EPISODE 559

[INTRODUCTION]

[0:00:00.3] JM: Go is a language designed to improve systems programming. Go includes abstractions that simplify aspects of low-level engineering that are historically difficult; concurrency resource allocation and dependency management. In that light, it makes sense that the Kubernetes container orchestration system was written in Go. Erik St. Martin is a cloud developer advocate at Microsoft, where he focuses on Go and Kubernetes.

He also hosts the podcast Go Time and has written a book on Go called *Go in Action.* Recently, Erik helped build the Virtual Kubelet project, which allows Kubernetes nodes to be backed by services outside that cluster. If you want your Kubernetes cluster to leverage abstraction such as serverless functions and standalone container instances, you can use a Virtual Cubelet to treat these other abstractions as nodes.

Erik also discussed his experience using Kubernetes at Comcast, which was a great case study. Near the end of that episode, he also talked about organizing GopherCon, a popular conference for Go programmers. If you are organizing a conference or thinking about organizing one, it will be a useful piece of information for you. There's lots of discussions about the hazards of organizing a conference.

Full disclosure, Microsoft where Erik works is a sponsor of Software Engineering Daily.

[SPONSOR MESSAGE]

[0:01:30.5] JM: Today's episode of Software Engineering Daily is sponsored by Datadog. With infrastructure monitoring, distributed tracing and now logging Datadog provides end-to-end visibility into the health and performance of modern applications. Datadog's distributed tracing an APM generates detailed flame graphs from real requests, enabling you to visualize how requests propagate through your distributed infrastructure.

See which services or calls are generating errors, or contributing to overall latency so you can troubleshoot faster or identify opportunities for performance optimization. Start monitoring your applications with a free trial and Datadog will send you a free t-shirt.
Go to softwareengineeringdaily.com/datadog and learn more, as well as get that free t-shirt. That's softwareengineeringdaily.com/datadog.

[INTERVIEW]

[0:02:31.3] JM: Erik St. Martin is a cloud developer advocate with Microsoft. Erik, welcome to Software Engineering Daily.

[0:02:36.6] ESM: Thanks. Glad to be here.

[0:02:38.4] JM: Yeah. It's great to have you. We're Going to talk about Go and some various systems that are built with Go. When Go was started, when the language Got started, the primary languages around that time for low-level systems programming I think were just C and C++. There were a few other fringe languages, but mostly it was C and C++. What were the shortcomings of the C and C++ language environments?

[0:03:06.7] ESM: I think there's a lot of complexities in the languages themselves, especially as they've been expanded over the years, especially when you talk about C++ and then some of the versions there. There's a lot of foot guns there. I think that what they were looking for was one, a language that was more simple and easier to reason about without having 20 ways of doing the same thing and confusion looking at code.

In addition to that, some of the newer languages, one of the things people love about them is a garbage collector, where you're not having to manage your own memory and worry about those things. I think the primary motivation was also to design a language from the ground up that thought about concurrency from the beginning. Because when we think about threads and some of the patterns that we use for concurrency and parallelism, these were after-thoughts, right? All these languages that are being used as systems languages now, they were added on after the fact as far as instead of thought about from the beginning.

[0:04:10.0] JM: What did that lead to? What were those low-level – because historically, low-level programming is really difficult around these issues of concurrency, or resource sharing. I think about semaphores and mutexes from college when I had to program in my operating systems class. What are the language mechanics that Go introduces to make those concurrency problems easier?

[0:04:37.0] ESM: When we think about Go, the two primary ones are Go routines, in which we can think about as lightweight threads. Most people use them, just treating them as threads, but under the covers these things are actually multiplexed across real operating system threads. The difference between a Python, or Ruby is that there's no global interpreter lock and things of that nature. Then in addition to that, the concept of channels, which I want to say was squeaker knew squeak that they borrowed that from.

This is based off the CSP paper, the communicating sequential processes white paper. Essentially, it's just a way of passing data across Go routines in a synchronous way. They can be used as cues if you use buffered channels, but this allows you to handoff control of the data to a different Go routine.

[0:05:29.8] JM: Did they borrow from the actor model of concurrency for that? I think the channel aspect was primarily based off of the CSP paper, and I think it was new squeak. There's a couple talks by Rob Pike, where he talks about history of the language and it Goes through and I think Oberon and some other languages, some of the different concepts were borrowed from. One of the things that people love about Go is there's like 25 keywords. It's really easy to reason about. You can fit the entire language spec in your head at one time.

[0:06:03.0] JM: Was that another one of the – what were the objectives of the creators of Go? You mentioned Rob Pike. This is a guy that's been around forever designing languages and systems. What were the objectives of the creators of Go?

[0:06:16.7] ESM: They wanted to create a new systems programming languages and they were primarily targeting C programmers and C++ programmers with concurrency being at the forefront of it; a very simple language that people could reason about, because they were looking at designing a new language for large projects, where tons and tons of people were working on it and people could easily just jump in and out of it without –

Because when we look at most languages that have been around for a long time and have – I don't want to call it cruft, but they accumulate features over the years and then everybody has their own way of doing things. I think that was some of the primary considerations when designing it.

[0:07:00.6] JM: When did you start working in Go? What pushed you into starting to build programs with Go?

[0:07:06.5] ESM: I started in late 2011. Well, I looked at it in 2009 when it was first released. I worked for Disney at the time and a few of us huddled around and looked at it, and I was like, “Ah, this is really cool.” It didn't really sell me. The one thing that I find hard about Go where it's like, okay what's your elevator pitch? Why should you use this?

It's like there's not a silver bullet. Everything that's there and everything that's not there, and it really took me using it to fall in love with it. Late 2011 I started at a company and there was a small service that was running. It was getting a lot of traffic and it happened to be written in Go. None of the rest of the team really wanted to learn Go, and I've always been a – I love learning new technology, so I'm like, “Why not?”

I looked at this a couple years aGo and it looked interesting. I'll learn it. Then that was really as I started building stuff, especially highly concurrent stuff with it that I really started falling in love with it.

[0:08:05.9] JM: What are the kinds of applications that are a Good fit for Go? What were you working on at Disney? I guess, you could you could talk about later at Comcast as well. What are the kinds of things that you've seen be a Good fit for Go?

[0:08:20.5] ESM: At Disney, they may be – actually I think I've heard of some groups using Go now, but at the time they were primarily a Java shop. This was just programmers geeking out, looking at a new language. Yeah, as I said, the language was really designed for systems programmers and what they didn't expect was so many of the Ruby and Python crowds to come to it, because a lot of people are starting to build these large-scale applications using DjanGo and Rails and they're starting to run up against performance issues and they're looking for something else, and they don't want to Go to C or C++.

That felt really great middle ground for people. Probably in the same vein as me, they started experimenting with it and as you write code with it, you really start to fall in love with it. Right now, a lot of people are writing API servers in Go. You get a lot of that. There's some people doing web development in Go. It's not quite there.

[0:09:14.9] JM: These are people that are moving from Rails, or DjanGo, or Nodejs to Go?

[0:09:20.4] ESM: Correct. You can find story after story of large Silicon Valley companies doing this even, Fortune 500 companies that are migrating over to Go some of these restful services and things like that.

[0:09:33.8] JM: What's a tooling like for that? Because I mean, you obviously with Ruby and Python and node, you have this giant package ecosystem. Is the Go package ecosystem, does it rival that?

[0:09:44.7] ESM: One of the beautiful things about the language in addition to just writing code in it and being able to jump in and not really be as confused by people's code is that it compiles down to a static binary, which makes deployment a lot easier. Because if you come from a Ruby or Python background, you're familiar with having to make sure all the modules are over there and make sure the right version of the interpreters there and all these things and you just didn't have that problem with Go. You just, here's the binary in Go and cross-compilation from the beginning. Not necessarily the beginning, but from early on was there – there's very little that you have to do to get this thing to run on Mac. You can develop on Mac, but then compile it for Linux to deploy it. Windows support is built-in.

[0:10:33.4] JM: Those compilation issues, that was another one of the focuses of the original mission of the language that the dependency management, the fact that it compiles down to a single binary without dependencies. This simplifies the process of taking something from your laptop and making sure it runs also on production servers. Why is it so useful? Why is it so important when deploying to modern infrastructure?

[0:10:59.0] ESM: When we think about deployment, especially when Go was created we have these dependency management things and especially if you have multiple applications that are depending on different versions of the same modules and you get into that dependency nightmare. Just having the static binary to deploy out becomes a much simpler process.

Containers, I would say helped. The introduction of containers allowed us to bundle these dependencies inside an isolated unit that we could move around. The containers hadn't yet been invented. I don't want to say invented, but modernized and popularized.

[0:11:40.8] JM: Speaking of containers, we've done a bunch of shows on the usage and the adoption and migration to Kubernetes. We've done fewer shows about the actual creation and the implementation of Kubernetes. Kubernetes is written in Go and this might be a perfect example of what kinds of applications Go is useful for aside from just your run-of-the-mill web application.

Kubernetes is a systems, essentially a low-level systems application for managing your cloud resources, managing your containers across your different nodes and the pods within those nodes and the containers within those pods. Why was Kubernetes written in Go?

[0:12:25.8] ESM: This one is probably a better question for the people who worked on the team, but I vaguely remember hearing that when they were looking at trying to build an open-source version of Borg, which is what Kubernetes is based off of and that's the orchestrator that Google uses internally to schedule out containers, they were looking at more modern languages. Of course Go being created inside Google I think had some appeal in addition to it, the concurrency primitives and the static compilation. This just took off.

There was probably the fact that it leveraged Docker under the hood in the beginning. There was probably some degree of using Go because, Docker also uses Go, but I wasn't involved in that decision. It's a lot of speculation there.

[0:13:12.8] JM: Fair enough. You eventually wrote a book on Go. *This is Go in Action*. What did you learn about the language while writing *Go in Action*?

[0:13:20.6] ESM: That's a good question. That was a few years ago now since we wrote the book, but I can tell you that I know Go better because of writing the book. It's really interesting when you think that you know a topic really well, but then you want to write about it, or speak about it you, do a lot more research to make sure that you're speaking accurately and dug through a lot of compiler and standard library just to make sure that we're presenting topics the right way.

Even outside of writing the book, there was a lot of – because everybody was new to Go, it took a few years for a lot of idioms and best practices to evolve and there were common mistakes that a lot of people, including myself made from the very get-go. I overused the crap out of channels when I first started. I was like, "Oh, I don't need mutexes. That's what channels are for." There's still a degree of using mutexes, right? If you're just trying to manage state with inside say a struc, like a mutex is probably your best bet. A channel is more about passing control of data between two processes.

[0:14:26.7] JM: What are some other lessons of managing concurrency intelligently in GO that you learned while writing the book?

[0:14:32.8] ESM: I think there's a lot of when we look at some of these things like learning best practices, some of the things are how to intelligently stop Go routines so you don't end up leaking them. They don't spin out there and run forever when you do things like fan out, fan in and you want to cancel a request, how do you do those things?

There was a lot of patterns that emerge there. Now we have the context package, which makes it much easier to do that. I wish I sat down and made a list of all these slight fails, because I know I've made a ton of mistakes and I know I look at code I wrote back in 2011, 2012 and it appalls me.

[SPONSOR MESSAGE]

[0:15:20.5] JM: Azure Container Service simplifies the deployment, management and operations of Kubernetes. Eliminate the complicated planning and deployment of fully orchestrated containerized applications with Kubernetes.

You can quickly provision clusters to be up and running in no time, while simplifying your monitoring and cluster management through auto upgrades and a built-in operations console. Avoid being locked-in to any one vendor or resource. You can continue to work with the tools that you already know, so just helm and move applications to any Kubernetes deployment.

Integrate with your choice of container registry, including Azure container registry. Also, quickly and efficiently scale to maximize your resource utilization without having to take your applications offline. Isolate your application from infrastructure failures and transparently scale the underlying infrastructure to meet growing demand, all while increasing the security, reliability and availability of critical business workloads with Azure.

To learn more about Azure Container Service and other Azure services, as well as receive a free e-book by Brendan Burns, go to aka.ms/sedaily. Brendan Burns is the creator of Kubernetes and his e-book is about some of the distributed systems design lessons that he has learned building Kubernetes.

That e-book is available at aka.ms/sedaily.

[INTERVIEW CONTINUED]

[0:16:56.5] JM: We've had a number of different authors on the show who have written technical books and they always say that the process of writing a book – you hear the same thing from people who speak at conferences or whatnot, the process of writing a book, or a talk hardens your understanding, that's why it is pretty useful for people to teach what they're working on.

I want to talk a little bit more about Kubernetes. You were working on Kubernetes when you were at Comcast and you worked at Comcast for I think about a year and a half. I've seen you speak about building systems there. With the specific application of Kubernetes that you talked about, that you worked on at Comcast was not something I had heard before. You actually touched on some interesting ways that TV signal actually makes its way to your television set. Can you explain why you were working with Kubernetes at Comcast?

[0:17:47.3] ESM: Yeah, so I've actually been working with Docker and Kubernetes since they were release on Github. It's not necessarily because I'm always pointed in the direction of what the new hot technologies are, it was actually by happenstance, because back then in 2013-2014 we hadn't even started GopherCon yet.

Go hadn’t reached the level it has now, so we used to just crawl Github looking for interesting projects written in Go. At the time, I happened to be trying to work on this concept of a framework for building distributed systems. It fell in-line as a better way of doing without the problem I was trying to solve.

Brian Ketelsen and I actually have contributed to both Docker and Kubernetes. We wrote the DNS service-based discovery logic for it. We got involved in the project very, very early on. I happened to be in between jobs and Comcast reached out. I got hired there as a systems architect. Basically what they were looking at doing was replacing this physical hardware that multiplexed video channels together into one stream and sent it to the devices that actually modulate the signal on to the cables that make it out to your house.

What they were looking to do is now moving away from that hardware was trying to design a distributed and fault-tolerant way of doing this and as well as making it easy for the video engineers to manage these things. We started thinking about the design and Kubernetes got tossed out there in the beginning.

It was like, “This is a really, really specific problem and we're probably better off building our own system,” and at the time, I had spent the past few years building distributed systems. My manager at the time was just like we should really look at Kubernetes. I started thinking about it like, “Okay, well what would this look like if we had to make it work in Kubernetes?”

Things fell in place. You start looking at it and you start realizing all the things you would have to build, the scheduler and all these things to build your own distributed system to orchestrate these things out and do failover and recovery, and you start thinking about all the components that exist inside Kubernetes. That's really when my vision of Kubernetes shifted from an orchestrator for containers, which is basically what most people would describe it as.

I started thinking it more in line of a framework for building distributed systems. It's extremely modular, you can change out the scheduler, you can run them side-by-side, you can write your own networking plug-ins, you can write your own types and controllers for those types. That's really when that hit me, just the extreme power in that.

At the time they have this thing called third-party resources, which was how you introduced a new type into Kubernetes. It was really only being used by people developing features for Kubernetes and it just hit me like, "What if we created the type that represented the video stream and then there's just a YAML file to represent the inputs and outputs and all these things and we build our own controller?"

The controller basically just creates other types within Kubernetes, in this case a pod for the multiplexer and some of the other things that needed to run beside it. We could use node affinity and things like that to make sure that it ends up on a host as close to the house as possible. Just the sheer power of that just astonished me and it really excited me.

For the past couple years, I've actually been trying to preach that. I gave a talk in Ghent, Belgium back in February talking about this; Kubernetes isn't the top layer of the cake. We need to start thinking about what those things are and the power of that and teaching people how the control plane works and how you can inject yourself and how and why you might want to customize a particular component.

It's worked really, really well for Comcast. They Got a lot of power and functionality with only having to design the parts that were specific to our domain. We had to do some stuff around network plugins and things like that. We had to add support for real time threading to Docker, but at the end of the day we saved herself a ton of development time.

[0:22:04.3] JM: If I recall correctly, the project you were working on was migrating this old way that signal gets delivered to your television via your set-top box to a newer system that delivered the television signal over Internet Protocol. That's the degree of low-levelness that this was, right?

[0:22:28.6] ESM: Correct. When we think about the way TV has worked for ages, I'm not sure how long QAM has been around, but at least the 80s, I would say.

[0:22:37.2] JM: Was that QAM?
[0:22:38.8] ESM: QAM is Q-A-M, Quadrature Amplitude Modulation. It's essentially how the signal goes across the coax cable. When we think about the way a traditional set-top box works, it works a lot more like the radio in your car. Coming across that coax cable depending on the city you live in and how many channels are there, you've got 1.5 gigs to 3 gigs of video just crossing that wire.

When you change the channel on your set-top box, you're essentially just tuning to a frequency on the cable. I think it's 6 megahertz, and then your demodulating out a channel. Standard definition, you can fit up to eight in that one frequency hi-def, it might be two. That's all your set-top box is doing is it's tuning to that frequency and then demodulating out that particular video stream.

Even video on-demand, there's a set frequency that allows you to go back up to the cable center and you're like, “Hey, I want to watch this movie.” It looks to see this particular program ID on this frequency is free so I'm going to start pushing the video down that frequency and then I'm going to tell your box to tune to that.

Then when we start thinking about the Xfinity systems and the fact people use Netflix and Hulu, these are all IP-based. This is on demand. Then you can think about the way that they save bandwidth here is just CDN, right? If a hundred people in one neighborhood are watching the same Netflix thing, it's just cached in a CDN.

That's the way all the new stuff works and that allows to free up bandwidth and things like that, but you have to support all the old customers. We have IP video and QAM coming across, well you have the multicast video coming across the same backbone, so you're essentially duplicating traffic. In addition to replacing this hardware, what this multiplexer did was consume the same content that the IP boxes were consuming through the CDN and then pushing it out as the coax video.

[0:24:47.1] JM: To implement the Kubernetes solution to this, do you have to work much in Go to create those custom third-party resources? Or is that just the creating something in YAML file? What's the developer experience like for creating that custom solution you had to make?

[0:25:05.3] ESM: Yeah. We used Go. It happen to be a language of choice within the team that I was on, and the client library in Go is always going to be up to par and have all the newest functionality and be the most tested and supported, because Kubernetes itself is written and Go. They have been working on the client library and other languages.

Even if you don't use say a client library that's offered by them, you're just talking to the Kubernetes API server through rest call. You could use any language. There's just some helpers for creating these specific types and things like that.

[0:25:42.7] JM: What were the lessons from that migration, speaking broadly? Because it sounded like a pretty long and technical process.

[0:25:51.5] ESM: I mean, there were some lessons learned around running real time threads inside containers and things of that nature. There were some things we had to create and ultimately we removed, because in future versions of Kubernetes they implemented features that we could use instead of our own custom stuff. We spent a lot of time on networking obviously, because this is a very specific problem we had to solve.

I guess, there were some difficulties too in the way ad insertion worked and opening up ports and things like that. I’m trying to think. I mean, it's actually pretty straightforward to create your own controller and when you break down even the controllers that are implemented inside Kubernetes, their job is pretty straightforward. If we think about the scheduler, the implementation of the scheduler and bin packing and things like this is complex the algorithm.

When we think about what the component actually does, it's just a long pole to the API server asking it to give it any pods that don't have a node name assigned. Then it does its thing. It decides which node it should be on based on annotations and using bin packing and things. Then the rest of its job is just to set the node name, push that back to the API server and that's all the scheduler does.

From that point on now, it's onto the couplet, which is the thing that runs on every node and it's doing a similar thing. It's just long-polling the API server saying, "Hey, I want to know about the creation updation, update or delete of any resource that has my node name," and then it reconciles.

[0:27:27.5] JM: Right, so the couplet is the node agent. This is an agent that runs on each node and implements pod and container operations. I want to talk about the virtual Kubelet project that you worked on. Maybe first it's good to describe in a little more detail what the Kubelet does.

[0:27:45.8] ESM: Yes, so I'm assuming they're show notes and I have a blog post where I walk through this, if diagrams help. The Virtual Kubelet as you said is the agent that runs on every one of the worker nodes, the nodes that are actually going to run pods, and it's responsible for a number of things. It handles communicating with the image registry to pull down the containers and things like that and to clean up after itself, it sets some firewall rules and things like that, it communicates with the container runtime Docker, things like that's actually create the container.

Essentially, what its job is is just to watch the API server looking for pods that's assigned to it and then reconciling the difference, like the API server says I'm supposed to have this pod running, but I don't. I need to create it. Or I just got notified that somebody deleted this pod, so now I need to shut down this pod.

I mean, it does some other things as far as mounting and config maps and volumes and things of that nature, but it's really the workhorse on the node. It's the thing that does the most work. Everything else is just changing state inside Kubernetes. If we think about, like say a replication controller or a replica set, its job is just to monitor replica set objects and then see how many pods match those labels, and then create a new one or delete one based on that. They're really just creating and modifying objects, which within the Kubernetes API server.

[0:29:16.0] JM: Since you joined Microsoft, you started working on the Virtual Kubelet project. This allows nodes within the Kubernetes cluster to be backed by services outside of that Kubernetes cluster, so rather than using just a node, rather than interfacing with just a node or a virtual machine at the lowest level, you could potentially interface with serverless functions, or with ephemeral container instances, like the AWS Fargo or Azure Container instances.

Explain the motivation for the Virtual Kubelet.

[0:29:55.2] ESM: Yeah. It's interesting. We have AKS, which is our managed Kubernetes offering, where we manage the control plane for you. We're not responsible for the team, myself included that created Virtual Kubelet; aren't responsible for the initial idea of doing this. I'm not sure who triggered it, but Brendan Burns wrote the first implementation of it in typescript called the ACI connector, and it did the same thing.

Myself and the team I worked with are responsible for this concept of making it its own project, where it's modular, where anybody could plug in these interfaces and things like that and writing it in Go. I think the motivation was when you think about the managed Kubernetes offering, you still have to manage the nodes, and some people don't want to manage the nodes either, so it's this green field, just blue sky idea of, well what if you didn't have to manage the nodes either? What if you could just have a nodeless cluster?

For anybody who's not familiar with what ACI is, you can essentially think about it like pods as a service, a group of containers and you just tell Azure like, “Hey, I want you to run this thing and done.” That could work for a lot of people. They don't have complex scenarios, where they have these services that need to do service discovery between each other and all these things. They just have a couple pods that are rather isolated and just ran.

A lot of what it's used for is also – if we think about just doing batch work. Sometimes you just have these processes you just want to run to completion. They’re doing database cleanup, or all those things. Having this idea of not necessarily running the workload on your main cluster is also appealing, because you could burst out therefore when traffic crosses the boundaries of how much resources you have in your cluster.

Or maybe you just want to run your batch work out there and you've Got CICD pipelines, where you're trying to run unit tests for every commit. These things can basically just run and you pay per second as they run. There was a lot of appeal for some scenarios like that.

[0:32:06.9] JM: The story is that you've got your Kubernetes cluster that's running your application nodes for your startup, for example. You have a set of nodes that are up, that are serving the containers to you. Basically, you've just got this the set of resources that you have a fixed number of virtual machines. You could spin up more virtual machines to service this cluster over time if you wanted to, but for the most part you've got a fixed cluster or otherwise you have to spin up more machines.
Throughout the day, you might have bursty workloads. Maybe you have more users that jump on the website, their startup manages, or perhaps you just have MapReduce jobs that are scheduled throughout the day and you have bursty workloads there, or like you said, you have CICD workloads and you want to spin up an instance of your application, or you want to spin up a bunch of instances of your application so that you can parallelize a bunch of tests across those different instances of the application.

For these kinds of usages, you would want to have ad hoc scaling of your cluster. The cheapest way to do that may not be to purchase more VMs on the fly and then spin up containers on them, the most cost-efficient way might be to use serverless functions, or container – the stand-alone container instances that you're just spinning up ad hoc. Am I getting it right? That's the motivation of this project?

[0:33:34.0] ESM: Yeah. I mean, that's exactly right.. There's there's a few different motivations there. I mean, especially if you think about CICD, right? You've got a couple of options. You could have a VM just sitting there for it. Most the time you probably don't want to affect your production cluster for the CICD jobs, so you could have VM just sitting there and you could have dead days where there's not a lot of commits and then you can have ones where people are backed up.

VM time is VM time. If it takes two hours to complete these things serially, you're still paying the same amount of money as you are spinning it up and running on them in parallel and just having it done. I think there's a lot of appeal there to just hey, just run this until completion in ACI, I pay per second, and I'm never paying for idle time.

The creation of Virtual Kubelet is really interesting too, because we talked about Kubernetes objects. A node is really just an API object that exists, so you can contact to the API server and just submit something just like you do a pod and you're like, “Hey, here's this node that exists in the cluster,” and the API server takes your word for that. Then the Virtual Kubelet just behaves the actual Kubelet does by monitoring for things that are assigned to it. Then it just translates these objects from a pod definition, which it sees as part of its watch into whatever backing service you want that to be.

Really it could be anything. The power of Virtual Kubelet is the first implementation for us was ACI, right? There's a lot of appeal with that. You could have a provider inside Virtual Kubelet, which spins up a virtual machine and then runs just that pod inside your virtual machine for a complete isolation. There's some appeal there as well for people who need that type of isolation for multi-tenancy and things like that

[SPONSOR MESSAGE]

[0:35:41.0] JM: Failure is unpredictable. You don’t know when your system will break, but you know it will happen. Gremlin prepares for these outages. Gremlin provides resilience as a service using chaos engineering techniques pioneered at Netflix and Amazon.

Prepare your team for disaster by proactively testing failure scenarios. Max out CPU, black hole or slow down network traffic to a dependency, terminate processes and hosts. Each of these shows how your system reacts, allowing you to harden things before a production incident.

Checkout Gremlin and get a free demo by going to gremlin.com/sedaily. That’s gremlin.com/sedaily to get your free demo of how Gremlin can help you prepare with resilience as a service.

[INTERVIEW CONTINUED]

[0:36:40.9] JM: To describe the abstraction, the architecture of the Virtual Kubelet in more detail, one constraint of running a typical Kubernetes cluster is that you have to be working with nodes, that is typically you're usually working with a virtual machine. The Virtual Kubelet lets you use the abstraction of a provider, instead of the abstraction of a node.

This provider abstraction is what lets you tap into serverless, or ACI, or whatever other abstraction you want to conjure up that fits the bill of a provider that looks like something that Kubernetes can interface with. Can you describe in more detail what a provider is?

[0:37:23.8] ESM: Yes. Like you said, when we think about how the interactions behave in the Kubernetes world, the Virtual Kubelet represents itself as a node in the cluster, therefore the scheduler can schedule things to it and all of this stuff. From that point on, the Virtual Kubelet's responsibility is just to make sure that happens. Inside the core Virtual Kubelet logic is all the reconciliation loops and watching Kubernetes and asking the provider like "Hey, what pods do you have running?" Then determining the differences and saying, "Oh, well you need to leave this one."

From the provider interfaces responsibility, there's just a simple interface, tell me all of the pods you have running right now, create this pod, delete this pod, things of that nature. It doesn't really care how you make that happen. The sky's the limit for what you can think of for running your container. It just cares that you can tell it what you're running and it can tell you what you need to do to reconcile with what Kubernetes says you're supposed to be doing.

[0:38:26.0] JM: Describe the process of writing the software for Virtual Kubelet.

[0:38:31.1] ESM: The team and I getting together to do that?

[0:38:33.3] JM: Yeah, yeah, and the architecture and the scoping and just the process of architecting this.

[0:38:38.5] ESM: Yes, it was really interesting. We were presented with the ACI connector that was written in Typescript and we fell in love with the idea. I'm like, "There should definitely be

Go," because everybody in the container world is using Go. If we want more people to contribute, that should be the language. We knew we needed to rebuild it in Go.

Then the idea came up that, hey, this should really be like a first class project that people can use especially when we start thinking about Kubernetes and the cloud space and what if you don't even want to manage your own VMs? You just the power of the orchestration of Kubernetes.

We kicked around ideas and talked a little bit about the design, but it didn't really start coming to fruition until KubeCon. The primary developers that were going to be on the project are all the Go people who are cloud developer advocates at Microsoft. .Then we had some other teams who jumped in as well. It was, I think eight teams across Microsoft that came together, so we built it into the Azure tool. People built CICD for it and all of these things. It was really amazing to see all the teams come together.
Basically what we decided was none of us seemed to be able to fit this into our daily schedules, so let's fly out to KubeCon a week early and we'll just rent a room in the hotel and just hack on it for a week. We threw together the first prototype across 18 – I forget how many people were there. I've got a master list on the blog post giving a shout out to everybody, but I think there was like –

[0:40:10.1] JM: That sounds pretty fun.

[0:40:11.0] ESM: Ten people. Yeah, we just got together for a week and just hacked on it. We had some people from the ACI team there, which is great because, when you couldn't figure out things with the API and they weren't working right, they could they were there immediately to help.

[0:40:23.4] JM: That sounds like such a great approach. This was in KubeCon Austin, the recent one?

[0:40:27.6] ESM: Yes.

[0:40:28.1] JM: Cool. Yeah, and I think, what you mention there it was Brendon Burns handing this thing to you and it was originally written in Typescript. I just find it funny that Brendan, he continues to I guess lead the evolution of Kubernetes through these prototypes. I mean, he did this also with – what's that? The other project he's working on. Was it called?

[0:40:52.4] ESM: Meta particle?

[0:40:53.1] JM: Meta particle, right. I talked to him a bit about that and that's another really ambitious project, where he threw together a prototype and he was like, "Hey, I think this is cool." That seems to be his way of contributing art is he makes a prototype to illustrate what he's doing and then gets other people to be excited about it and latch onto those projects.

[0:41:14.1] ESM: Yeah, I mean it's fantastic and I love the people are starting to think about these things. Gabe Monroy, that's where I got the – I think he said something about Kubernetes was never meant to be at the top layer of the architectural cake. The talk I gave in Belgium, I quoted halt and catch fire saying, Kubernetes isn't the thing. It's the thing that gets us to the thing.

I think a lot of us who have been around the space for a few years are starting to recognize that. That people are really starting to adopt these things, but everybody's still focused on using it as is. It's really exciting to find people who are as ambitious as meta particle is. Maybe that's not how things look, but I love that people are starting to experiment with these things and what could a good level of abstraction look like on top of Kubernetes. Because it does require a lot of knowledge to learn and use these things.

Then especially if you start talking about being an operator too – that's always the big thing I tell people when they're looking to do to adopt Kubernetes is I work for Microsoft, so what I love you to use IKS? Absolutely. I think the real power in having a cloud provider do this stuff is just the operational expertise.

You don't have to have a Kubernetes expert in every company. There's a lot of ways that distributed systems fail in really interesting ways, and we're stacking up all the operational knowledge into every individual company. I think there's a lot of power in letting somebody else worry about that. Well, you focus on your own business domain.

[0:42:57.9] JM: I've done a number of shows about Kubernetes. One of the things that was interesting that came out of all those shows that I did is the architecture for multi-cloud and how that's going to look. It seems like in the near future, many companies are going to have Kubernetes clusters on different cloud providers.

You might have a Kubernetes cluster running on Microsoft and you might have also Kubernetes cluster running on Google and you use different cloud specific services in conjunction with those Kubernetes clusters. For example, on Microsoft maybe you're using Cosmos DB and you want to have a Kubernetes cluster for that database to interact with, so you can stand up an API server just right next to that, the Cosmos DB setup that you have.

Then on Google maybe you're using Google's, I don't know pub/sub service together with your Kubernetes instance. Do you do you anticipate these different Kubernetes clusters on different cloud providers and how do you see those different Kubernetes clusters communicating with one another?

[0:43:58.8] ESM: It's actually really interesting, because I do see that happening a lot both in people experimenting some people running multi cloud, because they have a Kubernetes cluster running in their own data center. Then they're leveraging cloud provider for some other service.

Things like Kubernetes, it's becoming a commodity. All the major cloud providers have a Kubernetes offering. We're bound to see more and more companies spin up in the same vein as when slices and VMs became popular right, like lots and lots of people offered those things. I think we're going to continue to see more people offer Kubernetes and it going to become a commodity, and that's a good thing.

Like you said, some of the power that comes in cloud providers is the other services they offer. They've got a lot of machine learning things and cognitive services and databases that you don't have to worry about operating that can run at just massive scales, and their offerings just continue to grow. We're going to get into doing things like quantum computing and things like that. These are all things you want to mess with locally.

I do see that people are probably going to do that. A lot of the things that differentiate cloud providers are those additional offerings. I think as all of those offerings continue to build out, people will end up having clusters in each.

I would think you'd probably try to isolate these things in the same way that you do micro-services, where they each have their own role and there's light communication between them. It's possible people end up using Federation. Federation is still going, kind of ongoing and undergoing some work and I'm looking at the way that would work.

[0:45:39.5] JM: Federation, meaning what?

[0:45:41.4] ESM: Kubernetes Federation, now they call it multi-cluster. Essentially, the idea is to have a cluster of clusters. You could talk to the Federation API and say I need four of these things running to in each cluster and it would go out and create the resources in the individual clusters. You wouldn't have to communicate with them, and it would ensure that that happens. It knows how many are running in each cluster, which one it needs to create another one in and things like that.

I mean, there's a group of really, really smart people working on that and there's definitely been some headway there and I think they're looking at what the next steps are. That's also a possibility, but there's also people who think that that's not the way to solve that problem. That's that's the great thing about new technology, right, is that we can all experiment with different ways until we figure out what the best ways of doing things are.

[0:46:33.0] JM: It's for sure. Now to call back to the start of our conversation, your time spent working on Virtual Kubelet, did that articulate even further some of the reasons why Go is useful as the systems programming language?

[0:46:49.1] ESM: I think for me, not so much, just because I've been deeply ingrained in the GO world for so long and I've created so many things surrounding Kubernetes and distributed systems in Go. One thing that really did come out of that was the importance of open source and just how amazing that can be. After that, Microsoft actually spun up a group of cloud developer advocates who solely focus on creating open source projects and contributing to them. That's what myself and Brian Ketelsen and Jessie Frazelle do now.

[0:47:21.0] JM: Go actually has a large community. What do you see is your role in the community?

[0:47:27.4] ESM: I like to think I'm a decent Go developer. I try to contribute to as many projects as you can, but a lot of my focus is building a community. We do our podcast, which is Go Time. Brian Ketelsen and I organized GopherCon. We founded that, the first Go conference back in – well, we started planning in 2013, but 2014 was the first year. That was by accident. We kept saying for years like, “I wish there was a Go conference.”

Then somebody on Twitter was like, “You should just do that.” We were like, “Well, how hard could it be, right?” Now we're over 1,500 people. We’ll probably cross 1,800 this year. That was something I never thought that I would get into.

[0:48:10.8] JM: Do you run it? Is it like a business? Do you run it as a business?

[0:48:14.3] ESM: I mean, we have our day jobs. We don't run it as like a for-profit thing. I t's really been a hobby project for us. One of the great things about working for Microsoft is they allow us to use our company time on these things, like going through all the CFP proposals and things like that.

Yeah, it started just because we really wanted the Go conference. We thought it would be awesome. Ignorance is bliss. We’re like, “How hard could it be?” First year was 750 people. Yeah. As that happened, I really, really fell in love with the community aspect of things and bringing people together and mentorship and things like that.

[0:48:55.1] JM: We do meetups for Software Engineering Daily. The meetups are always pretty fun. You have great conversations in person and we spend so much time holed up in our offices, or in our apartments that it makes the in-person interaction all that much sweeter. I was thinking for a little bit about doing a conference, but then I learned a little bit about the economics of it, especially if you want to do it in San Francisco. It can be murderous. There are a lot of stories of people trying to do tech conferences and just losing their shirts on it.

[0:49:29.7] ESM: Yeah, there's a lot of things that I wish I never learned, like hotel attrition. What that essentially is when you block a bunch of rooms for your attendees, and especially if you're in a major city, you have to do that or there's the risk that they can't get hotels and then they don't buy tickets. You have to pay for rooms that are not booked.

Yeah, Brian and I almost lost our houses the first year, because most people didn't book their hotel until the very last minute. We were sweating bullets over a $100,000 in hotel rooms and nobody's booking these things, what are we going to do? Now we've grown more comfortable with that, because we realize the times of year and how close to the event that we should be looking at. We know that people will buy their tickets early, but booked their hotels late.

It's still a scary thing and it's a lot of risk, because when you get to GopherCon skill now, I think we're sitting on close to $500,000 in hotel rooms right now. Wi-Fi, like three days I think last year was $40,000 for Wi-Fi.

[0:50:38.9] JM: I have heard about this. They charge you so much for Wi-Fi.

[0:50:43.6] ESM: It's insanity. One of the things that I do do, I guess I'm not as public about as I should be is that Brian and I actually talk with a lot of people, especially people who are organizing the other GopherCons around the world. I'm happy to answer questions for people who are interested in starting conferences and things like that, because I think all technologies need it, and somebody has to Go out there and take the risk and be the first one.

I can give some advice on things that – some stumbling blocks and things you'll learn along the way. One of the best things I can say is stay small at first. When you get to convention center size, everything changes. You have to worry about union labor, you have to have insurance policies, you have to have paramedics, you have to worry about fire marshals, all kinds of stuff. If you stay small enough where you can operate inside of a hotel, if you're spending a lot of money on food and beverage and things like that inside the hotel, a lot of times they're forgiving on the hotel attrition.

A lot of times if you have a hotel room block and food and beverage budgets that are large enough so you can get them to give you the space for free. Wi-Fi is cheaper. A lot of times there's a meeting planner that you get connected with inside the hotel that will help you with all these things. Yeah, if you can stay small, two to four hundred people and inside of a hotel, you can make it a lot easier on yourself.

[0:52:03.7] JM: All right, well I'm going to stick to podcasting for a while. In what ways is Microsoft getting involved in that Go community?

[0:52:10.7] ESM: Yes. I mean, they've hired a lot of people. There's been a lot of press on the scooping up of Go people; myself, Brian Ketelsen, Jessie Frazelle, Ashley McNamara, [inaudible 0:52:22.1], and there's some people on the AKS team, like Aaron Schlesinger that does the Go in Five Minutes Podcast.

Internally, the AKS teams and things like that all use Go. Go is a first-class citizen in Azure, so we have a very, very good as your SDK, which has been rewritten to be a lot more idiomatic. In addition to that, a lot of us they allow us to and push us to go out and speak and teach people in the GO community to do our podcasts, to do our conference, to contribute to open source projects, things like Virtual Kubelet. They're really, really getting deeply ingrained in the GO community.

[0:53:02.7] JM: You are part of Go Time as you mentioned, and that's a podcast about Go. It's part of the changelog network. I am a big fan of changelog, a big fan of Adam. What's your experience been podcasting? What have you learned in that process?

[0:53:18.4] ESM: Yeah. It's really interesting and we could probably have a whole episode on impostor syndrome. For as much as I've done in the GO community and all of these things, I spent a lot of years hidden. I've never really liked being in the public eye. I feel like anything that I've done, anybody could have done really.

The podcast came around. We started talking about doing one and I was like, I really should start blogging or speaking and as we were talking before the show I DJ’d when I was younger, so the idea of putting myself out there it sounded a lot easier to stick a microphone in front of my face, because I was used to doing that and you do the persona.

We started doing it and I don't think we really anticipated for it to be as successful as it has been. It's really awesome getting some of the guests we get on the show and the topics that end up coming up. Sometimes even the accidental topics that come up are just really amazing. You find it hard to stay on topic. You're like, “Wow, this is really, really great.” Just re-themed episode.

[0:54:25.0] JM: I hear you. I mean, that's why I started this and I – just like you encourage people with the conference, I encourage people to start podcasting, partly out of self-interest because I want more podcasts to listen to, but probably because it's just an excuse to call up very interesting people.

Actually a close friend of mine recently was telling me like, talking to you is sometimes really annoying, because you just ask questions like you're on a podcast all the time and it made me – maybe have to check myself and be like, “Oh, actually. Okay, I guess I should recalibrate myself to the pedestrian conversation sometimes,” because you can't just have every conversation be like a podcast. Not every conversation is super technical and podcasty.

[0:55:06.5] ESM: Yeah, and ours we've tried to keep – one of the things we wanted to do from the beginning was not be a 20-20 question and answer, so we really, really tried to set it up where it's really we're having – they're really a co-host of the show, where it's just us sitting around at a table or something, just talking about Go and the listeners get to be a bird on the wall, a fly on the wall.

That makes it a little easier to not feel like you're interrogating somebody. Sometimes they do come out that way. One of the most interesting things that have come out of podcasting for me is that you don't really realize that you have a recognizable voice until you meet people at conferences that recognise my voice.

[0:55:50.2] JM: Right. They're like, “I thought you were just a disembodied voice.”

[0:55:53.3] ESM: It's just it's really interesting and cool. They're like, "Go time?" You're like, "Yeah." They're like, "I heard your voice." For me, I have a twin brother, so I'm used to hearing the sound of my own voice.

[0:56:03.0] JM: All right. Well, Erik it's been great talking to you. A really hood conversation and I look forward to seeing what else Microsoft is doing in the GO world and seeing what you do in it as well.

[0:56:12.8] ESM: Yeah. Thanks so much for having me on. This is fun.

[END OF INTERVIEW]

[0:56:18.9] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plugins. Use the value stream map to visualize your end-to-end workflow. If you use Kubernetes, GoCD is a natural fit to add continuous deliver to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on the fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team, who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations.

You can check it out for yourself at gocd.org/sedaily. Thank you so much to ThoughtWorks for being a long-time sponsor of Software Engineering Daily. We're proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]