EPISODE 843

[INTRODUCTION]

[00:00:00] JM: Kubernetes has evolved from a nascent project within Google to a thriving ecosystem of cloud providers, open source projects and engineers. Tim Hocking is a principle software engineer who has been with Google for 15 years. Tim joins the show to talk about the early days of the Kubernetes projects and the engineering efforts that are underway 5 years into Kubernetes.

At KubeCon EU 2019, two of the prevalent subjects of discussion were service mesh and serverless, particularly the Knative project. Tim gave his perspective for how different projects that are adjacent to Kubernetes are developed within the community.

We have a new version of Software Daily, our iOS app available in the iOS App Store. The Android version will be up soon. Software Daily is a place to access all 1,000 of our episodes in one place. You can find all the shows related to a particular technology, just streaming data, or cryptocurrencies, or Kubernetes. You can connect with other listeners through the comment section. You can access our transcripts and our related links to each episode. Everything in the app is free. Although you can also become an ad-free listener or support the show for $10 a month or $100 a year by going to softwaredaily.com/subscribe, and we're booking sponsorships in addition to that. If you are interested in advertising on the show, you can go to softwareengineeringdaily.com/sponsor. We're hiring two interns for software engineering and business development. If you're interested in either of those positions, send an email with your resume to jeff@softwareengineeringdaily.com with internship in the subject line.

[SPONSOR MESSAGE]

[00:01:57] JM: As the database component of the popular LAMP Stack; Linux, Apache, MySQL and PHP, MySQL remains one of the most popular database management systems among developers today, but there can be a lot of pitfalls without a cloud-based database service. Think deployment efforts, continual patching, painstaking performance optimization, complex high-

availability mechanisms, intricate security and compliance concerns, and continual worries about scalability.

Azure database for MySQL combines the community addition of the MySQL database engine with all the benefits of the cloud, free from the complexity of infrastructure and database management so that you can focus on building exceptional apps.

Azure provides fully managed enterprise-ready MySQL database service. You don't have to manage the infrastructure. It's all handled for you by Microsoft. You can fully benefit from enterprise-grade features, such as high-availability, scalability and enterprise-grade security and compliance features for your database solutions.

Learn more at aka.ms/mysqlintrovid. That's aka.ms/mysqlintrovid. You can also find that link in today's show notes. Thank you to Azure for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[00:03:36] JM: Tim Hockin, welcome to Software Engineering Daily.

[00:03:37] TH: Thank you. Good to be here.

[00:03:39] JM: You've been at Google for 15 years. Every engineer who I've talked to from Google has a crazy story about early days of engineering at Google. Give me a crazy story about scaling Google that I won't hear from anyone else.

[00:03:53] TH: About scale in Google.

[00:03:55] JM: Scaling Google.

[00:03:55] TH: Scaling Google. So, yes, I have been here for a very longtime. When I started the entire – My platform and SRE team was half of a floor of a building. So, I wasn't on the SRE team. I wasn't working on search at the time, but I was able to observe it very closely firsthand. So, I've seen a lot of firefights, a lot of really interesting bugs that nobody at Google would hit.

Sort of bugs that happen one in a million run hours we would hit every day, and it gave us a really interesting opportunity especially around kernel bugs to reproduce things that had existed in the wild for a longtime, but nobody had been able to track down, because we had a very large fleet of very homogenous workloads that if there's a bug in that kernel, we're going to find it. It's always a lot of fun to dig down. I'm a low-level person. So I like to dig in at those levels.

[00:04:48] JM: Before that, you're at Sun Microsystems. You spent five years at Sun from 1999 to 2004. How did your experience at Sun contrast with Google?

[00:04:59] TH: Sun was a very different company. I came into Sun actually through an acquisition of a startup called Cobalt Networks, built these little micro-servers. So that was my first taste of playing with systems and system management. In fact, we built a product at Cobalt, which if you squint, looks a whole lot like Kubernetes for a single machine. I didn't realize that until well into the Kubernetes process, but I realized that sort of the architecture ported over and held really well.

Sun was a big company, and by the time I joined it was already sort of in trouble trying to figure out what its role was. Linus was eating away at the low-end of Sun's product offering. As often happens, if you let the thing eat away at the low-end, the low-end becomes the middle and the middle becomes the high. Pretty soon, Linux was eating Sun's lunch and Sun didn't really know what to do with it. So I worked on Linux at Sun, but I wouldn't say that it was the most satisfying work, because it was hard to see how it fit into the larger picture of what the company was doing or even what Linux was trying to do.

[00:05:56] JM: If I understand the history correctly, in the 90s, Sun workstations were the de facto server to use for startups, right? There was some period of time where Sun workstations were the thing to use.

[00:06:10] TH: Yeah. Sun workstations were the only real thing for engineering. It was either Sun or SGI really. Sun servers was – Like that's what you did for the internet. Their old campaign was were the dot and .net, right? Sun was how you did everything. Linux came along in the late 90s and became a really viable thing very quickly. Nobody really realized what it was

doing. It was a 1 CPU machine. Sun would sell you these 100-core giant servers and Linux came along like, “We can do 1 CPU. We can do 2 CPUs. We can do 4 CPUs.” Before you knew it, it was like 4 CPUs is a lot of compute actually in a single server for the 90s. Put yourself in ’99, 2000 timeframe, Linux could serve a lot of traffic, and it was cheaper than Solaris and it was pretty much as reliable as Solaris in terms of the hardware support. The x86 hardware was a whole cheaper than the Sun hardware. So people made these tradeoffs of like – Solaris was a fantastic operating system. From an engineering point of view, it’s just beautifully done, really well executed software, but it was closed. It was expensive. It only ran on Sun’s hardware.

Sun tried to do this Solaris x86. They put it out and they took it away and then they put it back out and they open sourced it. It was a big mess. They never committed to x86 as a platform, because they thought their hardware was better. But x86 and Linux is like prototypical example of worse is better, or good enough is good enough.

[00:07:36] JM: The reason I bring up Sun is I think it’s worth having historical context and historical reflection for how good we have it today, for starting a company back in the 90s relative to today. You have these upfront cost that actually kept a lot of people from starting companies. How would you compare the upfront cost of the pre-cloud, early 90s or mid-90s, whenever Sun was the most dominant to what we have today?

[00:08:14] TH: That’s a great question, really interesting thing to think about. If you were a startup, like I was at Cobalt. I was in a startup, and one of the things I had to do as part of the startup was help to run the website. We literally ran our website on a server that was in our office connected by a T1 to I don’t even know what internet provider, but that was our internet connection.

When the server crashed or there was something wrong, I would go into the server room and go fix it. The idea that anybody would do that today is laughable, right? You want a sever, you just spin up a cloud instance or even less. You use a serverless or you use an HTTP hosting site or whatever and you have infinite scalability. You don’t have to worry about, “Well, what happens if I have a sale or my product becomes really successful or somebody tweets about it?” Not that Twitter existed back then, but if somebody caught wind of what we were doing and the load came in, we didn’t have a story for how to scale that. So we had an array of servers that was

our internet service and we did it ourselves. Yeah. It's sort of interesting that today that was thousands, tens of thousands of dollars that we had to spend to buy that equipment and maintain the T1. T1s weren't cheap either. Then we had all these workstations that we had to provision that were really high-end engineering workstation where today I can just do a lot of work in the cloud really easily. The idea of a cloud top, right? Yeah, there was a ton of upfront investment –

[00:09:36] JM: Cloud top?

[00:09:38] TH: Putting your desktop in the cloud.

[00:09:38] JM: Oh, sure.

[00:09:39] TH: Right? Whether you're using –

[00:09:41] JM: By the way. You can see me. I've got a Macbook here recording the audio and I've got a Chromebook where I do most of my preparation and work, and I'm gradually lifting and shifting as many workloads from my Macbook to my Chromebook.

[00:09:57] TH: Absolutely. I mean, what better example can I offer than like Slidware, right? Mac still has a corner on the market for building slides, but things like Google Slides is becoming really viable. If you look around KubeCon today or this week, you'll see a lot of people who build their slides in Google Slides or other online slide software that you don't need powerful local application to do anymore, and these things are free. You don't have to spend millions of dollars in licensing to get software that you used to have to buy.

[00:10:27] JM: Yeah. Okay. So that's on the SaaS side. Going back to the questions of infrastructure and the technology you need to build a startup, there was, like you said, the eating away of the proprietary machines and the proprietary operating systems with Linux and the LAMP Stack. Then eventually we got to Cloud. So I think Cloud – From what I can tell, Cloud was perhaps the steepest drop in the expenses for starting a company. But when did – And when and why did people move from Sun workstations to, I guess, Linux running on commodity hardware. That is the term that I use. But what is it? I don't really even know what that means.

[00:11:16] TH: I mean, put yourself in sort of the late 90s, early 2000s. You could buy a Spark 20 workstation for a couple of thousand dollars, and it was a beautiful, powerful workstation built for engineering, right? It came with those big 21 inch CRTs and they weighed like 8,000 pounds, and it was a great station, or you could go to Dell and you could buy an x86 machine for a thousand dollars and you could put Linux on it for free. Now, there's the old statement of Linux is free if your time is worth nothing, right? But there was a lot of people out there, myself included, who figured out how to do Linux and realized it wasn't that awful. You just had to invest in it once. So I figured out how to debug Linux and get it installed in machines and I could turnaround and take commodity hardware that was really fairly cheap for the time and build an equivalently powerful workstation to a Sun for a quarter or a tenth of the cost.

When you start to add that up times 100 developers, times 1,000 developers, that's real money, and that's just at the workstation side. Now you talk about the servers. Again, very few people, even today, need this sort of web scale applications that the Googles and the Twitters have to deal with. So for the vast majority of these customers, they didn't need what would Sun call the ultra enterprise servers, these big, thousand-core capable, scalable systems with light, I forgot what the – Interconnect was called at this point, but they were these giant refrigerator-sized systems, right? But this is what Sun was selling. This is the scientific computing, the high-performance computing. These machines were enormous and they were beautiful pieces of engineering. Nobody really needed them. This was a classic 90-10 problem or 99-1 problem. 99% of users do not need what that is, but that makes you a ton of your revenue, so you spend all your engineering on that.

What we happen, what I saw happen from the inside was the Sun sales team had no reason to push the low-end products, because they weren't making as much money on them. They weren't making as much commission on those low-end products. So things like the Cobalt products that we brought in, it was a thousand dollar server and it could run 200 website on a thousand dollar server. Why would Sun sell that when their commission was going to be pennies versus they'd put all their energy into an ultra enterprise account where they could sell one server a year for $20 million and make a giant cut? That sort of culture of commissions and high-end sales and focus on the high-end and, really, the perfection, they missed the forest for the trees.

[00:13:51] JM: You joined Google in, I think, it was 2004. Then 10 or 12 years later you were one of the founding engineers of Kubernetes. Describe your role in the founding of Kubernetes.

[00:14:06] TH: There's been a lot of noise about sort of the origin story around Kubernetes. There's a lot of sort of myth around sort of where it came from and how it began. From my perspective, what happened was we had a reorganization inside Google. We had our technical infrastructure team, which ran Borg. That was where I was working, and we had our cloud team, which is running Google Cloud, which was very nascent at the time, and we got smooshed together. I said, "Guys, go talk to each other."

We started having these regular meetings to talk about, "Well, what could we take from technical infrastructure into Cloud? What would make a good product? What would solve people's problems?" So the folks up in Seattle started to do this prototype of what eventually became Seven, which became Kubernetes. We, down in Mountain View Sunnyvale, we're working on how do we take the idea as a Borg and turn them into a cloud product? Could I do Borg as a service? That was one of the ideas that we were floating around.

The original ideas that we had were very, very different from what Kubernetes end up looking like today, but there was a really core pieces that stuck. When you started to glue these teams together you realized there's a lot of experience between these two teams. The cloud team knew how to build cloud products and knew what was going on with the cloud space. The infrastructure team knew the history of Borg and where all the bodies were buried and how not to step on the same landmines. So this sort of synthesis of these teams brought forth this idea of, "We can do this," and we agreed very early on that it had to be open source. It had to be not just open, but radically open. Different than almost anything else.

We came out really aggressive and we took the prototype that these guys had put together and quickly turned it into something that we could show people. Obviously, we couldn't brand it Seven. So we came up with Kubernetes and showed it off at DockerCon really early. It was very skeletal. But from that day, people could see what it was up to. They got it. A handful of people came up to us afterwards and we're like, "I understand what you're doing with this and I want to be part of it." That sort of lift just took the project.

[SPONSOR MESSAGE]

[00:16:13] 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]

[00:17:10] JM: What I find very interesting about the Kubernetes history, and we've seen this with TensorFlow. We're seeing it with Istio, with Facebook, we saw it with React. I don't know if you're familiar with the React project at all. But when an open source project can clearly layout its vision, even if it doesn't have the full technical underpinnings, it may even have substantially weaker technical underpinnings than the current market leaders. You look at Kubernetes versus Mesos. When Kubernetes came out, Mesos was much further along in terms of something that was ready to actually run production workloads, as I understand the history.

But the fact that there was a clear vision, the fact that there was content, the fact that there was documentation, it put a stake in the ground for how dedicated Google was to making this vision a reality. The cynical way of thinking about it is like, "Oh, it's like fake it till you make it," but the realistic mentality is that this is how you get buy-in from a gigantic fractured developer world. We don't live in the 90s anymore when software development is this provincial Linux ecosystem living across a couple of forums. It's this enormous world where you have to evangelize with significant resources.

[00:18:59] TH: That's really insightful. I agree completely. I do not in any way dispute that Mesos was technologically way ahead of Kubernetes at the time we launched Kubernetes. I mean, anything. A bunch of shell scripts was far ahead of Kubernetes, when we launched Kubernetes. For a longtime, a bunch of shell scripts was our biggest competitor in terms of orchestration solutions, right?

[00:19:21] JM: Literally. Not a shell script. Chef scripts, Ansible scripts.

[00:19:24] TH: There was a survey that was done, it was like, "What are you using for container orchestration?" and it came out number one. This was I think open stack a few years ago. Number one was Kubernetes. Number two, shell scripts … and Mesos was in there somewhere. It's sort of funny, we laugh that like a bunch of shell scripts. What is Kubernetes? At the early days, it was a bunch of shell scripts.

But the best technology doesn't always win. Mesos was a very big, very powerful system that was focus on very big, very powerful users, and Kubernetes sort of came along and started to eat its lunch from the bottom, or recognized a pattern from my history here. I saw this sort of happening right away and I knew that the thing that eats the bottom guy's lunch is probably the guy who's going to win.

Again, another case of sort of worse is better where Kubernetes is not as fully formed as Mesos was at the beginning, but the fact that it was not fully formed is actually – I agree with you completely. The thing that got people hooked, right?

We went to communities and said, "Look, we need help. Here's what we want to do with it. Here's our vision. Here is sort of our existence proof. We can talk about Borg. We publish the Borg paper. Here's the things we did with Borg, and if you read the last page of the Borg paper, it's the things that we don't like about Borg. Mistakes that we made, things that we want to change, and there is a forward reference to Kubernetes." It was like, "We're going to try to fix a lot of these things in Kubernetes."

People saw this and they realized we could jump in on this. We could have a meaningful impact. We could be part of the community. Not just somebody that Google takes patches from, but owners of the community. I think it was a really critical point that, one, we made this thing open source. That we made it really clear to folks like Red Hat that you're as much a project owner as we are. You have full access.

We made Clayton Coleman committer on a project basically right away. Clayton came in and he very immediately demonstrated. He's a smart guy who knows what's going on. Has a ton of experience. Has a very different perspective on things than the goodwill perspective, which has made the system so much more reliable, so much better. It's a different system than it would've been if Google had designed it by itself. The fact that companies like Red Hat are so deeply committed to the success of Kubernetes is because they were able to get into it and be owners of it from the beginning.

[00:21:34] JM: As time has gone on, Kubernetes has advanced far beyond the dog-eared napkin with a vision on it that it started as. Now you've got something that is running lots of production workloads. It's got an entire ecosystem around it, and you can kind of see Google starting to lay out quite a futuristic vision for what the cloud could look like, where you have Knative and gVisor and Cloud Run and you can really see these things potentially fitting together in a way that makes cloud workloads run much differently than they do today. Whatever is the furthest timestamp in the future that you can see clearly into whether it's 5 years or 10 years, how are we going to – First of all, what is that timestamp, and how are cloud workloads going to be different at that point relative to today?

[00:22:49] TH: So this is the Kubernetes 5 year anniversary. So everybody wants to sort of look out five years, and it's hard enough. If you asked me 5 years ago where Kubernetes would be, I would not have –

[00:23:00] JM: You would have said something much further back, right? Like much – It was moving much faster than you anticipated, right?

[00:23:05] TH: It was moving way faster than we could have planned for. We joked at the beginning that we had a crystal ball, because we built Borg and we knew exactly what people

are going to ask for, and we were largely correct. Where we missed the mark was they're asking for things way faster, then we iterated on them through Borg So what took Borg 10 years of evolution to figure out certain problems we've gotten to in three and four years within Kubernetes.

To some extent, we've run out of crystal ball. People are asking for things that are just new and different and to some sense were making things up as we go now. We're solving new problems as supposed to building on things that we already well understood. That's not exactly true. Like you picked on Knative and Istio, these are building on things that have been done in Google and they're an entirely separate project, because they're so big in scope themselves.

So playing out five years, what do I think is going to be very different? I hope that Kubernetes proper isn't that exciting. I hope it's just an assumed piece of ubiquitous infrastructure that you can get anywhere from anyone. There's no real cost associated with it. It's the sort of thing you learn in school as how else would you run a distributed system.

What I think will be interesting for cloud workloads will be the move up the stack. The more you can move up to stack, the better off you're going to be in general. The less you're coupled to your infrastructure, the less you understand what's going on below you. The more freedom you have to build better implementation.

So things like Knative are I think are fantastic, because it gives you a lot of the freedom of Kubernetes to build whatever container infrastructure you want to build, but it puts some limits on you and it says, "If you fit this pattern, we can do really cool stuff with you." So the API becomes, "I'll send you an HTTP request. You'll send me HTTP response. If you can live with that contract, I can scale you up, scale you down, scale you to zero, do pay-as-you-go usage-based billing." There're amazing things you can do with the system like Knative. So you push users up the stack. You don't worry about the fact that it's Kubernetes or that is using Istio or whatever else is going on beneath the covers. That's infrastructure. That's the new hardware. It's the thing that people don't want to have to learn.

Serverless is the same vein in general, whether that's functions or Knative-style containers of the service or the sort of Kubernetes as a service thing that people have talked about and trying

to build in various forms. I think over the next 5 to 10 years, we'll see that just continue to get more and more abstract. But I don't mean that in a bad way. I mean that in a good way. People will be able to focus on the things that they care about and not the things that they don't care about. Giving developers the freedom to not worry about infrastructure is amazingly freeing. Don't worry about what happens if the server crashes. That's fine. We'll handle it for.

[00:25:41] JM: So the Knative story, what I find interesting about that as I was unpacking it in some interviews I did with Knative people is it was very different than kind of the AWS serverless story, or the Microsoft serverless story, or the open source function as a service stories that people were trying to build on Kubernetes, because the Knative vision is that there is no function as a service. There is no container as a service. There is a gradient between those two things, and yet when I – I use Firebase for a couple apps I'm building. On one of them, I use Firebase Functions, which I know runs on Google and I think it's Google Cloud Functions under the surface. I hit a cold start with that. Why am I hitting a cold start on Google Cloud Functions if Google has the Knative secret sauce underneath? You assume Knative, if your contract is just send an HTTP request, get HTTP response. I assume you've solved the cold start problems somehow. Where is the magic? Why am I missing out on the magic?

[00:27:00] TH: So, it's a fair question. Like everything in the space where the technology is evolving. Just a minute ago I said if you decouple, then we can be better implementations. So the implementations of cloud functions will get better as the infrastructure gets better and your app doesn't need to worry about it. You're just going to get better. You're going to get better cold start time. You're going to get less cold starts and more warm stars.

I don't know the details of exactly what would happen in this particular situation, but there are reasons that you can end up being cold start if you've been scaled down or whatever. As we get better at building those implementations, you will see those things go away, but your app didn't change. You don't have to recode your application to get these better behaviors, to get lack of cold starts, to get the scale to zeroes. We didn't ask you to rewrite your application in order to get the pay for what you use sort of billing model. It's like if you follow this contract, there are a million things we can do without having to call you. Which I think is super-powerful. So, why did it happen in this case? I don't know. We'd be happy to look into it for you.

[00:28:00. JM: No. I wasn't trying to go there. But like in terms of the Knative stuff, what I'm just intrigued by is if you have implemented this internally somehow, like if Knative is the internal open source exposure of the API that you exposed to developers that are deploying applications to Borg and they can scale down to zero, or maybe they can't scale down to zero and avoid the cold start problem. Maybe you have some config logic that like allows them to say, "Look, I always need to keep at least one instance of this app so that I can avoid the cold start."

I guess what I'm trying to understand is how can you open source this perspective that there is no function as a service. There is no container as a service. There is a gradient between them, and eventually you're not going to have to worry about this cold start issue if it's not deployed in – It seems like it's not deployed in Google Clout yet. So how do you know that this is a solvable problem?

[00:29:00] TH: So, to be clear. What we have – I don't know how Firebase functions is implemented to be clear. But with cloud functions, for example, is not exactly the same as the thing that we use inside Borg. It's different. It's built for slightly different use cases.

We have ample usage of like AppEngine inside Google. AppEngine has somewhat even more opinionated model than Knative. It has this managed runtimes. At least AppEngine classic had these sort of very managed runtimes time. Even with AppEngine, you hit cold starts eventually. Because if you don't take any traffic, you're going to hit cold starts.

AppEngine classic has like something like a quadrillion users or something ridiculous. It's a very large number, and the vast majority of them take zero QPW. They don't take any queries ever, and it doesn't make a whole lot of sense to keep those things resident in-memory. If they're all in free tier and they literally serve no traffic. They take one query a month or something, and it's probably from a bot.

So for these sorts of applications, like the only way you can get the efficiency is to page them out effectively. Now, for the busier application, there's no reason why we wouldn't have an application. I have an instance running for you all the time ready to go. So, again, I have no idea exactly –

[00:30:12] JM: What about the application type that it only gets a – Maybe this is a rare application type. I'm not realizing it. It only gets one query per month, but that query needs to be very low-latency.

[00:30:26] TH: I would guess that that's a pretty rare –

[00:30:27] JM: [inaudible 00:30:27].

[00:30:29] TH: I'm not actually part of the AppENgine team. So I have no idea what the control knobs are there. What we experience for a lot of these people is they're in the free tier. They want to use this service. They want to serve their API, but they're not actually willing to pay anything for it. So, there's sort of a bounds in the quality of service that anybody could offer for that.

I'm sure there's a way to say I really need this to be low-latency, and like I'm willing to pay X-dollars to keep it low-latency I don't know what the X is. That's seems like a reasonable thing to offer if there's demand for it. I'm not a product person. So I have no idea if you are alone in this sort of thing or if there are millions of you just waiting to come out of the woodwork.

[00:31:07] JM: When I did this show gVisor a while ago, what I found interesting about it was – And maybe I'm totally mistaken about this. Actually, first of all, I guess I should ask you how familiar are you with gVisor.

[00:31:19] TH: Medium.

[00:31:19] JM: Medium? Okay. I find it interesting because it seemed like it was potentially a layer that could be used to swap out Linux is the underlying operating system of a cloud provider, or it's at least abstraction kind of gate between containers and an underlying Linux host operating system. Would you ever see it as desirable to swap out the underlying Linux operating system with like – Is there a better operating system we could build for running modern server workloads?

[00:31:53] TH: That's a tough one. Building an OS as fully featured as something like Linux is incredibly difficult. I mean, look at the thousands and thousands of human hours that have been ported – I mean, thousands is the wrong order of magnitude, hundreds of thousands of human hours that have been poured into Linux and the development of it.

So, to say casually that I could replace it for general purpose applications is pretty daunting. GVisor is an implementation of the Linux syscall API. So, in that sense, it's sort of an OS of its own. But we didn't have to figure out what that API was going to be. We know what it is. It's Linux. What we have to figure out is what those APIs mean in terms of the security context.

A ton of what gVisor does and one of the reasons that it's as lightweight as it is is it passes through a bunch of stuff to the underlying OS. It doesn't have to implement a comprehension of everything that's going on. It doesn't have hardware drivers, which is three quarters of what the Linux codebase is.

So, could we replace Linux with something else? Yeah, probably, but I don't think we'd want gVisor to be a conversion layer between, say, Linux and Windows. Windows is sort of the canonical other OS example, but having it be a translation layer I think would be a disaster. The fact that gVisor can pass through so much of it is what makes it a viable thing.

[SPONSOR MESSAGE]

[00:33:22] JM: DigitalOcean is a reliable, easy to use cloud provider. I've used DigitalOcean for years whenever I want to get an application off the ground quickly, and I've always loved the focus on user experience, the great documentation and the simple user interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU optimized droplets, perfect for highly active frontend servers or CICD workloads, and running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance. The prices on standard instances

have gone down too. You can check out all their new deals by going to do.co/sedaily, and as a bonus to our listeners, you will get $100 in credit to use over 60 days. That's a lot of money to experiment with. You can make a hundred dollars go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure, and that includes load balancers, object storage. DigitalOcean Spaces is a great new product that provides object storage, of course, computation.

Get your free $100 credit at do.co/sedaily, and thanks to DigitalOcean for being a sponsor. The cofounder of DigitalOcean, Moisey Uretsky, was one of the first people I interviewed, and his interview was really inspirational for me. So I've always thought of DigitalOcean as a pretty inspirational company.

So thank you, DigitalOcean.

[INTERVIEW CONTINUED]

[00:35:30] JM: So you worked with Joe and Craig and Brendan and a bunch of other people on the foundations of Kubernetes. What kept you at Google working on Kubernetes specifically? You could've gone to a different company and spread the Kubernetes love or you could've started a company around Kubernetes. What kept you at Google? What was it specifically about Google?

[00:35:57] TH: It's a great question. It's one I think about a lot. I love working at Google. I think the people that I work with are phenomenal, world-class people. At the end of the day, that's the thing that drives me the most.

Joe and Craig are also fantastic people. Geographically, they were up in Seattle. We were down in Mountain View Sunnyvale. So, I didn't get nearly as much day-to-day face time with them. So the coupling that I had with those guys was certainly lower than the people I have in my local office. That's just II think a truth of distributed development.

That's not say that I like them any less, but when they departed to go do their thing, good for them, that's great. I appreciate the enthusiasm there. The project would absolutely not be what it

is without their input. Not their input. It is their brainchild, right? Brendan included. They chose to go other directions for whatever reasons they had.

Joe and Craig had worked on GCE beforehand, the Google Compute Engine, which became Google Cloud Platform, which was an incredibly ambitious project. It was flying in the face of everything the Google knew how to do. I wasn't part of that project, but from – Actually, as the other side of it, as the people in Borg where they came to us with feature requests all the time. I can only imagine how challenging that project was. They built sort of a camaraderie and a rapport through –

[00:37:19] JM: So the Google Cloud folks would come to the Borg folks with future requests?

[00:37:23] TH: Oh, yeah. All the time. The Borg folks of course were very busy. Borg is actually a really small team, and they got the same answer from us that we got, the Kubernetes team got from cloud initially, which is, "Who are you? Go away."

They wanted a million things out of Borg so that they could build VMs in an efficient way. Át the beginning, we didn't know if this was going to be a thing. We didn't know how much resources to pour into that project. The Borglet team was 15 people. Do I have two people that I can carve off to go work on cloud? Not really.

[00:37:52] JM: So that's a pretty interesting problem that you just highlighted, which is basically cloud users want or perhaps need VMs. Google doesn't run VMs. They just run containers on bare metal, right? So you have to either – But why is it – Okay. So Google Borg is just not made to run virtualized workloads.

[00:38:16] TH: Well, it is now.

[00:38:16] JM: It is now. Okay. Interesting.

[00:38:18] TH: So we had to adapt Borg. Borg made a lot of assumptions about what a Borg job was going to do. When you want to run a VM, those assumptions are not always right. So we had to build things into Borg that made it possible to run VMs efficiently.

A great example was network interfaces. We're not going to pass through the VM network into the physical network. There's this Andromeda SDN layer in between. So how do I get an application to be part of the Andromeda network?

From a Borg point of view, that's an app problem. You go fix it in your application. From the application point of view, that's way more privileged than I want to give the VM process, and they wanted Borg to be able to sort of link the two ends of the pipe together for them. So we had to build that sort of a capability into Borg specifically for these sort of VM use cases, and that's just like one. There's a thousand of these.

[00:39:10] JM: How did they finally get somebody to pick up that ticket?

[00:39:13] TH: I mean, it's not that it's a ticket. We have organizational quarterly goal planning and we talk about is this important to the business? Why is it important? Executives talk to executives and they share roadmaps and vision. I mentioned earlier the reorganization. Part of the fact that we got reorg was to stop some of the tension between these teams who needed a lot of the same things but end up building them in two different ways, because what the clouds team needed to build, they had to build on top the cloud stack. What the Borg team needed, they built on the Borg stack. So there are piece of infrastructure that we didn't need to be replicating. So this reorg that ended up fostering the idea of Kubernetes was really pretty critical to the success of the organizations.

[00:39:54] JM: I just did this series of shows on Facebook, and what was interesting about them was Facebook almost missed mobile. There was a period of time where they were just ignoring mobile and then all of a sudden they realized, "Oh my God! We're in the middle of a platform shift."

It's very interesting that sometimes the technology companies that you had presumed to be most in the know take a while to pick up on something. Microsoft didn't see the cloud for a while. Google didn't see the cloud for a while. Why was that? Or did Google even see the opportunity and it was just like, "We've got other things we're working on. We're like working on Gmail and stuff. We're opting out of the cloud for a while."

[00:40:34] TH: I think it's closer to the later. There's 1 million things happening all the time. Nobody knows what's going to be important, and you can't – Even at a company the scale of Google, you can pick up a dozen engineers and just say, "Go explore this thing because it might be important."

Now, obviously, some people are always paying attention. So guys like Joe and Craig were paying attention to what Amazon and Microsoft were doing with cloud and they were like, "This is a big deal. Guys, we should be part of this. We need a cloud product." In some cases, through force of will, they made that happen.

Other places, we've made bets on things, and they've been wrong, and that's okay. We tried to open source our internal container technology just before the time that Docker came out called out, "Let me contain that for." It turned out that nobody cares. It didn't matter, because Docker had a very different way of doing things.

I'll be honest. I initially looked at the first version of Docker when it came out and I went, "Who cares? It's a poor man's version of what we can do already inside Google. So I don't this technology really matters to Google." I wasn't yet working within the cloud product customer-facing external world. I was working inside Google, and inside Google, Docker is still not widely adapted. It's not really possible in the Borg system, because Borg has a very different perspective on things. But it just didn't matter, right? Boy, was I wrong.

When I started to look at it from the how would I take this and make it available to the outside world, suddenly I was like, "Oh, well Yeah, Docker seems like an obvious starting point." Their UX was so much – I don't want to say better. It's very different from what Borg – I'll say better.

[00:42:05] JM: Great logo also.

[00:42:07] TH: Very cute. The Docker user experience comes at it from such a different perspective than Borg. We always joke inside Google that everything we build inside Google is three quarters finished, because we never put the plaster on the walls. Everything is exposed

wiring and open rafters. Because Googlers can tolerate that because they have my phone number if they have a problem. That's just not true for the outside world.

So Docker came along with plaster walls and paint and beautiful curtains and said, “Here’s the UX for how you run a container,” and they nailed it. They got to the heart of what made developers excited. When you combine that with this sort of larger experience that you can get from a system like Kubernetes and you say like, “Developers are happy. Administrators are happy.” That's a blend that doesn't happen that often.

[00:42:53] JM: I also think it's hilarious that Google saw the future way too early with AppEngine. [inaudible 00:43:01] that's the way the people want an infrastructure, basically. Too bad, Google was 12 years early. But I think we already did a show on AppEngine. You got to close off. You got somewhere to be. I want to ask you one more question.

So I've been to KubeCon. I’ve been to Google Cloud Next. I've been to Google I/O, and across these different conferences, I feel like KubeCon is the most kind of in the past, like re-factoring older applications. Google Cloud Next is sort of like a futuristic version of the present. Then Google I/O is like in the distant future. It’s like this panorama of different places in time. Do you get that sense also?

[00:43:49] TH: That’s an interesting perspective. I’ve never actually been to Google I/O. It’s really hard for Googlers to get into I/O. There are so many limited, so limited spots to get in that unless you’re working in one of those product areas, there’s really a very low change that you’ll be able to go. I would love to go to Google I/O some time. I’ve just never had an opportunity. I’ve never worked on one of those projects. So I get the announcements at the same time everybody else does.

Modula, whatever they’ve told us internally, it’s an interesting characterization. I wouldn’t say that KubeCon is really about the past. I mean, I think it’s about pushing forward. Google Next, we spent a lot of time talking about Anthos. Anthos is our hybrid multi-cloud solution from Google Cloud. It’s Kubernetes. But who’s the user-base for Anthos? It’s really enterprise customers who have one leg in the past, whether that’s in their own data centers, their monolithic applications, their VMs, whatever. The reality is those things exist and they’re not

going anywhere anytime soon. I wouldn't say that that's a futuristic vision any more than Kubernetes is. It's like a helping hand is how I would rather see it.

I/O, I love. I love the things that come out of I/O, because I love gadgets and I love the future-looking stuff and I'm always the first one in line to buy the next Google assistant, whatever technology is out this year. I'm trying to find an excuse to buy a new phone. But it's an interesting characterization. I can see how people would look at these three events and see very different themes. It's definitely true. The things that I want to talk about here at KubeCon are different than the things I want to talk about at Next. It's a different customer. It's a different user.

[00:45:26] JM: Are there any interesting elements of Google infrastructure that you would like to see in the open cloud, but they're hard to port to the open cloud model that we have today?

[00:45:42] TH: Well, that's an interesting question. The things – I'm an infrastructure nerd. So the things that excite me most about Google and doing things in Google is a lot of the infrastructure. Little details, little integrations, like monitoring. When I write a job, a Borg job for Google – First of all, my C++ program, or whatever, it doesn't call me, and it calls Google Main. Google Main sets up a bunch of stuff, and that includes the RPC network and all these cool look asides for on-the-fly rate limiting and access control and all these neat things that I don't have to do in my application. In some sense, you get that with Istio. On the monitoring side, you get all these really cool metrics out of the box, all the libraries that I'm using for my Google application are already instrumented.

So the minute I turn on my application, I can literally write my Hello World app and I go look at a URL of that app, and it's got hundreds of metrics that are collecting information about what my application [inaudible 00:46:37] and it calls this. How many bytes of log line have I written? Those sorts of like really find-grained metrics, and those are all being collected automatically. I didn't do anything. They're automatically being ingested into a logging system. They're automatically being merged with all my Borg information. So I can tell at this point in time, I was doing this many bytes per second of logging, and my CPU usage was X, and my memory usage was Y, and all these things are just automatic.

I can correlate my logging versus my CPU – I'm picking on logging, because it's a trivial thing, but it's instrumented. I can correlate these things just automatically out of the box. They're automatically retained for a short but reasonable amount of time, and I can just go to the monitoring dashboards and start graphing right away. I didn't have to configure it. I didn't have to export anything. I didn't have to set up an ingestion. I don't have to wait for hours for it to kick alive. It's just there from the minute I run Hello World.

So I'd love to see that level of auto-magicness continue to go through Kubernetes. Istio is fantastic and that you get a lot of these cool network integrations. It's not as seamless as it needs to be. Prometheus is a fantastic piece of software. It's not as automatic as it needs to be. Instrumentation across libraries, things like open census, open tracing. Those things are not as ubiquitous as they need to be.

As we get more and more of this, as we build out this ecosystem of cloud native, I'm using quoty fingers, that these things become more and more pervasive, and then you can get that automatic.

[00:48:08] JM: Yeah. That's inspiring. I guess that is cloud native. At first as you were saying, I was like, "That's language native." But then I realize, "No. Actually, you need the cloud to be able to have the guarantees of having a place to throw all these instrumentation –

[00:48:23] TH: It's not even language specific. I don't care if you write in Python, or Ruby, or Go. You can export the same metrics and get the same tracing all ingested into the same systems and you can produce the same results.

[00:48:33] JM: Tim Hockin, thanks for coming on.

[00:48:34] TH: Thank you for having me.

[END OF INTERVIEW]

[00:48:39] 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.

[END]