a whole lot of polish. You can use the app to access all more than 1,000 plus episodes in one place. You can find all shows related to a particular technology, like streaming data or cryptocurrencies, or Kubernetes or learning to program. You can connect with other listeners through the comments section. You can access our transcripts and our related links.

Everything in the app is free, although you can also become an ad-free listener and support the show for $10 a month, or $100 per year. You can find that subscription information at softwaredaily.com/subscribe if you want to support us. You can become a sponsor of Software Engineering Daily. If you want to go to softwareengineeringdaily.com/sponsor to air ads, or to

partner with us on content, I realize it's probably not smart to advertise that sponsorship offering right after the ad-free offering, but most people don't pay to become ad-free subscribers.

We're hiring two interns also. We are hiring an intern for software engineering and one for business development. If you're interested in either of these positions, you can send an e-mail with your resume to jeff@softwareengineeringdaily.com. You can put internship in the subject line that would help me filter that e-mail to the correct folder, in case we get dale oozed with internship offerings. That said, let's get on with today's interview with William Morgan.

[SPONSOR MESSAGE]

[0:14:46.3] This episode of Software Engineering Daily is sponsored by Datadog. Datadog integrates seamlessly with container technologies like Docker and Kubernetes, so you can monitor your entire container cluster in real-time. See across all of your servers, containers, apps and services in one place with powerful visualizations, sophisticated alerting, distributed tracing and APM. Now, Datadog has application performance monitoring for Java.

Start monitoring your microservices today with a free trial. As a bonus, Datadog will send you a free t-shirt. You can get both of those things by going to softwareengineeringdaily.com/datadog. That's softwareengineeringdaily.com/datadog. Thank you, Datadog.

[INTERVIEW]

[0:15:40.7] JM: William Morgan, you are the CEO of Buoyant. Welcome back to Software Engineering Daily.

[0:15:44.9] WM: Hi, Jeff. Thanks for having me.

[0:15:47.4] JM: We had a container orchestration wars. It turned out to be winner-take-all with Kubernetes winning. Is the service mesh market winner-take-all?

[0:16:00.3] WM: Wow, start with the really difficult questions. I don't know. I'm not sure and I think it's a little early for anyone really to be able to know. Certainly, I would like Linkerd, which I

am heavily involved with to win, I guess, in some sense. Really, I think for me it's more important that the sorts of functionality that Linkerd can give to the world is available to people.

Yeah, I don't know. I think it's tempting to frame things as a war. I don't know how helpful it is. I think it's a little early for anyone to say, “Yeah, there's going to be – one is going to take them all and all the others are going to be left in the dust.”

[0:16:41.0] JM: Kubernetes seemed like a winner take-all sensibly, because container orchestration, it’s a hard technical problem, there's lots of network effects. There's network effects in the developer community. There's all network effects within a particular company, so you don't really want multiple container orchestration systems in your company. What are the network effects of service mesh, or can you imagine a world in which there are multiple service meshes and how would that contrast with the multiple container orchestration framework potential future that we could have had?

[0:17:16.5] WM: Yeah, yeah. That's a really interesting way to think about it. I was largely divorced from the container wars. I cut my orchestration more, my orchestration teeth that at Twitter, which was a mazal-shop until that was very – that was how I thought about container orchestration, well, orchestration. I guess, we didn't really have containers as we know them today.

Then when I left Twitter 2000 – circa 2014, the rest of the world was adopting all these crazy things, Docker, and starting to see some glimmerings of Kubernetes even way back then. I wasn't present in the industry for a lot of that, so I'm a little reluctant to speak about the container orchestration wars.

I guess, one difference that I see between adopting something like Kubernetes and adopting a service mesh is that Kubernetes has a very large API surface area. It's a lot of technology to adopt, right? I think the same thing is true of Mesos. Maybe a little less so of console, which seemed of Nomad. I'm sorry, which seems to be a little simpler.

I think in contrast to that large surface area, the service mesh can be a lot simpler. I think we have purposefully with Linkerd, trying to make it as simple and as lightweight as possible and

make it as little of a commitment as possible, which runs a little counter to I think our natural tendencies as engineers, which is we want to build these amazing platforms and everything's a platform and everything is going to be take over the world.

We've worked hard to focus on Linkerd, especially being seen as more of a tool. I think in that world, although we have yet to encounter a company that runs Linkerd, as well as another service mesh, I think that's certainly a possibility. We definitely have encountered companies who run multiple orchestrators, because once something is in, it's very hard to remove it. I think especially in more mature organizations, you'll find lots of different, this historical strata of technology that they've adopted. None of it really goes away.

That's all to say, I think there may be some dynamics in the service mesh world, specific to the technology that make it a little less one-size-fits-all and a little less okay, it's going to take over the entire organization.

[0:19:36.4] JM: What should be the surface area for a service mesh abstraction, if you want to keep it lightweight and you want to keep it tear outable?

[0:19:49.5] WM: Yeah. Yeah. That's a good question too. I am always a little reluctant to start putting in abstraction layers until there's enough exploration of the space. One of the things that was announced at KubeCon, or will be announced at KubeCon, I guess, and will be live by the time this podcast goes out, is literally a service mesh abstraction called the surface mesh interface that Microsoft has been developing, that is an abstraction layer over Istio, Linkerd, Console Connect and in the future, potentially some other service meshes.

The way that abstraction was defined was by looking at the core functionality across those three service meshes, which are quite different in many ways, but they did have three core bits. One was around telemetry and observability. One was around policy. Then one was around traffic shifting. Since those were common across all three implementations and made sense as a set of functionality, that the service mesh model as a whole supported. That was a pretty national starting point for SMI. SMI as it is today, has three interfaces, obviously that's not complete and there's a lot more that has to be built out and I think that will be a long process in building out

SMI. It feels there's been enough exploration of the space that at least some of the core functionality across multiple service meshes. It's at a point where we can it.

[0:21:18.5] JM: I could imagine that working, because you have a world today where a given enterprise has five or 10 logging and monitoring and analytics tools and it's not problematic. We don't need to integrate all of these things, or I guess to the extent that we need to integrate them, maybe we can build some higher level abstractions, although that doesn't really happen.

I guess the Microsoft service mesh interface idea would be that you have some commonalities between these different service meshes. Some service meshes may have a superset of those functionalities, but they all have routing, they all have load balancing, maybe some other features; telemetry. If they have those commonalities, if that is the spec for the service mesh pattern, then we can at least build a common interface that is at a higher level around those commonalities.

[0:22:19.7] WM: Yeah. Yeah, that's right. I think it's really interesting to look at and maybe this gets back to your earlier question, why was there not a common interface defined for container orchestrators, right? Why was that so much of a A versus B versus C experience? Whereas, we're able to find these bits of common functionality across service meshes. I think that goes back to my earlier comment about the nature, or the scale of the technology and the API surface area and how much of an investment you're making in one of these things, where it's much lighter for the service mesh, than for picking a container orchestrator.

[0:22:55.2] JM: As Microsoft has been developing this service mesh interface, what have your conversations with the company been like?

[0:23:04.5] WM: Well, so it's been very much an alignment between what we wanted and what they were trying to get at. Ultimately, Linkerd users, which is the audience of people that I care the most about and whose lives I want to improve, will be helped by having an interface that is generic across Linkerd, because it allows integrations to be built on Linkerd without the integrator having to feel they're being locked into a particular mesh.

I think this opens up a wide world of integrations. One of the things I'm most excited about is there's a project called Flagger from Weave, which is a very cool idea. I mean, we've been talking about in Linkerd land for quite a while, which is okay, when you do a canary deployment, or a Bluegreen deployment, you're doing this partial traffic shifting from one version of code to the other, you should actually be looking at the success rate of the new code. If that success rate is dropping, well then he should stop the deployment. You should roll that back. That's a natural combination of two of these service message APIs; one around traffic shifting and one around telemetry.

Flagger prior to SMI worked with Istio, and I think had worked with at mesh, but those integrations were built one at a time. Now with SMI, Flagger is able to build against any of the service meshes that support SMI. That's great for Linkerd, right? That's great for the service meshes and that's great for the industry as a whole.

[0:24:36.9] JM: That's cool, because that's a higher-level pattern that you would want in any application.

[0:24:42.9] WM: Yeah. Yeah, that's right. That's right. The service mesh, it doesn't make sense for the mesh itself to give you that, because that involves a certain amount of business logic, it involves a certain amount of organizational, “Well, how are we going to do this? What's the process for deploying new code?” All that stuff, it sits at a higher level than the mesh does. It makes a ton of sense for it to be a separate project. I mean, Flagger is one use case you could also imagine any CICD system doing a very similar thing at the tail-end of a deployment pipeline, right? Rather than just cranking the code out there, okay, passed all the unit tests and integration tests, crank it out. You could do the same thing. You can do a partial rollout and measure the success rate. Then if things are going south, then stop the roll back and go back to the old code. That's one example. There's a couple other cool examples in the SMI demo of other use cases. Ultimately, it's really good for Linkerd and it’s really good for Kubernetes users everywhere.

[0:25:36.7] JM: There are some products that were out earlier than you started working on Linkerd and Buoyant. Well, I think these companies are older, but there's Kong, HashiCorp’s Console Connect. Was Console before or after Buoyant?

[0:25:55.8] WM: Yeah, Console was –

[0:25:56.7] JM: There before Buoyant.

[0:25:57.4] WM: Yeah, but it was a key value store.

[0:25:59.5] JM: Key value store. They've explained that to be a service mesh thing. Then Nginx is obviously old, but it's rebranded as having service mesh characteristics, because it was service mesh before service mesh was a thing to some extent. These older technologies, how did they differ from what you are able to do with Linkerd since you're devoted to the identity of a service mesh company? You always have, although you start with the service proxy, I guess. That's before the term was changed into service mesh. What advantage do you have over those other technologies, or how does your platform differ?

[0:26:41.6] WM: Yeah. I think the answer is the same for Linkerd for almost any service mesh. How does a service mesh differ from API gateways? How does it differ from Ingress? I think, we've had proxy technology since the advent of TCP/IP networking. We've always wanted to have proxies for a variety of reasons. The difference is I think in the set of immediate features and in the set of use cases that are supported. The focus for the service mesh of course is on what data center terms, what we used to call east-west traffic, rather than north of south. This is a communication that happens between services, rather than communication from the outside world that goes through your application then hits a database.

I think the answer is the same for Linkerd as it is for almost any of the service mesh projects out there, which is what you're asking is basically what's the difference between the service mesh and something like an API proxy, or an API gateway, which Kong was, at least originally. Or something like Nginx, which has primarily been focused on Ingress use cases. How do I get traffic from the outside world into my data center?

The answer is that the service meshes focused on what – in data center terms, we used to call east-west traffic, which is rather than north-south, which was traffic coming in from the outside world and going to our app servers and then hitting our database, instead it is traffic between

applications, right? What's driving the need for that is the move to microservices ultimately, right?

If you look at Kubernetes and Docker and things through that lens, really what those technologies allow you to do is to adopt microservices in a way that solves a lot of the pain around deployments and orchestration, right? When these companies, when companies are adopting and organizations are adopting Kubernetes and Docker, they're making it easier and easier to adopt microservices as micro services. Then suddenly, the service message becomes very relevant, right?

At its core, the service mesh has a bunch of user space proxies, at least that's the implementation we see on most of them. We've had user space proxy technology since the beginning of TCP/IP programming. It's really more a question of well, what is the set of functionality that the service meshes supports, that is useful, that's necessary in the world of microservices when services are communicating to each other?

The answer is well, the communication between services actually is quite different and it's weird in a lot of different ways. There's a lot of parallels to handling ingress traffic, to handling API gateway traffic, but it's not the same. I'll give you one example, which is on the internet, we like to use TLS, right? When your browser connects to a website, you'll see a little green icon and it says HTTPS and you feel safe and happy, because the encryption is being – the traffic is being encrypted between your browser and the website and that avoids a whole class of scary things that you don't want to have to deal with, like people intercepting your traffic and knowing what you're doing.

It's very similar concepts in the service mesh world, where something like Linkerd can provide transparent encryption and identity, right? Part of that TLS communication is not just hey, I want to encrypt stuff, but also you have to prove to me that you are actually yahoo.com, right? We have these cryptographic ways of saying, “Okay, I know that the person I'm talking to, or the server I'm talking to is yahoo.com.”

The same sense the same class of problems exists in the world of microservices, right? What you want is okay, service A talks to service B, you want both A and B to be aware of the identity

of the service. Then you want them to start encrypting the traffic. This falls into this general pattern, or concept of zero-trust networking and there's a whole fund set of things we could talk about there.

My point is the mechanics and the requirements of TLS in between services is actually quite different from the mechanics and requirements of TLS at the edge. For example, at the edge you need to rely on these public certificate authorities. You're relying on something like Verisign to put their stamp of approval and Verisign does a bunch of due diligence to make sure that presumably then, and that yahoo.com is the certificate – the people who hold the certificate for yahoo.com are actually Yahoo.

In the service mesh world, well, we don't really care about those external entities. We care about an internal entity, right? We need an internal CA. We care less about a one-way validation of A validating that was talking to B and we care more about two-way validation. We want B to know that it's A, as well as A talking to B. While it's using TLS and while certificates are being exchanged, the mechanics are and the requirements are quite different. That's a very long explanation of one of the sorts of differences of east-west, or service communication, versus north, south, or ingress communication. Does that make sense?

[0:31:40.2] JM: It does. When I talked to HashiCorp, that TLS feature is the first thing that they have gone after with Console Connect, with their rebranding of their key value store agent gossip system, which was console. They added TLS handshaking to that and called it Console Connect. Then Console Connect went through the Pokemon evolution from agent-based key value store to service mesh agent, plus TLS encryption manager. In that evolution, is there any important difference between the quality of service that somebody using Console Connect is going to get, versus somebody using Linkerd?

[0:32:34.7] WM: Quality of service. That's interesting. I think, at least in the way that these systems use TLS and provision identity, I don't think there is a substantial difference. I think there's a lot of details that are different. For example, so Linkerd let's see, we released 2.3, I think was last month. One of the things that we did was we turned mutual TLS on by default. That means if you install Linkerd in the Kubernetes cluster by default, it provisions a little certificate authority, the certificate authority is issuing keys to each of the proxies, the proxies

are rotating those keys every 24 hours, they're tied to Kubernetes service accounts, because there's a bunch of nice properties about service accounts. That all happens out of the box for you.

Linkerd doesn't have, at least not as of 2.3, a way of extending that identity outside of Kubernetes, although that's coming up on the roadmap. Console Connect by contrast, is started from the perspective of while we're running outside of Kubernetes and now we're going to give you these hooks for building into Kubernetes. If your requirement was today, like in the next 24 hours I need to have an identity system that spans external and internal. I need these things and that's your primary use case and Console Connect is a better choice.

If you're setting up Kubernetes and a Kubernetes-specific solution works for you then today, Linkerd I would say is probably a better choice. Those are the sorts of differences. It's not really a fundamental difference between how we're going to think about identity and how we're going to think about encryption. We're all trying to build on top of these well-established primitives that have been around for a long time, like TLS and X509 certificates.

[SPONSOR MESSAGE]

[0:34:28.5] JM: Deploying to the cloud should be simple. You shouldn't feel locked in and your cloud provider should offer you customer support 24/7, because you might be up in the middle of the night trying to figure out why your application is having errors and your cloud provider's support team should be there to help you.

Linode is a simple, efficient cloud provider with excellent customer support. Linode has been offering hosting for 16 years and the roots of the company are in its name. Linode gives you Linux nodes at an affordable price, with security, high-availability and customer service.

At linode.com/sedaily, you can get started with 2 gigabytes of RAM and 50 gigabytes of SSD for only $10. There are also plans for cheaper and for more money. Linode makes it easy to deploy and scale your applications with high-uptime and simplicity. Features like backups and node balancers give you additional tooling when you need it. Go to linode.com/sedaily to support

Software Engineering Daily and get your application deployed to Linode. That's L-I-N-O-D.com/sedaily.

Thank you Linode for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:35:58.0] JM: Let me posit something that I think is going to be an opportunity for differentiation. You can correct me if I'm wrong. It seems like we're in the nascent stages of more infrastructure deploying this security, or workload identification stack of open policy agent, spiffe and spire, where you have better workload identity, better zero-trust policy management. That seems to have some tough engineering problems, because you have to have the question of how do new policies get pushed out from the policy manager? How do you rotate those policies and things like that? Am I off there, or does that seem like an area that's going to be – it's going to require some tough engineering and it's going to be an opportunity for you to differentiate?

[0:36:55.2] WM: No, you're not wrong. Definitely, there are a lot of unsolved and significant challenges in that world, especially the sorts of things that spiffe and OPA address are these very generic questions around well, how do we codify policy? In the general sense, in a way that can be applied to all sorts of different policy across all sorts of layers of the stack? That's what OPA is very good at. Spiffe is the counterpart for how do we specify identity in a way that can be applied anywhere in your stack. Those are both very important problems for enterprises.

For Linkerd, we take a much more tactical approach. I don't disagree that this will be a point of differentiation. For me and for most of the Linkerd maintainers, we are a little less focused on the how do we solve generic enterprise problems? We're much more focused on okay, if you have adopted Kubernetes and you need to get XY and Z in place, how can we give that to you in a way that isn't as rapid as possible, that is as slow config and that's as low management overhead as possible?

That's why for us, it made a lot of sense to ship a CA that runs in the Kubernetes control plane, in the Linkderd control plane in Kubernetes and to do all the TLS stuff for you, so that out of the

box, you don't have to do anything. You just get, effectively zero-trust security for your Kubernetes cluster.

Now ultimately, this is a question of what's the starting point? Because on the roadmap for Linkderd is okay, give you the ability to actually use a CA that's outside of the cluster, because that's important once you get to multiple clusters and gives you the ability to extend the proxies that you can run Linkerd proxy outside of Kubernetes. That involves figuring out how to do identity.

All these things will happen, right? It's just a question of what's the starting point. Is the starting point okay, I have to solve this very hairy enterprise problem in a way that's generic across a lot of use cases, or do I have to solve this very, very focused Kubernetes-centric problem? We've taken the latter approach.

[0:39:07.0] JM: Google started a project called Istio, that's been out for I think a year now. How production-ready is Istio?

[0:39:17.4] WM: I'm definitely not going to speak for the Istio project.

[0:39:19.8] JM: Well, they didn't either. That makes two of you.

[0:39:25.0] WM: I did really enjoy your podcast with Eric Brewer. I thought it was really interesting. I had a comment on Twitter about this too. I thought it was interesting how non-committal he was around when you asked that question. Well, so on the one hand look, as an engineer, right? What does it mean to say anything's production-ready? You're always a little nervous. You're always in the back of your mind, you're like, "Well, I know it has X, Y and Z and I think for these cases not going to."

It's always I think as an engineer, a little scary to say, "Yes, this is ready for production." Once you're an executive at a major cloud provider, it struck me as a little strange that it didn't have a more, I guess confident answer. I don't want to read into that. I thought it was interesting and I had a good conversation on Twitter about it.

Ultimately, there are companies that are running Istio in production, right? There are companies that are running Linkerd in production. By that definition, it's production ready. Then by many other definitions, there's always things that you – there's always more things that you want.

[0:40:33.7] JM: Now, I'm not trying to criticize Istio relentlessly, I just think their marketing strategy is hilarious, because of how pervasive it is, despite the fact that – In terms of design differences, one positive thing I will say about Istio is and one way, I think it notably – it's different than Linkerd. Not better or worse, different is the fact that Istio is plugged into the envoy ecosystem. Envoy has that level of traction as an open source project, where there are multiple companies who have multiple big companies with tons of resources.

Lyft having the interest in envoy. Envoy seeming like a thick – it's a well-developed ecosystem. It's going to develop more. Istio maybe gets to draft off of that. Now we're already seeing the downside of that, because there's some overhead to, I guess the deployment complexity of Istio and I have no idea how much that has to do with envoy. I don't really understand. I don't know what your sidecar model is. In any case, you own the entire stack. I mean, it's open source. You've been focused on Linkerd for a longer period of time.

Anyway, I see these two ecosystems as the Istio ecosystem and the Linkerd ecosystem as both having looking at from an engineering perspective, both having a viable potential future, at least to the extent that this is not a winner-take-all container orchestration war. Give me your perspective for the design differences between Istio and Linkerd that people should consider.
[0:42:20.1] WM: Yeah. How to unpack this? There's at least four answers that came to mind while you were asking that question. I guess first, I should take a step back and say I don't want to talk about Istio just to talk about Istio. To me in an ideal world, I would never think about Istio, because what matters to me primarily is can I give Linkerd users the tools that they need to be effective at their jobs, to build these systems, to build the platforms for their developers and their organizations? None of that, none of that critical focus requires thinking about Istio at all.

On the other hand, we are asked about Istio on a very regular basis, so we've developed these – a way of thinking about it and set of answers and things like that, so I find myself having this conversation much more than I would really want to.

Let me address the envoy thing first. Envoy is a very widely adopted project, and I think it's a pretty good project, although I haven't used it myself. Part of the reason why it's so widely adopted is because it's a great building block. It's like, okay, we've got a wheel. On top of the wheel, we can build a motorcycle and we can build a car and we can build a semi-truck and they all have different purposes, but they're all using this awesome wheel under the hood, or not under the hood, I guess. Under the axle.

[0:43:43.5] JM: Engines have wheels, I think. Gear is a kind of wheel.

[0:43:46.8] WM: I'm a computer guy, so I don't know. I don’t know how that stuff works. I just press the button and make the thing go. Ultimately, I don't think it matters what the proxy is under the hood. Linkerd doesn't use envoy. We use a thing that we call Linkderd proxy and it's written in Rust and there's a bunch of cool stuff that we like about it. Ultimately, I don't think it really matters.

When you're adopting Istio, when you're adopting Linkderd, when you're adopting some service mesh, what the underlying network stack is I don't think really affects you in a really direct way, even if you have an existing envoy deployment, using something like contour for ingress, or something like that. It doesn't really change a lot for you. Every usage of envoy, because it's such a good building block, it's coupled with some system to drive envoy, right? That's like, okay, contour is going to drive it for ingress, or Istio is going to drive it for the service mesh.

When you're interacting with things, when you're operating things, when you're monitoring them, you're thinking at thinking about it at the Istio level, or you're thinking about it at the contour level. The fact that it uses X, or Y is much less important than is it solving these problems for you, right? Is it doing it in a way that is not expensive, either for the machines to run, or more importantly, for your brain to comprehend?

If it is, right? If it's if it's solving problems for you and it's not insane to operate, then congratulations, you made a good technology choice. If it's not, then it’s not. It's really more about I think that level of analysis, than okay, what are the components under the hood, or under the axle? Finally, to get to the Istio question, so I know a lot of the Istio engineers and

they are really good engineers. They‘re probably much better engineers than I was back when I was an engineer. Now I'm just an e-mail –

[0:45:44.8] JM: You’re a suit.

[0:45:46.7] WM: Yeah, that’s right. I’m just a suit. The thing that always seems so interesting to me is I think Istio is a bad project. I think it's bad and I don't mean that in a way that the people who work on it are bad, because I think they're really good, right? The question for me is what's the disconnect? What happened there? How did you get engineer who were so good working on something that is not good, right? As measured by a bunch of factors, as a measured by human beings actually trying to use it in practice.

I think the answer comes down to and you touched on this earlier, I think it comes down to marketing. I think there was so much marketing that was placed around Istio and there was such a force behind it that it smothered. It was this wet blanket that just smothered any possibility of having a really honest, or direct relationship with users, right? As a result, you get something that is at least, I've got to hedge all this because I never run Istio in practice. I'd say the majority of the Linkerd momentum these days is people coming from Istio and so I’m like, they're telling us why. The level of complexity, the level of resource overhead, all that it's just – it's really much more than it than it should be.

In that sense, I think the value props of Istio are great, like the things that it does are very similar to things that Linkerd does, at least for now. Those are all good things. The cost is so high. I think it does a disservice to the users to say hey, this is a service mesh and look, it's super complicated, super heavy, but you're going to have to do this, because the service mesh it's going to be this critical part. I think that's really bad. I think that really is a disservice for the users, because now everyone has in their heads they're like, “Oh, service mesh is a complicated thing. Yeah, we don't want to do that.” Put that off to the last possible moment and it doesn't have to be in that way, right? That's not a core attribute of the service mesh. That's an attribute of Istio.

With that said and again, every person who I've met works on Istio is awesome and it's great. That's the weird tension in my mind. The focus that we've taken with Linkerd is quite different.

For us, we're not trying to convince you to get onto a cloud provider, where we're just going to operate at – we're going to operate Linkerd for you and some of the complexity doesn't matter, or whatever.

Our adopters are the operators. It's the same folks who were bringing in Kubernetes, who are bringing in – in some cases, bringing in Docker and containers to their organizations. They are the ones who are using Linkerd, who are bringing it in. By that lens, we have to make Linkerd as simple as possible to understand and to operate, because if we can give you the value props of the service mesh, if we can give you the features around telemetry and security and reliability, but the cost of using those features is so incredibly high, then I think we haven't done our jobs.

The focus for us has been, even at the expense of features has been how do we make Linkerd the simplest, lightest and fastest service mesh that we can, so we can give you the value props, we can give you all the cool things on the mesh, without you having to burn a whole bunch of system resources, or even worse, use your precious brain cells on understanding a bunch of really complicated APIs.

[0:49:14.4] JM: Istio really made me question what is my role as a podcaster, or journalist, or whatever. The reason that that happened is because two KubeCons ago, Google started carpet-bombing the cloud native community with Istio marketing. I think this was Copenhagen if I recall. I was like, “Okay, well I didn't know that much about service mesh back then.” I was like, “Okay, cool.” This is Google's service mesh equivalent to Kubernetes or Tensorflow or whatever. They're going to take the strategy that they did with Kubernetes and Tensorflow. They're going to carpet-bomb the developer community with this positive Google love. This is the open source, open cloud candidate from Google.

Then I would talk to people and be like, cool, so I'm hearing all these Istio talks and seeing all this Istio marketing. Like, do you know anybody who's using it in production? They’re like, “No, I don't know anybody, but I'm sure it'll happen eventually.”

Then the next KubeCon rolls around KubeCon North America, the one in Seattle, and it was the same thing. I would be having conversations with people and be like, “Man, I'm excited about Istio.” I'm like, “Okay, do you know anybody that's actually using it? Do you know anybody that's

having success deploying it?" They're like, "No. I mean." I'm not going to be using Istio for a while either. I'm not going to be – we don't need a service mesh right now. We're barely even getting our Kubernetes strategy off the ground. We're two or three years away from having a service mesh.

The thing is in two or three years, Istio is not going to have these performance issues. Istio is going to be one-click to deploy, it's going to be simple, it's going to have the security policy, it's going to have load balancing, it's going to have everything you need out of a service mesh. It made me wonder, is there anything wrong with that? Is there anything wrong with the pre-emptive marketing, with the fake it till you make it, with the build the network effects before you have the software to effectively justify those network effects?

I don't know the answer to that question. That's part of what made me interrogate Eric Brewer, even though I really – I like Eric Brewer. I admire him. I may have left him with a bad taste in his mouth and he may think I'm the Kara Swisher of the software engineering world. I don't want to be that. There is no other marketing check on the software engineering marketing world. It's all manufactured consent, were all purchased through sponsorship dollars, me included. For me, it was really existential to see that. I still don't really have an answer to it, but –

[0:52:10.1] WM: Istio caused you to have an existential crisis.

[0:52:12.5] JM: It did. Well, which is great, which is great. I mean, I'm happy to have my journalistic integrity called into question when I look in the mirror, thanks to Istio.

[0:52:28.6] WM: Wow. Okay, maybe we'll get there with Linkerd one day.

[0:52:30.8] JM: Not to turn this into the therapy session, but I just wanted to unpack where I was coming from with that interrogation of Eric Brewer. Okay, I'll be fine.

[0:52:41.0] WM: I think it's that's a great question to ask. I think, is it okay to do that? Well, I think, okay from which perspective? Certainly it's okay from the perspective of Google, because you want this thing to be out there, right? It's going to take some time to build it. Well, let's lay the groundwork, right? We got to make this project successful. If what you care about is making

Istio successful, then okay, we're going to use every means at our disposal. Why not? It's open source. It's not like we're forcing people to use it. We're just heavily encouraging them.

That's from the other perspective is to me, it doesn't seem right to put something out there and not be forthright about its properties, right? Obviously, I'm super biased in this conversation, right? Because I've been watching this from the perspective of someone who was like, "Well, hold on guys. We already had a service mice over here." You can't trust anything that I say, because I've got a horse in the game, or in the race.

Trying to put that aside as much as I can, it just doesn't seem right, right? This is like the vaporware approach to things and it doesn't seem right to do that. Life would be a lot easier. Yeah, let's put it this way, life would be a lot easier for me if Istio was awesome and I could just hop on that bandwagon, right? That would solve a lot of things. I don't want to have a service mesh fight just to have it. It's not a great use of time and energy to duplicate all the stuff, but I feel we're in the situation where we have to do this because it's not solving things for people that we care about. Like for the users who are out there with Linkerd, if I could tell them, "Hey, just use Istio. It'll solve all your problems."

Like, "Then we don't have to maintain this thing and I don't have to sync all this money into all this open source engineering." Honestly, that would solve a lot of problems, but we can't do that. It's not right. Ultimately, I feel we have a duty and a responsibility to not just Linkerd users, but to the Kubernetes ecosystem as a whole to give them what we know is valuable, right? We know that everything about the service mesh is hugely valuable, because we've seen it. We've lived this life, even back in the days of Twitter when there was no Kubernetes and there were no sidecar proxies. What we were building was effectively the service mesh. A whole bunch of details are different, but we know firsthand and everyone who works on Linkerd knows firsthand just how valuable this stuff is. It just doesn't seem right to not give the world the best possible implementation of that.

[0:55:18.9] JM: You raised 10 million dollars from Google Ventures. I know that Google Ventures is not a direct arm of Google, but I found it Shakespearian and hilarious that you did end up raising 10 million from Google Ventures.

[0:55:37.5] WM: Yeah. It made a lot of sense. I'm wearing a couple hats in life these days. Let me take off my Linkerd maintainer hat and I'll put on my Buoyant CEO hat, because it was Buoyant that raised the money. Although, really what we're doing with it is investing it right in Linkerd. I will admit that there's a certain amount of enjoyment that I got out of the fact that there's ultimately Google money that's going to Linkerd. Though, I guess there's also, that money goes back to Google, because we do testing on some of our testing on GCPs. I don't know. This big, that can set your cycle somewhere in there. All that I know is that the landlords of the Bay Area are the ones who are ultimately damn rich off of the service mesh, right?

The true winners of the service measure wars are the Bay Area landlords. It was a really good match for us. GV had a history of investing in a really good open source infrastructure companies. CarOS being the primary one in our minds. The partner that we worked with there, Dave [inaudible 0:56:35.9] was great, understands the space really well. They also made it clear to us that although they had the letter G in their name, they were not – like you said, they're not a strategic arm of Google. They make these investment decisions based on what they think is going to win in the market, what they think is going to have the best technology, what they think has the best team around it. By that measure, they wanted to invest in Buoyant. Again, speaking was my Buoyant hat, we didn't need the money, but the relationship made a lot of sense for us for those reasons.

[0:57:08.0] JM: Do you have any predictions for the next few years, the next I don't know, three to five years, however far you can predict somewhat confidently into the distance about the, I guess the interaction between the major cloud providers, or the degree to which the wars between AWS and Google will spill out into the open source world, or interactions between, or people going into multi-cloud? What's your perspective on the different major cloud providers? Any thoughts that I can get from you that I wouldn't hear anywhere else?

[0:57:47.0] WM: Oh, probably not. I think, by virtue of focusing so heavily on the cloud native audience, I see Linkerd adoption in Kubernetes clusters on a variety of clouds. That's what I understand most viscerally. I think the question of the delicate inter balance between the cloud providers ultimately comes down to what is the enterprise hybrid cloud strategy and what makes sense there? It doesn't make sense –

[0:58:16.7] JM: Sorry, do you mean hybrid cloud or multi-cloud?

[0:58:18.8] WM: Let's say multi-cloud, multi-cloud. You're right. It's much better to see, I'm already getting myself in hot water.

[0:58:24.0] JM: Well, the hard part is the on-prem plus cloud thing, right?

[0:58:26.8] WM: Yeah. No, I mean multi-cloud. How critical is that? Because if that's really critical for companies, right? If it's really critical, and not just some companies for a large class of companies, because there are company like PagerDuty or something that have to be multi-cloud, because being multi-zone is not enough for them, multi-region is not enough. If something goes wrong, they have to be available no matter what. That's a special case, right?

For ordinary companies that don't have quite as strict a requirement around that level of availability does multi-cloud and make – is that a nice-to-have, or is that a core thing that we're going to have to have as we move into the next couple years? To me, I think that's what it comes down to. Because if that doesn't matter, then network effects, which you've brought up very, very early on are the things that matter the most, right?

Okay, we're on Amazon already, well let's just keep going with Amazon. Then at mesh will win over everything. Problem solved. If it does matter, okay well now, things get a little more interesting, right? Because now, abstraction layers become a lot more powerful, because you are going to have to deal with multiple clouds at the same time.

Do I have any special insight? Can you hear anything from me about this that other people couldn't tell you probably with much greater insight? No, no. I am in the little service mesh bubble and I can barely see beyond that.

[SPONSOR MESSAGE]

[1:00:01.7] JM: You probably do not enjoy searching for a job. Engineers don't like sacrificing their time to do phone screens and we don't like doing whiteboard problems and working on tedious take-home projects.

Everyone knows the software hiring process is not perfect, but what's the alternative? Triplebyte is the alternative. Triplebyte is a platform for finding a great software job faster. Triplebyte works with 400-plus tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves the hiring process by saving you time and fast-tracking you to final interviews.

At triplebyte.com/sedaily, you can start your process by taking a quiz. After the quiz, you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple on-site interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte because you used the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more since those multiple on-site interviews would put you in a great position to potentially get multiple offers. Then you could figure out what your salary actually should be. Triplebyte does not look at candidate's backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. I'm a huge fan of that aspect of their model. This means that they work with lots of people from non-traditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There's very little risk and you might find yourself in a great position getting multiple on-site interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out. Thank you to Triplebyte.

[INTERVIEW CONTINUED]

[1:02:21.9] JM: Let's wrap up with just some business strategy conversation. This cloud native space of company building, I don't have much historical context on the enterprise software industry. From what I do know, it seems a very different market than anything in history, because you have these major cloud providers, you have these gigantic enterprises that are going through a digital transformation.

It seems like a great place to be in the company that you are in, the company that you're building, because the market is rapidly expanding and it's meeting you where you are building towards. That doesn't mean that your strategy for going to market is straightforward. It seems like doing sales, managing deals, perhaps even deciding what is your structure for support engineering, or feature engineering, or the degree to which you're going to be a service integrator, versus just somebody who's building features and throwing them over the wall for somebody to buy through a marketplace. All these questions seem complicated to me. Tell me your current strategy for going to market.

[1:03:42.8] WM: Sure. Yeah. Let me take off my Linkerd and linking hat and put on my Buoyant hat.

[1:03:47.0] JM: You already did that.

[1:03:48.4] WM: Oh, well I had replace –

[1:03:49.8] JM: You're wearing that and your suit. You shouldn't be wearing a hat and a suit at the same time.

[1:03:54.6] WM: Well, and my monocle, okay. Full CEO mode. Yeah, so you're right. It is complicated. So far, I think Buoyant has been very lucky in that the stuff that we have touched upon with Linkerd was at the right place at the right time. The service mesh started to take off. A lot of this honestly was from our friends at Google, by promoting Istio so heavily, while that rising tide also lifted our boat, which was nice. We can take a little bit of credit for it.

Early on, it wasn't clear that focusing on Kubernetes was the best approach and so on. We were a little smart about it. Oh, yeah, we also spent a lot of time getting off the JVM. Obviously, that was either a genius move, or a genius move that had been preceded by a dumb move.

Leaving all that aside, we've been we've been lucky and that everything that we've done in the open source world has been very – has garnered a lot of attention and that's making it easy to fundraise. Obviously, there's only so far that that can take you. There's two levels here. The first is one of the things we're definitely seeing with Linkerd is the very early adopters of Linkerd

could do everything themselves, right? They would take it, they'd download the code, they compile it, they would deploy it and then only occasionally would they reach out to us and be like, “Hey, we need help with this.”

We're starting to encounter now, and I think this is a sign that the service mesh market is maturing. We're starting to encounter a set of companies who just need help, right? They don't have the ability to take on massive infrastructure, or even not so massive infrastructure projects and feel confident in themselves.

We're starting to find a set of company who just needs help with Linkerd. From my perspective, we have two choices here. One is to say, “Okay, well you're on your own. Good luck.” The other is to say, “Okay, we can help you. We can we can do support, we can do services, we can make you successful.” It has to be done in a way that's economically viable for Buoyant, right? That there has to be a commercial relationship there. If what you care about is being successful when adopting a service mesh, well we can be the company to help you do that.

That is ultimately supporting services. I think from the macro picture, it's rare to have a company, not impossible. It's rare to have a company, a very fast-growing large company, at least that it's based purely off of supporting services. My view is those are critical. Those are critical components. Those are not sufficient, but they are necessary because you have to give people mechanisms to be successful with your projects. I think that's particularly true of the infrastructure world, where it's not – it's not like you drop this thing in and okay, “Hey, we've installed Slack, or hey, everyone can sign up for Slack now and we don't have to change anything.”

You've got to do stuff. You got to do stuff. As easy as we make Linkerd to get started, pretty soon you're like, okay, well we've got our existing CA and we need to use that. Then okay, well now we need to deploy the Linkerd proxies outside of Kubernetes and we've to get identity working. It gets complicated when you start extending into, especially when you extend into what I will call legacy infrastructure and trying not to use that in a pejorative sense. That's the first answer.

The second answer is which I think is the more compelling answer and speaks to maybe the bigger vision that Buoyant has, is that the word service mesh does not occur anywhere in Buoyant's mission statement. Buoyant's mission is purely around helping human beings get to the point where they can be really successful in adopting microservices, right? That's independent of Kubernetes, is independent of Docker, it's independent of any of that stuff.

The entire industry, if you step back, the entire industry is going through some massive transformation, right? As a transformation in how software is built. It's something you can't stop, right? At least for the vast majority of companies, you are moving on to the cloud, and when you move on to the cloud, you have to do this stuff. You can argue, you can fight, but eventually you're going to be there.

We have to give people the tools to make that happen. Service mesh is certainly one part of that, right? Linkerd will solve a bunch of very engineering-focused, engineering-specific challenges for you, but there's a lot more to it than that. This again, it goes back to my experience that Twitter, when Twitter moved into microservices, again no containers, although we had the JVM and we had C group, so we had the containerisks. No Kubernetes, but we had Mesos. We had a whole lot of services that changed everything about the company, right?

Just forget about the technology, the way that engineering teams were communicating with each other were operating was totally different, the way that HR worked was different, because the way that the teams were structured was being different. The way that finance worked was different, because now we had all these services and some of them were really expensive to run and some of them were cheap. Every aspect of the company started changing. That's what I think, that's the really interesting set of things that Buoyant, as opposed to Linkerd be can be helpful with, right? Can we help organizations embrace that change? Not just adapt to it, but embrace it. That requires a lot more than just solving the technology under the hood.

[1:09:21.2]JM: It sounds it would be awesome to just hire a bunch of support engineers and deploy the support engineers to helping out with the large enterprises that have Linkerd problems, because not only is that probably profitable, immediately profitable, or near close to immediately profitable. You get a great feedback loop into what enterprises need from you,

which will eventually allow you to expand into market adjacencies that you'll see coming, thanks to your enterprises.

[1:09:54.0] JM: That's a really good point. We're a bunch of open source nerds. We have no idea what happens inside, or at least we originally did not have an idea of what happens inside the engineering mindset of someone who works in a really big company that has these 24-month roadmaps and all these initiatives. We had no idea. Every commercial relationship that we add at Buoyant has tremendous value. The money is nice of course, but just having access sitting in those meetings, understanding what exactly is happening on these and what the timeline is and what the roadmaps are and what the challenges they have to face, that's been hugely valuable for us. That's another big, big benefit of developing a really strong support and services relationship with your customers.

[1:10:36.0] JM: Let's close on a just an engineering question, because you alluded to getting off of the JVM. I remember at KubeCon, I guess two years ago at this point, it was the Austin KubeCon. The night I got there, there was an event because you were announcing your new service mesh conduit, or your new service proxy conduit. Conduit came out and I was like, “Okay, wait. They have to service things now, or proxies, or meshes, or something. It's like, this one's in Rust. Then over time, it became Linkerd2, where you rolled it into Linkerd or something.

It seems like it's worked out well for you in retrospect. What I'm wondering is did the strategy play out as you intended, or does it seem like a – does your strategy seem wise in retrospect? What was an unconventional thing? Let's spin up a second product, which gives us a green field to play around in and then we get to roll that into the old product, or replace the old product with the new product. I mean, it looks pretty nice in retrospect. I just don't know if that was how you envisioned it upfront.

[1:11:40.7] WM: Yeah. Well, that's the hard question to answer. Did we do this right? Even in retrospect. I think, there's a world in which – We were looking at Rust back in 2015 before Linkerd even existed. We had some early prototypes in Rust and we were like, “Oh, this is really cool.” At that point the language was changing. The code would stop working, because the language had changed underneath it. We're like, okay.

We don't really want to be on the JVM, but it doesn't feel we can really build on Rust right now. We had just come out of Twitter, so which had all those technology built out. It had already production tested all of it for us, until we were able to tell the story early on by building on what I would call the Twitter stack of Finagle, Scala, Netty, the JVM. We were able to tell the story very early on of hey, this is production ready, because look, Twitter uses it, scale, Pinterest uses it, scale. There were a bunch of companies that were using that stack and scale.

That's a hard bar for a startup, especially to get over is like, hey it's ready for production, because you're just some no-name startup there. That was the real value, I think. Maybe, there's a world in which we never did that and we just stuck it out with Rust and we could have avoided having to do a rewrite last year, which is effectively what we did.

[1:12:55.2] JM: You started out with Java.

[1:12:56.7] WM: We started out – No, no. I don't know. Rather than starting with the Twitter stack, which was Scala on JVM.

[1:13:01.8] JM: Oh, sorry. Rust in the beginning.

[1:13:02.0] WM: Yeah. Maybe we should have just sucked it out. I don't know. That's an interesting thought exercise. Either way, what had happened was Twitter and Linkerd got very, very popular very rapidly. It was clear to us, even certainly well before Austin, that the JVM was going to be too much of an impedance mismatch for what people wanted.

There were people who deployed Linkerd as a sidecar, the Linkerd JVM as a sidecar. Even though it was at a 150 megs, they were like, "I don't care, because these are all Java apps and it's 2 gigs." Then there were other people who had these 20 meg Go microservices and they're like, "I'm not going to add a 50 meg sidecar." I think there's a second inflection point there, where we could have just been like, "You know what? We're going to address enterprise stuff only and enterprises all on JVM, so we're just going to be the enterprise service mesh and adjust that JVM market."

[1:13:54.6] JM: I want to work out with GraalVM. Maybe GraalVM would've lowered your footprint.

[1:13:59.5] WM: Yeah, it's certainly possible. Though Graal has not been – we've been keeping our eye on it. It has not moved as fast as – it still, last I checked, not able to run Linkerd, because there's some complicated stuff that Finagle does especially does and that Netty does. It's not just regular JVM code. This is super advanced JVM code, right? Scala and Netty and all this asynchronous stuff.

Anyways, although as a side note, IBM's open J9, JDK has improved things a lot from Linkerd, from Linkerd JVM. Okay, anyways, so yeah, getting back to the conduit thing, so we were like, okay, it's not going to be a long-term solution for us to gain. We have to get off the JVM. We were really nervous about what we were doing. We knew we were going to build on top of Rust and on top of Go. Those were natural choices. I actually wrote a long article about this for InfoQ. If you search for InfoQ and Linkerd V2 or something, you'll be able to read exactly what I'm going to tell you.

[1:14:56.5] JM: You got to promise, publish your next article on Software Engineering Daily.

[1:14:59.2] WM: All right. Deal. Definitely.

[1:15:01.7] JM: It's on InfoQ. You hear that?

[1:15:04.2] WM: Yeah, that's right.

[1:15:04.9] JM: InfoQ is my Istio.

[1:15:08.6] WM: Well, I'll just have to have a –

[1:15:09.8] JM: Just kidding. I'm competitor-focused. I worked at Amazon.

[1:15:14.8] WM: Right. Good. We wanted to do something that we knew was going to be risky, which was rebuild stuff on top of Rust on how to go ,and we were worried at the time if this didn't

pan out that it was going to destroy the Linkerd brand, because Linkerd was getting a ton of adoption. Even with the JVM being an impediment.

We decided to call it conduit, to just sandbox – sandbox a weird term. To make it a separate project with a different brand and to see how much interest we could get doing that, and to treat it as an experiment. If it played out, then great. If it didn't, then fine. Linkerd is still going to continue on. That was gosh, I can't even keep my timeframes right anymore. I think that was the tail-end of 2017. Then all of 2018 was putting tons of resources into conduit, first half of 2018.

It was working. Everything was working. We had early adopters using it, there were bugs, there were weirdnesses, but everything about it just felt right. It felt like this is what the service mesh should be. By the middle of the year, we were like, “Okay, it's clear this is the path forward for Linkerd, right? Let's merge this thing in.” Then in September of last year, we launched officially G8 Linkerd 2.0, which was that same conduit code base with a whole lot of extra development on top of it. That's where the majority of our new adoption has been and certainly the majority of our engineering effort has been on Linkerd.

Now we're at 2.3, next month we’re at 2.4. We're catching up with the 1.X feature set. We had a huge feature set, thanks to Finagle with 1.X. With 2.X, we're now catching up 2.4 we’ll have traffic shifting, so you'll be able to do Bluegreen deployments and things like that. It'll have SMI support. It'll have all these cool things. 2.3 already had all the TLS stuff that we can never actually do really well with the JVM, because it was so hard to get it deployed as a sidecar among other reasons. That's the story.

Yeah, there was certainly a period of time where it was confusing for people to have these two things, especially from one small startup. Hopefully, we don't have to talk about conduit ever again, we just have this beautiful Linkerd and everyone uses it and just very happy with it.

[1:17:22.4] JM: Yeah. No, I think it worked out really well.

[1:17:24.8] WM: That's the back story.

[1:17:26.2] JM: I mean, from conversations I've had with people about Rust, it really – it does seem like the right language for the job. It gives us a bunch of really nice things. It gives us the ability to – the Rust is in the data plane, so it's the actual proxies. The control plane is all on Go. What Rust gives us – it gives us a couple things. First, it gives us the ability to write native code, which is great, so we can be as fast as possible, which is important for a user space proxy. There's no garbage collection. There's none of that. It's basically as fast. The Rust people will clean it faster than C or C++.

Then it also gives us a lot of awesome security guarantees around the way that it manages and enforces memory usage, sidesteps a whole class of buffer overflow exploits and things that historically have been very problematic with C or C++. Then for the Scala programmers in us, it gives us these really nice higher level abstractions, where we can do things like, hey, all the memory for this request – we're allocating memory as this request comes in. Then you want to then free them. Everyone in the request terminates. In that way, you keep your latency profile really, really sharp, because you're doing all the memory allocation, you're amortizing that really evenly over the request flow. Okay, but you can do that in Rust by like – I'm going to get myself in trouble, because I don't actually program in Rust.

[1:18:47.5] JM: Careful. The Rust community is serious.

[1:18:49.3] WM: By deleting that feature, or just terminating the feature, whatever the right word is, sorry Rust people. Then because everything has been – the ownership and everything has been enforcing these nice zero-cost abstractions, you can delete all the memory for that request in a really nice straightforward manner that doesn't involve thousands of lines of spaghetti code. It's been a really nice move for us, especially coming to it as Scala programmers, who were familiar with abstractions and higher order programming.

[1:19:17.3] JM: Well, let's leave it at that. William, thanks for coming back on the show. It's been really fun talking.

[1:19:21.2] WM: Jeff, it's been a pleasure. Thank you for having me.

[END OF INTERVIEW]

[1:19:27.3] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source, it's free to use and GoCD has all the features that you need for continuous delivery. You can model your deployment pipelines without installing any plugins. You can use the value stream map to visualize your end-to-end workflow. If you use Kubernetes, GoCD is a natural fit to add continuous delivery to your cloud native project.

With GoCD on Kubernetes, you define your build workflow, you let GoCD provision and scale your infrastructure on the fly and GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team, and they have talked in such detail about building the product in previous episodes of Software Engineering Daily. ThoughtWorks was very early to the continuous delivery trend and they know about continuous delivery as much as almost anybody in the industry.

It's great to always see continued progress on GoCD with new features, like Kubernetes integrations, so you know that you're investing in a continuous delivery tool that is built for the long-term. You can check it out for yourself at gocd.org/sedaily.

[END]