EPISODE 831

[INTRODUCTION]

[0:00:00.3] JM: Modern server infrastructure usually runs in a virtualized environment. Virtual servers can exist inside of a container, or inside of a virtual machine. Containers can also run on virtual machines. Kubernetes has allowed developers to manage their multiple containers, whether those containers are running in VMs, or on bare metal, which means servers without VMs.

As organizations expand their Kubernetes deployments, the overhead of those deployments is becoming a relevant concern, so-called Kubesprawl can occur within organizations due to a lack of best practices on when new clusters should be spun up or spun down, and when those clusters should be shared by teams, or shared by services.

Paul Czarkowski is a principal technologist with Pivotal. He joins the show to discuss virtualization, Kubernetes and the state of the cloud need of ecosystem, as well as how Pivotal is thinking about all of these issues.

[SPONSOR MESSAGE]

[0:01:12.8] 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. 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.

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. As a bonus to our listeners, you will get a $100 in credit to use over 60 days. That's a lot of money to experiment with.

You can make a $100 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, and of course computation. Get your free $100 credit at do.co/sedaily. Thanks to DigitalOcean for being a sponsor.

The co-founder 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. Thank you, DigitalOcean.

[INTERVIEW]

[0:03:20.1] JM: Paul Czarkowski, you are a principal technologist at Pivotal. Welcome to Software Engineering Daily.

[0:03:24.4] PC: Thanks so much for having me.

[0:03:25.7] JM: I'd like to talk to you today about Kubernetes and containers and virtualization, as well as some of the blog posts that you've written. Let's start with just virtualization broadly speaking. Describe the evolution of virtualization technology as you have experienced it in your career.

[0:03:43.3] PC: Sure. I remember the first time I came across virtualization was VMware. I was working at an ISP back in Australia in would've been like the late 90s, I believe. We saw this thing called VMware. Like most people at the time, we saw it as a way to do disposable tests and development infrastructure. We ran mail servers and web servers and stuff. We would run, I guess test versions of those in VMware. Just so you know, not needed so many servers. That worked really well.

Then at some point, we started to see it push into more production use cases. When VMware started bringing out the multi-hypervisor management tooling and they brought out product tooling to increase the resiliency, by doing migrations from one host to another and stuff like that. We started to see ways that not only could it increase our ability to run more workloads per server, but also to add resiliency to what was at the time, fairly fragile applications.

We all started running it in production using VMware, obviously stuff like KVM and other tooling. VirtualBox came out. VirtualBox was great, because it helped us do the same thing on our desktops, so we could run multiple operating systems, which for people like myself who have insisted on running Linux on the desktop for a long time was great, because we could have somewhere to run a Windows VM, where we could run Outlook and our other Windows-specific tooling.

Then we had this cloud thing showed up and it taught us that actually maybe the best place for the resilience is actually in our applications and the way we deploy our applications, versus the infrastructure itself. Because as you push the resiliency into your application, you actually inherently make it more suitable for changing the scale of it, so quickly deploying more, dealing with failures and stuff like that.

That brought us to the cloud thing. Then we started to get cloud native. We started to say okay, now we have this cloud thing. Let's actually write our applications to take best advantage of being deployed to clouds. Then of course, we got – we then started looking at containers. Obviously, the purists would tell you containers have existed for longer than virtual machines on Solaris and BSD and stuff like that. Really, until Docker came along, they weren't something that the average person could use.

[0:06:22.1] JM: Indeed. You've given a great overview for reasons we might want to virtualize our technology in various ways. We can get security benefits, isolation benefits, continuous delivery benefits. If we're talking about virtualization, there is a distinction that you have drawn in some of your blog posts, between soft tenancy and hard tenancy. Could you describe in more detail what those terms mean?

[0:06:51.6] PC: Yeah. I guess, first of all, tenancy is basically when more than one user, whether that's people from other companies, or even people from inside the same company want to use the same infrastructure, or the same platform. Then the soft and hard tenancy aspect comes in based on how the resources are actually divided up amongst those users.

With hard tenancy, you usually have some almost physical boundary. Obviously, physical machines with their own VLANs are a very hard tenancy. They have their own network. They have their own physical machine. With VMs, you emulate that same hard tenancy in virtual machines, so you get most of the same separation. Then as you slip into things like containers, you no longer have that physical barrier.

Containers are really – especially Linux containers are just separations of workloads on the same kernel. There's always some form of opportunity to get in through the kernel into another container. It's less obvious how you can restrict a given container to a certain set of resources on the physical machine. There's some security concerns and there's also some performance and your noisy neighbor type concerns that have been somewhat solved in VMs.

Now the question is what tenancy do I need? Then when I figure out what tenancy I need, then what's the best way to achieve that tenancy, given the fact that I want to use containers, or given the fact that I want to use VMs? You could work your way back from there.

[0:08:34.7] JM: Just to emphasize the differences between containers and VMs, could you distinguish between their tenancy models, explain to what degree is the tenancy of containers soft or hard and the same question for VMs?

[0:08:52.2] PC: From VM, you do have that physical separation. With the containers, Linux containers by themselves don't really have a good understanding of multiple users. Yes, there is some isolation using C groups and namespaces and stuff like that, but they're really more about separating their runtime environment from each other, versus creating a secure environment in which two workloads could run side-by-side in a highly secure way.

Then when you look up the stack, say using Kubernetes, introduces some extra tenancy options, like namespaces. A lot of the actual underlying Kubernetes infrastructure tooling, like

the API server, the Kubelet, etc., don't really understand that tenancy. The Kubelet, if it's given instructions will just attempt to do what it's told, versus actually validating that. The person has the rights and access to do those things.

Above the API, Kubernetes has enacted some of that tenancy, so you do have role-based authentication, etc. In the actual Kubernetes components themselves, that doesn't always reach all the way down. You see some weird things that you might not expect. For example, you can basically see every DNS. DNS is a great example. When you create a pod and a service on Kubernetes, Kubernetes goes ahead and creates DNS for it.

You can then learn about what's installed by looping over the DNS and seeing what's available, and you could even potentially poison the DNS and add your own pod to a given load balance, or to a DNS A Record and have – but potentially cause some traffic to be redirected to the wrong pod, either accidental or nefarious purposes.

[0:10:56.0] JM: I'd like to now survey some aspects of the virtualization world, the Kubernetes world, the cloud world, and look at this issue of tenancy and different virtualization models from a variety of perspectives. Because to my mind, that's a good way of presenting this information in a way that's a little more accessible, or will perhaps convince people, this is why you should care about the tenancy models, or at least this is a survey of how the tenancy models might impact your day-to-day work, as a developer who is probably not building a Kubernetes managed service. Probably if you're working with Kubernetes, you're implementing it at your company.

Let's actually start there. You have a quote, “If you are trying to develop a Kubernetes strategy, you have already failed.” Now this is highly disjoint from some of the marketing messages I have encountered. What's wrong with me trying to develop a Kubernetes strategy?

[0:12:03.6] PC: Yeah. I mean, I will admit there is definitely a little bit of clickbaitiness to that quote. Really, what I was trying to get at is Kubernetes is not a solution to a problem. Kubernetes is a platform. Really when you're developing strategies at the high-level in a business, you need to be focused on the problems that you have and solving those problems.

Kubernetes maybe an implementation detail of that strategy, but it shouldn't be the strategy itself. You think back a couple of years ago, there was a fad of having an OpenStack strategy and a couple of large enterprises and large vendors actually had press releases about their billion-dollar OpenStack strategy. If you look back at those, you'll see that most of those companies either got very quiet about what they're doing with OpenStack, or have publicly said that the OpenStack wasn't what they wanted. Didn't solve the problem that which I solve. That was because they're making these huge bets on this relatively unknown thing, that they didn't have a great understanding of. That's treating technology as an enterprise strategy.

Whereas, if you look at it as an implementation detail, you usually focus on a much smaller problem. You might be just focused on a particular business unit, or a particular set of applications, and you're making a much smaller bet and you give yourself a little bit more leniency to fail and learn from those failures and re-implement and work your way through it and come up with something that is actually solving whatever problem you had, versus that well, we're just going to install a giant Kubernetes cluster and magically, all our problems will go away.

[0:13:56.9] JM: I love this point so much, because it's really easy to get caught up in look, I need to migrate to Kubernetes, we just got to get this done, we got to move everything to Kubernetes, we got to move the legacy monolith into a microservices architecture, etc. That may be right, depending on the business priorities. Ultimately, you as an engineer need to be aware of the business priorities. If something is going on in the business that necessitates perhaps building out a new service, you may have to wait for your elaborate Kubernetes migration strategy. That's just the reality.

By the way, in the meantime, while you're building whatever new feature or tacking additional software onto your existing monolith, maybe “serverless technology” advances to a degree where, now you would be developing a “serverless strategy” if this was your mentality. If the Kubernetes opportunity passes you by, maybe now you're spinning up something with serverless functions.

[0:15:05.9] PC: Yeah, absolutely. I think we're seeing that already with folks that went from having an OpenStack strategy, never actually got a thing, so then they switched to having a

Kubernetes strategy. They'll probably never have a thing before they're like, “Oh, now we need a serverless strategy.” I think that's definitely a concern.

What we should be looking at is figuring out how we can get access to these platforms and these different abstraction layers, without needing to make a large bet on them, and without needing to have a multi-year process on figuring out how to operate it, or how to figuring it every single best practice, before you start utilizing it. We need a way that we can start experimenting as developers, as operators and as cohesive business units. We need to be able to start experimenting with these different abstraction layers and figuring out which ones are best for the types of workloads that we currently have, and also the types of workloads and applications that we want to build in the near future.

[0:16:07.7] JM: As we experiment, we're going to implement certain patterns, perhaps incorrectly. We may only realize that they are incorrect in retrospect. We may end up spending up just one big Kubernetes cluster, and then later on realize that actually, we should have spun up a bunch of smaller Kubernetes clusters. We may spin up a bunch of small Kubernetes clusters and realize, perhaps we only should have had two or three.

[0:16:33.4] PC: Yeah, absolutely.

[0:16:35.0] JM: I think you articulate this with the term ‘Kube-sprawl.’ What is Kube-sprawl?

[0:16:41.3] PC: Right. Kube-sprawl is generally, when you end up with a bunch of containers running in Kubernetes and you don't really know what it all is. You're not really tracking it all. We have the same with VM sprawl for virtual machines, right? Especially when we were – I saw quite a few orgs just doing lots of unmanaged, say KVM servers and they would just spin up random VMs across random hypervisors. It was like, you no longer knew what I was running where and how it was running.

It's really important to have a reasonable, cohesive plan on how you’re going to move forward, obviously at early stages, experimenting and learning is super important. Don't block yourselves in that but. As you're experimenting and learning, say with Kubernetes, you should also have

folks that are focused on what's our long-term management of this going to look like, based on the things that we currently know?

As you hinted at, you're going to get it wrong and you're going to get it wrong three, four, five times, before you figure out the best way for you to use it in your own org. Definitely, some organizations are going to function better with one or two large clusters. Others are going to function better with 10, 20, even a 100 clusters based on how that organization is structured and how it works. I feel like, you're familiar with Conway's law, right?

[0:18:10.7] JM: Is that the thing where an organization represents its software architecture, or something like that?

[0:18:16.7] PC: Yeah. Basically, your software will represent the communication structures inside of your organization. I think your infrastructure fits that same thing. If you have lots of teams building up microservices, doing a whole product-focus development, you may be better off with lots of clusters. Whereas, if you have a large command and control, like developers write code and then operators run that code, instead of commanding control type culture, then it may make sense for you to have just one or two large Kubernetes clusters, right?

It's important to acknowledge the type of organization you're working in. Then maybe some aspirational, like this is the culture and this is the company we want to be. Then figure out the best way to meet those needs with the infrastructure that you're going to get.

[SPONSOR MESSAGE]

[0:19:19.8] JM: To get ahead today, it helps if your organization is cloud native. The 2019 Velocity Program in San Jose, June 10th through 13th, we'll cover everything from Kubernetes and site reliability engineering, to observability and performance, to give you a comprehensive understanding of applications and services, and stay on top of the rapidly changing cloud landscape.

I attended Velocity conferences early on in Software Engineering Daily and it helped me grasped the times and the fast-pace of change in today's times. There are twin revolutions in

open source software and cloud. To understand the nuances of these twin revolutions, going to conferences can really help.

At the Velocity Program in San Jose, you can learn new skills, you can learn technologies for building and managing large-scale cloud native systems, you can connect with peers to share insights and ideas. Join a community focused on sharing new strategies to manage secure and scale the fast and reliable systems that your organization depends on. You can get expert insights and essential training on critical topics, like chaos engineering, cloud native systems, emerging technologies, serverless and much more.

Listeners to Software Engineering Daily can get 20% off most passes to Velocity by using code SE20. You can go to velocityconf.com/sedaily and use discount code SE20 on bronze, silver and gold passes. O'Reilly Velocity Conferences are a great way to learn about new technologies. I can speak from personal experience on going to those conferences. You can go too. It's San Jose, June 10th through 13th. Go to velocityconf.com/sedaily and use discount code SE20 to support the show and get 20% off. Thank you to O'Reilly.

[INTERVIEW CONTINUED]

[0:21:39.9] JM: In a Kubernetes cluster, you typically have – at least, I learned this from your blog post, something like four to six virtual machines at least. If you're getting a bunch of Kubernetes cluster spun up, that will lead to 4X to 6X VMs per Kubernetes cluster. Explain the VM count that is needed to run a cluster and whether or not that's problematic.

[0:22:09.4] PC: Yeah. I mean, you can run a single machine Kubernetes cluster. For any number of reasons, which are hopefully obvious, that's not something you would want to do in production. You need to have resiliency for Kubernetes itself and also resiliency for your applications running in Kubernetes.

Usually, that means you have say three Kubernetes controllers, or masters. Then you have three or more Kubernetes workers, which is your applications are going to run, right? Because you need your Kubernetes control plane to be able to deal with some failure in the system and

you need your applications to also be able to deal with some failures in the system. Generally speaking, you will have six as a minimum; three four masters, three four workers.

There is some argument, which is if you just have one master and you're able to repair and replace that master very quickly, then maybe it's not as important to have three masters. Because if you have three masters, obviously you need to figure out and learn how to cluster etcd and have the data store recover from failure and expand to new systems. Or as we just have the one, if you've got really good backups and restore methodologies. If you can restore your master in 10 minutes after failure, then maybe that's okay, because all of your workloads will continue to run, even if your master nodes are down.

There's some argument over which is the right number for masters. Effectively, six is the minimum; three masters, three workers. Then you're probably not going to run enough workloads to make that worthwhile. I think the golden point, the point where you start to see the value is say around 10. You might have the three masters and a six, or seven a worker nodes. That gives you plenty of resources, assuming you're using large enough VMs to run a significant number of containers, but also have enough extra resources on those VMs to deal with failure. Because when a node fails, Kubernetes will reschedule Kubernetes workloads on to other nodes, if they have the spare resources to handle them.

I think around 10 is probably that good number. Then that to me, 10, also that's usually enough to handle a business units applications. If they've got a bunch of microservices, say six or seven microservices, each running a couple of replicas with maybe a database running in your cloud infrastructure somewhere, then I think that gives you plenty of resources for your applications to be scheduled, for them to be rescheduled as failures, for you to be able to upgrade Kubernetes from underneath them without having to get a bunch more VM, so that you have enough to migrate them around. Yeah. In my experience, around 10 VMs tends to be the sweet spot. If you're going for the idea of having multiple clusters, versus one or two big cluster.

[0:25:21.9] JM: Is this problematic at all? Is this too much resources that we have to devote to a cluster?

[0:25:27.9] PC: As long as you have pretty good utilization on those clusters, I don't think that's bad. I mean, yes, you do have the almost lost resources on your masters, because you can't really schedule work to them. That's possible. That's generally made up for by being able to have a little bit more systems – more containers running on the same VM, or more applications running on the same VM. Also, the value you get out of the orchestration and value recovery generally makes it worth that overhead.

[0:25:59.6] JM: Why is it important for us to consider more resource-efficient ways that we could run Kubernetes workloads?

[0:26:08.2] PC: Every enterprises is interested in saving costs. That's a big one. There's also obviously the environmental benefits of having less servers running to run more infrastructure. I would say, less enterprises are as interested in that as just the sheer savings in having less compute resources are running to handle the workloads that you currently have. There's also some argument over the number of employees you need to be working on a particular thing. Often, you used to talk about how many sysadmins you would need per server, or whatever. As enterprise, you still use that metric, then having more applications deployed to each server, or each pieces of infrastructure, again saving costs, etc.

[0:27:06.6] JM: Describe some of the ways that you see promise in potentially reducing the footprint of these Kubernetes sprawling workloads.

[0:27:19.9] PC: I think there's a lot of benefit to really looking at what you're trying to achieve, what you're trying to run, and building out Kubernetes clusters that are very specific to that. When you have smaller clusters and more of it, you can make them, okay, the types of applications we're running are very CPU-intensive, but not very memory-intensive. Or maybe they need GPUs.

You can build out clusters with specific types of infrastructure and resources available to them for those applications. Without saying, to make GPUs available to every single worker node across a giant cluster, even if you only have a small number of applications requiring that infrastructure.

[0:28:09.0] JM: From the point of view of Pivotal, because Pivotal builds a managed Kubernetes service, if you could cut down on the resource consumption of a given Kubernetes cluster, that would lead to dramatic economic improvements for how much it would cost to run these workloads across your infrastructure. It seems this would be particularly relevant for a managed Kubernetes service.

[0:28:36.7] PC: Yeah. What Pivotal does is we actually do a managed Kubernetes service, but on your own infrastructure. We don't provide infrastructure. We work either on top of VMware in your data center, or on top of any of the public clouds. Most of the any cost savings would be directly realized by the customer using our product, Pivotal container service, not by Pivotal ourselves.

[0:29:03.8] JM: Can we talk more about the managed Kubernetes service world? Because I think, there's a little bit of variety among how these different managed Kubernetes services have been implemented across different providers. What are the subjective decisions that these different Kubernetes services are making?

[0:29:22.2] PC: Yeah. I think when you look at it from a higher level, most of the managed Kubernetes services are fairly equivalent. Obviously, some are more mature than others. You could argue that Google's has a higher maturity than say, Azure's or Amazon's, purely because they've been running up for longer. Also, they invented the thing.

When you dive deeper, you can see different cost structures and different types of resiliency. For example on Google, they give you the masters for free. You're only paying for the instances in which your applications are actually running. Whereas on Amazon, you pay for the masters themselves as well. There's definitely some different financial implications.

Then say on Google, they're starting to support Istio out of the box, so you can do a check box and have extra systems installed into your cluster to help you do service mix and stuff like that. There's definitely some differences there. Then upgrade methodologies, they all upgrade in different ways. Obviously, I think your first thing is are you currently using a cloud? Is that cloud's Kubernetes offering good enough for you? Then determine from there, whether you need to look at the others based on the different types of features they support, how they like to

do upgrades, and if they have easy ways to tie them back to your own infrastructure, in your own data center, if you try to do multi-cloud and that thing.

[0:31:00.0] JM: Do you see companies running multiple Kubernetes managed services?

[0:31:05.8] PC: Yeah. We definitely see that. Obviously, if they want to have workloads in different clouds, we see that. Then with our own product, we see a combination of folks will use PKS in their own data center, and then they can choose if they want to use PKS on their cloud of choice, say Google. Or if they want to use GKE and use Google's built in Kubernetes clustering.

There's a number of decisions to be made there. You could argue that it's going to be cheaper infrastructure-wise to run GKE on Google, than it is to run Pivotal's Kubernetes product on Google. You then don't necessarily have the exact same interface into your clusters. I mean, sure, Kubernetes is Kubernetes, but there are definitely some leaky abstractions, and the different platforms are going to have different versions of Kubernetes available to you. It may be more important for your business to make sure that you have the exact same operational tooling, the exact same software managing the cluster, whether it's on your an infrastructure, or in the cloud. Or you may be more interested in the costs of it. We see both, depending on the type of customer it is. That goes across the different clouds, as well as not just GCP and your own infrastructure.

[0:32:27.9] JM: Revisiting the subject of virtualization, I want to go a little bit deeper on some ways that the footprint of these different Kubernetes clusters that are going to be running within an organization can become more efficient. As we've discussed, the efficiency gains could be realized at the level of a company that's operating their own Kubernetes cluster, or a company that's adopting a managed Kubernetes service, if the Kubernetes service takes on a different virtualization model that is more resource-efficient.

In any case, if we've got containers running on VMs and those VMs are running on some host operating system, the management system as I understand it is you have virtual machines that are managed on by a hypervisor. It's running on this host operating system. Then we're often

running multiple containers on each of those VMs. Is that correct? Is that the correct model I should have in my head?

[0:33:29.9] PC: Yeah, that's right.

[0:33:31.2] JM: Okay. Is there a hypervisor system for managing the containers in a given VM?

[0:33:40.2] PC: Well, really Kubernetes is that hypervisor. Kubernetes runs a tool called Kubelet on every single worker node. That worker node basically is responsible for scheduling your different containers on your different workers. Yeah, in a way, Kubernetes is acting like the hypervisor.

One thing that's starting to show up is we're starting to see these micro-VM style hypervisors, like Kata Containers, Firecracker, etc., where you actually have VMs, but they have the performance of containers. We start to see one container per VM. We're going back to you have a hypervisor that's running a bunch of VMs, that one-to-one to containers. When you do that, you suddenly start inheriting the security posture and the tenancy models of VMs versus containers.

That then means that you could potentially start running multiple tenants on the same hypervisor, because you get those tenancy constructs, versus if you're relying on Kubernetes and containers to provide that tenancy. You wouldn't necessarily allow those two users to run inside the same kernel. I think we're going to start to see in the public clouds, I believe we'll start to see some benefits around them, having not needing to have worker nodes that are specifically assigned to a particular user. Instead, every single time a user spins up a container, they'll get that micro VM and it'll be running on a shared hypervisor.

They get most of the benefits of a container, but then also most of the benefits of a VM. I think it'll be a while before we see that move down to people running their own Kubernetes clusters, because it's going to be a lot harder. Also, I think the efficiency savings won't be as pronounced when you're talking about tens or hundreds of containers, as opposed to when you think about all the containers running across all of Google's Kubernetes engine. There's hundreds of

thousands, I would imagine. At that scale, you're going to see a fairly substantial decrease in physical infrastructure required to handle that workloads.

[0:36:09.7] JM: I have heard that at Google, they run containers directly on the hosts running on Borg. They don't use virtual machines. Why are we even bothering with this virtual machine layer? Why not just run a Kubernetes instance on each of our hosts?

[0:36:31.7] PC: Yeah. That is an excellent question. To start with, Google has a lot of strength and skills and running and managing large fleets of physical machines, to the point where they have a lot of custom hardware. Their automation tooling is incredibly strong for dealing with physical machines. The rest of us don't necessarily have that level of automation out of the box with physical machines, but it's fairly easy for us to get that automation, using some virtual machine type system. That is your first benefit.

Then your second benefit is around – if you want to have multiple clusters, you can run multiple clusters on a smaller number of physical machines if they're in VMs, because you get that tenancy model that makes possible to run three or four different users, VMs on the same physical machine where you might not want to have three or four different users, containers, running on that same physical machine. I think there's a fairly nuanced set of answers to that and that's just a couple of things.

[0:37:48.7] JM: Does this necessity, or this adoption, this continued support of VMs in the broader industry, does it speak to maybe just wearing an additional seatbelt if you're not Google? Because it provides this additional security benefit, this additional isolation benefit. Yeah, Google knows how to run containers directly on the hosts, but that's just a little harder to do while maintaining security and proper isolation. Would you attribute it to that at all?

[0:38:22.8] PC: Yeah, absolutely. If you look at how Google runs their customer workloads, right? In GCP where you're getting instances from Google, you are getting your own VM. Now they do some weird, like VM container trickery, but at the end of the day, you are getting your own fully isolated VM if you get up a GCP instance. Then if you get a GKE, Google Kubernetes Engine, a Kubernetes cluster, you are getting VM instances that are dedicated for your Kubernetes cluster.

While they run a lot of their own infrastructure across one big, flat set of containers using Borg, they don't necessarily expect their customers to do that same thing on their infrastructure. I think that's a pretty good indication of if it's just me, I can use all the machines and containers I want. If it's me and three or four of my closest friends, or perhaps not closest friends, perhaps we shouldn't just share that same set of infrastructure.

[SPONSOR MESSAGE]

[0:39:35.3] JM: Today's episode is sponsored by Datadog, a cloud scale a monitoring service that provides comprehensive visibility into cloud, hybrid and multi-cloud environments with over 250 integrations. Datadog unifies your metrics, your logs and your distributed request traces in one platform, so that you can investigate and troubleshoot issues across every layer of your stack.

Use Datadog's rich customizable dashboards and algorithmic alerts to ensure redundancy across multi-cloud deployments, and monitor cloud migrations in real time. Start a free trial today and Datadog will send you a t-shirt. You can visit softwareengineeringdaily.com/datadog for more details. That's softwareengineeringdaily.com/datadog. You will get a free t-shirt for trying out Datadog. Thanks to Datadog for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:40:39.3] JM: We've done a show about gVisor that will air by the time of this episode. gVisor is a tool for providing abstraction away from the underlying Linux kernel. In that show, I learned a little bit about the interface between containers and the kernel space. The fact that when you're accessing kernel space, you make a syscall. You go from the user space, you make a syscall into the kernel space, so that the underlying kernel infrastructure can do something for you that's useful, that has to do with some lower-level resources.

One problem is that there's 300 something syscalls and most of them are unnecessary, perhaps for a lot of the workloads that we have just – and if we're talking about cloud infrastructure. Even though they may not be that useful to us, they do present a security surface area. With gVisor,

you have this layer that I guess negates a lot of those syscalls, or just makes them into null operations, so that you have a lower set of security concerns.

I don't know how closely you've looked at gVisor, but does it seem like, perhaps a solution that might eventually get us to a place where the more general enterprise could run just on – the containers on hosts?

[0:42:00.2] PC: Yeah, potentially. I don't know a ton about gVisor, but my understanding is that they effectively provide a syscall translator, that collects your syscalls and then actions them through a separate system, versus you talking directly to the kernel fiber syscalls. I would imagine, there is some overhead there. I wouldn't think it would be any more, or less overhead than say using something like Kata Containers, or one of the other micro VMs.

I think, the general use case for that thing will probably be more at the micro VM level, because it's a much easier thing for people to know, run and understand, because it does follow that same VMs running in hypervisor model that people are used to. I think maybe at Google scale with Google level sophistication, gVisor might be the right solution there. I think for most people, something like Kata Containers, or one of the other micro VM platforms is probably going to be more where we look at for the general enterprise and general users.

[0:43:10.3] JM: When we're talking about these lighter VMs, like the Kata Containers, what are we subtracting from the virtual machine that makes it take up less space?

[0:43:21.0] PC: Yeah. That is a good question. I can probably answer some of it, but I don't think I have all the knowledge to give you a really good answer.

[0:43:28.0] JM: Okay. Okay, no problem.

[0:43:29.1] PC: I think for some things, like in a traditional VM, you're emulating floppy disks and you're emulating USB and you're emulating CD-ROM drives and all these things that when you want to just run one single application you don't need. I think in a way, the micro VMS are just cutting out all of that extra nonsense, which is probably going to give you some performance benefits. It's definitely going to give you some security benefits. If there's a

vulnerability in the floppy disk emulation layer, well if you don't have a floppy disk emulation layer, then the ability to try and hack your way through that, it just goes away.

I think that's where a lot of the performance improvements come from, just the fact that all of these subsystems that you don't actually need just go away, and you don't actually even have access to them.

[0:44:25.7] JM: I'd like to zoom out a little bit, because I feel we've talked in some detail about the different future alternatives for running Kubernetes workloads and container workloads. I'd like to change the topic just to more generally, what is going on in the Kubernetes/cloud world. I'd like to start with service mesh. You've already given the perspective on the Kubernetes strategy thing, perhaps being a mistake. Is it a mistake to try to develop a service mesh strategy?

[0:45:03.6] PC: Yeah. I think it's in the same category of mistakes. Not everyone has a service mesh problem that needs to be solved. If you're just learning how to use Kubernetes, adding in all this extra stuff you need to learn as on top of it, isn't necessarily the way to go. I always recommend that people just start with fairly basic Kubernetes and figure out how to operationalize it and how to run applications on it.

Then as they start to hit problems that are starting to look a little bit like problems of a service mesh might solve, then start looking down the service mesh way of doing things. I also recommend that folks look at what languages they're using and see if the value they're getting out of a service mesh is available directly in the language itself.

For example, Spring which is a java thing that Pivotal works on, has a lot of things that do service meshy things. They handle circuit breaking, they do service discovery, they do config servers, they do gateways, all directly in the native spring interfaces. You don't need some external service to expose those to you, which if you can do something directly in your language, versus throwing that control app to something else, it's generally going to be more resilient and more suitable for your needs.

I think it's one of those things where everyone involved in running a software from the applications, to the operators, to the architects, really need to understand the – like back to what you said earlier, you need to understand the business and the business goals and the full end-to-end value stream of delivering software.

Once you know that, you can start to make really good, intelligent decisions around what you need out of a platform, what you need out of the languages you're using and how you're going to deploy those things.

[0:47:03.5] JM: If a pivotal customer were to come to you, and I don't know how much interaction directly with customer you have, and said, “Look, I don't care what you're telling me. I want a service mesh and I want it now.” Does Pivotal have a recommended mesh to go with? When you're watching this battle of the different service mesh providers play out, what does that lead you to recommending?

[0:47:29.7] PC: Yeah. I mean, I think the main player in the service mesh is Istio. There are certainly some others. There are definitely some arguments about Istio being very heavy. I've seen some folks talk about the Istio side cars that need to be installed in every pod, costing you maybe 300 meg of memory per container, or per pod. Certainly, there can be some – that can add up if you're running thousands of applications, thousands of pods. There’s definitely some folks talking about more resource-efficient options. I think right now, the only viable option is Istio.

[0:48:10.3] JM: Really?

[0:48:11.8] PC: Yeah. I mean, and again, I'm not a service mesh expert, but certainly all the conversations we have with customers that are interested in service mesh are all Istio-focused. I think that's probably because Istio’s blessed by Kubernetes in a way.

[0:48:29.1] JM: You mean, blessed by Google.

[0:48:30.4] PC: Well, yes. That's a very, very important distinction. It is blessed by Google. If Google does it, it must be good for all of us, right?

[0:48:38.0] JM: See, this is something that irritates me a little bit. I mean, maybe I shouldn't be irritated. Irritating is the wrong word, but it's this self-fulfilling prophecy thing, where okay, yeah, Istio is the way to go, because Google's going to put all the resources into it and they're going to tilt the direction of the CNCF and the direction of the Kubernetes community. Cool, all right, we'll have Istio, and it'll be the best service mesh probably, because the community will marshal around it. That's great. That's a self-fulfilling prophecy.

It's weird, right? Because who's using it in production? Not as many people as Linkderd. As far as I can tell from talking to people, I believe Monzo Bank is using Linkerd in production. I don't know. Man. Do you feel that way too? You know what I'm talking about?

[0:49:29.6] PC: Yeah, absolutely. Yeah. Generally, all the cool, new shiny things we're talking about are not the things that people are actually running and using, because there's so many downsides with coming, with running the cool, shiny thing, especially when it's quite heavy. Whereas, if you've got something that tightly focuses on a particular problem you want to solve, well just use that thing. Definitely, I hear what you're saying and I agree.

I think where Istio starts to become interesting is when you look at that next level-up abstraction of Knative, where Knative is a new thing on top of Kubernetes and some of the things it needs to do what it does provided by Isito. If you look at it as the next abstraction layer up the platform stack, where you're not necessarily dealing directly with Istio, Istio is now an implementation detail of your platform, instead of becoming a thing that you have to deal with and run on top of your platform. I think it's a little bit easier to swallow.

[0:50:34.6]JM: Yeah. The other thing that makes it easier for me to swallow is from what I can tell, Envoy is a legitimately good piece of technology. They were like, "Well, this is a really promising sidecar thing and this fits into the open-source vision for the world that people actually want pretty well. Whatever, let's build a service mesh around it." I don't quite remember what the Linkerd sidecar model is, but perhaps maybe it's not a technology that's as flexible as Envoy. I guess, and maybe that's the architectural point I need to dive deeper in before I pass judgment on anyone.

[0:51:13.7] PC: Right. Yeah, I agree. Envoy on its own is brilliant. It makes for an excellent ingress controller, it makes for an excellent proxy system. When you tie it into Istio, Istio is like – Envoy is only one component of Istio, and there's all sorts of other stuff going on to provide circuit breakers and TLS between all of your applications and other stuff. That's where Envoy in itself, brilliant. There's a lot of folks using Envoy right now in production for a number of different use cases, whereas –

[0:51:51.2] JM: Sure. Lyft.

[0:51:53.3] PC: Exactly, Lyft. Whereas, all combined with the rest of Istio, most people don't have all the problems need to be solved with Istio, so why inherit all of the things you need to do to run Istio, just to solve two of the 10 problems that it tries to solve themselves?

[0:52:11.8] JM: Yeah. I mean, it's almost a little bit of a hangover type of situation from the container orchestration, where it's like, all right, container orchestration war has ended. We can all admit, Google just wins every battle. Let's just concede everything to Google right now. Let's all marshal behind Google. Let's just do it, because I want a better infrastructure. I want to stop worrying about this stuff.

[0:52:31.2] PC: Right, absolutely.

[0:52:32.4] JM: I like Google generally. I like Gmail. Okay, one last topic that you're an expert on, because you're at Pivotal. We have seen this – speaking of Google, Google basically highlighted this anthos thing, like the platform where the service integrators connect with big enterprises that need service integrators. I think this is painting a picture of a future where big enterprises are going to need more and more help with their software. How do you think that relationship between a systems integrator, like at Pivotal – I think Pivotal is an excellent systems integrator when it wants to be a systems integrator. It can also build excellent software. Tell me what the future relationship between systems integrators and enterprises is.

[0:53:21.0] PC: Yeah. I think the relationship between them is important. It's going to get even more important. I think what we're seeing is, Pivotal now is more of a platform integrator than we are a systems integrator. We help provide the different platforms that an enterprise might need.

Then we use systems integrators and other folks to then help tie in the rest of it into some easy to plug in marketplace, or whatever you might call it. That the systems integrators writes the software, or helps build the software to run on our platform, and then we turn it into something that's able to be consumable in a marketplace-type scenario, so that a user of that platform can say, “Oh, we need this other thing as a middleware, or as a database layer, or as something else.”

The platform, whether it's Kubernetes, or whether it's Cloud Foundry works with that marketplace to get those systems deployed from the systems integrator, again, whether it's running on some other infrastructure, or whether it's running inside your own Kubernetes cluster, and just provide the integration points of a URL, a username and a password, or an API key or whatever else. You get that everything as a service type way of working.

Whereas an application deployer, the only thing you really need to care about is your own application. You allow the platform and the service broker and the marketplace to provide all of the other pieces you need to make your application run, whether it’s databases, search, message queues, Kafka, any of those kinds of things.

[0:55:02.7] JM: Okay. Well, Paul Czarkowski, thank you for coming on the show. It's been really fun talking to you.

[0:55:06.5] PC: Yeah, thanks. Thanks for having me on.

[END OF INTERVIEW]

[0:55:11.6] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It’s open source, it’s free to use and GoCD has all the features that you need for continuous delivery. You can model your deployment pipelines without installing any plugins. You can use the value stream map to visualize your end-to-end workflow. If you use Kubernetes, GoCD is a natural fit to add continuous delivery to your cloud native project.

With GoCD on Kubernetes, you define your build workflow, you let GoCD provision and scale your infrastructure on the fly and GoCD agents use Kubernetes to scale as needed. Check out

gocd.org/sedaily and learn how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team, and they have talked in such detail about building the product in previous episodes of Software Engineering Daily. ThoughtWorks was very early to the continuous delivery trend and they know about continuous delivery as much as almost anybody in the industry.

It's great to always see continued progress on GoCD with new features, like Kubernetes integrations, so you know that you're investing in a continuous delivery tool that is built for the long-term. You can check it out for yourself at gocd.org/sedaily.

[END]