EPISODE 501

[INTRODUCTION]

[0:00:00.0] JM: In 2011, platform as a service was in its early days, it was around that time that Gabe Monroy started a container platform called Deis with the goal of making an open sourced platform as a service that anyone could deploy to whatever infrastructure they wanted.

Over the last six years, Gabe has had a front row seat to the rise of containers and the variety of container orchestration systems and the changing open source landscape, every container orchestration system consists of a control plane, a data plane and a scheduler. In the last few weeks, we've been exploring these different aspects of Kubernetes in detail.

Last year, Microsoft acquired Deis and Gabe began working on the container services that are related to Kubernetes within Azure. Asher container service, Kubernetes service and container instances. In this episode, Gabe talks about how containerized applications are changing and what developments might come in the next few years.

Kubernetes, functions as a service and container instances are different cloud application run times. With different SLA's, interfaces and economics. Gabe provided some thoughts on how different application types might use these different run times.

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

[SPONSOR MESSAGE]

[0:01:30.2] GM: Your company needs to build a new app but you don't have the spare engineering resources. There are some technical people in your company who have time to build apps but they're not engineers. They don't know JavaScript or IOS or Android and that's where OutSystems comes in. out systems is a platform for building low code apps. As an enterprise grows, it needs more and more apps to support different types of customers and internal employee use cases.

Do you need to build an app for inventory management? Does your bank need a simple, mobile app for mobile banking transactions? Do you need an app for visualizing your customer data? OutSystems has everything that you need to build, release and update your apps without needing an expert engineer.

If you are an engineer, you will be massively productive with OutSystems. Find out how to get started with low code apps today at outsystems.com/sedaily. There are videos showing how to use the out systems development platform and testimonials from enterprises like FICO, Mercedes Benz and Safeway.

I love to see new people exposed to software engineering. That's exactly what OutSystems does. OutSystems enables you to quickly build web and mobile applications, whether you are an engineer or not. Check out how to build low code apps by going to outsystems.com/sedaily.

Thank you to OutSystems for being a new sponsor of Software Engineering Daily and you're building something that's really cool and very much needed in the world.

Thank you OutSystems.

[INTERVIEW]

[0:03:22.2] JM: Gabe Monroy is the lead PM of containers at Microsoft Asher. Gabe, welcome to Software Engineering Daily.

[0:03:27.8] GM: Hey, thanks for having me.

[0:03:29.4] JM: Yes, it's great to have you. You were involved in containers several years ago, you created Deis back in 2011, what was the landscape of platform as a service like in 2011?

[0:03:45.5] GM: Wow, takes a little time to rewind back there but yeah, at the time, Heroku was extremely popular, that was one of the things I definitely remember, you know, I spent a bulk of my career around then in New York City, working with lots of different companies in the financial

services sector and the big bit of feedback I got from everyone was that they wanted that Heroku thing but they wanted it running on their own servers.

You know, what they were saying was they wanted PaaS but they wanted what we were referring to at the time as private PaaS. You know, that was something I started building way back when and containers were really just a nibble detail for how you would build a PaaS system if you're familiar with socials like Cloud Foundry, you'll know that on those solutions, have had containers embedded from them since day one. Heroku as well. And so as I was looking to build this private Heroku that people were asking for – that's what led me to containers early on.

[0:04:44.5] JM: Back then, were companies just uncomfortable with the idea of being in the cloud, why was it that they wanted a platform as a service but running on their own hardware?

[0:04:56.1] GM: Interestingly, most of the people I was talking to, well, probably 50/50 but a significant number of them were interested in the private Heroku actually running on cloud, what they didn't want is they didn't want a multi tenants shared PaaS environment and the reason for that is in my experience, pretty straight forward. Due to the – what I term as sort of the failure of the first generation of PaaS.

Where PaaS worked, it worked brilliantly, flawlessly, right? You hear just code to cloud, you know, get Push and you're stuff's running and everything was beautiful. The problems came in when those abstractions that were – the things that you used to simplify everything and allow you to get to the cloud so easily, over time, those obstructions would start to fail you and you'd realize, actually, I need to tune this thing and the platform is actually blocking me from being able to do that.

Now, the platform authors and the people who were building the platforms will say "Hey, look, that's by design," of the customers would say, "Well, I need more flexibility than that." As a result, this active braking through the past obstruction caused customers to have to fall all the way down to the IaaS layer. That can be pretty painful if you're used to sort of get pushed Heroku master, right?

Having to deal with VM's and configuration management, all that is pretty painful. Folks who were asking for this private platform experience, what they were really saying when I dug into it was, they were saying, "We want to use the platform abstraction where it suits us. When we have to fall down to IaaS, we wanted to work with the rest of our IS. We don't want to have to run a separate island for those two things." That was really the big driving force behind a lot of that.

[0:06:39.5] JM: How did those conversations inform your design decisions when you started working on Deis? I guess I should give a little more color. Deis is what you used to describe as an open Heroku, this was a platform a service that was open source, that people could deploy wherever they wanted to.

How did those conversations that you were having in 2011 informe the design of Deis?

[0:07:03.7] GM: Well, it was really important for us to meet the operations engineers who were really our customers, meet them where they were. The really early versions of the dues PaaS, Actually used Chef, not a lot of people have realized this but we actually use chef because that's what the operations seems to be comfortable with.

That's what they were using to manage the other IS in their infrastructure, some of them are puppet but you know, Chef was pretty popular back then. Over time, as people started to get more into containers and Docker, things like [inaudible] became more popular, we had to adapt to sort of the technology stack that the cutting edge operations team was using at the time.

What I would say is, we were on the one hand, crafting a user experience for developers which was really around, just enabling developer productivity, you know, agility, faster time to market, just making the process of software delivery easier.

While at the same time, constantly improving the operational model for how we did that, making sure that it was based on what kind of technologies, high functioning operations teams we're using at the time, you know, started with Chef, moving into Docker, moving into [inaudible], later into Kubernetes, before actually appeared and we supported multiple schedulers and just making sure that we could do that reliably the whole time.

One of the things I'm really proud of actually is that we were able to do a series of re-platforms of the Deis technology stack while simultaneously keeping the developer experience, not just stable, but improving it over time and then also re-platforming over time to improve that the sort of underlying sub straight, we're able to do that with minimal disruption while keeping a strict Senva compatibility throughout the entire life cycle of the project.

Something that taught me a lot actually about how to manage open source software projects.

[0:08:50.6] JM: I'd love to talk a little bit more about the open source management conversation later on. But to give people more of a feeling for what Deis was and some of the decisions that you made when you were building it, a container platform is often defined by the control plane which is the centralized system for managing things.

You've got the data plane which is the application containers and then you've got the scheduler which is what is gathering the resources and allocating the resources based on whatever's available at a given time and scheduling them out to become application containers on the data plane.

There's been a variety of these container systems over the years that are built with various schedulers and various ways of doing the control plane and the data plane. What are some of the design decisions that somebody has to make when they are architecting one of these container platform systems that we have seen several of over the last several years?

[0:09:54.7] GM: That's an interesting question, I think the term control plane and data plane, it actually derives from your network equipment and the idea there is that, the control plane is how you can figure the network equipment, be like a firewall or a router or a switch.

Then the data plane is the path through which you know, the packets flow. Same thing is true in storage systems, right? You got your data plane for your storage system and – sorry control plane for storage and then the data plane or data path for the storage subsystems where you actually read and write blocks to and from.

Now, it's critically, the work gets done the data plane. Making sure that your data plane is fast and doesn't have any extra hops and is extremely reliable is probably the most important design consideration, right? You need to optimize for that data plane because that's where the value of the platform comes from.

The control plane is really how you operate the thing and manage the thing and for that you want to make sure that it can be updated easily, you know, you can survive certain degrees of failures, also things like security, authentication, authorization, you know, backup, restore, disaster recovery, all that, you know, it's very important for control plane services.

Yeah, you're optimizing for that data plane performance is probably the biggest one.

[0:11:09.5] JM: You mentioned that Deis was a system that was built to be able to accommodate multiple different schedulers and some of the schedulers that we have seen are things like docker swarm and Kubernetes and marathon which is Mesos – the scheduler on Mesos.

Tell me about what you have to do to build a system that can accommodate different schedulers?

[0:11:33.5] GM: Well, the first I'd say is just making sure that you have a good abstraction around how you interact with the data plan of the cluster, right? This is just sort of standard architectural best practice, making sure that you have a generalized interface for how you want to run containers on an orchestrator.

If you're experienced at all in writing software systems, chances are, you will have built something that is – your good abstract interface that you can reuse across multiple orchestrators. The real trick comes in when you're trying to target three orchestrators at the same time, you're actually bound by the lowest common denominator feature set.

Which means that you know, if one of the orchestrators supports a really cool new feature like IP per pod in Kubernetes, well, you can't actually use that, you know, in your interface because it's not supported by two out of the three orchestrators, right?

You have to consistently track what type of improvements are happening to make sure that you're improving that lowest common denominator feature set. That's a lot of work and what we ended up finding over time was that you know, customers were really better served by us taking a bet on one orchestrator.

Because really, at the end of the day, the people who were using Deis, they were developers, right? They weren't really concerned with the details of the orchestrator, the operations teams, you had to serve those developers who they cared a little bit more about which orchestrator is being used but not enough for us to keep supporting three over time.

While I was proud that we did it and while the team and myself certainly learned a lot in the process, I wouldn't advice other people to do something similar.

[0:13:10.4] JM: I think this is something that the entire industry was dealing with. I'd actually love to get your perspective on this. I've been doing this show for about two and a half years and for a considerable amount of that time, I was doings some coverage of the container space and talking to people about Kubernetes and Mesos and Docker Swarm and talking to vendors, you know, I would go to expos, vendor expo halls at the conferences and you talk to the vendors and they're supporting Kubernetes and Mesos and Docker Swarm.

It seemed like there was a lot of overhead for having to maintain compatibility with all those different systems and you know, one of the things that seemed positive about the end of the container orchestration, I mean, maybe you'll agree with me that the wars are over and Kubernetes has kind of won but everybody can kind of row in the same direction and not have to think so much about compatibility with the other systems.

Would you agree with that thesis?

[0:14:11.7] GM: Yeah, I would. You know, the one thing I would add to that, you know, yes, Kubernetes has won, you know, all the major clouds are now offering Kubernetes as a service support or have announced it.

Some are further along than others. Even vendors like Docker Mesos sphere have – to a large measure of acquiesce to discuss demand for Kubernetes based solutions so in that sense, it's good but in another, I'd say that these are never winner take all scenarios, right?

Kubernetes is going to be the dominant orchestrator going forward, that's clear but you know, things like docker swarm and Mesos and even things like ECS on Amazon, those will live for a while because people have made investments in them are going to back away overnight.

[SPONSORE MESSAGE]

[0:15:01.6] 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 into any one vendor or resource, you can continue to work with the tools that you already know such as helm and move applications to any Kubernetes deployment.

Integrate with your choice off 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 demands.

All while increasing the security, reliability and availability of critical business workloads with Azure. Check out the Azure container service at aka.ms/acs and the link is in the shownotes, thank you to Azure container service for being a sponsor of software engineering daily.

[INTERVIEW CONTINUED]

[0:16:29.3] JM: You mentioned with the open source aspect of dealing with Deis that you learned a lot and this is something I'm kind of dealing with. We have this set of open source projects with software engineering daily and there are some things that I have found to be

unexpected even with just this project which is fairly simple, it's like these apps that people can open – that people can download and basically, it's open source ways of running software engineering daily apps on their phones.

There's just simple things like always keeping the repositories in a state where people can stop by and check it out and things like going through the GitHub issues in a timely manner but this is even with just a simple project. I imagine with Deis, there's a whole lot more volume of stuff that you had to deal with and also you were building a company around the system.

What were the pros and cons to building a company around an open source project?

[0:17:27.2] GM: Wow, yeah. Well, the first thing I'd say is that we weren't really expecting the project to take off in the way that it did, it really caught us all by surprise. I mean, we were trying to build things for the customers that we were talking to and we sort of open sourced it and were a little bit shocked at how quickly it took off and how quickly we were, I don't want to say burdened because you know –

The contributions were very welcomed but certainly, the amount of open source maintenance that we had to do was – it was hard for a team as small as we were at the time. It was tricky but what I will say is that when we got disciplined about it and you know, what I mean by disciplined is, making sure that we were doing dock first development, right?

So that we would write documentation on features before we would build the features and we were ideally read documentation to describe the features then we would write tests to proof out when we were done with the feature and then we would implement the feature.

I mean, we do all that in the open and then we use sort of GitHub planning processes that were very open sourced friendly, you know, doing a lot of that stuff, not only did it make for a better software because the processes were good processes by which it creates software, it also allowed for us to accept external contributions that were quite meaningful.

We actually had a period of time where we had external maintainers on the project. I mean, these are people, some of which – I'd never actually still to this day, I've never spoken with who were tremendous contributors to their code base in Deis, reviewers.

They were functioning as core members of the team and they were doing it all in volunteer basis and part of the reason they were able to do that is because we did all the project planning and basically, everything revolving around getting software delivered for the Deis platform.

We managed to do all of it in the open. When you do that successfully, like I said, not only does it lead to better software outcomes, it allows you to scale your software effort through legit open source engagement. That's one of the pros. I think on the downside, I mean, it's tricky to monetize a business like that, right? When you're building software for free, you know, there's not a lot of really great open sourced business models.

That's a thing that you know, not only us but a lot of companies ended up wrestling with. That was probably one of the bigger challenges we had to deal with.

[0:19:52.3] JM: Yes, well, I need to consider that option of the documentation first and then implementation – I actually hadn't heard of that but that makes complete sense. That's pretty common for open source projects?

[0:20:05.3] GM: No, although, for ones that have some significant rigor around them, projects like Deis but also projects like Kubernetes, right? Kubernetes is the way you add a feature to Kubernetes is you write the docs for the feature. You know, people review the docks, they discussed the docks and once that's clear then you go about building the feature, right?

It's just the better way to go about making software, collaboratively I'd say. Now, obviously, some of that process and overhead doesn't make sense if you're a smaller team and you can kind of – you're optimizing for agility and you're building an MVP and that sort of thing. When you're dealing with an infrastructure software project and every change is pretty critical.

Changes can result in outages of mission critical software that's running on top of your platform for example. The bar is a lot higher and things I talk first, development, end up being pretty important.

[0:20:49.8] JM: Deis was eventually acquired by Microsoft and since joining Microsoft, you've worked on a couple of different projects, you've worked on as your container service and Azure container instances among other things, describe the projects that you're involved with.

[0:21:08.7] GM: Sure, yeah, there's a lot that we're doing in Microsoft now so obviously, Azure container services is one of the popular things and when I came on board Azure container service was really focused on docker, Mesos and Kubernetes and since then, we've really shifted our focus to where our customers are which is on Kubernetes and that's why we launched the new azure, the AKS version of our service, the next version of Azure container service which is a managed Kubernetes.

In addition to that, we have azure container instances which is something I'm really proud of, it's a first of its kind, containers running in the cloud, no VM overhead, micro billing, very innovative service. Also, service broker technology, we recently had an open service broker for Azure, this is something used to glue container workloads to things like azure data services that could cause missed DB or Azure Blob storage or Azure Postgres, things like that.

We also have a bunch of developer tooling that we built. Things like helm, draft, brigade which are really designed around empowering developers who are using Kubernetes to be more productive with the platform and yeah, lots of other stuff too, mostly in sort of the CNCF space.

I actually sit on the board of the cognitive computing foundation where I represent Microsoft and so really, you know, most of the technologies represented there are part of the – what we refer to as the Azure containers prior to family.

[0:22:35.9] JM: I would like to go deeper on the Azure container instance idea because this is pretty interesting and this is almost kind of getting back at your roots of wanting to do the open Heroku sort of thing because I think of the Azure from the user's point of view of just – I mean, if

you're an average user in a dorm room who is building an application and you just want to deploy some node app, you probably don't need a VM.

You probably don't need an entire Kubernetes cluster. It seems like the abstraction of just a single container is the right abstraction for the average kid in a dorm room building an online game or some basic ecommerce application.

Can you describe what an Azure container instance is behind the scenes? What goes on when I spin up an azure container instance and what is it running on behind the scenes?

[0:23:31.6] GM: Well, there's limits to what I can say but I'll share with again. You know, what I'd say is that Azure container instances is – it's very simply, it's your container running in the cloud, right?

You don't have to deal with VM's, it's built by the second. So it's a micro billing model and it's just extremely easy to get up and running, AZ containers create the path to your container image, you know, if you need have pool secrets, needs to run it, that's authentication, that sort of thing. Specify how much CPU and memory if you want and boom, it's running, right? That's it. Extremely easy to get to use and yeah, it's a pretty powerful concept, I can tell you that it's actually one of the interesting things about is this is not just an individual container.

If you look at the API for it, it's actually a container group so you can run, it's equivalent to a pod which is a group of containers that share the same process name space that can all sort of run together as a co-scheduled unit, right?

That allows you to do all different side car patterns and things like that. That's really what the primitive is. As far as what's powering it, I can tell you that there's a lot of concepts that we've taken from systems like Kubernetes that are pretty evident in the architecture of ACI and I think that enhance a little bit to how it's being powered behind the scenes. Can't say any more than that.

[0:24:55.7] JM: I don't want to get you in trouble. When you think about it from the user's point of view. I imagine container instances in this spectrum. When you think about the next

generation run times, you've got serverless functions on one side of the spectrum where you just throw code up and it runs for as long as you need and then it spins back down and it basically takes up the smallest footprint of resources imaginable.

Then on the other end of the spectrum, you've got Kubernetes where you're managing an entire cluster yourself and then somewhere in between that, you have a single container instance. Tell me if you think that is a fair spectrum and maybe talk about what are the different workloads when people are considering serverless functions versus single container instances, versus an entire Kubernetes cluster.

How should they think about where to throw their run times?

[0:25:53.3] GM: Sure, well, I think the best place to start is with the word serverless which is just you know, one of these words that's we're saddled with that is not very descriptive and you know, it's problematic –

[0:26:02.0] JM: We keep doubling down.

[0:26:03.8] GM: Yeah, everyone knows it's important so we keep talking about it, right? It is important, right? You know, the best definition I've heard for serverless is it's a conflation of three things. The first is a system that has invisible infrastructure which is to say that you're not worried about things like virtual machines or network cards or things like that, right?

The infrastructure doesn't appear to you. On the second is that it's micro built, right? People want to pay for what you use and that extremely granular level, that's the second dimension of these systems. The third is what we refer to as an event based programming model.

Which is to say that your functions or whatever they are will respond to an event. That's the method by which you invocate these different serverless functions. The first example that you provided with the serverless functions are functions as a service as I like to call it, it's all three of those things, right? The infrastructure's invisible, it's micro built and it's event based programming model, right? With Azure container instances, ACI, it's actually the first two without the third.

It's invisible infrastructure and it's microbilling but you don't have to use the event based programming model. I think that's really important because if you've ever tried asynchronous programming and you got into sort of callback based, no JS or maybe Twisted Python or some other event based programming model, certain problem to manage, it's just great, right?

Wow, it's so much less code, so much easier to reason about, And then for other problem domain, it's actually not great. It can kind of be kind of a pain to work with and you wind up in callback hell and it's just a complete disaster and then you go, wait, if I write this in a bunch of synchronous code, you know, it's 10 lines of code and I know exactly what's happening versus this twisted mess of callbacks, right?

You know I think we're the event based functions based programming model fits, that's terrific but there's a lot of cases where you don't want that or where that's too limiting for other reasons and that's where ACI is just really perfect. I actually think it is the best of server list because I think everyone wants invisible infrastructure and micro billing and I think sometimes you want the event based model but not all the time.

You know what I like to think of and like to describe it, you know I have worked in the past for a while the majority for past was 12 factor is super limiting, right? That's what everyone would say, "Oh why would I want to?" "You know my app is this in fit 12 factor therefore past doesn't work for me." Well, if you thought 12 factor was limiting wait until you meet functions based event programming. I mean it's extremely limiting as to the model and what you can do with it. So having something like ACI in your tool box is I think extremely important.

[0:28:50.6] JM: What about the scaled down quality of the functions as a service? That seems quite a useful aspect of it to me. I mean that's one of the applications that I see for functions and service that generally from what you said, I agree with you. I would rather have the ways that I can architect applications in my head, ACI is the container instances would make much more sense for most applications but certain situations where you just want eventually to get to a zero runtime.

I mean I guess you could use – could you just use container instances for those instead of the serverless functions?

[0:29:26.0] GM: Here's where I wink-wink and sort of tell you a little bit about what we have planned here coming up. Imagine a world in which you had a Kubernetes cluster where the control plain was free which is by the way it is free with AKS and then you had a add your container instances runtime that was wired up to Kubernetes directly where you only paid for the containers that you were running in Kubernetes for by the second, right?

So in other words, you had serverless Kubernetes. So if you think about that, serverless Kubernetes gives you all those benefits right? You only pay for the containers that you are running, if you only want to run containers of process and functions and then scale them down to zero you can do that. If you want to run some match processing and then kill your batch processing or your workloads, you can run that while keeping your Kubernetes control play running all the time.

So you really can get those benefits of a scale to zero or what Brendan Burns likes to refer to as the zero to end the scaling model. You can scale all the way from a cost point of zero, all the way up to end notes and seamlessly all the way through. It's pretty powerful stuff and something we are going to be offering at the container level soon enough.

[0:30:40.5] JM: That sounds fantastic and I don't know if this is how the as your function as a service platform works but we did a show about how you build one of these things and the architecture that I am aware of is instead of the code actually running on a server when you don't need it, it is sitting in a database. So you just have raw text, you have a program that you would like to run for users when your users need it but for most of the time, it's just sitting in a database.

It's a database entry of raw text and when the user requests it, the code gets taken out of the data base, scheduled onto a container and then run on the container and then the user requests to get service with that container's response to your request and then of course, assuming this is how a similar to how you do your functions as a service scheduling, that spin up time where

you're taking the code out of the database and scheduling them onto a container and responding to the user's request, that is known as the cold start problem.

And that is the knee-jerk reaction whenever anybody wants to say, "Well you know you can't get a free launch with these serverless functions." So maybe you could tell me about – well I guess if you are a serverless Kubernetes isn't public information quite yet maybe you can't tell me that much about how you are working against the cold start problem but maybe you could give me some guidelines for how you start to approach the cold start problem.

[0:32:09.7] GM: Well you know what I would say is that if you are running your functions runtime on top of a serverless Kubernetes environment let's just say, you have control over how you tune that because you know better than anyone what your cold start parameters need to be and maybe you know what times a day users are going to be coming in or maybe you want to precede a certain number hot or warm container instances to process your workloads.

You get more granular control over that versus just outsourcing it for everybody. That is one thing I would say, the other thing I'd say is that look, these functions runtimes like as your functions. You know there are just like the first party functions as a service offering. I mean we are just getting warmed up to use the phrase here in terms of how we're going to be optimizing these cold start times and so I would expect a lot of improvements there as the months go by.

And for a lot of workloads, my hope is that that's not going to matter for people who are using as your functions. If it does I think there is a lot of other options for you including things like AKS and ACI that will give you a little bit more control and flexibility.

[0:33:16.9] JM: Yeah, have you seen any scheduling problems that look particularly difficult? Like when you are building as your container instances where there any scheduling problems that looks particularly hairy?

[0:33:28.5] GM: Yeah, I guess one of the things that I'd say is there is some widely published limits as to how large some clusters can get using traditional container orchestrators and that's true regardless of what the backend of technology is. So one of the true gear things for us is at

the scale that we operate, we actually have to have multiple levels of scheduling. That's not a problem that most folks have to deal with but it's something that we have to deal with at Azure.

So that was tricky, besides that I think scheduling is a pretty well drawn path. I think where things get interesting or when you start to make state into your scheduling decisions and things like data locality, federation is also interesting when you are dealing with multiple regions and trying to place workloads that have access to data as one of the dimensions on scheduling or the fastest path to data. Other things like temporal scheduling.

Scheduling based on time so you can ideally optimized batch heavy clusters to run batch workloads over weekends and things like that. Lots of different dimensions to scheduling but I would say nothing super strict. I think federation though is probably what I've seen is really the next frontier in scheduling.

[0:34:42.4] JM: What does that term federation mean?

[0:34:44.8] GM: Federation means wiring up multiple control planes so that they can be serviced by a single federated control plane. So imagine I have a cluster running in Azure's east US, I have another one running in west US, I have another cluster running in west Europe and I want to have three separate AKS, Kubernetes environments running but I want to talk to them as if they're one cluster. So I stand up a separate federation control plane that has those three different clusters as backend resources.

And then I interact as users interact only with that federated control plane surface and that actually will do interesting things regarding and dictating the scheduling, do different things around workload placement. For example it might know that this user should default workloads to east US or it should automatically spread workloads of this type across all three regions, right? That's where you have that extra layer and room for different types of scheduling optimization.

[0:35:45.9] JM: Well that sounds extremely useful like I have heard about if you are a cloud provider operator, it's not necessarily easy to roll changes out to all of the different availability sounds and I'm sure it is appealing from the standpoint of a cloud provider to have better

abstractions for doing that and it makes sense to do that in the control plane, a control plane that's communicating with all the other control planes.

[0:36:13.4] GM: Yeah and you know as with all things, it's like is the complexity worth it right? Because the other way to tackle that is to just have smarter clients, right? So you could tackle the same thing by having just the client that was talking to your three control plans and the logic was embedded there right? How much value to do you get by adding a federated control plane in front? It's still up for debate because again, you're dealing with things like least common denominator.

You know features for example, your federated control plane will only support API's that are available as in all three clusters. So if there is a newer API available in one of those clusters, well you can't access that API through the federated control plane and lots of other complexities involved with a system like that as you might imagine. So I think it is an interesting spot today.

We are starting to see some early use of federation and a lot of really good used cases for why it matters and yet, I'm not convinced and in a lot of cases that just smarter clients may not be the better answer.

[BREAK]

[0:37:23.5] JM: If your app or website is successful, people will abuse it. Dealing with the abuse internally comes with massive opportunity costs. It slows down your product roadmap, it requires teams of specialists and custom infrastructure. If your company is concerned with credit card fraud, account takeover, fake accounts or user generated content problems like spam, fishing, upsetting imagery, hate speech and cyber bullying, checkout smyte.com.

Smyte is a customizable platform for identifying bad online activity in real time build by engineers from Facebook, Instagram and Google. Smyte is also hiring. If you want to work on a modern platform with Kubernetes, Kafka, React and lots of data engineering and machine learning, send an email to jobs@smyte.com. Smyte helps prevent bad actor on sites like Quora, Task Rabbit and Meet Up. Check them out today at smyte.com.

And if it sounds interesting to work at, send an email to jobs@smyte.com. You can also checkout the episode that I did with Pete Hunt from Smyte where he talks about some of their infrastructure and it's a fascinating platform. So I hope you enjoy that episode, if you check it out and checkout smyte.com if you've got problems with bad actors.

[INTERVIEW CONTINUED]

[0:39:02.4] JM: So in that model and forgive me because I guess I don't know too much about Kubernetes but in that model where you are talking about having a central control plane that federates requests out to a set of control planes like you've got the central control plane sitting in Redmond somewhere and it communicates with the control plane in US East 1A and US East 1B and US west 1A and you've got control planes in each of those places that are managing the workloads of cloud services over an entire datacenter.

Is the way to look at that logically is it that in that centralized control plane, you have like this is the same Kubernetes system as the other control planes or are we just talking about a communication between different Kubernetes instances? Is this a hierarchical structure or is it a flat structure?

[0:39:58.5] GM: Well you know the idea is that it should have abstractive any into one big Kubernetes cluster, right? So the idea is that you're just dealing with one big Kubernetes cluster but you get a little bit of scheduling affinities and magic kind of mixed into the model and that's in keeping with sort of the purpose of the Kubernetes API. So in that sense I'd say it's really trying to flatten the objects rather than trying to craft a hierarchy.

I also said that Kubernetes is pretty anti-hierarchical. You know the whole system is really built upon the concept of loose coupling through labels and key value labels and label selectors. So hierarchical is just not how Kubernetes works in general.

[0:40:44.5] JM: Then given that you've work at Azure now, I always love to ask people who work at cloud providers, can you tell any crazy stories or cool things that I wouldn't learn unless I was working at a cloud provider up close?

[0:40:59.0] GM: Oh man, there's not a lot I can say publicly. I think most of it though has to do with just the scale of it. I mean if you look at Azure, not a lot of people know this but we have more regions, cloud regions that any other cloud provider. The scale in which we operate is just breathtaking and what it takes to manage resources at that scale is honestly, before I join I didn't really know what I was getting into. I'd heard people just speaking about it but it's something you have to really see to believe.

Yeah, it is pretty hard to wrap your head around the sheer magnitude of it and also things like the amount of customers that Microsoft has access to and if you think about it, really everyone or most people are a Microsoft customer on some level whether that's Windows server or Office or sequel server or Azure or –

[0:41:59.7] JM: X-Box.

[0:42:00.8] GM: X-Box, I mean anything right? Pretty much everyone is a Microsoft customer these days and that really becomes very apparent when you are on the other side of the fence and just dealing with – you know I just consider myself very lucky to get to interact with the type of customers that Microsoft brings every day. It's a very rewarding experience.

[0:42:22.1] JM: I would like to talk a little bit about modern deployments of Kubernetes. So if I am deploying Kubernetes today like let's say I am just rolling my Kubernetes out at the bank. Like I ran a bank and it's got thousands of employees and many engineers, what's a good rule of thumb for how many clusters I want? Do I want one cluster per team or how should I think about how many clusters I want?

[0:42:47.5] GM: That's a great question and this is something I talked a bit about it at KubeCon recently and you know, there's no one size fits all answer is what I would say. I think the dream scenario is you have one big cluster and it's safe enough that everyone can use it for all different uses, dev test broad across multiple teams, that's all fine and part of the benefits of that is that you have the driver utilization really high.

You can run that one cluster at 80% plus utilization and that's going to mean a lot less cost in terms of compute but also a lot less operational costs because ideally you have a smaller team

who can operate that. In practice what happens is that the control plane is not something that people trust. You know people want to have fault domains for the control plane for that cluster. So for example, I don't necessarily trust that upgrading my dev cluster or the upgrade of Kubernetes from this version to the next version isn't going to break stuff.

So if you're not confident in that then well you need to split your cluster so that you can test your upgrades you know separately, right? And so that's part of the reason why you see dev test in prod clusters spinning out because people want to give the control plane a test drive and they want to make it part of their software testing journey as they promote code. So that's pretty important and then the other is on the team dimension is do you have proper role based access controls and proper security segmentation inside your clusters to make sure that you don't have noisy neighbors?

Are you labelling your workloads properly? Do you have with regard to CPU and memory limits and things like that? Do you have proper tenancy configuration? Are you allowing privilege container executions that can in theory do bad things to the host? What's your security posture for a given Kubernetes cluster as it relates to other teams, friendly teams? And so what we see is for people who have good security posture, they'll be more comfortable running multiple teams.

But they might not be comfortable with the control plane upgrade. So for those folks, you will find them running dev test and product cluster and just upgrading them to Kubernetes version. For more conservative teams, you'll see a matrix of dev test prod for each team and I think that there's a lot of people who are still there right now and then layer into that often times people will have devs have their own Kubernetes environments to do their own dev sort of in the cloud and that adds another dimension to it.

With that said, I think as the controls and some of the tunables and the isolation primitives get better. I think we'll start to see more of a trend towards single larger clusters.

[0:45:29.5] JM: Great answer. What if I am an IOT company? Let's say I am an oil refinery and my oil refinery is several football fields in square footage, square mileage. I've got sparse internet across my oil refinery and I've got plenty of things that would be considered IOT

devices. Could I connect all of those across a single Kubernetes cluster and what would be the risks of doing that when I have intermittent internet connections between the different nodes in my giant oil refinery?

[0:46:05.4] GM: Yeah, that's a great question and I can tell you that we are actively exploring Kubernetes for these used cases. It is an interesting area here. Kubernetes has the concept of heart beating between the different nodes. Think of an IOT device and the Kubernetes control plan and so one of the things that we're doing for example for Azure stack which is like this cloud in a box appliance that you can buy from Microsoft today is the idea that you'd be able to have an IOT device talking.

To a Kubernetes environment running on an oil rig somewhere and that Kubernetes environment can be federated with a cloud based Kubernetes environment and you could in theory have all of these things working together. You know obviously the trick is as you said, if you have poor links between your devices and sort of the home Kubernetes on the edge that can be tricky but that is also going to be problematic regardless of what your this is your management system you are using.

For example when you need to push down an update to the IOT devices themselves, you need to rely on connectivity to inform the device that it needs to get an update right? So on some level, you have to assume there is some connectivity between the IOT devices and some edge device, can that be Kubernetes? Technically, yes, but there are certainly more work to do there to figure out how feasible to fund this.

[0:47:30.6] JM: Another random example I was thinking of is what about a connected car or a smart car where you've got the cars driving around and the biggest question I have is would it makes sense to deploy Kubernetes to a car like that? Maybe you would have different nodes that are managing the wheels or managing the dashboard or have you seen anybody using Kubernetes in a car?

[0:47:54.2] GM: The things that I have seen people using Kubernetes for would freak you out. Yeah, not all of which are advisable mind you but I mean yeah, you know in theory Kubernetes is a system to enforce desired state for compute resources and anywhere where you have that

problem, you could in theory use Kubernetes to drive towards those solutions. Now it could be overkill, it could not be the communication model where the nodes heartbeat to Kubernetes control plane on it.

An interval might not be ideal for that particular model, the form factor might not be ideal but I can tell you that people had run Kubernetes on raspberry pies to extremely good effect. So we know it could scale down really small. I think the questions are more around what does your control surface need to look like? How are you updating your workloads? Is the car really the source of truth? Are you trying to federate that to a dealership or some manufacturer?

And that is going to be pushing down updates and I think that's really where they questions lie and Kubernetes as sort of the mechanism by which you distribute software to different components in a system like a connected car. I think that in a lot of ways less interesting because there's just lots of different answers for that.

[0:49:11.0] JM: Okay, so I won't keep you much longer. I just asked a more far flung question, I talked to Brendan Burns a couple of weeks ago and one of the things he said that took me by surprise was he said that we might be able to see a return to the model of people buying single purchase binaries of software and I think what he was referring to was the fact that since we have Kubernetes as a runtime that's on every cloud provider now, regardless of whatever cloud I am going to deploy to I'm going to have access to a Kubernetes cluster.

So that means you could have a cross cloud app store which seems like something we haven't really had before because as long as you could define let's say a helm chart for how to deploy a Kubernetes app, then you could have that as aesthetic binary and you could sell it for $99 or something like that and you could just run it on whatever cloud provider you prefer and like that could be a business model. Do you think that we'll see changes like that in the way that people purchase and deploy software?

[0:50:20.5] GM: Yes, I think we will see that. With that said, I think we will see it because I think all of the ingredients are there. We have not just Kubernetes on multiple clouds but we have a commitment from the CNCF membership, you know Microsoft, Amazon, Google and others to ensure that we're delivering conformant Kubernetes which means Kubernetes that works

everywhere the same way for customers and so that sub straight gives us the ability to have that sort of unified compute environment.

That combined with helm charts is a packaging mechanism by which you can distribute that kind of software which is seeing in a massive uptake in the community and part of the Kubernetes project itself gives you the other ingredient there and you know I think it is only a matter of time before we figure out how to get the rest. The only other thing I'd say is it's notoriously difficult to build these sort of market place solutions where you connect up providers and ISV's and customers all together.

It's not clear that there is as much customer demand these days to pay for software, right? I think that's questionable, these are certainly for some things databases. I know people are still paying good amount of money for it but you know the software that you are talking about $99, you know I am curious what is that. That people are going to pay $99 for and is there going to be enough of that to incentivize ISV's to participate and to incentivize customers to show up and purchase. Maybe, maybe not I don't know.

[0:51:49.7] JM: Yeah, I was trying to think about that. I was like maybe something like Zen Desk where you have a help desk service and we know Zendesk is a great service, kind of expensive and there is a lot of mid-market companies that would be looking to save a little bit of money and maybe if they could pay $99 as oppose to paying, I don't know, $800 a year or whatever they pay then yeah maybe.

[0:52:11.6] GM: Yeah, maybe and yeah I think you know the thing I wonder is for a solution like Zen Desk, I think other similar sass software I think a lot of the benefit to end users comes from the operations right? And the fact that they're taken on the operational burden, right? But if you can make that sufficiently automated via Kubernetes and applications package via helm, potentially that could be a viable model, who knows?

[0:52:36.0] JM: Yeah, who knows? All right Gabe, well great conversation. The time flew by and I look forward to talking to you again in the future.

[0:52:41.8] GM: Thanks for having me. This was fun.

[END OF INTERVIEW]

[0:52:46.6] JM: GoCD is an open source continuous delivery server built by ThoughtWorks. GoCD provides continuous delivery out of the box with its built-in pipelines, advanced traceability and value stream visualization. With GoCD you can easily model, orchestrate and visualize complex workflows from end-to-end. GoCD supports modern infrastructure with elastic, on-demand agents and cloud deployments. The plugin ecosystem ensures that GoCD will work well within your own unique environment.

To learn more about GoCD, visit gocd.org/sedaily. That's gocd.org/sedaily. It's free to use and there's professional support and enterprise add-ons that are available from ThoughtWorks. You can find it at gocd.org/sedaily. If you want to hear more about GoCD and the other projects that ThoughtWorks is working on, listen back to our old episodes with the ThoughtWorks team who have built the product. You can search for ThoughtWorks on Software Engineering Daily.

Thanks to ThoughtWorks for continuing to sponsor Software Engineering Daily and for building GoCD.

[END]