EPISODE 1101

[INTRODUCTION]

[00:00:00] JM: A service mesh provides routing, load balancing, policy management and other features to a set of services that need to communicate with each other. The mesh can simplify operations across these different services by providing an interface to configure them and to make settings like on these load balancing or policy management aspects of services. And there are lots of different vendors who offers service mesh technology. AWS has App Mesh, Google has Istio, which is open source. Buoyant has Linkerd, which is also open source. HashiCorp has Consul Connect. And unfortunately, not all of these service meshes play well together. They don't necessarily communicate with each other.

Luke Kysow is an engineer at HashiCorp where he works on Consul Connect, and he joins the show to talk about service mesh usage adaption and possible strategies for maintaining multiple service meshes within a single organization. Of course, at a large enough company, these different teams setting up different service meshes is troublesome and it would be useful for these different meshes to be able to communicate with each other.

Before we start, I want to mention, I am looking for companies to invest in. If you're building a company for developer tools or a company that requires heavy engineering, send me an email, jeff@softwareengineeringdaily.com. And also, if you are looking to write for Software Engineering Daily, you can send an email to write@softwareengineeringdaily.com, or jeff@softwareengineeringdaily.com. I'd love to hear from you.

[SPONSOR MESSAGE]

[00:01:42] JM: Errors, and bugs, and crashes happen all the time across my software. Most often, these crashes have to do with obscure exceptions that come from React components, failing to render on the client device. Source maps and stack traces would be useful, but in many cases, I'm not able to identify the root cause because the error is occurring on a client

device. It's not on my infrastructure. And what can I do about that? I can use Sentry. Sentry.io can quickly triage and resolve issues in real-time.

Visit sentry.io/signup and use code SEDAILY to sign up for two free months. You can simply choose the platform that you're integrating with. Sentry works for every major language and framework, from Rails, to C#, to Java, or React Native, and you just install the SDKs. You integrate it with your application. After that, errors will be caught by Sentry and you'll be alerted the moment that they occur so that you can triage, and resolve, and keep your users happy with functional applications. If you're building an application and you want to monitor your errors and your performance, try Sentry by visiting sentry.io/signup. And new users can use code SEDAILY for two free months.

Thank you for being a sponsor of Software Engineering Daily, Sentry, and you can go to sentry.io/signup and use code SEDAILY if you're curious about it.

[INTERVIEW]

[00:03:15] JM: Luke Kysow, welcome to the show.

[00:03:17] LK: Thanks for having me.

[00:03:18] JM: It's 2020. Describe what role a service mesh plays in 2020.

[00:03:24] LK: Yeah. Well, I think this all started back when we started on this path to microservices, and people were looking for ways to manage kind of this exploding complexity, because as you know how like in-apps and you need to manage all the infrastructure for that. And that's where Kubernetes came out of. And that's just enabled this complexity to keep going. And so where service mesh is it's a recognition that the network was the next place where we needed to put software in between everything. So where the service mesh comes in is eventually you reach this level of complexity where you need higher level tools to manage things. For putting Docker containers on a host, eventually we were like, "Okay. Well, we need a

higher level tool to manage this." So Kubernetes or Nomad. I think you're reaching the same conclusion where you're like, "Okay. Well, now we have all these things running everywhere in disparate networks. These networks don't necessarily communicate with one another. Some of them have duplicate IPs where aps or application lifestyle is more complicated. And so I think that's where service mesh comes in, and it's like, "Okay. Well, we can't hardcode these things in our app. We need operators or managing thousands of services. We need this higher level abstraction, this programmability of our network where these calls are going so we can manage these things better." I think that's where service mesh comes in in 2020.

[00:04:49] JM: Who wants a service mesh? Does everybody need it or are there some cases where some lover of abstractions and unnecessary complexity, and I've installed my service mesh and I'm instantly regretting it.

[00:05:04] LK: Yeah. I think that's a very interesting question, because I think the answer is not everyone wants a service mesh nor needs a service mesh. And that's also probably why although there is a lot of work going on in this area, the actual production usage isn't really there and not taking up at the speed that we saw something like Kubernetes happened.

Anecdotally, I was at like the Service Mesh Con at KubeCon last year, which is you think a room full of people that are kind of on the cutting edge of Service Mesh, and there's probably 200 people in the room and they ask, "Put up your hands if you're running service mesh in production," and probably like 10 or 20 people put up their hands. I think that is an example of like how people don't necessarily need a service mesh right off the bat, and you're going to add a lot of complexity when you start putting proxies in front of everything.

I think the folks that need a service mesh, usually like the service mesh usage kind of starts at the edge. It's like, "Okay. I don't need it within my Kub cluster, but I need to connect my VMs to my Kub cluster, or my other Kub cluster to another Kub cluster." So you start to see the mesh happen around the edges. And then as that starts to be useful, then you can see it coming into play within the apps themselves where you're running like proxy everywhere in your Kub cluster.

I think you got to reach a level of complexity where it's worth taking on that additional complexity so you do get those higher level constructs that you can program and you also need a team of people that is ready to spend time to dedicate to operating this mesh. So you're looking at like some larger organizations, I would say, is when it starts to become really useful.

[00:06:41] JM: What needs to be configured when I'm setting up a service mesh?

[00:06:46] LK: Yeah. I think like the first place to start, at least most of the service mesh, is you need a proxy in front of all of your apps, which is really easy in Kubernetes and pretty hard on some of these other platforms. So that's kind of like the first place that you start is, "Okay. Well, how am I going to deploy this proxy alongside my apps?" And in Kubernetes, this is really, really easy. All the service meshes will have this mutating web hook where every time you schedule a new pod, it will come in there and it will just add in a sidecar to there, and now you have your proxy running alongside your app. Then once you have that, it's a matter of the control plane that controls like what happens, like what does proxy does when it receives traffic? So in Kubernetes, this is just running like a deployment.

Those are kind of like the two pieces that kind of you need to have in order to start using that service mesh. And then you need a use case and something that you want to do it with, taking metrics or doing canary deployments, blue-green deployments, or doing migrations, or fail over. So those are kind of the components.

[00:07:50] JM: You work at HashiCorp. Describe HashiCorp's strategy around service mesh does far.

[00:07:56] LK: Yeah. HashiCorp is an interesting path towards it, because we've had Consul, which has been our networking and service discovery tool for a really long time now. And that kind of cut its teeth in the day where folk were moving to a lot of infrastructure as code and they were running microservices in lots of nodes and VMs and bringing those up, and they're starting to be more than just like four servers in your DC. You're starting to have like hundreds or thousands of servers. So that's where Consul came along where it was the idea where you run

Consul on all of your nodes. You know now what all of your nodes were, because this is before schedulers, and you know what services are running on them and you can use – We detect failure, and that's used to route to all your different service across your data center.

That's where we started. And then we started seeing Kubernetes, the rise of Kubernetes come, and it's the same kind of problem where you have these nodes coming in and out of existence and you need to attract them. However, now these are pods that are coming in and out of existence and you need to track them.

What HashiCorp's vision for pretty much all our products is multicloud, the idea multicloud and multiplatform. So the idea that you can use the same tool, the same workflow across multiple clouds and multiple platforms. Whether you're provisioning an EKS cluster, an AKS cluster of VMs, you can use Terraform. Whether you're provisioning, whether you're storing like secrets in a database, or you're storing like certificates, you can use Vault.

So HashiCorp's vision for Consul is wherever your app is running, we own the network for that app and we help you connect your workloads together and monitor those network workloads and know where they are. So that's kind of where HashiCorp's vision is going for service mesh. So that really fit with Consul where we're already tracking where your apps were at and what their addresses where. And so it made sense to expand into Kubernetes and worked in like Kubernetes native way for that.

[00:09:55] JM: Describe the conversations that you have with customers around people who are deploying and operating a service mash. What is useful to them? What is extraneous? What are the difficulties? Tell me about the actual case studies.

[00:10:12] LK: Yeah, for sure. I think a lot of it is developer-driven. So the operators are like, "Here, we have some Kub clusters for you," and the developers were like, "Okay. But I want to do this stuff that I've been reading about with service meshes. So I want to do canary deployments and I want to see all my metrics for all maps, and to I want to help with migrating, with making these big routing changeovers." So that's definitely one of the big use cases where

the developers are coming to the operators and they're like, "Hey, I want a service mash in my Kubernetes cluster." And then from the operators 'perspective, they're thinking around like, "Okay. Well now, I have team A and team B and they're running different Kubernetes clusters." But they have to call each of the services.

Well, how am I going to allow this routing to work? Am I going to need to have thousands of firewall rules between pods with IP's that don't exist? So they're looking at it often from like this security perspective. How can I ensure that this traffic is going to be encrypted without having like manage like massive PKI infrastructure on my own?

So those are definitely some of the use cases that are coming up. And then you also just have, again, like things aren't getting less complex. They're getting more complex. So you're having multiple Kubernetes clusters now. And so the operators or are looking for a way to manage kind of all these complexity in one place and have kind of a global view of everything.

[00:11:31] JM: There are different service meshes. You've got how HashiCorp Consul. There's Glue. There's Istio. There's Linkerd. What's the current comparative diagnostic of the different service meshes?

[00:11:48] LK: Yeah, for sure. I'm almost not an expert in Istio and Linkerd, but I think like the general view is – And maybe say it in a couple of words, like what's the tagline? So for Linkerd, I think it is user experience and simplicity. For Istio, it has all the features. For Consul, it is multiplatform. That's kind of I think a good summation of where they're at and what kind of they're targeting.

To dig into it further, Consul, we're really focused on running across multiple platforms kind of that. It was like our ethos of HashiCorp , is like one workflow, not one tool, right? So we're focused on being able to run a VM just as well as we run on Kubernetes. And then I think from the Istio side, it's incredibly fully featured service mash, and you can basically do anything that you wanted. That does tradeoff complexity. It's a little bit harder to configure maybe. That's definitely one of its biggest strengths. And then from Linkerd's side, I think they're focused –

Really focus on the user experience on statistics and being able to kind of like see stuff out of the service mesh really, really easily and having a really nice install and user experience workflow. I think that's kind of where I would say the three products are at.

[00:13:03] JM: So one of the reasons we are having this conversation is due to a discussion around multi-mesh. So the concept that I have my billing service somewhere in the company. You may have the check-out service elsewhere in the company. Each of us has our own Kubernetes cluster. Each of us has our own strategy for how we want to manage the inter-service communication. We have our own preferences for how we want to do service proxying. Our own preferences for how we want to do service mesh. And that said, we may need to interoperate with each other. The billing service may need to interact with the check-out service. So the services may need to call each other. Therefore, we need to have some interoperability between these different services. And if I've got an Envoy proxy and I need to communicate with an Nginx proxy elsewhere, maybe there's some interoperability issues. How interoperable are the different service meshes?

[00:14:08] LK: Yeah. Right now, I don't think they are interoperable. And so if you had like your billing service and check-out service in two different Kub clusters and one was running Istio and one was running Consul, I think you'd have to look at like ingress as kind of like you exit the service mash in terms of you're not being encrypted with like the TLS for that mesh and you're not subject to the routing rules for the mesh. And so you're just ingressing into another service mesh the same way you would ingress that that service mesh didn't exist and you were ingressing it to a service that was just running there.

So right now the meshes are not interoperable, and so that's kind of one of the reasons. There's a project coming out of VMware called Hamlet, and this is the idea of multi-mesh federation. So a protocol that each mesh can implement that will allow meshes to share the list of services that they have and to route between different meshes without necessarily having to like come in through an API gateway or something like that. Basically exit your data center. So that's kind of where we're at right now.

This project, it's being pioneered by VMware, but there's a bunch of folks looking at the spec. Google is looking at it. HashiCorp is looking at it. Pivotal has been contributing to it. So that's kind of a nascent project, and we'll see where that goes. But that is kind of looking to solve this problem of multi-mesh. And I think – I mean, being candid, I think each mesh would preferred that you just run their mesh everywhere. But I think the reality of it is that you are going to have these multi-mesh environments either because of acquisition where you acquire a company that's running a different mesh and you now need to talk to those services, or because of massive organizations where one side of the organization has chosen this one mesh and it's kind of deeply ingrained into their infrastructure. Yeah. I think you are going to reach these situations where we do need the ability to have mesh federation.

[00:16:09] JM: Mesh federation. So there're already people listening who are probably saying, "Oh my God! This service mesh things is such unnecessary overhead. And now you're going to tell me I've got to put additional federation infrastructure. Can we just level set again? Like is this extra overhead, do we really need all these stuff?"

[00:16:29] LK: Yeah. I think you only need it – There's always a tradeoff, right? That's the thing with everything. So if you want to be able to control the network between these clusters programmatically and you have a valid reason for doing so, then you are going to need something there, right?

There're a lot of really powerful things you get out of running meshes and federating them together. You get like unified metrics. You've got to be able to use the same routing rules you're used to to do multi-cluster, like canaries and migrations. I think you're right, that like listeners should be a little bit worried about this and take it with a grain of salt and think about like, "Okay. Well, do I actually need to federate these meshes, or can I just treat that other – That check-out service running in in this other cluster as just like another API endpoint, like Twitter or Facebook's API and just call it directly? They'll put up a load balancer for me?"

But then you need to immediately deal with, "Okay. Well, how do we do security? How does certificates work? How can we do MTLS between those two apps? Okay. So now we need

some function that's going to start provisioning these certificates between these two clusters." So I think as you do get into the weeds there, you realize that, "Oh! We do need something here that's going to help us manage this complexity." So yeah, it does definitely sound a bit like over the top on the face of it, but I think once you dig into the problems that you might want to solve, you do run into the need in some cases for this complexity because it does solve a lot of the complexity that you're going to be facing anyway.

[00:18:06] JM: Let's get into some of the work that you have been doing with Consul. So in terms of implementing a service mash, has the sidecar started to take on any additional roles these days? Like the last time I did a lot of coverage of the service mesh, there was kind of the security management where you kind of have the service meshes is doing some access control management. You have the telemetry. You have the potential congestion management, like exponential back-off, and retry, and redirecting to backup service instances, things like that. And perhaps, now, Federation is another role that the service proxy is playing. Tell me more about what the sidecar is doing these days.

[00:18:57] LK: Yeah. I think the biggest development that we've seen, at least on the Envoy proxy side, is this idea of a web assembly. This is the ability to write code in any language that can compile down to web assembly and have it run in the request path on the proxy. Like the proxy is running this code for you. It's pretty exciting development. I mean, this is again kind of like where if complexity rises to a point where you need software involved, this is kind of where that's solved. So you can see something where companies could write their own custom authentication policies where they have some very, very peculiar requirements where if a request comes in like /Jeff, then it has to use this TLS certificate. It needs to have like this bearer token. And then if it comes in, like /loop, you know we're going to JWT authentication, something like that. Something that would be very, very hard to encode in the CRD and have the service mesh do. What if we can write this code ourselves? Compile it down to WebAssembly and distribute it to all of our Envoy proxies? And Envoy will run this and decide right there at the proxy right at that level whether that request is allowed to come through the service or not. So that's kind of one of the cool things that I think we see starting to come out now.

Solo has their own. I think it's called something hub where it's the idea where people can share this, a community of WebAssembly plugins that people can use. I think you've kind of nailed like the main use cases, and the proxy hasn't really evolved too much beyond that. But WebAssembly is definitely something that's coming down the pipe.

[00:20:38] JM: What about issues with deploying the mesh? Are there any usability issues that – Well, I guess we should get specifically with Consul. When you see people trying to deploy the mesh, are there any usability issues that you've managed to make simpler? I imagine there're some people in the audience who have tried to deploy a service mesh and have gotten frustrated by the process. So maybe there're some usability issues to simplify.

[00:21:04] LK: Yeah, for sure. I think the biggest problem you have with deploying a mesh is if it works, you don't know. So if it's running in the proxies, they're all transferring traffic as expected, then there's no immediate output necessarily. Everything is just the same. It's one of those classic like demos where, "Hey, look. I did this migration. Nothing changed. So therefore, successful."

For that, what we look at is having the ability to see like the health of these different parts, like the proxies that are going through them. The service mesh is being used for. We also look at how can we get these statistics out to users so they can see that, "Look. Now you're actually seeing stats coming through. So therefore the proxies are in the request path and they're receiving traffic." That's kind of one of the big things, is like it's so transparent that you don't really know if it's working.

And then the other thing that we're really focused on from the Consul side is, okay, Kubernetes in general is a lot easier to deploy a service mesh into, because you have the ability to basically mutate everything that you're deploying into your cluster. But if we look at other platforms like VMs, it's a lot harder to provision it and deploy these service meshes. If you look at a lot of them, they are very focused on Kubernetes. And if you're running VMs, you're doing things like editing Etsy hosts, or you're manually provisioning like for every VM that comes up, you're manually provisioning a CRD into your Kub cluster so it can actually see that VM. Or you're like

having Kub config on the VM so that it can like talk to Kub API. So that's where a lot of our UX focus is, is how can we make the experience on VMs and on Kubernetes and on the next platform the same?

Luckily, we have like many, many years of Consul working great on VMs. And we have the ability to tie into automatically detecting when new VMs come up and when they die, doing failure detection there through gossip. And so that's another place where we're looking at UX.

[00:23:10] JM: So tell me about your work specifically. What have you built within Consul?

[00:23:16] LK: Me personally?

[00:23:17] JM: Yeah. What do you work on?

[00:23:19] LK: Yeah. I'm on the Consul Kubernetes team. My focus is on making Consul work great, very Kub native way on Kubernetes. Over the past year and a half that I've worked there, we've worked a lot on the bring up UX. Consul has a lot of interesting pieces to it because it used to run only on VMs. And so we've worked a lot on having all these things automated when you install Consul into Kubernetes. So you have a very, very simple – You just run install on Helm chart and you're up and running. We've worked a lot on the security side of things. So making sure that you have TLS between all the different components, you have ACLs enabled, because Consul has its own ACL system.

Recently, what we've been really working on is the federation piece. So multiple Kubernetes clusters and multiple VM clusters kind of all join together in a secure way and in a really easy way to get it all up and running. So just recently, we released a feature that I'm really excited about where in basically four steps you can take Kubernetes cluster running on any cloud and a Kubernetes cluster running on any other cloud, or a VM cluster. And you can federate them together in four steps. And boom! You can now route from one pod on one cluster all the way over to another pod on another cluster across public Internet if you want to with full MTLS security between those pods even though they're on different networks. And it's just really easy to set up. So that's something I've been working on recently.

[SPONSOR MESSAGE]

[00:24:52] JM: This episode of Software Engineering Daily is sponsored by Datadog. Datadog integrates seamlessly with more than 200 technologies, including Kubernetes and Docker, so you can monitor your entire container cluster in one place. Datadog's new live container view provides insights into your container's health, resource consumption and deployment in real-time. Filter to a specific Docker image or drill down by Kubernetes service to get fine-grained visibility into your container infrastructure.

Start monitoring your container workload today with a 14-day free trial and Datadog will send you a free t-shirt. Go to softwareengineeringdaily.com/datadog to try it out. That's softwareengineeringdaily.com/datadog to try it out and get a free t-shift.

Thank you, Datadog.

[INTERVIEW CONTINUED]

[00:25:48] JM: Consul started as basically a lock server, if I recall correctly, similar to Chubby. It has had this strange progression to becoming a service mesh platform. Can you recast why that happened or reiterate why that happened? Why did it go from being a lock server, like an etcd kind of thing to becoming a service mesh?

[00:26:12] LK: Yeah. I think like for the longest time, Consul has been known as the distributed keyvalue store. It's kind of what its tagline was. And I think what we saw is, "Okay. Well now, in order to use this distributed keyvalue store, you have like these client agents across all of your nodes, and then it was kind of this natural place for service discovery. So why don't we have those services register with those client agents?"

It did evolve from like a KV store, keyvalue store, to a service discovery mechanism where you register your service when it comes up on your node. And then you can use DNS to route between the services and you have an API now where you can ask like where are my services

and how healthy they are and add health checks and all that stuff. So that was kind of where it started. And so but then we saw folks using these in Kubernetes. And you didn't really need that service discovery mechanism anymore, because that's part of Kubernetes. Kubernetes has its own DNS just the way Consul has Consul DNS.

What we did still see was people were using Consul as like a homegrown service mesh. And I before I joined HashiCorp, was doing the same thing. We had Nginx running and we had Consul Template. Consul Template is a tool that will take the information in Consul and we'll write out config files. And so what we wanted was the ability for us to talk to the local Nginx server running on any node, and we could use a path to route to a service where that service was running.

For instance, like /billing would go to the billing service, /checkup would go to the /checkout service. And so the services would be kind of – They wouldn't need to know about where all these services are and you won't have any problems with DNS like latency and TTLs if the service has changed. And so we use Consul for that, because Consul knew where all those services were.

You already had this kind of – This use case for Consul where it's like, "Okay. It knows where all the services are, and now I have the need for a mesh either through migrations or for reliability or just because I want that programmability. My complexity is such at a level that I need to be able to program it."

So it was this natural fit for us to look at, "Okay. Well, folks are already doing this with Consul. They're already hooking up Consul Template and HA proxy and Nginx to build these kind of homegrown service meshes. Why don't we start doing this for them?" Because there's a whole bunch of problems that you can't really solve when you're just hooking up Consul with Nginx, like certificates, and MTLS, and rotation, and authorization, and things like that. And so that's where Consul kind of naturally evolved. And we actually focus first on L4. So the first release of Consul Connect, which was what we called the service mesh, was focused only on L4 connections. Basically like, can this service make a connection over to this service? Yes or no?

And it used MTLS for that. But it wasn't really focused on a lot of the L7, so stuff around routing based on HTTP headers or about retries and things like that.

So we were very focused on security, because that's all lot what we're hearing for our customers, was they had all these environmental, all these firewall rules. Everything was changing really quickly, and they already have this information in Consul. So why can't Consul also figure out like whether service A can route to service B? And that kind of got us on the path of service mesh. And then we've continued on from there.

[00:29:23] JM: Have there been many challenges that technical implementation challenges that come to mind? I can think of all these things you need to build; service discovery, routing, security features, heterogeneous environments. You need to handle containers and VM's. What's a particularly difficult implementation challenge you've had to build around?

[00:29:45] LK: Yeah. I think definitely the implementation challenge of having certificates be distributed and be refreshed and having the ability to rotate like the root CA and cross sign and heavily certificates pushed out. That was definitely a pretty complicated challenge. And then the other one that comes to mind is, as we talked about how Consul has been around for a while, it's been around from before Kubernetes.

The Consul model of what a service is doesn't match up directly with what Kubernetes model of what a services is. So for instance, in Kubernetes, you have a pod, which is part of the deployment. And then you have a service, which has endpoints. And then in Consul, we have a service, and we have service instances.

So I think that's been an interesting challenge, is how do we kind of mesh is the operative word there. How do we mesh these two views of the world in a way where we can still have the same experience on VM's over there that we do have on Kubernetes? And how do we kind of map those two concepts between each other? That's been definitely an area of work that we've had to do.

[00:30:53] JM: How do you monitor a service mesh deployment? Specifically Consul?

[00:30:59] LK: Consul outputs metrics, just the same as many of the service meshes. So it has like endpoints that you can call with Prometheus to script those endpoints, or you can hook it up with Datadog to scrape those endpoints and get all those metrics out. Because we run Envoy proxy, it has the same metrics that you expect from something like Istio. So you hook up Prometheus usually to scrape the Envoy endpoint and pull out those metrics, and then you can go from there. You can build your dashboard. And you can you do the same thing with Datadog or any of your other metrics tools of choice.

[00:31:33] JM: And nodes in Consul are communicating over a gossip layer. Have there been any difficulties in implementing that gossip layer or have there been any updates to the gossip layer?

[00:31:46] LK: Yeah. To help your viewers like understand. So Consul has this idea of a gossip layer where all the nodes in your data center are all talking to one another, and because it would be ridiculous. We have data centers with like tens of thousands of nodes. For every node to talk to every other node, it has this idea of gossip where one node can talk to another node and ask like, “Hey, are you okay?” Then if it doesn't hear from that node, it will then talk to another node and be like, “Hey, is node Jeff okay? I didn't hear from him.” And it won't talk to every other node, but eventually all the informational will kind of get gossip throughout the cluster and everyone will kind of converge on a state, which is like, “Okay. Well, the Jeff node hasn't responded. So we think he's in trouble.”

Yeah. When we look at kind of running in Kubernetes, you don't really need that anymore, because Kubernetes is a source of truth. Kubernetes knows whether your nodes are down or not. Kubernetes knows whether your pods are healthy or not. So that has been an interesting place for us where, right now, we still run the gossip layer in Kubernetes and we still kind of have our source of truth for whether Kubernetes node is or up or down. And luckily, like the way Kubernetes works, usually they converge upon the same answer, which is like the node is down. But Kubernetes does a different way with like Kubelet reporting out.

But I think what we are looking towards is especially as we go to like other platforms, is the ability to farm out this gossip layer to where we're running the orchestrator, and we not need to run ourselves for failure detection. Instead, just rely upon like the Kubernetes API to tell us whether something is up or down. But right now, we do run the gossip layer in Kubernetes.

[00:33:24] JM: Let's talk about multi-mesh in practice as you have seen it. So you've laid out earlier that you think we are headed towards a world, or we already are in a world where there is this mesh federation issue. What does that mean for the developers of service meshes such as yourself?

[00:33:47] LK: Yeah. I do think it is a future problem that we need to kind of be ahead of now. But I don't think you're going to – I mean, as I talked about at the beginning of the show, a lot of folks aren't even running one service mesh, let alone multiple. So I do think it's a problem that kind of is pretty far in the future, but something that we need to kind of get ahead of now. And so as a service – And that's kind of like what our job is, is service mesh developers.

And so I think what it means is we're going to have this kind of agreed upon spec and way to interoperate with different service meshes the same way you have the ingress controller in Kubernetes where all the different ingress controllers can basically – They use the one ingress back and they implement that, and you can switch out these ingress controllers for each other, kind of. There are lots of annotations probably involved with switching them. I think you're going to look at kind of that where you have a standardized spec that everyone will implement.

And then the interesting part comes in is whether – So we're talking about federation between, say, two Kub clusters and two different service meshes. And so now you implement this spec that allows you to federate. So the Istio cluster conduct the Consul cluster. But does Consul use that same spec within its own service mesh when you're doing Consul to Consul communication, right? Does Istio use that same spec within two different issue clusters? That's an interesting part.

I think, often, these commonality specs, they lose a lot of kind of the special sauce of each mess, kind of like the SMI spec , which is the service mesh interface spec where the idea was

you have one CRD that would be implemented by the different meshes. And I think one of the problems with that is that then if you have special sauce that you want to layer on top, the users can't really access that. I think that is one of the challenges that we need to think about is like how do we start to layer – Yes. We can implement these specs, but then do we actually use them internally within our own mesh?

[00:35:39] JM: What would that standardized interface look like? How would the meshes be talking to each other?

[00:35:46] LK: Yeah. There're a couple pieces to it. One is which services do you have over there? So there's basically like an endpoint. You call and be like, “Okay. Give me your list of services.” And then it's, “Okay. Well, now I have this list of services. How do I talk to that service? What endpoint do I call from my service mesh over here?” And then you have a whole other problem, which is very complicated around identity and security. So you are not going to be running – If you look at the way federation works across like within a same mesh, with different clusters within the same mesh, often you're required to share the same route CA. Each mesh can decrypt and encrypt each other's traffic, right? But when you’re talking about different meshes, they’re not going to be running the same CA, and you probably don't want to be running the same certificate for all your meshes.

So there’s this whole layer of, “Well, how do I trust that this request coming from this other mesh, which is encrypted with a different CA, how do I even decrypt that traffic and how can authorize against that?” That’s the topic of like SPIFFE federation, which is being worked on by the folks over, I thinks, Sitel. And so that’s the idea of like, “How can we –”

[00:37:01] JM: Acquired now. I think they got acquired by HP.

[00:37:03] LK: Oh, do they? Okay. Well, good to know. I hope they're still working on it, because it’s definitely a topic that we need solved. So yeah, the sharing of services, there is like the endpoint, “Okay. Well, how do I actually talk to that service?” And then there is the issue of authentication, or like TLS encryption, like how do you exchange these certificates and decrypt

each other's traffic? How do you rotate those certificates? How do you trust that the first certificate that you've got from that other cluster was actually the right one? Kind of that first step, that bootstrapping problem. Those are kind of like the things that you need to solve when we look at mesh federation.

[00:37:36] JM: And do you think there's going to be some kind of diplomatic effort? Do you expect to be some CNCF sort of diplomacy around standardizing this stuff, or do you just expect it to be worked out by itself?

[00:37:50] LK: Yeah. I don't 100% know. I'm a little bit more on the ground implementing the spec than on like the politics side of things. But I think – If you look at it, you're thinking, "Okay. Well, if you're mesh vendor and you implement this federation spec, then does that mean that folks don't need to use your mesh everywhere?" Which is probably what your desire is, right? Because they can just federate with whatever mesh they want.

But on the other hand, it could be good, because maybe somebody wouldn't want to use your mesh in a certain situation because they can't interoperate with the one they're already running. But now they could use your mesh and then interoperate with their mesh. I think that's probably like the considerations that are being done on the business side. I don't know if it will be part of a CNCF project or a foundation of some sort, but I would imagine like any kind of open, like whatever happened with the SMI spec, where I think that is lived in like a public GitHub repo and anyone can contribute to it. I think I could see that the same thing happening for Hamlet for the mesh federation spec.

[00:38:48] JM: Wait. So the service mesh interface spec, how does that differ from what you're working on with Hamlet?

[00:38:56] LK: Yeah, for sure. It is definitely kind of hard to piece the two apart. The SMI spec is the idea that within my one cluster, I can set intent. I can set up like authorization rules. I can set up L7, like canary rules. I can set up like retry rules. And instead of setting these things up using like the Istio CRD or the Consul CRD or the Linker CRD, I create these rules using this

SMI, the service mesh interface common CRD. And then if I were to swap out my meshes, then they would work the same way. So that's within one cluster.

And then the idea with Hamlet, with the service mesh federation, is where you have like – You only have one mesh running in one cluster and one mesh running in another cluster. How do those two communicate with one another? And it's not really dealing with the routing rules. It's more just about like can I just share these services so I know where they are, and then therefore I can route to them? So it's more focused on like sharing what exists between the clusters and making sure the certificates can work for the communication.

[00:40:04] JM: Talking more about what you have specialized in, specifically which is Kubernetes and getting Consul working well with Kubernetes, tell me about that implementation and the Kubernetes ecosystem more broadly. What have been the engineering difficulties in building Consul for Kubernetes or targeting the Kubernetes platform for Consul?

[00:40:32] LK: Yeah. I think a lot of the difficulties and the challenges stem from how the consul model of the world doesn't 100% mesh with the Kubernetes model of the world. And so we need to kind of deal with that. So one example would be that the way Consul was originally built was the idea where you have these – You're provisioning these AMIs. You know what service is going to run on that AMI. And so you include along with that service deployment config file that describes the service name, the port that it's available on and any metadata. So that is actually deployed. It's a config file that lives on disk. And went Consul also starts on the same node, it reads that config file and it tells the rest of the cluster, "Hey, I have the service running on my node." It's a little bit different where each node is saying, "Here's what I have running on it."

But when you look at the Kubernetes world, it doesn't work like that. Kubernetes tells a node, "Here's what you're going to run on." So it has that information about what is running in the cluster at a centralized level, versus the Consul's original model was it has that information on each node. So it's a decentralized model and it just trusts that the node says, "Oh, I'm running this service." "Okay. Cool."

When we look over at Kubernetes, well, we don't know. The nodes get provisioned. They don't have anything on them. They don't know what service they can run on them, right? Luckily, Consul as an API. So when a pod comes up, now we tell Consul, “Hey, this services is running on you know.” And when the pod exits, we tell Consul, “Hey, that service is no longer running on you.” So that's one example of how we had to kind of change the way we run at Kubernetes.

The other example is there is like ACLs, for instance. Consul has its own concept of ACLs, whereas Kubernetes, everything is tied to a service account. So how does that mesh with Kubernetes? When a pod comes up, what service does that authenticate with or as into Consul, right? Consul has its own concept of ACL. So what ACL token does this service get? And so we basically pair the service account within ACL token. And so we kind of need to meld these two concepts together. That's a lot of what our work does, is how do we translate kind of concepts to Kubernetes concepts in a way that makes sense for Kubernetes users? It's a lot of work, but what you do get is when you are running on VMs or on the next platform, you have this idea of ACLs can work the exact same way across all these different platforms. And so it's a little bit harder to get it working on Kubernetes in the first place. But then once it is working, you’re going to have the same user experience, the same workflow, whether you're on VMs or on Kubernetes or whatever the next platform maybe.

[00:43:09] JM: I'd like to take a step back and talk about HashiCorp more generally, because it's in a position in the software ecosystem. It's sort of like a Switzerland. Nobody really competes with HashiCorp, or maybe that’s a way of saying everybody competes with HashiCorp. So just a few random questions about HashiCorp. First of all, Nomad. Nomad is this scheduling system that came out around the time of the container orchestration wars. And as I understand, people still use Nomad, and it has some design differences between Kubernetes. I know you're not on the Nomad team, but just from being at the company, Imagine you have some perspective. Why are still people using Nomad?

[00:43:57] LK: Yeah. I mean, this is a little bit timely, because Cloudflare recently wrote a blog article where they used Nomad extensively for everything. And that's 10% of the Internet going through Nomad essentially. So there're a lot of people using Nomad. I think a lot of the reasons is there's a little bit of a backlash over the complexity of running Kubernetes. If you can run it in

a cloud and they handle that complexity for you, then that's all well and good. But if you're something like Cloudflare, or you're running on data centers, Kubernetes is very, very complex to run and to manage, and Nomad is very focused on kind of the user experience and simplicity and only doing one thing, kind of the UNIX philosophy. Like doing one thing and doing one thing well. So it's very focused on scheduling. That's what it does. It doesn't try to do a bunch of other stuff, which people do need in some use cases. Bbut other folks who have other solutions for those things, like service discovery, or secrets, or whatever, like for Vault. They might be using Valut for secrets. They are happy to just pick a platform like Nomad that just does that one thing and does that one thing extremely well.

Nomad also has a little bit different design. So it was originally focused on extremely fast scheduling. So we had like the C10000K problem, where we could like schedule 10,000 services in like seconds because we use an algorithm of like optimistic scheduling where just like it automatically assumes that it can be scheduled and kind of condole all of these tasks really quickly. So it does have a different use case and it's still heavily used by a lot of different companies that's making money. It more than pays for the engineering team, and it does make sense for HashiCorp to keep working on it, and we're still pushing kind of more and more features into it and it's becoming more feature-rich. And I think a lot of folks are looking at it as they turn away from kind of the complexity of Kubernetes.

[00:45:45] JM: Very interesting. So what is it like being at HashiCorp these days? Tell me more about some of the goings-on and the changes and the product development.

[00:45:56] LK: Yeah. It's a really exciting place to be in. It's kind of that hockey stick growth that we're seeing right now. So I think it's interesting. We're seeing a lot of pickup for some of our products. For instance, Terraform. Terraform is the de facto way to provision infrastructure. That's pretty cool. I mean, that's something that's like everybody who's providing infrastructure, most of them are going to do it through Terraform. And so now what we're looking at for that is to make the experience a little bit better. We have Terraform Cloud, which is the ability to like collaborate on infrastructure in a web UI and work with a bunch of developers and a bunch of organizations and set up security rules and everything. So you can't just like spin up like EC2 massive instances.

So we have like each product is kind of doing really well in and of itself. So Vault is the de facto way to store secrets. If you're thinking about like a secret, most folks are thinking, "Okay. Well, how do I put this into Vault?" And so we're really seeing all these products accelerate. Have more and more users and more and more features. And then the other thing that we're really working on is this idea of cloud. So more and more what we're seeing is companies don't want to run these really complicated and important core infrastructure tools themselves. They don't have enough people that do it, or they just know that that's not their specialty. They don't want to be the experts in running Vault. They would just want someone to run Vault for them, or someone to run Consul for them.

So we've had a big investment in our cloud team, and the idea of the cloud team is that we will manage our tools for users. We have an SRE team that's on call for them, ensure they're up and running. And just like you can like click a button, get an RDS instance in Amazon. You can click a button, get a Consul installation that's running in your infrastructure, but is managed by us. So that's another big, big shift for us.

[00:47:48] JM: What do you see is the competitive dynamic between HashiCorp and the major cloud providers?

[00:47:55] LK: Yeah. HashiCorp has really, really good relationships with all the cloud providers, which is interesting. I think that's like hats off to Armon, and Mitchell, and Dave, kind of the executive team as they're building these relationships. HashiCorp helps the cloud providers, because if we take the example of Terraform, we're helping people run a ton of stuff on those clouds, right? So we're helping the cloud providers like sell more to their customers. So from that perspective, it's really good for them, and we're helping people migrate these apps to the cloud and build bigger deployments into the cloud. And so that's all really, really good for the cloud providers. So we do have a pretty good relationship with the cloud providers.

And then I think one thing that's interesting with this whole HashiCorp cloud platform, which is the idea that we run a managed service for you on the clouds, is there was kind of like this

interesting war almost between open source companies and the clouds. I think you covered it pretty well in some of your shows, where the clouds, some of them were kind of like building their own managed services and selling those to their customers using an open source technology. And then the open source technology, that was kind of one of the ways that made their money, which was building these managed services and selling them.

So there was kind of this like big licensing war where they change their licensing, so you can do that. And I think I'm not sure how it's all revision, but when we look at Hashi or cloud, we're working – Like the clouds are definitely aware of what we're doing and they're happy, because in the end, we're helping them run more resources on their clouds and helping customers like be more successful on the cloud, which is really good for them. So when we look at HashiCorp, we have like Consul, which can run on Azure, and you can install it through the Azure marketplace. Again, this is like you click a button, install it, but all the servers in there, everything are being managed by a certain. And then same thing for recently announced. We recently a HashiCorp cloud platform on Amazon, and the idea is you go through HashiCorps portal and you click button install Consul into a VPC running in Amazon and you peer that VPC with your VPC. And now we're managing these Consul installation for you, but it's all running kind of on Amazon.

[00:50:06] JM: Well, just to wind down, do you have any predictions for what will change in the next few years around the Kubernetes and service mesh ecosystem.

[00:50:18] LK: Definitely. I have some guesses. I can throw them out there. I think you've seen Kelsey Hightower say this too, where Kubernetes is going to get more and more kind of like something that people don't care. So the same way you don't really care about Linux. It's just there and it works for you. I do see that we're going to see more abstractions on top of Kubernetes and it's going to become less of this really big important thing, and it's just more kind of like the way things are in the world.

I think that's one thing that's going to fade into the background, and you're just being more focused on like the Kubernetes API and how do you describe your app into Kubernetes API and less focused on Kubernetes itself.

And then I think for service meshes, I don't think there's going to be a consolidation of service meshes where it is one mesh wins out over everything. It's interesting that Kubernetes did kind of went out it was like at the top – Sorry. The top container orchestrator. But if you look at a lot of other different technology spaces, there hasn't been one tool that has like been the one that everyone uses. For instance, CICD, there's Jenkins, CircleCI, Drone, Metrics, there's Prometheus, there 'Datadog, right? There're lots of different areas where there's multiple tools that exist and coexists. So I think service mesh, we're not going to see a consolidation up on one service mesh, but each mesh is going to kind of find its niche and thrive in that niche and have more people use it, in the pie is kind of grow. I'll go back to that talk about how in service mesh gone like 20 people out of 200 were running in production. Well, you can see that there's just way more people that are going to end up running a service mesh and there 'going to be more users of each mesh. And I think that's where I see it going.

[00:51:56] JM: Okay. Well, it's been really great talking to you, Luke. I'm very grateful for you sharing your time and hopefully the service mesh ecosystem gets better and not worse. So it's a very boilerplate thing to say. But nonetheless, I hope it doesn't get worse. Thanks for coming on the show.

[00:52:18] LK: Hey. Thanks, Jeff. Thanks for having me.

[END OF INTERVIEW]

[00:24:29] JM: Join us on August 26, 2020 for GitLab Virtual Commit! An immersive 24-hour day of practical DevOps strategies shared by developers, ops pros, engineers, managers and leaders. Attendees will hear from U.S. Air Force and Army, GNOME Foundation, State Farm, Northwestern Mutual, Google, and more and more about problems solved, cultures changed, and release times halved. Come and be part of a community of people just as passionate as you are about DevOps. Register today! softwareengineeringdaily.com/GitlabCommit

Thank you to GitLab for being a sponsor.

[END]