EPISODE 845

[INTRODUCTION]

[00:00:00] JM: Containers offer a lightweight abstraction for running a server. Cloud providers are able to manage billions of containers from different users allowing for economies of scale so that each user can pay less.

Today there is a variety of ways that users can deploy containers on a cloud provider. These containers can run in managed Kubernetes clusters, in functions as a service, or in long-lived, standalone container instances. User preferences are getting more sophisticated with some users showing an interest in Knative, an open source serverless system originally created at Google.

Whichever container deployment system you choose, your application and its multiple servers need a way to route traffic, measure telemetry and configure security policy. A service mesh abstraction can help serve these use cases.

Lachlan Evanson has worked in containers and Kubernetes since before the container orchestration wars. He was an engineer at Deis, a company which built an open source platform as a service running on top of containers and Kubernetes. Deis was eventually acquired by Microsoft, where Lachlan now works as principal program manager of container compute. Lachlan joins the show to discuss containers, Kubernetes and the service mesh interface, which is an interoperable service mesh layer, which Microsoft launched with Buoyant.

A few quick announcements; there are some conferences I'm attending in the near future. Datadog Dash in New York, July 16th and 17th; and the Open Core Summit, September 19th and 20th in San Francisco. We are hiring two interns for software engineering and business development. If you're interested in either position, send me an email with your resume to jeff@softwareengineeringdaily.com with internship in the subject line.

FindCollabs is the company I'm building. We've launched several new features recently. If you have a cool project that you're working on, I would love to see it. I check out every new project that gets posted to FindCollabs, and I've been interviewing people from some of these projects on the FindCollabs podcast. We have a new Software Daily app for iOS. If you have checked out the app in the past, you would probably like these new updates. We've really freshened up the interface and improved reliability, and you can become a paid subscriber if you want to get ad-free episodes. You can go to softwareengineeringdaily.com/subscribe. All of these details are in the show notes.

Let's get on to today's show.

[SPONSOR MESSAGE]

[00:02:52] 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]

[00:03:49] JM: Lachlan Evanson, welcome to Software Engineering Daily.

[00:03:52] LE: Great to be here. Thanks for having me, Jeff.

[00:03:54] JM: You spent some time in the trading technology world, and trading technology is like its own planet. You have requirements for high-performance. You have high scalability, high stakes. There are lots of money at stake, and there's lots of money to be spent on resources.

But it's not like consumer applications. It's internal applications. Tell me about the unique requirements of building software for traders.

[00:04:24] LE: So, yeah. You're referring to a job I had at a company called TradingScreen, in which I worked in both Tokyo and New York City. What we would build was to deliver a software platform where banks could trade with each other. So it was a sovereign company that would build both platform and distribution and network. In my specific role there, I was responsible for the core network when we would connect directly to exchanges, software exchanges.

So that game is around latency, building very big pipes with very short latencies so that these orders can make it into the market with minimal hiccups. So it was very high frequency and high volume stuff. So those concerns were how to design for in those days, but that's really where I entered, distributed systems and building them, which led me to things like Kubernetes.

[00:05:19] JM: What lessons did you learn in that environment that you carry with you to today?

[00:05:24] LE: I would say the biggest lesson that I have is understanding failure and how failure affects the customers at the end. So, obviously, network failure, not many people are thinking about the network, just pipes. Obviously, that was my job. So people would only get upset at me when there was a failure in the network obviously.

So understanding how things fail in the distributed nature where somebody is trading in New York City on a market in the CMA in Chicago. How that is actually connected and how that data flow works is really just the basis of a distributed system. So understanding that in things like Kubernetes gave me incredible concept, context and insight into how to design and plan for failure.

[00:06:13] JM: Another thing I could think of, in the trading environment, there's all these weird integrations, like old systems, new systems, old protocols, new protocols. It's kind of like the world where easing our way into with Kubernetes, like Kubernetes and cloud native stuff having to wire together, really old systems and containerized COBOL. That archeological dig field, was there an archeological dig sense in the trading world?

[00:06:49] LE: Yeah. You're talking about these very old systems and you're trying to build new software to deliver services that are dependent on those systems operating. So you get in to a lot of abstraction building and translation work. So you're basically in the job of building shims that can connect one data source or one data stream, translate it to a common language, route it across a platform and then maybe translate it back to some older framework protocol. Yeah, it's all about building abstractions. As you mentioned, it's kind of the same position that we're in now in things like Kubernetes.

[00:07:24] JM: You eventually wound up at Deis. Why did you joined Deis?

[00:07:28] LE: I joined Deis because I had gotten into distributed systems and been responsible for building a cloud platform for a company who is trying to deliver containerized software both on on-premise and in public cloud. The work that I did there led me to gain a lot of skills, and I also realized through open source that I really enjoyed helping people attain these skills for themselves and understand. Deis was in the mission to kind of making Kubernetes easy to use through open source tools, and I had the opportunity to go and work with them. I had met them many years earlier and I invited Gabe, who is the CTO of Deis, over to my office to check out some of the platforms we built.

Subsequently, some of those tools I was able to jettison. We've built them in-house with things like Helm and Kubernetes. So I got a taste for open source through – It was in open stack in Kubernetes at that time, and Deis was doing Kubernetes, and I was in Kubernetes – I've been in Kubernetes for quite some time. So I had skills and I was really willing to go out and help other people get into the community, understand how it worked and start solving problems for their businesses. So that's what Deis was looking for, and I thought it was a good fit.

[00:08:47] JM: Deis had this really interesting acquisition story, where the Deis team, if I recall, Deis was acquired by Engine Yard, which was a platform as a service provider. Then Microsoft acquired Deis from Engine Yard. Was that the history?

[00:09:05] LE: Yes, that's correct.

[00:09:06] JM: That was interesting, because Microsoft was realizing, “Oh my goodness! We need to get into the container game in a really big way. Who can we go out and acquire? Let’s go for Deis. Let’s get the Deis team.” What was the acquisition like from your point of view?

[00:09:25] LE: The acquisition was fantastic from my point of view. The onboarding was great. The continuity was great, and we came on as the Deis team, which was a small knit. We’re only 30 odd people team into Microsoft, but we were welcomed with open arms. We obviously had Brendan Burns, or was already over at Microsoft, and we landed in a good spot and we had a very clear mission as we came on. We were given the resources that we needed to be successful.

So out of that acquisition came services like the Azure Kubernetes services, and a lot of the tools – So we had this heritage at Deis of building open source tools to solve developer problems. A lot of that heritage has come into Microsoft as well with tools like Helm that we maintain. But we’re really focused on creating that great developer experience and that really solid platform. So that kind of acquisition at that time was great for me personally, and I think most of the team would agree with me. It’s been a smooth journey and Microsoft has been very welcoming to all of us.

[00:10:31] JM: So you are around for the container orchestration wars before Kubernetes won out. From your point of view, why did Kubernetes win?

[00:10:42] LE: So, for me, and I was in the trenches during this time.

[00:10:46] JM: You were.

[00:10:46] LE: I was in the trenches, and this was pre-Deis. For me, I’ll just speak from my experience. There were other container orchestrators out there and I certainly looked at them all. But Kubernetes had – It was just a story of the right place at the right time. So we were already using containers in the company I was working for at that time. I issued a challenge to my team when we were looking at Kubernetes, because we had heard about it, and this was back in maybe late 2014.

How long is it going to take us to get a workload or Kubernetes up and running and get a workload running on it, because we had already had containerized workloads at that point. We were able to do it in a matter of hours. So that was just a really compelling first stab for us, and I think it was one of the first things we tried.

So that is – And that API just gravitated with what we had. We had service needs at that point, so we needed long-lived services, and Kubernetes met the needs even pre-1.0. So I think it was a right tool at the right time and it did the things and met the expectations of what we needed. So, for us, it was a tool that we went and used. But I think that also spoke to the power of the API and the depth of the API and the kind of operational behaviors and runtime behaviors it provided or what people were looking for.

[00:12:06] JM: That's consistent with what I hear from other people, although at the time, Mesos was more advanced technology, right? Maybe even Docker Swarm might have been more advanced or fully feature – I know Mesos at least. Why was the experience with Kubernetes more fulfilling for you than what you could have gotten out of Mesos?

[00:12:30] LE: I can't really answer that question, because I haven't looked in Mesos in depth. It was just the right tool at the right time, and we landed on Kubernetes. I know that Swarm was coming out around that time, but I don't think it was quite out. So it was there.

[00:12:44] JM: Any other reflections from the container orchestration wars?

[00:12:48] LE: No. I think just the success of the open source community and the way they really built a community from the ground up and had people out there learning about Kubernetes, solving problems, iterating on the software really quickly to solve problems that people were having. We're still on a quarterly release. Actually, it's a three-month release cadence since the project started. So we're releasing new software every three months.

So that means a lot gets fixed in a very short amount of time and new features are added all the time to meet the needs of the people that are using it. So I think that kind of nascent building community was instrumental and getting a mindshare around solving problems, which led to Kubernetes becoming extremely popular.

[00:13:34] JM: What do you at Microsoft today?

[00:13:36] LE: I am a PM. I have a team of PMs, and I work on upstream software. So open source software.

[00:13:42] JM: Explain what upstream is versus downstream.

[00:13:45] LE: Upstream is software that's in open source, and we can take a dependency on that to build services that are downstream. So we can either maintain or contribute to that software. Something like AKS could be a downstream implementation of an upstream project [inaudible 00:14:02] Kubernetes. So we pull that in and we make it a service and we make a great user experience. Kubernetes is the upstream project that's backing that.

[00:14:11] JM: So you work on open source software.

[00:14:13] LE: I work on open source software across the whole stack, from developer tools, to abstractions on top of Kubernetes, right down to Kubernetes. So I'm on the PM side. So it's about sustain contributions to open source software and helping solve developer problems in open source.

[00:14:31] JM: I saw your role was I think principal program manager of container compute? Is that right?

[00:14:37] LE: Yes.

[00:14:39] JM: So what I presumed that was, and you can correct me if I'm wrong, I was thinking you would be involved in this palette of abstractions that we have. First of all, you have the managed Kubernetes, managed Kubernetes service. You have functions as a service, and you have – I think they call it OCI, or no. Sorry.

[00:15:01] LE: ACI.

[00:15:02] JM: ACI, Azure Container Instances. Do you spend a lot of time thinking about those different abstractions, the different ways of running containers?

[00:15:10] LE: Yeah, absolutely. So quite recently we released an open source project called a Kubernetes event-driven autoscaler, KEDA. That was actually built with Azure functions team. So they saw a need and a problem space for their users to deliver Azure functions on Kubernetes and they needed some componentry to do some autoscaling. Our team actually went and helped the Azure functions team develop working an open source, because it was something newer for them, that our team is constantly working in. So a project like that is a way that we see some value that we can give to the community and develop that in open source so that other members of the community can benefit from it as well.

[00:15:53] JM: Let's talk about those different compute abstractions. So let's say I have a blog of computation that I want to run in the cloud. I have managed Kubernetes. I have container instances. I have functions as a service. What are the different application requirements that would map from my application to those different compute abstractions?

[00:16:24] LE: So, I would probably ask that question a little differently or answer it a little differently.

[00:16:28] JM: Sure.

[00:16:29] LE: So as you go up the stack, it's about different tradeoffs. What are you getting from the platforms from something like a functions as a service system as supposed to Kubernetes or container orchestration role? It's just about how much of the abstraction you want to understand or control. So in a functions as a service system, you're handing over your source code and having interpreted, packed up and delivered. Functions as a service is event-driven. So there are different SLAs as to when events are triggered, consumed.

So if you have something that might be very time sensitive or requires a very intricate set of language features, functions as a service may not be for you. So it's about the right tool for the right job. You might want to pop down to container orchestration, because you want something slow latency, can autoscale and you can control the autoscaling. So you have a few more of the componentry of the underlying system exposed to you. Containers fills about the right abstraction. You can take your source, deliver your libraries, package it up in a container and deliver that to a distributed system like Kubernetes and have it orchestrated.

So it's about how much of the infrastructure do you want to know or control and how much do you not want to control. There are different tools for different job. So they're all valid and important, but it's how much of that you want to take on and understand for whatever problem you're trying to solve in your business.

[SPONSOR MESSAGE]

[00:18:05] JM: DigitalOcean is a simple, developer friendly cloud platform. DigitalOcean is optimized to make managing and scaling applications easy with an intuitive API, multiple storage options, integrated firewalls, load balancers and more. With predictable pricing and flexible configurations and world-class customer support, you'll get access to all the infrastructure services you need to grow. DigitalOcean is simple. If you don't need the complexity of the complex cloud providers, try out DigitalOcean with their simple interface and their great customer support, plus they've got 2,000+ tutorials to help you stay up-to-date with the latest open source software and languages and frameworks. You can get started on DigitalOcean for free at do.co/sedaily.

One thing that makes DigitalOcean special is they're really interested in long-term developer productivity, and I remember one particular example of this when I found a tutorial in DigitalOcean about how to get started on a different cloud provider. I thought that really stood for a sense of confidence, and an attention to just getting developers off the ground faster, and they've continued to do that with DigitalOcean today. All their services are easy to use and have simple interfaces.

Try it out at do.co/sedaily. That's the D-O.C-O/sedaily. You will get started for free with some free credits. Thanks to DigitalOcean for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[00:20:07] JM: This is what I find so interesting about the range of things available here, is it seems like there are so many cases where if you're an insurance company, for example, why do you care about running Kubernetes? Why wouldn't you just want container instances, managed container instances? That is one thing about this whole space that seems surprising to me is that there are not more people going after ACI or Fargate and just saying, "Just give me a container instance." Why does everybody want to manage a Kubernetes?

[00:20:46] LE: So it's that promise of interoperability and not being locked into a specific platform. So when you're making decisions about where you should run your software, do you want to limit your options as to where it will run for longevity, or you might have different – You might be coming from – Let's talk about use cases here. You might have a data center already and you want to kind of extend your workloads out as you're going through this microservices journey out on to the public cloud.

So what's a system that can run in both places where you can create identical experiences and move your workloads over one at a time, and Kubernetes is a system that can run on-prem, on public clouds, different public clouds. So that promise of, "If I deliver software to that abstraction, the only bet I need to place is on that abstraction. Where that abstractions runs and how that is delivered, I don't mind. That's opaque to me. But I know that if I move it from on-prem to on cloud, it will run the same way," and that's the promise of Kubernetes.

So people only have to place a bet on that API abstraction, which is Kubernetes, and then they have the flexibility to make the decision about where that runs later. So I think that's why people are interested in things like Kubernetes, because this is part of a larger journey of companies coming to public cloud, changing the way that they're building software. So, yeah, that's why –

[00:22:17] JM: But these are not like Heroku Dynos. These are like you're deploying your container to a Docker container. Its ACI is I believe – It's a Docker container. AWS Fargate is Docker container. I don't think there's any kind of proprietary lock in, and if you want to connect it to a Kubernetes cluster later on, you could do that through like virtual kubelet, for example, right? Why do you need the control plane, the Kubernetes control plane, to be managed by you?

[00:22:49] LE: So if you're writing to something like ACI, you're writing to the ACI API, which is not portable of Azure, or Fargate. I'm writing to Fargate. So I can't move that Fargate workload to Azure.

[00:23:06] JM: Sorry to interrupt you, but what about that API is proprietary.

[00:23:11] LE: So there's no runtime that's going to implement it on any other platform. If I submit – In ACI, you're submitting an ARM resource. I need the ARM APIs and the equivalent implementation that can take that.

[00:23:24] JM: ARM, what is it?

[00:23:25] LE: ARM is the Azure Resource Manager. So you're coding to the Azure Resource Manager API or you're coding to the Cloud Run API if you were using Cloud Run or Fargate –

[00:23:37] JM: That's the Google – Google one is the Cloud Run. Oh, okay.

[00:23:38] LE: So it's then not portable. So you can't build all these tooling around that API. Then you decide that maybe you want to try somewhere else and then you got to go redo your tooling. So have a look – Let's flip that on its head. I use Kubernetes. I put in virtual kubelet. I connect that to ACI. That workload is now portable, because the abstraction I've backed on is Kubernetes, not ACI.

[00:24:02] JM: Got it. Because then you say, "Hey, Kubernetes, spin up a container," and it says, "Cool! Spinning up an ACI container."

[00:24:08] LE: Yeah, and that model, it's translating, it's handling the translation between the Kubernetes interfaces and ACI. So that's kind of the beauty of something like virtual node. You can get the serverless container. I don't really worry about running a node or a VM. I don't really care about the underlying toil that I need for that infrastructure. I just want to run a container. But I want that still to be interoperable and portable.

[00:24:34] JM: This sets us up for a conversation around Knative. Have you looked at that project very much?

[00:24:41] LE: Yes, I have.

[00:24:42] JM: What are your thoughts? So Knative came out of Google. It is their spin on an open source serverless system where you say, “I want to have scale up and scale down for a container image. It’s an open source thing.” They have an installation of it called cloud run that you can use on Google Cloud. But since it’s open source, other cloud providers could have Knative on their cloud. Give me your diagnosis of Knative.

[00:25:17] LE: So I would phrase this question as –

[00:25:21] JM: I like you rephrasing my questions for me. Thank you. Please.

[00:25:23] LE: What is the problem that Knative is trying to solve, right?

[00:25:26] JM: Okay. Yeah.

[00:25:27] LE: And we’re going into this common abstraction problem, right? So people see common patterns emerge from running functions as a service workload. So Knative is a set of componentry build serving and eventing that provides base level abstractions in which you could build FaaS platform on top of something like Kubernetes. It’s the same thing. I’ve got a democratized abstraction Kubernetes. Now I want to build a FaaS. Can you build the abstractions? Well, if you’re going to build a FaaS, you’re going to need serving. You’re going to need eventing. You’re going to need build event serving.

Now, I can built a platform because you’ve commoditized those abstractions. As a user, I don’t care how they’re implemented, because that should be opaque to me. I just need to build – So I can go write. They’ve democratized those set of tools in open source so that people who want to build a system on top of it have those building blocks.

[00:26:33] JM: You’re making a great pitch for Knative right now.

[00:26:35] LE: Well, I would say, liken it to the announcement we made with server mesh interface. So we made an announcement yesterday about service mesh as being hard. A lot of people are using service mesh. So what service mesh interface is trying to do, it's trying to commoditize –

[00:26:53] JM: Okay. Wait. Let's talk about Knative first. We'll get to the service mesh stuff later. I want to go deeper on Knative first.

[00:26:58] LE: Sure

[00:27:00] JM: So you've presented a very appealing vision for Knative. Do you plan to offer some Microsoft equivalent of Cloud Run?

[00:27:09] LE: We planned of what customers are looking for the problems that they're looking to solve. So we're driven what customers are asking us. They're coming to us not asking for infrastructure tools. They're coming to us with problems. I have this workload. I have this use case. I have this business problem. So the way we approached this is how do we build tools to solve those problems. We want to listen to our customers, and we listen to our customers and we help them solve their problems.

[00:27:43] JM: Okay. The thing that Knative solves, maybe this is just like Google going off on a building and engineering thing that nobody actually needs kind of path, or maybe just Google needs it. Because what they say with Knative is there is no function as a service. There is no container as a service. All there is is gradient of spin up and spin down and there's a cold start problem that happens if you don't specify that I always need one of these things running, something like that. It's basically the unification of functions as a service and container as a service so that the user – I mean, it will be the user problem that they will be solving with this if I was to take the words out of their mouth, hopefully, is, "You just bring us compute, and we run it for you in a reliable way. There's going to be a cold start problem if you go down to zero, zero QPS. But otherwise, we're going to keep it high uptime and keep the scalability there for you."

Are you suggesting that's not really needed by anybody, because function as a service and container as a service and Kubernetes as a service gives actual customers enough options for running their compute? They don't need this additional Knative solution? Help me understand.

[00:29:12] LE: No. I'm not suggesting that they don't need it, but I'm suggesting that if there are customers and community members with a problem to solve and that provides the solution to their problem, then that's certainly needed. But how that problem is solved, people don't come to us with technology problems. They come to us with business problems and they're trying to serve their developer community internally or externally. How can they build software and have it run in a way that need it to run? So, objectively, whatever delivers is that for that customer, however that is solved, they're okay with.

[00:29:55] JM: You haven't seen the problem that Knative solves, at least in your customer base.

[00:30:03] LE: I have seen people wanting to run function as a service. Again, it's this abstraction. I'm betting on an abstraction, like the Kubernetes API I mentioned earlier. Is that abstraction portable? Is it interoperable? Can I move it around? Those are questions that people are answering. So tools like Knative and SMI are seeking to build platforms that are interoperable, so solve that problem.

[00:30:36] JM: Okay. All right. Interesting. Very cagey. It sounds like we'll have to do another show in the future on this topic. Let's talk about what has been announced, which is SMI. Explain what the service mesh interface is.

[00:30:48] LE: So, what we were seeing from customers was a common set of problem. Service mesh is about taking it up the stack. We've got Kubernetes, but what do we need next to deliver, to solve our business problems and help our application developers? Service mesh has been around for quite some time in this space, and what has emerged is a bunch of different APIs to delivery service mesh with a cross-section of features. Again, if you want to use a specific service mesh, you have to code to their specific API, which is generally not Kubernetes native. So you have to go and touch another system to program it.

So what we're offering with SMI is a specification to unify that abstraction for service meshes. So our customers, again, are coming not saying we need a service mesh, but we want things like traffic management. We want to be able to route services the way we want to be able to route them. Think of problems like canarying deployments, so upgrading. We want telemetry. We want to know how many RPS, how much successes, how many failures we're having service to service. We want to be able to impart policy. Can service A talk to service B?

With SMI, we've defined those three APIs in policy, traffic management, and telemetry, and it's a specification, which any of the service meshes can then implement. So that as users, they can use any service mesh they like and they actually don't need to worry about it. They use a common abstraction for service meshes and who implements that service mesh. It doesn't matter. So if I want traffic management, I can say, "This is how I want my thing routed," whether it's Linkerd, or Istio, or HashiCorp console, or super glue. The user doesn't actually mind as long as it's delivering the features they need for traffic management. So part of that announcement, which is just a common set of APIs for the service mesh workloads without defining a runtime for them and having the runtimes implement those specifications.

[00:33:01] JM: All right. Now, that's a very diplomatic abstraction to insert here. But wouldn't it be easier just to pick a winner? Can't you just pick Istio or pick Linkerd and say, "This is the service mesh that we endorse."

[00:33:19] LE: So I think people want to be able to – At the moment, it's an all-in decision. Again, it's about portability and interoperability. I lay down a service mesh. There might be some features I want from one service mesh and others I want from another. Why does it have to be an all-in decision that you have to make immediately? Once you've made that decision and you code to those APIs, then you can't change and you don't create an ecosystem of, "Can the best tool for the right – May the best tool for the right job come to pass?"

So it's enabling the ecosystem partners to come in, have a single abstraction and be able to implement the tooling. They want to be able to implement the tooling. This is not a new concept CNI. So the container network interface. CSI, the container storage interface. The ingress resource in Kubernetes is we define a specification and then people can implement the runtime of that specification, and that creates an ecosystem of tools and runtimes that do different things, and people have the choice to pick whatever one they want.

[00:34:29] JM: Describe the surface area. What that does that interface, the service mesh interface, what does it provide?

[00:34:35] LE: So, today as of launch, we have three discreet APIs. One for access policies. Can service A talk to service B? We have telemetry, RPS, failure rates, things like that. The wonderful, rich information that service meshes give you, and traffic routing. We started there, because that's what customers were actually asking us to solve. How do we solve this?

[00:35:02] JM: Connections from service A to service B?

[00:35:04] LE: Yeah, policy.

[00:35:06] JM: That's security policy.

[00:35:07] LE: Yup.

[00:35:08] JM: That's like can the user who made the request for service A to service B, does that user have the right security access policy? Then what was the third one?

[00:35:20] LE: Traffic management. So saying that waiting traffic.

[00:35:23] JM: Oh, so like A-B testing kind of stuff or canarying. Right.

[00:35:28] LE: Yeah. So we expect that more use cases will arise and more specifications will arise. But it's created that ecosystem that as part of the onstage demo, we had Istio delivering traffic management. We had Linkerd working with metrics, and we had HashiCorp console –

[00:35:48] JM: Oh, for security policy.

[00:35:49] LE: For security policy.

[00:35:49] JM: That's beautiful!

[00:35:50] LE: So they all just implemented it, and the only thing that I would have to code as a user is that common SMI abstraction, and whoever brings the runtime as a user, the user gets the abstraction that they're looking for there. Because you don't necessarily care how you do traffic management. You just know you want traffic management.

[00:36:11] JM: Did the Istio community take part in that service mesh interface launch?

[00:36:19] LE: So, the Istio community, we worked with Kinvolk to actually build the Istio shim. Kinvolk is a community member in Kubernetes that work on building different tools. So we work with them. They came in and helped us build the tooling around Istio.

[00:36:37] JM: I need to talk to those people, because I saw a presentation from them about Firecracker earlier in the week. I was like, "Firecracker? Why are you working on that?" Then now it's like service mesh interface. Who are you future people?

[00:36:48] LE: So, I think the thing to think about with something like Istio is what do you need to change to make Istio support SMI. You have two things you can do. You can either write a translator, so a piece of code that watches SMI and translate it to Istio primitives, which is what the Kinvolk team wrote with the – Or you can make Istio natively watch the API and implement it itself.

For the demo we showed, they were basically translated. So they would watch the SMI, translate it to their speak. The beauty of this is you don't actually have to leave the Kubernetes abstraction. You get all the information in Kubernetes. So when you're deploying these abstractions, it's Kubernetes native.

Whereas before, you might have to go and program against a different API to get the feature that you wanted. Now, you don't have to leave Kubernetes. Again, it's about that single pane of glass that you've got that abstraction and that API. You want it to be portable. Kubernetes is very extensible. So we can plug in custom resources to do things like SMI.

[00:37:57] JM: I like the abstraction, but only in today‘s context. If one of the underlying service mesh is wins, then you don’t really need the shim. But I guess in this kind of environment, you probably don’t have winners. You probably just have people who win –

[00:38:16] LE: Think about it like let’s take a look at CNI, right?

[00:38:20] JM: Talk about CNI.

[00:38:22] LE: So CNI, the container networking interface. Basically, when Kubernetes was started, there was no single interface where different providers could bring the best of their networking tech to the game. Things were compiled. You couldn’t change the networking stack.

CNI was introduced and companies like Tigera with Calico, and it created a whole ecosystem of companies, tools, open source tools, that implement it. Then I can say, “Well, I need this from the network. I don’t have to change Kubernetes. I tell it to use this provider rather than that provider,” and I get the experience that I need from that cluster. So it’s left the choice to me as how it wants to be implemented if I’m a cluster admin. But it doesn’t make me choose one or the other.

CSI is a more recent one, which is a container storage interface. So all the storage vendors are coming in and providing their plugins to their great storage systems. Just because you build CNI doesn’t mean there’ll be one storage system to rule them all. There are different operational factors.

CRI. So we have CRI-O, we have Containerd, we have gVisor, we have Kata Containers, we have all these different container runtimes that have different ways that they run containers with different security boundaries.

[00:39:41] JM: What is a container runtime?

[00:39:42] LE: A container runtime is a thing that takes the container that you packaged up and [inaudible 00:39:47] execution environment for that container.

[00:39:50] JM: And Containerd is one of those, or what's –

[00:39:53] LE: Yes. So Containerd is one of those. CRI-O is another one of those.

[00:39:59] JM: That's container runtime initiative-

[00:40:01] LE: I think it was – Is it CRI open source? Red Hat pioneered CRI-O. I call it CRI-O. Yeah. I don't know the acronym –

[00:40:13] JM: You're not into those [inaudible 00:40:14]. So there's so many weeds to go into and you only have so much time. So much you can focus on.

[00:40:20] LE: Right.

[00:40:21] JM: But, generally – Okay. So this is interesting, because I've gone to three or four KubeCons and this is like one of these subjects that I haven't really delved into. I had gone into the CSI, the container storage interface a little bit. I've touched a little bit about the CNI to some people. But now I'm starting to see the general trend, is when you have a situation where there are competing components, you find the common interface between them that kind of brokers the competition. It gives you the common interfaces, and those competitors, they actually should be interested in having that interface defined, because they know what they're competing over and they're both going to be confident, competitive companies.

They're going to say, "Hell, yeah. We'll compete over this interface. Let's get it on. Let's define a common interface so we're going to have customers to fight over," and it gives the customers a safe interface to buy into. That's exactly why it would make sense for Microsoft to build this or to lead the effort to make the service mesh interface and to say, "Look, our customers are coming to us and they're telling us we want a service mesh. We don't feel comfortable going with either Istio or Linkerd." You say, "Well, gees! We just want to help you out. We want to make you feel more comfortable buying into this ecosystem. Let's make the common interface and let the competitors duke it out.

[00:42:05] LE: Yeah, it's a common pattern that we see. Really, it's about can we agree on the common set of problem we're trying to solve and lock them in and make it opaque to the user as to how they're implemented. So they don't have to actually – If the person is responsible for choosing their technology under the hood. But as a user, why should I have to program to one specific API if that doesn't make me portable at the end of the day? In this abstraction layer cake, all the abstractions are like these. We have a standard interface and then we have a tool where you can pick whatever – For each need of a customer. Because customers all have different needs. Whether they're storage, whether they're networking, whether they're service mesh.

So we see that need and we allow our customers to make it a choice for what's right for them. Of course, we want to create great experiences in these ecosystems. So we felt that doing this would actually help customers move forward with decisions and service mesh.

[00:43:11] JM: Are there companies that you worked with as initial customers to get the service mesh interface launched and tested?

[00:43:24] LE: So we basically worked with companies to get their requirements. What are they looking for in a service mesh? When they say they want a service mesh, what do they mean? Of course, we've been doing that over a longtime. Service mesh has been around for quite some time in the ecosystem. From there, we wanted to rally different people in the space and see if they thought it was a good idea. As part of the announcement we announced with several other companies who are interested and said, "There needs to be that kind of growth in a new space to prove its value and there'll be different implementations come out for that kind of problem space." But then there'll be a point where we try to re-aggregate back and create a standard. This is kind of, again, a common pattern.

We just felt like the time to re-aggregate and work with all the people in the ecosystem who are looking at service meshes to say, "Do you think this is a good idea?" They all came back and said, "Yes." So now we can actually start to work on building those implementations and gauge how that's working in front of customers and how they're using it. But I've been demoing it at the booth since the announcement and the feedback has been quite very positive. The simplified experience and the burden of not having to choose upfront and making it portable and not being locked in seem to be – Which for the things we were looking at seem to be what I'm hearing a lot of feedback around at the booth.

[00:45:01] JM: Do you think that there is too much concern from enterprises about lock-in? Because the way I look at it, is if you're an insurance company or you're a manufacturing company, you don't care about lock-in. All of these stuff is a rounding error. You just want to move faster, right? It doesn't matter if you're locked-in.

[00:45:28] LE: I can't comment on how they're feeling about lock-in. We do hear customers coming and being concerned about that being one of their concerns and a top priority. I can't comment whether there's too much fuzz around it, because I'm not the one in the position making the decision.

[SPONSOR MESSAGE]

[00:45:56] JM: Commercial open source software businesses build their business model an open source software project. Software businesses built around open source software operate differently than those built around proprietary software.

The Open Core Summit is a conference before commercial open source software. If you are building a business around open source software, check out the Open Core Summit, September 19th and 20th at the Palace of Fine Arts in San Francisco. Go to opencoresummit.com to register.

At Open Core Summit, we'll discuss the engineering, business strategy and investment landscape of commercial open source software businesses. Speakers will include people from HashiCorp, GitLab, Confluent, MongoDB and Docker. I will be emceeing the event, and I'm hoping to do some on-stage podcast-style dialogues.

I am excited about the Open Core Summit, because open source software is the future. Most businesses don't gain that much by having their software be proprietary. As it becomes easier to build secure software, there will be even fewer reasons not to open source your code.

I love commercial open source businesses because there are so many interesting technical problems. You got governance issues. You got a strange business model. I am looking forward to exploring these curiosities at the Open Core Summit, and I hope to see you there. If you want to attend, check out opencoresummit.com. The conference is September 19th and 20th in San Francisco.

Open source is changing the world of software and it's changing the world that we live in. Check out the Open Core Summit by going to opencoresummit.com.

[INTERVIEW CONTINUED]

[00:47:58] JM: I guess what I'm curious about is when people say lock-in, what I think of oftentimes is some proprietary provider that gets you locked in and then it costs a bunch of money. They just can rachet up the subscription. They can charge you more and more every year, and it's onerous. But you still get to move quickly. It seems like lock-in in the modern context actually means if you get locked in, that means you're on technology that gets outdated and gets deprecated relative to the ecosystem. So you're actually looking to keep up with the open ecosystem rather than just cut costs with your lock-in, avoidance of lock-in.

[00:48:47] LE: Yeah, I think customers are seeing the benefits of open ecosystems and how it can be a proving ground for new concepts and technologies. So, again, it's about making a decision on an abstraction. What is the right abstraction? Because you got to remember that when you place that bet on an abstraction, you then build tooling around to support that abstraction. So if you are in a case where you need to move for any reason, what's the cost to move? Understanding that cost to move or being able to mitigate it by using an open source abstraction I think is something that is – They're probably thinking about closely.

With things like Kubernetes and having that portability, it's been one of the stories that we've heard time and time again, because we're still going through this cloud journey with enterprises and they're looking at the long game of how they move software around and how they build it and distribute it. Of course, they want to take it into consideration, because we're not looking at a short game here. We're looking over many, many years.

[00:50:01] JM: What do your customers want out of multi-cloud support?

[00:50:08] LE: I think multi-cloud support is really – Maybe it's just the vendor lock-in or being able to have flexibility to move. So, again, and not have to re-tool. I think that's multi-cloud story again is around having a single administrative interface to deliver software regardless of where it runs. So I want to run on cloud A or cloud B, but I want to use the same system. That is obviously attractive, because the alternative is to build a system for cloud A, build a system for cloud B and somehow stitch them together.

[00:50:52] JM: But you see a lot – Or I've seen a lot of that in practice where a company builds – Let's say a company got started when AWS was the only cloud provider. A company like Thumbtack comes to mind. Yeah, let's just take Thumbtack. Thumbtack is a gig economy platform. They're doing really well. They started on AWS, but they use data services from Google. They want big query on Google. So they stitch together cloud resources for upside rather than for downside protection.

It seems to me like this – I mean, first of all, I agree that what you described is a desirable outcome. You want – It would be great if you could mirror all your bucket storage between AWS and Microsoft, for example, and AWS and Azure so that you have a reliable bucket storage. Maybe you don't trust a single cloud provider. You think a cloud provider doesn't have the reliability guarantees that they say they do. So you mirror over to a different bucket storage system. That I think of as downside protection.

There's also upside opportunity. I should be able to go get CosmosDB, a service that is not available on other cloud providers. I should be able to go and get BigQuery from Google. I should be able to go and get Red Shift from Amazon. Do customers want that kind of thing, the ones that you're talking to? Do they want the upside opportunity as well as the downside protection?

[00:52:25] LE: Yeah, I see both cases. Yes, absolutely. I would frame it in the what is the system that's going to benefit us the most. In the case of something like CosmosDB, if that makes sense for you, you can deliver it. We've worked on tooling to allow even that to be

Kubernetes native. Again, it's just like the runtime implementation. If you have a common abstraction, the runtime implementation that suits you best or what's best for you wins. It's about you've only made a bet on that abstraction. Where it lands, then you have more flexibility.

[00:53:05] JM: But if I want – Okay, the runtime abstraction is Kubernetes. That's great. If I use managed Kubernetes in one cloud provider and I want to connect it to the managed Kubernetes in another cloud provider, how good is that story today? Because what I hear from some people is that IM policies make it not trivial, because you have, "Okay. Yeah, I run Kubernetes in this cloud provider and I run another Kubernetes in this other cloud provider, but the identity and access management systems in those cloud providers are different. In each cloud provider, the IM is deeply integrated with the Kubernetes. So Kubernetes becomes like closed. Am I misunderstanding something?

[00:53:52] LE: No. I don't think you're misunderstanding. There are sharp edges like that, and let me speak to one specific example. We're working on policy in Kubernetes right now in a project called Gatekeeper under open policy agent, and we're working with companies like Google to solve Kubernetes policy using a cloud native compute foundation incubator project called Open Policy Agent.

Now we can talk about policy in the same way in Kubernetes using – This is a case. Kubernetes is growing up. Enterprises are showing up to Kubernetes now. We need Kubernetes policy. So over on the upstream team at Azure, we're looking at those needs and looking at the open source community and saying, "How do we create an experience that is interoperable and portable." So we went and we created this project, which is now called Gatekeeper and we collaborate with other cloud providers. Because at the end of the day, we want them to have a great experience with policy on Kubernetes and we can offer an integrated solution on Azure for that. But if they want to go and take that and run that somewhere else and on-prem, I want for it to be portable so they can have that experience on-prem, because they may have that.

This is a case where we go out and we see a customer. We go in and open source. We try and build consortium so we get rid of those sharp edges. Because at the end of the day, we're all in solving this problem. We know we can give them a great experience on Azure. But if they're coming, speaking a different language, then we have to do different types of work. If the language we're speaking is open policy agent, if you're on Google, Azure, I'm using those as an example because they're collaborating in the community with us now. Suddenly, a lot of those sharp edges disappear. Like you were mentioning, I have this identity management. I have that identity management. I have this policy engine. I have that policy engine. So we're trying to come together and build tools in the open and then provide great enterprise experiences. So we built Gatekeeper in the open and we have Azure policy for AKS, which is built on Gatekeeper, which gives you that integrated great experience on Azure. This is ways that we can now get a customer to have policy regardless of where they're running, but we know we can give you a great experience on Azure.

[00:56:17] JM: So could Gatekeeper be – Or I don't know what that project is. That sounds interesting though. But does this provide – Because I've done a show on open policy agent, but that does provide or does Gatekeeper or open policy agent provide a way to, for example, translate my AWS IM policies to parallel Google Cloud IM policies?

[00:56:43] LE: No. It works as a generic policy engine. So it's not looking at IM policies. Although there are other endeavors that take care of policy and having it –

[00:56:53] JM: It seems like it's a such a hard problem.

[00:56:54] LE: At a higher abstraction. Again, we're building them into – So we have a project called Aad Pod Identity, which is open source. So it's tied into Azure Active Directory. So your pods or your workloads get a unique identity and it's Kubernetes native. So you're not touching Azure Active Directory. You're working within the confines of Kubernetes, but you have identity that service A can talk to service B through pod identity.

So if that's portable and another cloud provider implements that, suddenly you're not worried about IM as much. You're worried about identity. Does identity A have identity B. How that's implemented under the hood. Does identity A able to connect or identity B. How that works under the hood is not actually your concern anymore. It's about building these – That's the value of something like SMI and open policy agent, because if we use open policy agent, we can actually have – Those policies are portable.

[00:57:56] JM: So what you're saying is if you solve the workload identity problem, IM is less of an issue.

[00:58:05] LE: It's an implementation detail.

[00:58:07] JM: Right, and is that because if you solve the workload at any problem, then it just services permissioning to each other rather than you permissioning into a service and then having to permission yourself into all the different network hops.

[00:58:24] LE: Yeah. Right. Well, it's kind of you don't have to leave the Kubernetes API to do it all of a sudden. It's not like when I create an identity, I go over here and touch these APIs, then I write them up through here and then push them over there and then I can touch Kubernetes. It's kind of I go to Kubernetes. I create an identity. That all magically wires up to Azure Active Directory. Then I say I want to use that and I want to say that A can talk to B, and that's all wired up.

So nobody knows that it's active directory under the hood, but we've got that identity system there that can handle this. So solving that is an abstract then. It's just I have identities. I want workload identity. Can workload A talk to workload B. Wherever that's running and however that's implemented, I don't actually care as long as the runtime guarantees are there.

[00:59:15] JM: Okay. Well, it sounds like I need to have a conversation with Gatekeeper. Last question; have you been to all the KubeCons? How many KubeCons have you been to? Three or four?

[00:59:25] LE: I was thinking about it last night, my 5th or my 6th.

[00:59:30] JM: Okay. How is this one compared to the last one?

[00:59:33] LE: I am really amazed with the level of technical expertise in the community, because I've watched it through the years. So people are coming with really interesting problems and questions, which for somebody like me is really interesting to see how the community is building on top of Kubernetes trying to solve things like service mesh, identity, policy.

So now at the booth, it's so great to hear from the community as to what problems they're having, but these are really higher level abstractions now. Back in the early Kubernetes, it was how does Kubernetes work. How do I deploy Kubernetes? What Kubernetes do? Now it's like, "Okay. We're all okay with Kubernetes. What are going to build on top of Kubernetes?" The enterprises are showing up. We have this problem. How can the community help? So that's what I really enjoy. So the level of engagement and questions has been really invigorating for me at this KubeCon.

[01:00:31] JM: People are smart. People here are just really smart. One thing I like about this conference is I grew up – You know the game Magic, Magic the Gathering?

[01:00:39] LE: Yes.

[01:00:40] JM: I grew up playing Magic a lot. I played Magic tournaments. One thing I always loved about Magic tournaments is the people are just shockingly smart, and you're there to play like a stupid little card game and it doesn't really matter and these weird abstractions like spells and lands and magic wands and stuff. Ultimately you're there because you like socializing with these smart people. That's kind of how I have felt at these – I come at these conferences and I'm talking about software and CNI and CSI and these –

[01:01:13] LE: All the I's.

[01:01:14] JM: All the I's.

[01:01:15] LE: [inaudible 01:01:15].

[01:01:16] JM: Yeah, and these weird diplomatic conflicts between service mesh at a startup versus service mesh coming from a gigantic company. This stuff doesn't matter. It kind of matters, it's cool, it's really important – I mean, I think software that gets build is great. But what I can say with certainty is that I am really enjoying the conversations at these conferences.

[01:01:40] LE: Yeah. Generally, it short-circuits my knowledge growth into a really –

[01:01:46] JM: Right! Exactly.

[01:01:47] LE: I learned so much from everybody. I learned –

[01:01:50] JM: From like 3 or 4 days.

[01:01:51] LE: 3 or 4 days. It's six months in my office and three days here, I learned so much. So I come with a kind of open mind to learn and I really enjoy hearing what the community has to say and what they're trying to build.

[01:02:05] JM: Here's to KubeCon.

[01:02:06] LE: Yeah, here's to KubeCon.

[01:02:07] JM: Thanks, Lachlan Evanson.

[01:02:08] LE: Thanks, mate.

[END OF INTERVIEW]

[01:02:13] JM: FindCollabs is a place to find collaborators and build projects. The internet is a great place for communicating with each other. Why aren't we building projects together? Why is it so hard to find other people that will work with you on a new project reliably? Why is it so hard to find cofounders to start a small business with? We started FindCollabs to solve these problems.

I love starting new projects, software platforms, musical songs, virtual art, podcasts, and when I create these projects, I want to share them with people. I want people to be able to follow my progress, because as they see that I'm dedicated to consistently delivering quality work, maybe they'll be inspired to join me, and it doesn't have to be on a financial basis. It could be just to collaborate and to make artistic progress.

You can see my profile on findcollabs.com by searching by Jeff Mayerson. You can see the different projects I've made; games, podcasts, open source software. Some of these projects are incomplete. Some of them are well-developed, and this is how innovation and invention happens. Projects start out in an incomplete state, and if you can prove to the world that you work on projects with dedication and consistency, you will get other people to join you. Whether you want to hire them or if you just want to collaborate casually. You can see my profile and many other people's profiles, projects that they're posting on FindCollabs, and I would love to see your projects there too. You can check it out by going to findcollabs.com, F-I-N-D-C-O-L-L-A-B-S.com. You can post a project that you're working on no matter how incomplete it is.

The best projects in the world start with an inspiring vision, whether or not they have code attached. If you like to compete, the FindCollabs Open is for you. The FindCollabs Open is a $2,500 hackathon with prizes for the best React JS project, the best machine learning project, the best game. There are lots of prizes, and you can check it out by going to findcollabs.com/open and post your project.

Thanks for being a listener to Software Engineering Daily and I hope you post your cool ideas on findcollabs.com.

[END]