EPISODE 1416

[INTERVIEW]

[00:00:00] JM: William, welcome back to the show.

[00:00:01] WM: Thank you very much, Jeff. It's great to be back.

[00:00:06] JM: The platform of Linkerd is something we've explored a lot in previous episodes, of course. And last time you and I spoke, it was nice to hear that there's been a large growth and traction for the company. And the first thing I want to ask is, has there been any significant change to the technology of Linkerd and Buoyant? Or has there just been a broader acceptance that a service mesh is something that is desirable for the majority of companies?

[00:00:42] WM: Yeah. The technology has not really changed. I mean, we've added more features, we've made it more powerful, we continue to expand the set of capabilities that Linkerd has, but the core technology remains the same. We got a chance to do it the right way, when we did Linkerd 2.X, and those choices on the tech side have have held strong over the past couple of years. But there has been a change, I'll say, there has been a big change in the sorts of people who are coming to Linkerd and kind of audience, at least what we've seen.

[00:01:18] JM: What is that difference in user base?

[00:01:23] WM: Well, I forget, when you and I last spoke is probably a good 18 months ago, I think it's been a while. But for a very long time, the audience coming into Linkerd, were people very enthusiastic about the technology, people who are really excited about Kubernetes, were excited about service meshes, who were excited to get their hands dirty, and we're adept, often at deploying and operating open source projects. And that's great, it was wonderful. But over the past 6 to 12 months, we've seen people come in who are much less excited about the technology for its own sake, and much more in the camp of, I believe, in the value prop and I want a service mesh – I don't actually care that much about the details, I don't actually want it to be exciting and interesting operate, I kind of want it to just work. And for me, not to have to think about it. It's been a pretty dramatic difference in those two, in that shift in the audience.

[00:02:26] JM: With that change in user base, are there feature requests? Are there like differences in in what these newer companies actually need out of a service mesh? Or is it kind of the just the same monitoring and rate limiting and same features that they're looking for?

[00:02:47] WM: That's a really good question. I don't think it's really a dramatic change in the types of features they want. But it's certainly, the ways that you're using Linkerd is different. The early adopters were very Kubernetes centric, and you could kind of get by and say, "Hey, look, this thing works on Kubernetes. And if you're not using Kubernetes, it's not going to work." That was okay. Folks were coming into it today, often have Kubernetes running alongside other environments. So, we've seen a lot of interest in the ability to run the Linkerd's data plane outside of Kubernetes. And that's something that we're starting to really sink our teeth into. It's non-trivial for a couple of reasons. Less on the proxy side, and more on the kind of overall system side, but that's been a big change.

[00:03:32] JM: Now, when you say data plane, are you talking about the path that the data takes from individual service or container instances, and wherever that data is being stored, like a database for Buoyant to retrieve from?

[00:03:48] WM: Yeah, maybe we'll give the listeners a brief review of the service mesh in case they haven't had this drilled into their head a million times a day as I have. But the way a service mesh works is you've got a control plane, and you've got a data plane. And the data plane are these tiny little proxies that sit next to every service in your Kubernetes cluster in the form of the sidecar container. They intercept the calls and never calls to and from each of those services.

The control plane is some machinery that just sits off to the side and kind of helps you coordinate the data plane as a whole. And so, that model gives you the ability to do all sorts of features without the application really having to be aware of it. So far, for Linkerd, at least, we've been very Kubernetes focused. So, we've kept that the way you deploy the data plane, while you do it as a pi, you do it as a container in a pod, and it works really easily on Kubernetes and doesn't work anywhere else. To get to the feature we're calling mesh expansion, we need you to have the ability to spin up the proxies and not the control plane, but the core data plane proxy itself somewhere outside of Kubernetes and have it connect to a control plane in a way that makes sense in a way that propagates identity, in a way that allows you to do MTLS, and kind of all the other fun features that Linkerd has. So, that's what we're starting to dig into.

[00:05:09] JM: So, how does a service mesh fit into an engineer's day to day life? Is it something that I'm consulting to just look at overall service health? Or am I typically waiting for it to alert me that something is wrong? Am I using it to define certain service level objectives? Give me an idea of how, from what you've seen that from users, how a service mesh fits into somebody's average day?

[00:05:43] WM: Yeah, so I think it depends on what type of engineer you are. So, if you are a developer, and you're writing code, and your job is to build the business logic that's powering your organization, then ideally, you actually don't interact with the service mesh at all or, not directly. And the goal, at least in my mind, the goal of something like Linkerd is to make it to the developers are blissfully unaware of kind of platform features.

Now, on the other hand, if you are an SRE, or if you are on the platform team, and you are someone who's tasked with building the platform on which the application runs, and that platform could include Kubernetes, can include Linkerd, and could include CI/CD, it could include some code repository or code hosting model, then your interaction with service mesh is pretty direct, right? Because now it's one of those platform features that you're building. In that case, there's a couple ways that you would interact with it.

One, as you point out is with metrics. So, the way Linkerd works is with those proxies sitting next to every application, every network call kind of transparently going through those proxies, Linkerd has a wealth of information about not just the state of the network, but the state of the applications. Linkerd understands HTTP, and understands gRPC, it can tell you whether you're getting successful responses or not, you can feed all those metrics into some kind of monitoring and alerting system. And crucially, you can do that in a way that's consistent across all of your applications. It doesn't matter what language they're written in. It doesn't require any particular libraries to be instrumented. And in a way that it gives you a uniform set of metrics.

Now, I can't look inside the application. I can't tell you kind of the internals. But I can tell you, here's the success rate, here's the response latency, here's the request volume, here's how they're changing over time. So, that's one big way. Another big way that you can interact with it is on the control side. So, this would be things like, well, retries and timeouts, and load balancing, and circuit breaking, and things like that. So, here's features where I as a platform owner, maybe adding some configuration to Linkerd and say, “Hey, for this service, I actually want you – when calls happen, I want them to to go in this particular way.” Because I know this is a flaky application or whatever it is.

And then the third kind of primary way, is from the security perspective, where you as either a security engineer or just a security conscious engineer, are using Linkerd to do things like do mutual TLS, between all the pods in your cluster, or to instrument policy and say only these services are allowed to talk to these other services. So, if you're on the platform side, you fall into one of those categories, you're often interacting with a service mesh pretty directly. If you're on the developer side, hopefully, you read a blog post about it and have a shocked expression on your face, and then close the tab, and never think about it again.

[00:08:42] JM: Can you talk a little bit more about TLS? And why that's relevant to an infrastructure operator, and just give a little bit more context for the relationship between a service mesh and TLS?

[00:08:57] WM: Yeah, that's a topic near and dear to my heart. And in fact, I wrote a guide recently, I think it's called like the Kubernetes Engineer’s Guide to Mutual TLS or something. So, there's a lot to say about TLS. I’m going to try and restrict it to the basics. So, in general, TLS, is Transport Layer Security, it’s the thing that we use to secure connections, to secure network connections, right? And so, the most common example that we're all hopefully familiar with is when you use your web browser, and you're talking to a web server, and you see a little green lock icon, that means, “Hey, you've got a secure connection.” And what is secure mean? Well, it means number one, it's encrypted, but that's not enough. It also means that you have validated the identity on the other side. So, you’ve authenticated who's on the other side. And then there's also a third guarantee around integrity.

So, those three guarantees together, mean that you have a secure connection. And that means among other things, that bad guys and middle can’t snoop your traffic, when you're looking at cat pictures on Reddit, no one can tell what you're doing with some asterisks in there. And no one can kind of impose in the middle and do what's called the man in the middle attack, person in the middle attack, maybe we should say, and change what you're seeing so you actually get dog pictures back. So, that's TLS in general.

And then, in the world of platform engineering, and Kubernetes, especially, it turns out that one of the things that TLS is pretty good at is being a really convenient mechanism for ensuring encryption of data in transit, between pods in your cluster, and for validating identity on both sides. In this world, we modify it slightly, we call it mutual TLS. We validate both sides of the identity. So, my brows are talking to the web server, talking to softwareengineeringdaily.com, my browser validates that softwareengineeringdaily.com is who it says it is. But your website doesn't actually care about my browser, right? Like it doesn't care what my identity is. That's handled through other mechanisms.

For Mutual TLS within a Kubernetes cluster, when service A talks to service B, they're both validating each other's identity, they're establishing a secure connection, they're communicating across that connection, and then you as the operator as kind of the platform owner, you now know that I have secure communication, and someone breaks into the cluster, and they sniffed the network, they're not going to be able to get that data, which is important, especially if you have sensitive data. And there's some other nice guarantees, as well.

So, summarizing that very long, essay, Mutual TLS is a very convenient mechanism for getting encryption of data in transit, especially within a Kubernetes cluster. And a service mesh, like Linkerd actually, is a very nice way of giving that to you. In fact, we can do it even though TLS is like complicated and hard and annoying, we can actually give it to you in the context of a Kubernetes cluster, without you having to do a whole lot of work. In fact, we enable it by default. So, the moment you install Linkerd, and you mesh your pods, you actually have MTLS between all mesh pods.

[00:12:06] JM: And I'm curious, are there other things you could bundle into? I mean, MTLS, just getting MTLS as a kind of thrown into the benefits of having a service mesh deployed to your cluster is pretty nice. Are there any other features that you're thinking of building or could be other nice to haves to kind of bundle in with the service mesh functionality?

[00:12:38] WM: Yeah, so as soon as you have MTLS, one thing that you have is you have identity. You have like this really cool cryptographic proof of identity on either side. And on top of that identity, you can now start building policy. So, you can say, well, it's nice that your service A and you're trying to talk to service B and you want Linkerd to encrypt that connection and authenticate and all that. But is A allowed to talk to B? So, we know with the latest release of Linkerd in 2.11, we give you mechanisms where you can control that. You can say A, it’s not allowed to talk to B or only this type of communication is allowed to happen within the cluster. So, that's built on top of the same mutual TLS identity. So, it's not tied to network identity. It's all sorts of nice reasons why we want to do that. It fits into this model called zero trust security where the pod itself is the enforcement point. So, we're not relying on the host, we're not relying on the network, we're not relying on some centralized service. We're doing all of our security enforcement at the most granular level.

So that's one big feature. Of course, there's a lot to security beyond this. This allows you to capture connection level security, but there's also request level things you might want to do and request level policy that MTLS isn't really going to help you with and other classes of things like that. But yeah, policy is a big one for us. Like I said, 2.11 introduced that at the connection level and the 2.12 is probably going to continue that thrust, especially looking at policy around outgoing connections and not just incoming connections.

[00:14:10] JM: How do you test and verify and validate the security features?

[00:14:17] WM: Oh, it's open source. So, you just let it out there. And if someone has a big security incident, then you say, “Whoops”, and you fix a bug. The system works. Another hard-hitting question. We do it through a combination of things. Number one, Linkerd is a CNCF project. We're fortunate to be a CNCF graduated project, which is kind of a top tier of maturity. And one of the things that entitles us to as a CNCF regularly subsidizes audit, security audits by third parties of projects. So, we actually are kicking off our our next audit of Linkerd, I think in a week, so it's happening soon.

That's one thing, if we have kind of third-party audits by security professionals. Another thing is we go through a security. Sorry, we go through a code review process and no code can get into the system without it being signed off by someone who knows what they're doing. And then the third thing is we – and this is going to sound hand wavy, but we think really hard about security. We educate ourselves as best we can, and we learn from best practices across the industry. So, why are we doing Mutual TLS instead of doing something else? Well, because TLS for all of its warts, is an industry standard, and we can adopt it and there's libraries that we can rely on, and we're not like rolling our own. So, there’s a lot of decisions like that.

[00:15:42] JM: I remember coming by your office back when you were in San Francisco, and you were working on the dashboard for Buoyant. And I'd love to know more about how usability and user experience has played into what you've built out of the user layer, what the actual operator is interfacing with to configure Linkerd and make updates to it.

[00:16:15] WM: Yeah. So, user experience in general has been a big, big thrust for us, since the very beginning. Students of ancient history will know that Linkerd 1.x was the first version. And it's actually very different from 2.x. We started with some Scala libraries that came out of Twitter, because we had also been Twitter engineers. And it was very, very powerful, but also very, very complex. And when we rewrote things in 2.x, circa 2018, starting in 2018, one of our big goals was to make a system that was simple, and that was especially operationally simple, because we saw, what we saw with 1.x is we saw people who bought the value, the value prop of the service mesh, and believed in it, but had a lot of trouble implementing it. And that seemed needless.

So, since that 2018, which is now what, 30, 40 years ago, in subjective time, we have been focused on how do we – every time we add a feature, how do we make it simple for the user? What is the UX? What is the operational model? You have to have in your head to understand this feature, and as a result, Linkerd is known, I’m very proud, in fact, that Linkerd is known to be the simple service mesh, the one that is actually “easy to operate”. And I say, “easy”, because the reality is running any software, it's really hard, right? Especially if it's software that you are on call for, especially if it's software that other people are relying on, and you need to fix problems. It's difficult. No matter how simple we make Linkerd, it's still difficult.

So, a lot of that feedback, and a lot of our creativity and energy over the past year or two, around how do we operate Linkerd? And how do we help people operate Linkerd at scale? And how do we make life really, really simple for them, has gone into Buoyant Cloud, which is our

SaaS product. Buoyant Cloud, I'm happy that there's like, there's a free tier, so you can try it out, and you don't have to just believe me, but a lot of the time and energy that we've put into Buoyant Cloud has been with the idea of like, okay, let's say you have to run Linkerd, and it's as simple as we can make it but running software still sucks? How can we ease that burden for you? Can we take on alerting and monitoring for you, so you don't have to set that up yourself? Can we take on visibility into the mesh itself, so you don't have to build those dashboards? Can we take on things like expiring certificate alerts, so we give you plenty of warnings, so you're never surprised by the fact that you set a certificate up a year ago, and it suddenly expired? Can we give you the ability to understand some of the more sophisticated features like policy? Linkerd has a very powerful policy mechanism. And with any powerful system, it's possible to shoot yourself in the foot. Can we make it so it's very clear what's happening with policy? These are all the ways that our focus on user experience and our focus on simplicity has shifted our product and has driven this roadmap focused on the end user of Linkerd.

[00:19:20] JM: And as the workloads have gotten more heterogenous, as you've gotten a wider range of customers, presumably, some of them are on legacy deployment systems that aren't as standardized as just everybody doing Kubernetes. Has it become more difficult to serve the range of infrastructure use cases with the same user experience?

[00:19:46] WM: Well, we've certainly seen an expansion and the sorts of workloads that Linkerd is exposed to, even within the world of Kubernetes, it used to be, well, you'd have a deployment and you'd have a stateful set, and you'd have whatever, daemon set. And now there's much more sophisticated things like Argo rollouts and these other operators that are starting to come into the system that Linkerd has to be aware of. Kubernetes itself has some, I don't want to say works, but it has some sharp edges when it comes to things like container ordering. And so, anyone who's using cron jobs, like, we have to do some special stuff for. And then the other thing we've seen a lot of this is multi cluster. So, that's been a big expansion in usage, and that has its own special set of concerns, especially as Linkerd tries to mediate not just on cluster calls, but also calls between clusters, which could potentially be separated across the entire Internet.

So yeah, even in the world of Kubernetes, the sets of workloads that Linkerd is being exposed to is definitely growing. It's been manageable so far, I think, in part because Linkerd, tries to be pretty, pretty basic in how it attaches to the rest of Kubernetes. And Kubernetes, in turn, is pretty good at being a platform on which things are built. But it's not getting easier.

[00:21:11] JM: I'd love to know about some of the engineering challenges that you've encountered, maybe organizationally, or technically. As the company has scaled, it's interesting to hear that the core technology has remained relatively stable. It's been pretty much the same core technology, but I'm sure there’s stuff you have to fix day to day or various minor features you have to add. And managing that in tandem with a growing company has its complexity. So, I'd love to know about how you're managing the company and how you are kind of dividing up responsibilities.

[00:21:50] WM: Yeah, so there's technical complexity, and then there's like organizational complexity. And the two things sometimes have parallels, but often don't, especially since organizational complexity involves human beings, which are kind of squishy objects. So, I'd say on the technical side, what's been nice is that kind of the original model for Linkerd, at least for 2.x is one of horizontal scaling. So, yes, we've made the proxy itself a lot faster. We've made it able to handle much higher throughputs. We've done a bunch of investment there. And if you look at the benchmarks that we publish, you'll see the results of that effort. We've gone through and like tweaked the memory allocator, in Rust and things like that, and done a bunch of experiments to optimize that.

But the core design, you know, is one of horizontal scalability. And so, as you add more workloads to the system, well, you get more proxies, and the control plane itself can scale up and you can run multiple replicas of the control plane. So, we haven't really hit – on that side, we haven't really hit a bottleneck. On the organizational side, now I was talking about Buoyant, the company, I think one thing that we have started doing, which we probably should have done a long time ago, is having a really cohesive roadmap between Linkerd and Buoyant Cloud. It used to be that the two things were pretty separate. There's like a Linkerd roadmap, and it kind of went at its pace. And then there's a Buoyant Cloud roadmap and it went on its paced.

Nowadays, we've gotten a little smarter andas design and development goes into Linkerd, it's taking feedback from Buoyant Cloud, and obviously, as Buoyant Cloud, it's developed or taking a lot of input and feedback from Linkerd. And part of the reason why that's possible is because we run Linkerd ourselves. So, Buoyant Cloud itself runs on Linkerd. So, we are not just creators, we're also consumers of dog food. In fact, it feels like we're swimming in dog food. It's like, yeah, there's a lot of dog food.

But that's actually been really helpful. And having that kind of very direct connection has been gratifying because shipping open source, man, it's like shipping CDs. You kind of release, and then like, you put it out there on a Friday, and you're like, “Okay, there it is, world.” And then you go home, and you come back next week, and people have bug reports or whatever. You don't get very direct feedback. And the feedback that you get tends to be pretty negative, right? If you can actually run your service mesh yourself, then you get some deep insight. If you are suddenly on call for this thing that you're developing, suddenly very different relationship with your end user. That's been extremely helpful for the project. I think Linkerd has gotten a lot better because of that.

[00:24:39] JM: So, can you give me more insight into what it's like to run an infrastructure company? I guess, I'd love to know if there are some unknown unknowns from my perspective. What keeps you up at night? Is it ability to sell enough infrastructure software ability to keep up with competitors? Is it fear of technical outages? Or do you just feel like everything is running hunky dory at this point?

[00:25:11] WM: Oh, everything's great. It's a self-managing machine. I don't even have to do anything. I just get to sit around and do podcasts. Now, the reality is, it's a real balancing act. It's not just an infrastructure company, it's an open source infrastructure company, which has its own set of very unique and difficult challenges, because we have to balance two things. We have to balance, first of all, the open source community and the needs of the community, and we want that to grow, and we want Linkerd adopters to be happy and successful, ideally, publicly, loudly, publicly, successful.

And then of course, there's Buoyant, the business, which has to make money and that money is coming from Linkerd adopters, but we have to do that in a way that doesn't sour the community. So, a lot of that balancing act, and this is what I spend a lot of my mental energy on, and what we're flushing out, and I think doing a pretty good job on has been navigating that and knowing when to invest in community and when to start the sales conversation and when not to, because you don’t want to try and sell something to someone who doesn't want to buy it, right? That's not what sales is. That's maybe like, used car sales. But that's not what modern software sales is about.

Modern software sales is like, it's a collaborative and helpful relationship, right? If you really want to be successful at this, yes, you're exchanging value, goods for value, but that value has to actually be valuable for you, especially since pretty much everything we do is on a subscription basis. So, even if we manage to trick you into doing something for one year, well, if you go away the next year, that's not great. So, we have to make you successful.

So, navigating that complexity between the open source and sales, probably is more, has been more of a challenge to the company than just – or has more defined the way the company works, than just the fact that it's infrastructure software, although certainly infrastructure means that there's a set of constraints and instead of capabilities that the company has to develop.

The other thing that's been really helpful for us, I think, is that attitudes towards open source have changed. And I forget whether you and I have talked about this in the past, but I remember when I started in open source, which was a long time ago, we were passing around stacks of floppy disks, installing Linux. We were installing Slackware version, whatever, zero point something. And open source was like this thing. It was like, we're sticking it to the man. I don't have to buy Windows anymore, because I can just install Linux, take that Bill Gates. And it was this real kind of almost anti anti corporate thing.

Over the past, I'm going to give away my age, but over the past 30 years, or whatever it is, something like that. The relationship between the commercial and open source has gotten a lot friendlier. I think today, if you look at the really popular open source projects, there's always a company behind them, that is investing in those projects. It's not a nights and weekends thing. It's not volunteers. It's very rare for it to be like this volunteer only effort. Usually, there's an economic engine behind the project that's powering the growth of the project. And once you get comfortable with that, it's good for the project, right? It's like it incentivizes, it gives a project oxygen, you have developers, you have maintainers, you have people who are being paid to make this thing better. And it can be done, it can be done in a way that is not anti-community.

[00:28:56] JM: Have you been able to get any insight into how people are running their Kubernetes clusters, but just the from their macro perspective of seeing all these different Linkerd deployments, is the vast majority like people just doing EKS clusters or standing up their own Kubernetes on EC2 instances or using VMware? Have you seen any particular trends in how people are deploying and managing their Kubernetes clusters?

[00:29:31] WM: Yeah, we see a lot, especially when we have a commercial relationship with you, we get as deep as we need to get in order to make sure you're successful with Linkerd. So, we do occasionally see people who are running their own Kubernetes, not using a managed Kubernetes. We do occasionally see people who are doing stuff on Prems. Actually, we're seeing more and more of that, but not because it's growing, I think it's just because we're being exposed to a different audience. But the majority of folks that we see in the Linkerd, certainly in the open source community, are using a hosted Kubernetes, often on a cloud provider, whether it's SIBO, or one of the hyperscale providers. And they have kind of the standard Kubernetes challenges. How do we actually deploy applications effectively here? And what are the developers need to know? And how do we build up the platform and that stuff? The actual cloud provider under the hood does not actually affect the nature of those problems that much.

I'll tell you one change that we have seen, I mentioned this a little bit before, it's a shift of multi cluster. So, we introduced multi cluster functionality in Linkerd in like 2.9, whatever that was, like, that was a long time ago. And then like, it didn't seem to really be used. And I was like, “Man, there's so much buzz about multi cluster and no one's using it.” Now, two years later, or whatever it is, we see people doing serious multi cluster deployments. I think people have realized that Kubernetes itself, it's hard to do stuff in a single cluster, right? Like multi tenancy within a cluster is hard for a variety of reasons. CRD's are – the namespaces are nonhierarchical and CRD is like our cluster wide and stuff like that.

So, if you're doing multi tenancy, well, you got to do it with multiple clusters. If you're doing like high availability, well, you probably want things in different zones, while that's multi cluster. And you know, hey, it turns out two different teams in the company, both used Kubernetes, at the same time, where we acquired this other company, and they're using Kubernetes. Well, now you're multi cluster. There's all these reasons that you end up with multiple production Kubernetes clusters, sometimes running the same services, sometimes not, and then having to figure out how to communicate between them, which sometimes is trivial, it can be done right at the network layer. And other times, it's basically impossible and has to be well done impossible, impossible to do L3, L4, and you have to do something like Linkerd to do that. That's been the most noticeable trend, I think.

[00:31:59] JM: Has there been anything in the evolution of how people are managing their clusters that has surprised you? Is there anything, any any new developments you've seen?

[00:32:13] WM: Anything that surprised me? It’s interesting. Let me think about that. Has anything really surprised me? I don't think so. It's been a lot of the same challenges, and often the bigger challenges are the organizational ones. Yeah, Kubernetes has a learning curve, but like, you can learn it. And once you've learned it, like, kind of the operational semantics are clear, hopefully. So no, I don't think I've seen anything – I don't think I've seen anyone do anything really surprising. The changes that I've seen are things like, multi cluster being like kind of a bigger pattern. I certainly seen an increase in sophistication around security. It seems for a long time, it seemed like, people were struggling primarily to get things to run on Kubernetes, and security within the cluster was an afterthought. And then suddenly, now we're seeing security teams come in, and talk about Linkerd, and they're like, “Oh, we need Linkerd because we have all this shadow IT, and there's all these applications running on Kubernetes, and we need a way to secure it and to do micro segmentation.” I'm like, “Wow, that’s great.” I would never have had that conversation two years ago and we never would have had a security person come in and deploy Linkerd for those reasons. So, that's been a big shift. But is it surprising? I don't know. Maybe I'm just not easily surprised.

[00:33:41] JM: Have there been any issues where the service mesh deployment ends up causing latency or, or memory consumption issues, that prevents a cluster from operating effectively?

[00:33:59] WM: There's definitely, there have been bad cases where like the proxies has gone into some bad state and keeps growing and growing, and eventually has to be killed. Has it ever gotten to the point where it like takes down an entire cluster? Probably, probably. If we leave outside, if we leave aside like bugs and things, typically no, typically no. I mean, Linkerd does require resources to run, right? You're adding these proxies everywhere, every call between A and B now has to go through two proxies, not just one, but two proxies. So, that's going to add latency, and these things are going to consume memory and they consume CPU. But typically, the real costs are the applications themselves. So usually, the biggest source of CPU and memory is the application or if you're running Prometheus, and that's what's hard from all the memory in your cluster.

[00:34:58] JM: Is there any interface between Prometheus and the service mesh?

[00:35:05] WM: We expose metrics in a Prometheus compatible fashion. And then we have a Linkerd vis extension. You can little Prometheus and little Grafana, and gives you the dashboard and stuff like that. So, yeah, it's not critical, it's not required for the service mesh to run, but it's been kind of our default time series database, at least for like the little on cluster dashboard that you can get with Linkerd. Just throw in Prometheus and Prometheus sort it out.

[00:35:36] JM: Can you tell me more about your actual infrastructure and how Buoyant Cloud is deployed and architected?

[00:35:44] WM: Yeah, so, there was a little debate internally, when we first got started. It’s like, well, we don't really need micro services, and we don't really need Kubernetes. Like, it's going to be pretty straightforward application. Maybe we could just do something normal, on one hand. And another hand, it’s like, well, we also really want to dog food Linkerd, which means we're going to have to use Kubernetes, and we're going to have to write things to services, and eventually the dog fooding went out.

So, it's a little over engineered, frankly, because we really want to have it as a platform for running Linkerd. But you know, it runs on Kubernetes comprises multiple services, they speak HTTP and gRPC, they talk to each other, they serve up a web dashboard, believe we use React on the front end. We use the actual, the way you connect your cluster to Buoyant Cloud, is you add a – there’s a Linkerd extension you can add that’s a little agent and the agent kind of like talks to Linkerd and reports what it sees back up to Buoyant Cloud.

It's not actually that weird or interesting, it's pretty straightforward, at least in the world of, of Kubernetes. But we try not to make it particularly – we’re not actually doing anything that's that crazy. All of our real technical kind of deep tech stuff is in Linkerd itself. In Buoyant Cloud, it's pretty normal looking Kubernetes application. With one big exception. I think that, because now I realized, we actually do have a lot of metrics data. So, we have to do some fancy stuff, to handle all the metrics data, because one of the things we do is we take everything that your proxies are doing, we report that home, we back up to Buoyant Cloud, we host it, we separate it all out by customers, and no one's allowed to see anyone else's data and stuff like that. We give you graphs, and you can go back, and time windows, you can compare, latency versus rollouts, and a bunch of stuff like that. That part of it gets a little more sophisticated.

[00:37:39] JM: Can you tell me about standing up a successful support program? Because you you obviously have to service customers that have a wide range of issues. Do you have to build particular instrumentation that allows you to assist those people or build internal applications? And yeah, how does your support strategy work?

[00:38:05] WM: Yeah, this is great. So, a big part of what we do for enterprise customers is we help them run Linkerd, right? When we started doing that, we kind of adopted the same model that everyone else does, which is like 24/7 on call. And so, if Linkerd breaks, you call us, and you wake us up at three in the morning, and then we have until this time to get back to you. And then we're going to help you like reboot your pods, whatever. And after a while doing that, we were like, "Wait a minute, if we had a little more insight into what your application is doing and to save your service mesh, then things would be a lot better." That actually was the genesis of Buoyant Cloud.

So, the reason why Buoyant Cloud is a management project, it's like a Linkerd management product is because not only does it help you operate Linkerd, it also helps us. So, you give us access, we can log into your Buoyant Cloud account, we can see the state of your alerts, we can – if you have a very fancy support relationship with us, we can start calling you and we can start being proactive and we can say, "Hey, look, Jeff, like you got a certificate that's going to expire next week. Here's what you need to do X, Y and Z." And then then we flip the model around. So instead of it being this reactive thing, it's like, the emergency room, where, okay, wait until you break your leg, and then you can come to us and we'll help you fix it. We can actually be proactive. We can say, "Hey, look, you got to eat your apples and your vegetables and you got to drink milk or whatever you're supposed to do to be healthy."

[00:39:33] JM: Yeah, I'm looking for the days when Linkerd can tell me what to do on my diet.

[00:39:38] WM: We'll add it as an extension. You just have to add the proxy into your bloodstream somewhere, attach it to a 5g chip and you should be just fine.

[00:39:50] JM: What's been your experience of going fully remote as a company?

[00:39:54] WM: Oh, gosh, it's been interesting. I don't think we would have done it had the pandemic not kind of forced our hand. But I'm really glad that we did. It's actually been really good for the company. Even before COVID hit, we had been hiring folks in all sorts of different parts of the world, which I love. I love that model for a company. But we were still pretty SF based. And then, once the pandemic was going, and it didn't seem like there was an end in sight, and maybe it still doesn't, we decided to bite the bullet and just be remote first. And it's meant, wasn't meant. Well, it's meant that we're all spending a lot of time on Zoom meetings, which is not great. But it's meant that well, at least for me, it's meant that I actually have a lot more time that I can spend with my family and with my kids, which has been gratifying. I can see them throughout the day, even if just for 60 seconds here and five minutes there, they come home from school, and I can say hi to them. So, that part's been great.

And then, for the company, I think one thing that really helped us was a lot of the work that we're doing is asynchronous by nature anyways? Linkerd, big community project, we've got maintainers, who live in all sorts of different parts of the world. And so naturally, there's this asynchronous philosophy that didn't actually change stuff, that much there. Buoyant Cloud can be a lot more synchronous. And so, we've shifted the model there. We're having like live conversations over Zoom, or whatever it is. And over time, we kind of adapted to the model. I think the hardest thing for us was the junior folks. The more junior folks needed more structure than what was really available. And so, they were the ones where we really had to go out of our way to address. The senior folks who were able to stay engaged and kind of keep their eye on the big picture, and who knew more about the system. That, they were okay, and in fact, they kind of appreciated it. So, we had to change the way we were handling our junior folks.

[00:41:54] JM: Well, just to wrap up. Can you talk a little bit about what you're working on right now across the company?

[00:41:59] WM: Yeah. Gosh, more of the same, more Linkerd, more Buoyant Cloud. One thing that I'm really excited about for the next release of Linkerd is client-side policy. So, we're finally going to get to the point where we can do things like circuit breaking or egress control. There's fine grained policy on the roadmap, so we can do policy based on HTTP verbs, or gRPC methods, and a little bit further out from there as mesh expansion, which we talked about, pretty heavily in the earlier part of the show. Those are all things that I've been dying to get to for a long time, and we're finally at a really good place for them. You can tell there's kind of a security theme that's happening for Linkerd. A lot of our adopters are very, very heavily into like, “Okay, we need TLS and we need micro segmentation and all that stuff, and you need it to be zero trust.” So, that’s all feeding right into the roadmap, we tend to be pretty customer focused. And then Buoyant Cloud is right along there with it, like every new Linkerd feature, we're going to give you a really easy to manage it and understand the state across clusters and fit into your GitOps workflow. So, it's just more, more, more and more of the same.

[00:43:09] JM: Well, William, thank you for coming back on the show. As always, it’s a pleasure and nice to get an update on Buoyant and Linkerd.

[00:43:10] WM: Yeah. Thanks for having me. And don’t forget to go to linkerd.io and get your favorite Kubernetes service mesh.

[END]