EPISODE 976

[INTRODUCTION]

[00:00:00] JM: When the Kubernetes Project was started, Amazon Web Services was the dominant cloud provider. Most of the code that runs AWS is closed source, which prevents an open ecosystem from developing around AWS. Developers who deploy their applications on to AWS are opting into a closed controlled ecosystem, which has both costs and benefits.

The software industry has a history of closed and open ecosystems existing at the same time. AWS represented a huge closed ecosystem, and with the amount of money at stake in the cloud business, it was only a matter of time before a more open ecosystem emerged. Since the creation of Kubernetes, the world of cloud computing has evolved rapidly. Google and Microsoft have both invested heavily into Kubernetes, and Amazon itself has adapted to the newer competitive landscape with a Kubernetes offering of its own. Amazon has also made efforts to become more publicly involved in open source projects including Kubernetes.

Kelsey Hightower has been a part of the Kubernetes ecosystem since the project was started. He's one of the most recognizable faces in the world of Kubernetes, delivering keynotes, appearing on podcasts and co-authoring the popular Kubernetes Up and Running book. Kelsey joins the show to discuss the progress in the Kubernetes ecosystem and the competitive dynamics between Kubernetes and AWS.

the software world. So send me an email if you're interested in working with us, jeff@softwareengineering.com.

[SPONSOR MESSAGE]

[00:02:25] JM: If you are a SaaS or software vendor looking to modernize your application distribution to gain more enterprise adoption, checkout replicated.com. Replicated provides tools to deliver your Kubernetes-based application to enterprise customers as a modern on-prem private instance. That means your customers will be able to install and update your application just about anywhere.

Bare metal servers in a cloud VPC, GovCloud and their own Kubernetes cluster, vSphere. This is a secure way the your customers can use your application without ever having to send data outside of their control. Instead of your customer sending their data to you, you send your application to your customer.

Now, this might sound difficult and maybe you're not used to it because you're a SaaS vendor. You're a software vendor, but Replicated promises that recent advancements from tools like Kubernetes make it far easier than before, and the Replicated tools can help vendors operationalize and scale this process.

The Replicated tools are already trusted by noteworthy customers like HashiCorp, CircleCI, Sneak and many others. As a result, over 45 of the Fortune 100 already have an application deployed via Replicated in their infrastructure. That's a strong sign of adaption.

Go to replicated.com for a 30-day trial of the full Replicated platform. You can also listen to an interview with Grant Miller, the CEO of Replicated, that we did a while ago.

Thank you to Replicated for being a sponsor of Software Engineering Daily, and you can check it out for yourself at replicated.com and get a free 30-day trial.

[INTERVIEW]

[00:04:34] JM: Kelsey Hightower, welcome back to Software Engineering Daily.

[00:04:37] KH: I'm happy to be back. I think this is the first podcast where I've come back twice.

[00:04:42] JM: Awesome. Well, glad for that to be the first. I am just returning from KubCon and AWS Reinvent, and I'm trying to understand how these two ecosystems fit together. What is the overlap between the Kubernetes ecosystem and the AWS ecosystem?

[00:05:00] KH: I think one big overlap is this whole concept of cloud native and the patterns above it being rooted in open source. So we're starting to see that overlap before. I think earlier, it was all about cloud pick a vendor, and ideally they do a good job and you kind of get locked into their ecosystem for better or for worse. But now what we're starting to see between KubCon and AWS Reinvent is that a lot of these ideas are not open source. Ideally, you see things like Kubernetes and there are Fargate for Kubernetes adapting the same open source ideas, APIs and platforms that you see at KubCon.

[00:05:39] JM: Do these two ecosystems, the Kubernetes ecosystem and the AWS ecosystem, do they compete with each other?

[00:05:46] KH: Yeah, there's friendly competition I think on multiple areas, and in this course, the natural competition. If you think about Linux, before Kubernetes, we have Linux. All cloud providers are trying to host Linux for you in the form of some VM wrapped around a broader IS offering. It's going to give you things like load balancers, IEM and storage. The cloud native world is very similar, right? Open APIs, open platforms. Now what you're starting to see, there's a lot of collaboration when we talk about the source trees, like, "Hey, I want to contribute feature X, Y, Z because that's what my customer base needs."

So that's where you're going to see this collaboration, because we have another extension point and so forth on the open source side. The competition comes in when, let's say, we have an existing container runtime, but maybe we feel that we can do a little bit better or do something different or focus on security. Then we start to see competing ideas and offerings that may now kind of split the community. Now some people may decide that they don't want to run in a containerd, or let's say Red Hat's container runtime, CRI-O, and they just want to go to

Firecracker now. So that's a new entrance into the container runtime space. That's going to divert some of the attention.

[00:07:02] JM: We can dive a little bit deeper and some of those technologies, but take me inside your empathy matrix. When you think about your description of the playbook that you think AWS has been executing since the rise of Kubernetes. Because it AWS was very well-positioned to respond to the potential competitive threat of Kubernetes, and I think AWS has executed quite well. What is the playbook that they're running? Do you have a perspective for what that playbook is?

[00:07:38] KH: I think something that's fairly transparent is that they try to meet customers demand right there in their model, right? They want to be this customer-obsessed organization. When Google put out Kubernetes, we put it out for everyone, like Android, the Go programming language, Chromium. All of these things we tend to do in the open ideas, open implementations. I think what Amazon is seeing, I don't work there of course, is their customers want to be on top of this open wave of thinking and it's not just because it's open and free.

Right now I'm overseeing its open and better, right? These things are – Kubernetes feature-wise, is feature-rich, has a great ecosystem that's building around it, it's extension points. I can see this – If I was a customer standing from the side, if I were to compare ECS versus Kubernetes, if I can get both of those tools and a fully managed set up, I think I would lead towards Kubernetes because of all the things I just mentioned.

[00:08:40] JM: AWS is this mostly closed and proprietary ecosystem, and Kubernetes is open source. But most of the consumption of Kubernetes is through these proprietary managed Kubernetes offering. So even EKS, which is kind of a more opened flavor of ECS in some ways, it has its lock-in through things like IM policies, and all these providers have their own flavor of lock-in. I have not seen yet the ease of portability or like mirroring different Kubernetes like infrastructure systems between different instances of a managed Kubernetes provider.

So why does it even matter that Kubernetes is open source here? If we're going to be locked in to a specific cloud provider with whatever managed Kubernetes installation we choose to use, why does this open source even matter here?

[00:09:42] KH: So there's a couple of things. So you have to decouple integration work from lock-in. Google Cloud, Azure, AWS, we all have our own implementations of IM, and Kubernetes afford space to be able to integrate with any of those. There are different ways you can integrate. One way would be just expose your platform's native metadata service directly to the pods that run inside of Kubernetes. Another way is to link Kubernetes service accounts or identity and allow those to kind of pass-through or be transparent to the cloud platform.

So in the one scenario where you say, "Let's just make Kubernetes a trust domain what you're starting to see from the cloud provider." So that means Kubernetes can be first-classed issuing tokens that can be used to authenticate to the rest of the cloud platform, and that's just the recent thing that Kubernetes enabled over the last couple of years. So in that case you have to be a little bit more patient to say, "Is there something we can do better than the native integration that gives us that portability?"

I think the last thing here is why is it important that it's open source? First, so that you can actually get what you mentioned. So if a provider decides to lock you in, that's kind of their freedom to do that, right? However they choose to integrate. But the one thing we can do is we get the portability of the workflow, and I think the workflow is really what people are after.

So it's not about having all the cloud providers provide all the same APIs, all the same services. That race to commodity, maybe I think some people believe that there is a world where that makes sense, but I think in the way of Kubernetes is even if you have all of these differences, we still can describe them using the Kubernetes API. We still can integrate with them using the same workflow. So at least when you end up in that world, you can describe what you have and that makes it easier to port the workflow between the multiple providers.

[00:11:40] JM: Amazon's managed version of Kubernetes is EKS. Google's version is GKE. Have you heard any anecdotes about how the UX of these two managed Kubernetes versions compare to each other? I'm trying to understand what are the key design considerations between different managed Kubernetes installations.

[00:12:03] KH: So GKE is about a world where Google manages the control plane for you. So that means etcd, the scheduler, the API server, none of that runs today in your infrastructure, not in your project, not in your VPC. We control that. We manage it. We fully upgrade it.

Then inside of your project or your account, with GKE, you get the nodes, and the nodes are the things that we manage using the GKE API. So that means the nodes are fully managed. We create the nodes. We auto scale the nodes. We have a thing called auto repair. the goal is to make it feel like your cluster is fully managed by someone else and you just leverage the Kubernetes API, but you have enough there to customize. So if you believe you need to have a mix of standard nodes in your cluster plus some with GPUs or these TensorFlow processing units, TPUs for short, you can do that using the GKE API. So it's a fully managed thing.

Now let's go to EKS. The EKS that people use today, that is about getting a control plane, which will run in your project and your account. Then once you have that control plane, it's up to you to bring your nodes to the table. So you can use a cloud formation template. You can use some open source command line tools. I think thing Weaveworks has a really nice tool that people are using. I think it's EKS ETL. Then what that will try to do is create a node pool to attach to that control plane. So at that point, you have to kind of manage the nodes yourself. That's your line of responsibility.

The other thing that I don't think you mentioned was Fargate for EKS, and what that does is says, "Well, let's treat Kubernetes like a serverless platform and we're going to make some tradeoffs here." So this is going to get you closer to GKE, but the tradeoff to do that though is in the Fargate for EKS world, you can't do everything that the Kubernetes API supports. They make some tradeoffs. For example, block devices can't be used. There's a concept of a daemon set, so these logging agents or whatever agent you need to run on every machine, but not inside of every container or pod. Those are also not available. So some will say subset of Kubernetes to get you closer to that fully managed experience that you see in GKE.

[00:14:22] JM: You mentioned this Fargate product from Amazon, and the first instantiation of Fargate was just a standalone container product, and it was an ambitious and great idea. Just you want to spin up a single container almost like a little bit more of work and isolation properties that you would get out of a Heroku-like experience, I think. But initially, it did not

integrate with Kubernetes. Today it does integrate with Kubernetes. It integrates with EKS so that you can have this combination of a – You can have a lot of configurations of a combination of a managed experience and an unmanaged experience on the Amazon cloud.

Can you explain the importance of that Fargate standalone container product and its integration with EKS? Go a little bit deeper on the synergies between those two products.

[00:15:19] KH: So if you can remember, we step back for a second. There's EKS, which is Kubernetes- based. Bring your own nodes and manage it. Then there's ECS, which is their original container runtime that's a little bit closer to like Docker Swarm or some of these things that really kind of assume the world where there's one container per VM. Maybe you can pack them densely, but the idea was just one container and we would just automate that for you using more of a declarative config, right? This is kidn of closer to a Nomad is doing. Docker Swarm previously. That's the idea of a ECS, and that product was really early on, right? It predates Kubernetes for the most part, I would say.

Then Fargate has two flavors. Fargate's goal is to try to make the infrastructure disappear. So whether you like the EKS API or the ECS API, those are just two different APIs to manage containers, right? One gives you the whole power Kubernetes in the way of thinking about pods, one or more containers, plus volumes and secrets and environment variables and annotations. Just all the stuff that we attribute to the Kubernetes API, versus ECS, which has its own thing. Maybe it has its own community. It has its own way of thinking about this problem.

Fargate comes along and says, "Well, what if we were able to manage everything for you and allow you to move towards this idea of serverless, like this idea that the cluster is somewhere, but you don't interact with it directly, and instead you focus on either using a Kubernetes-like API or the ECS API to run your workloads and they'll give you a pricing model that's a little bit closer to the workflow being run, whereas in GKE, you pay based on the cluster of resources, right? So that means if you can do BIN packing yourself and get a lot of density per node, then you will have a tremendous cost savings. So it's not necessarily a workload pricing model versus a cluster pricing model.

[SPONSOR MESSAGE]

[00:17:24] JM: Over the last few months, I've started hearing about Retool. Every business needs internal tools, but if we're being honest, I don't know of many engineers who really enjoy building internal tools. It can be hard to get engineering resources to build back-office applications, and it's definitely hard to get engineers excited about maintaining those back-office applications.

Companies like DoorDash, and Brex, and Amazon use Retool to build custom internal tools faster. The idea is that internal tools mostly look the same. They're made out of tables, and drop downs, and buttons, and text inputs. Retool gives you a drag-and-drop interface so engineers can build these internal URIs in hours, not days, and they can spend more time building features that customers will see.

Retool connects to any database and API. For example, if you want to pull in data from Postgres, you just write a SQL query. You drag a table on to the canvas. If you want to try out Retool, you can go to retool.com/sedaily. That's R-E-T-O-O-L.com/sedaily, and you can even host Retool on-premise if you want to keep it ultra-secure.

I've heard a lot of good things about Retool from engineers who I respect. So check it out at retool.com/sedaily.

[INTERVIEW CONTINUED]

[00:19:01] JM: How does the Google vision of Knative – So Knative in my understanding is basically an open source serverless-like experience built on top of Kubernetes and it is not serverless in the sense of the ephemeral serverless concept. It's serverless in the sense of ephemeral if you need it long-lived, if you need it – The Knative spectrum of container life is as wide as it gets. Then Cloud Run is the idea of a managed Knative experience. How does that vision compare to the Amazon vision of the Fargate, ECS, EKS family of products?

[00:19:46] KH: I think you have to really understand the way Google has been doing things for about 15 years, right? We put out the MapReduce paper, and then upSpawn's the Hadoop big data ecosystem. We put out the Borg Paper, [inaudible 00:20:01], Mesos, Docker, Docker

Swarm, Nomad and so forth, and even Kubernetes comes from that lineage. Google has always been in this habit of publishing our ideas for others to consume.

Next we have open source projects that take these ideas and make them real. You can actually download them and use them, okay? We've done that for containers. We've done that for web browsers. We've done that for mobile devices. We've done that for programming languages. When you start to think about serverless, it's only natural that we do the same thing.

So the app engine team has been doing what people would now regard as serverless, right? This idea give me your code and I'll run it for you, and this is assuming you don't consider serverless to be only functions as a service, like Lambda, right? If you assume that serverless is more about a managed compute offering, weathered it's scaled to zero or not, doesn't really mean too much as long as there's maybe some people would say, "In the serverless world, we have a billing model that's closer to either A, the workload, or B, the request," right? So we have this more fine-grained billing model that isn't pay for the whole machine 24/7.

So let's get to Knative. So if you take the app engine platform and the ideas and all the things we learned after 10 years, what kind of API do you need to do that? Well, one, we already have this distributed system internally called Borg, and in the outside world, we have a distributed system called Kubernetes. These are both great platforms to build out a serverless offering. We've build cloud functions on top of Borg. We've build app engine on top of Borg, and Cloud Run on top of Bork.

So when you start to think about this is what would you do in the open source world? Well, on the open source world, you need the same control plan components that are missing from Kubernetes. One is serving. So Knative serving is about the ability to take a request and just-in-time, spin up a workload, and when the request is done, spin down the workload.

Knative also gives you a few more knobs to twist and say, "Hey, what if I want that workload to not have cold start?" Well, in Knative's API, you can say things like minimum number of instances. So that way you can have a little bit more fine-grained control over latency and things like whole start.

Take that API and what would it look like if it was fully managed? So in the Kubernetes world, we have this Kubernetes style API. So Knative doesn't need to expose the entire Kubernetes API. That's pretty big. So what Knative does is this focus on services. You can route traffic to different services. You can have different CPU types. You can have all of these things, and Cloud Run gives you the fully managed serverless feel, where you pay at the request level, but you get to use the same API, right? So this is a higher level abstraction.

Now, the last part of your question is how does this compare to ECS and Far gate? Well, what they're doing is exposing the full ECS API, and in many ways, the full Kubernetes API. So you don't get a new abstraction here. So you're still dealing with pods and services and all of these things and the Fargate for EKS world. Whereas in the Cloud Run or a Knative world, you try to end up with a subset of Kubernetes resources being exposed to focus on running containerized workloads, and the last thing here is the function framework. So if you do like functions as a service, this idea that there is an event and that event will invoke a function, then in that world you can use the Knative eventing services, and that's going to allow you to create custom events. So maybe you want an event off of GitHub.

Maybe you want an event off of PubSub. You can do all of that in Knative. There will be a broker that will keep track of everything. Then if you want to have a deployment model that's a little closer to Lambda and maybe not the whole container, or this is idea by function framework where you can give us a code snippet and we generate the runtime behind-the-scenes that can respond to those events.

[00:24:06] JM: When you think about this disparity in the two product set offerings between the Google vision of this stack of serverless to Kubernetes abstractions and the Amazon vision of the same category, the serverless to Kubernetes, how are you running all these different things? What kinds of applications do you want to run on these things? What's the API? What's the user experience? Etc. How do these respective product sets illustrate the long-term vision for each of these cloud providers? How do they reflect the long-term vision of Google cloud and AWS respectively?

[00:24:50] KH: I can't comment too deeply on the Amazon one, right? But in the Google one, when cloud first started, I think the virtual machine was the standard unit of compute. Here is a

virtual machine and you can connect it to networks, VPCs and load balancers and you can auto scale it. That was the unit of compute.

Google's vision, the unit of compute will be the container image. I can run a he container image on a VM. So if you go to Google cloud, you can do G cloud computing instances curate – – with VM. That's just going to give you a virtual machine that will automatically pull down a Docker container and just run it. Or you can go up a layer and say, "You know what? I really like the Kubernetes API. I want to compose my app of multiple container images and get things like automatic failover and just a way Kubernetes treats infrastructure." Then you can use the exact same container image and write it in Kubernetes.

Then you may say, "You know what? I want a fully managed environment. I don't want a cluster. I want Google to do the heavy lifting for me, do the load-balancing, IM integration, just everything." In that case, you take the same container image and you move it up to Cloud Run.

So the goal here is that the unit of compute now becomes the container image. Now, when it comes to workflows, you may say, "Well, I don't want to learn how to make Docker files or build these things." Well, we can give you a function framework, but here's the thing, we will always create a container image underneath the covers. Put it into a container repository so that way even though you're starting with the function framework, you can still take the resulting container image that you didn't have to build that we maintain and you can move it back down the stack all the way back to a VM.

Industry-wide, I see that this theme is taking over. In the Amazon world, Lambda is getting real close to being a container image. I mean, it's real close. Lambda layers, this ability to customize the Lambda format and packages. I'm predicting one day they'll just accept the container image to act as a function to be on par with the rest of the industry, and the other platforms that you've mentioned already treat containers as a first-class thing. It's just that Google feels to be all-in on this container being the unit of compute.

[00:27:09] JM: We could continue talking about the minutia of these container platforms, container runtime and stuff, but I have some other subject I want to discuss. It's the end of 2019 right now, and we're coming off of a year where a lot of the things that we considered axiomatic

about open source or that many of us considered axiomatic about open source are being questioned when we have license changes in ways that licenses and business models are changing kind of with a form of legalese or with a particular cloud provider, in this case, AWS, in mind as a competitive threat to the very nature of open source from the perspective of some companies and some open source communities.

How is the definition of open source changing today?

[00:28:05] KH: I don't think it's changing much at all actually. I think there's now it's so mature that now we're talking about the business aspects that should have been there all along. There's always been a legal foundation for source code. We're just starting to see it become challenged a bit, right? You just can't copy code from someone else proprietary or open and not give them credit for that, right? There're kind of various laws that pertain to how source code can be shared, or copied, or redistributed.

Open source, the various licenses that you can choose. So if you write source code, you have the right to choose the licensing model for your source code. Now some people could say one license is better than the other. That's their perspective. That's their right to do that. But the legal structure allows you to choose the license for the software that you write.

The next level of that is the business. How do you want to run your business? It's independent of open source, whether you want to share your source code or not. I think you're going to find it really hard these days unless you have a lot of other value, like fully managed or a lot of support. There's something else. But usually when it comes to software, you're competing with a broad global ecosystem of capable people that are able to produce software for almost zero cost.

Maybe they're doing it on the side because they're paid really well at their job. They have no interest in creating a business. Envoy is a great example of this. Matt Klein doesn't need to build a business on top of Envoy. So he can give that away for free and started Lyft, but tons of contributors who see the value and saying, "Well, if we all contribute to this, then it's a shared amortized engineering cost over this project, but we can use it in our larger services, whether it's a service mesh or so forth."

I think what you're starting to see now is people are experimenting with the various business models. I think there's a part of business which says some people will only pay for things that they have to pay for. So if you open a movie theater and you said buying a ticket to the movie was optional. My guess is a lot of people would just go see the movie for free.

Now, if every movie theater didn't have movie ticket prices and they just made you buy a seat or they made you buy something to eat, and that was a sustainable business model, then that will impact the market and make it where you can't sell a movie ticket anymore. I think when it comes to software, what we're starting to see is the previous model where you could charge a lot of money for software on a licensing fee, being challenged from the open source incumbents.

I think the last thing here is a lot of companies will start with open source. It's a great way to get the funnel of people into your funnel. They can see your software. They can use your software. Then when you try to build a business after that, now you have a challenge. Should the new stuff be open source as well? Is that the expectation from your user base, because probably they're not customers yet? They don't pay you, and maybe the expectation is not the same you would have with the paying customer. I think that's where it just gets a bit tricky.

[00:31:21] JM: Putting yourself in the shoes of these companies, and you don't have to go too deep on this answer if you don't want to speculate too much. But if you think like one of the – There're a lot of database companies that have changed their licenses, and the perspective of them changing their licenses is we don't want Amazon to be able to copy our codebase, offer an Amazon managed experience of the database that we have worked so hard to write, and basically have the best funnel possible for taking customers into that managed database experience.

They don't want to compete on the axis of incrementally better managed experiences because they are presuming that customers don't care as much about the incremental add-on of their managed database experience relative to the baseline features that come from simply standing up a managed and hosted and operationally-managed database experience that AWS can provide.

I think the counterargument to doing this is Amazon has placed a competitive constraint on this kind of open core business model. This is an opportunity for these other companies to rise to the occasion and build incredibly good software that is so good that customers would opt for going outside of the AWS ecosystem because the provider is known for being amazing. Instead, these companies are opting for changing their license.

Now these two things are not mutually exclusive. You can change your license and have a legal insulation against AWS, and then you can still build great features, whether it's KSQL or whatever, some amazing thing on top of CockroachDB or whatever it is, data provenance. But isn't there a certain boldness in the idea of open source and open core? Why not take the bold option? Why not ignore the opportunity for legalese, because we're software engineers. We innovate our way out of these kinds of situations. We don't turn to the legalese.

[00:33:55] KH: I don't want to say software engineers to understand business, because business has nothing to do in my eyes with just writing the best software, because the best software doesn't always win. That is a fact. The business part is – And sometimes it boils down to you only pay for what you have to. If I don't have to pay for it, why would I pay for it? That's just like business like 101. Then you get into other areas of you have to protect your logos, right? This is why Red Hat, one of the best open source companies in the world, but use that logo without permission, then it's going to be a discussion. Depending on how that one goes, there may be a discussion in court because they have to protect the brand. It's not just about software. There's a brand association to that software. Based on that brand association is what you're going to pay for. The support, the security patches, the things that go above and beyond just putting some code out there for people to download for free and call you when there's trouble.

There's a lot more that goes into developing software. Most engineers I know don't work for free. So their money has to come from somewhere. So every business is going to be unique. Every business is going to try a different way of attacking some of these challenges that they face.

One thing we really can do as observers is say, "We have to see if that's a good idea." Right now we may say, "Hey, that doesn't look like a good idea for me." But let's say five years from

now their business quadruples and it turns out to be a good idea, then that will become the next theme. That will become the next pattern that the next open source company takes because it has been proven to work.

So part of this whole openness is the ability to experiment with these ideas. We experiment with the GPL. We have the MIT license. We have the Apache license. We have the BSD license. Some people use GitHub. Some people use Bitbucket. These are all freedoms that allow people to experiment in the way that they see forward.

So if that was a bad business decision, then we have to wait and see. We can't really make a call from the outside and say, “Oh, look at that. That's a bad decision.” Because if it turns out to be a great decision, guarantee that someone’s going to copy it. I think that's going to be a thing that time will tell if that was a good decision or not. But the thing I like about the whole situation is you have the freedom to make those choices. That's the key here, because if you don't have the freedom to make those kind of choices, then you get put in the very bad spot.

If you, in your case, you could decide to build what will start to look like proprietary software. If you told me that you can only use the software with the license and it stopped working with that one, then you start to look a lot like every other proprietary vendor out there, because you’re trying to attack the problem. If I make all of my software free, people who have the skillset will use it for free, because the value I can provide may not be worth paying for. Every business is going to have to weigh that one out and make the best decision for them. So I don't know if there's a one right way of doing it.

[00:36:59] JM: Well said. Moving on down the subjects I wanted to discuss with you. At both KubCon and Reinvent, I felts like the idea of platform engineering has really made its way into a wider number of enterprises. I know that enterprises have always had some central – Well, for a long time they’ve had like central technology team that like once an enterprise gets to a certain point, the central technology team, they start making some longer-term plans about modernization and so on. But the idea of platform engineering has really matured, and I think the SRE book had a lot to do with it and just broader discussions.

This perhaps is the standardization of platform engineering tools have made things a lot easier, like people are really asymptoting towards Kubernetes, or they're asymptoting towards AWS, or these different things, at this point, they're here to stay.

Describe the decisions that an enterprise platform engineering team is making today.

[00:38:09] KH: Yeah. I think the decision as a preferable engineer is how do I make it easy as possible for the decisions, the ideas, products and services that my company develops gets into the hands of the customer. That's it. That's the endgame for a platform engineer.

So there're various ways of thinking about it. Maybe why start with a source code repository? What kind of controls do I want to put at the source code level? So maybe I require code reviews before things get merged. Maybe I have a tool that does static analysis for security vulnerabilities right there at the source code level?

Then I make another set of decisions. The thing that's going to actually to build that software and ready it for production. I'm going to start to think about the end-to-end build process. I'm going to sign may be container images to make sure I can trace it back to the original source repository. I'm going to attack my dependencies in a way where it's reproducible. Then I'm going to take that software and place it in a repository that follows the same set of principles.

Now that I have this end-to-end software governance process, now we can start to think about how do we deploy? If you're in a platform team, one of your goals and decisions you have to make is how do you securely and safely and repeatedly get this software into the customer? That typically involves deploying some server-side software. That means what tool are you using to do those things?

10 years ago, 20 years ago, maybe you had to build all those tools yourself like Google did, right? Things like Borg and all the things we use for security between the various applications. But now what you're looking at is a lot of these ideas, a lot of these fundamentals are baked into tools like Envoy, Kubernetes. These are all the fundamentals now. So they are there.

Now when you start to think about doing this end-to-end process, you're just coming together with a collection of tools that are starting to again a lot more synergy between them. Jenkins is now aware of how Kubernetes works. Kubernetes is now aware of how Envoy works, or maybe the other way around that these other tools can now get service information metadata from the platform itself. Same thing for metrics and logging. I think the decisions that that platform team is making is now being supported directly in the tooling that they're using versus having to build it from scratch.

[SPONSOR MESSAGE]

[00:40:47] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW CONTINUED]

[00:42:35] JM: Let's say I am at big organization. Maybe it's an insurance company. A big, successful insurance company with multiple technology teams throughout. There're service teams. There're customer-facing teams, whatever. Different teams throughout the company at this point have set up their own Kubernetes clusters. They've each got Kubernetes clusters.

At a certain point, you get to a point where Kubernetes clusters are strewn throughout the organization and they're not connected to one another. Does it make sense to have some kind of central Kubernetes cluster that I issue commands to and those commands are federated out to the other clusters, or is it okay to just have these disconnected disjoint community's clusters throughout the organization?

[00:43:29] KH: Yeah. I think it just depends on how the organization works. So I've worked with a lot of companies that you described, and what it turns out, it's not just one company. A lot of times these big companies are made up of many small companies. Maybe by division, maybe some acquisition, and they typically run a bit independent and their interface is really at the top. Meaning, you have to go through some load balancer or some special network to get to their service.

So the fact that they all have their own Kubernetes clusters really represents their true organizational makeup. So once you have all these Kubernetes clusters, let's start to think about how can we share tooling and workflow. I think there's two parts of this. There's the right side. How do I deploy things into those clusters? Then there's the read side. On the read side, that's about saying, "You know what? If everyone here is going to be running Kubernetes on their own, well, step one, maybe that's a good idea because at least now we have a system that has enough metadata and accounting about what's being deployed by who and when, and there's hooks in there that we can enforce companywide policies."

So first order of business would be first let's read and make sure that we can account for what clusters are out there. Once you have a catalog of all the Kubernetes API endpoints, I can then start to pull in all of that data to tell me what's running where, how big those clusters are, what versions and things around security.

The second thing a can start to do is think about either shared tooling. Maybe today those teams have set up their own clusters and they're hitting that Kubernetes API directly. Now in some cases, some teams will be responsible enough to handle that. What I really want though is maybe teams using some CICD process that is then kind of hard coding in many ways that this application goes to this set of clusters. Then that becomes the policy.

Really what I want is policy engines dictating where things go, because they may have something to anchor our discussions around. The policy for this teams application say they go to these clusters and they move from QA to production in this way. I think that's the second thing you want to do.

Now in terms of aggregating things via a big central cluster, that starts to makes sense, and that becomes a little bit easier. Once you have these other things in place, what's out there and a way to get things out there. Once you have that, then you can start consolidating clusters around boundaries that makes sense. Maybe three different teams are actually building things for the same region of the world. In that case, I can decide. Do I give them a dedicated cluster because of security and other boundary concerns, or do I give them a single cluster and maybe use things like Kubernetes name spaces to segment them between? You can't really make these decisions until you kind of decouple people from directly using Kubernetes.

[00:46:26] JM: The CICD step is becoming more important. It's becoming thicker. There're products for security and compliance and static analysis that are plugged into the CICD step, and then there's feature flagging, which offers another step for how to do a release. How does CICD look in the limit? I mean, you work at Google. I'm sure the CICD step at Google is as mature and would like a decade into the future if we were to compare it to probably what's available in the open today, or maybe not. I mean, can you tell me what is the future of the CICD step? How does it mature?

[00:47:10] KH: I think that tools get more mature. Meaning, all the patterns that you hear about or read about about how people are practicing, because it's a practice, CICD, like Canary analysis and feature flags. All of these things that we practice are starting to become software in their own right. That means we have more modules to pull from. So what we're starting to see is

almost this infrastructure standard library. You don't have to invent these things from scratch anymore. There's just enough out there.

The future to me looks like those things will continue to mature. The way I look at CICD, it's a way to serialize your company's culture, right? Ideally, if your company says we have to have manual intervention before we deploy to any environment, then the CICD tool should be able to support that company culture. If your company says, “When we check things in and they get tagged, they go all the way to production after a little bit of integration testing and QA.” Then, again, your CICD systems to support that.

I think the future of CICD is rooted in the practice of CICD. Right now I think there's a false belief that there is a golden way to do CICD. Meaning, I check-in code. It goes all the way to production without any human intervention. I think it’s that thought that makes people hesitant to adapt it, because that sounds scary.

The future is going to be when we start to align the cultural aspects of continuous delivery and improvement into the actual platforms. So it’s going to be the other way around. The tech is going to mature in its own parallel timeline, but I think people themselves, their understanding of delivering software is going to improve the most, and that's where I think you’re going to see the broader adaption.

[00:49:00] JM: There seems to be some difficulty in purchasing decisions these days. If you take a big enterprise, you have individual developers on the ground who want to purchase things to solve problems. Generally, they’re given not exactly carte blanche, but they have a lot of freedom to say, “Hey, I want to buy this thing as long as it's vetted by the security team, or whatever. We can buy it,” but in some cases not.

In other cases you have some central CSO or chief technology officer, whatever, that just can rule from on high and say, “We’re going to buy this,” and everybody in the company is going to implement it. What are some strategies that enterprises can use to make better purchasing decisions? Because you have some companies today that are just really struggling with their cloud bills, like they’re realizing, “Oh my God! We’re spending way too much money on software. We have no idea how these people throughout the organization are using it. We just

know we get this huge bill every month and we have no idea, should we stop paying it? If we stop paying it, what's going to break? It's kind of scary."

[00:50:07] KH: Yes, it's real tough. I mean, the way I think about this problem is I remember in high school there is this thing – Hopefully I wasn't the only one experienced it, the ethical school shopping for your school clothes. The best scenario is you go and you pick out your clothes and then your parents pay for it.

What you described is the enterprise saying, "I know you need school clothes. I'll be back and I'm going to go buy what I think you should wear," and they bring back all these clothes and some things fit, some things don't. Some things are not your style at all, and they say, "You're going to wear these clothes, because I bought them."

Now you're going to end up with a whole bunch of tension because, number one, you spent money on things that I don't want or maybe don't even need and you expect me just to use them anyway. Number one, I'm going to approach this whole thing with a bad attitude. I'm probably not going to wear and just going to go figure out how to trade these things and for something I actually do want, or dire constraints, I'm going to go out and sew my own clothes and take [inaudible 00:51:08] next year because I have to replace this atrocious wardrobe you put together for me.

The enterprise, I think the further away the person who has to use the tool is away from the purchasing decision, you're going to create the same dynamic. I think the key is I think most enterprises have every POC possible. Anything that's for sale, I see a lot of enterprises would just buy everything, and they buy it because they're kind of shooting in the dark and hope one of these things stick, right? They went to a conference. Company a that looks like theirs is using a tool and they're deploying 50 times a day. Let's just copy that.

Their problem with that is they don't understand that there's probably someone behind the curtains at that company that's doing the dedicated hard work. I would say when I worked in the enterprise, the way I tackled this problem – I remember, I was trying to roll out Puppet, and Puppet is a configuration management tool and I really want to replace the way we were deploying software. We were reading these Confluence docs, like a wiki, and we were copying

and pasting commands to deploy applications, and you'd do one server at a time to make sure that you didn't mess things up. If you did, you would try to rewind and cleanup and do it again. I was like, "This is a great case for a configuration management."

So I brought it in and I remember I was very excited because I was able to automate some of these run books, and then people looked at it and it's like, "No. that thing is too new. No one knows what it is. It's probably not secure, something, something, something. We're not doing it."

I thought about it was, it wasn't the fact that Puppet was the right tool that needed to happen. It was that I had to go to the hard work. Part of that work was going around for six months and helping every team outside of my area of responsibility included, package all their applications and RPMs and do it by working side-by-side with the developers and getting their buy-in. Once I was done with that, going through all of the run books and made sure that I had every step perfect and that people who wrote those run books had buy-in.

Two years later, now I'm able to use Puppet, because I got all of the buy-in by doing the work myself, getting others to help recruiting other people, teaching other people and inspiring them into action. No one thinks about that when they purchase software. They're not buying software and saying, "All right, now I'm going to go do the heavy lifting, and two years later we can use that software."

[00:53:37] JM: Let's zoom out as we wind down. Examine the industry in a broader perspective. There are companies that are interesting to me and some ideas that are interesting to me. Heroku is one of these companies. Heroku has been incredibly durable company, and since the rise of Heroku we've seen some next generation platform as a service companies. There's Netlify. Then we've seen the cloud providers offer these better and better managed experiences, but there's still kind of a wide scope of different things that the cloud providers do, and in some sense you want a more narrow experience, the PaaS products.

What is the future of PaaS? What's the next generation of the PaaS companies look like?

[00:54:19] KH: It's not about what you're doing, right? Running a container isn't all that exciting anymore. Herroku sweet spot was given your source code and behind-the-scenes they're

packaging it in some container and running it for you and you get an IP address. For a lot of people, that's all they need and that's perfect. When we start to get into more complex use cases where I want to do a little bit of machine learning, I need to store 50 petabytes of data. Then you start to look at does that platform have a place to store 50 petabytes of data at a competitive cost? Then how easy is it to authenticate to that particular set of data and how fast is that transit going to be to that data?

Then you start adding other things like databases, CDNs, and caches, AutoML, malware I give you a set of photos and you tell me if it's a dog or cat. All of these things become building blocks that when you start to say platform in a future, people are going to look down the list and say, “Hey, how many of these things that you have?” Because if you don't have them, then I'm going to have to go figure out how to get them, either run them myself, build it myself. That's not really – Then the platform seems to have gaps and lacking.

I think we’re well passed, and not for everyone. If all you have is a simple API or maybe a couple webpages, a lot of these things make sense. So these are specialized platforms on the future. I think we’re going to see a lot more specialized platforms, like the Cloudflare workers, where you can run some of your workloads at the edge in the CDN incapacity. So maybe you have a front door to a lot of your logic can live on the edge. Then you’re going to have some platforms that are going to be designed to be optimized for like storing lots of files.

Let’s say you have a photo service. Then you’re going to want to be close to some ObjectStore and you’re going to have all these platforms at your disposal. Guess what? The demands are going to be that they all be easy as Heroku, like Cloud Run, for example, at Google. You give us your container, you can give us a very small YAML file and we’ll deploy that container and we’ll scale it across multiple regions. We deal with load balancing fail over.

For a lot of people, that's a lot of the Heroku feel. If you just want to give a source code, again, you can use one of those function frameworks. But guess what? Users also want global load balancing. They want data stores like Spanner that deal with global data. ObjectStores, like Google Cloud Storage and things like AutoML to tie it all altogether. I think PaaS is just going to be a little bit more demanding because customers want to do way more than they were doing 10 years ago.

[00:57:07] JM: All right, last question. Tell me a subtle aspect of the Kubernetes ecosystem that is easy to miss.

[00:57:14] KH: The Kubernetes API is the most important piece of the Kubernetes ecosystem. That container run time, that container orchestration platform is the first system that was built using the Kubernetes API. If you understand that, you can understand that Kubernetes is moving to a direction. So when the web came out, you have web browsers and web servers and they make HTTP calls, and there are headers and there're bodies, but if you extract out the protocol piece, then you end up with RESTful APIs. Now we can start to build APIs. It's the foundation for GraphQL. It's the foundation for GRPC, because we extracted one of the most important parts of the web, which is HTTP.

If you think about Kubernetes, if you extract the Kubernetes API out from the container orchestration piece, then what you're left with is the API server, things like the RBEC, etcd for storage and now you have a way to describe I think using the Kubernetes style API. So if I want to spin up a database, this is not about running containers.

I can just define a custom resource definition and say, "How many nodes do you want in that database? How much do you want to pay for it? How much storage? Etc. Create that Kubernetes dial config, give it to the Kubernetes API, and then I can build my own control loops that call the necessary cloud provider APIs to give me that database. Keep that database running, and if I want to get fancy, I can add another CRD, custom resource definition, that backs up that database, migrates that database. Once people start to understand Kubernetes as this universal control plane decoupled from running containers, they'll see the true power what this whole Kubernetes thing is about.

[00:59:04] JM: Kelsey Hightower, thanks for coming back on.

[00:59:07] KH: Awesome. Enjoy being here.

[END OF INTERVIEW]

[00:59:17] JM: Cruise is a San Francisco based company building a fully electric, self-driving car service. Building self-driving cars is complex involving problems up and down the stack. From hardware to software, from navigation to computer vision. We are at the beginning of the self-driving car industry and Cruise is a leading company in the space.

Join the team at Cruise by going to getcruise.com/careers. That's G-E-T-C-R-U-I-S-E.com/careers. Cruise is a place where you can build on your existing skills while developing new skills and experiences that are pioneering the future of industry. There are opportunities for backend engineers, frontend developers, machine learning programmers and many more positions.

At Cruise you will be surrounded by talented, driven engineers all while helping make cities safer and cleaner. Apply to work at Cruise by going to getcruise.com/careers. That's getcruise.com/careers.

Thank you to Cruise for being a sponsor of Software Engineering Daily.

[END]