EPISODE 491

[INTRODUCTION]

[0:00:00.5] JM: Docker was released in 2013 and it popularized the use of containers. A container is an abstraction for isolating a well-defined of an operating system. Developers quickly latched on to containers as a way to cut down on the cost of virtual machines as well as isolate code and simplify deployments. Developers began deploying so many containers that they needed a centralized way to manage the containers.

Then came the rise of the container orchestration framework. A container orchestration framework is used to manage operating system resources, and there were a number of options that people started looking to for their container orchestration. One of them was Apache Mesos, which was an open source orchestration framework used by Twitter since 2010. When developers started looking for a way to manage containers, many of them chose Mesos. Some of them also chose Docker swarm. Swarm is a tool for managing containers that came from the same people who originally created Docker.

Shortly after these people were starting to look for container orchestration systems, Kubernetes came out of Google. Google had been using containers internally with their cluster manager borg. Kubernetes is a container orchestration system that was built with the lessons of managing resources at Google. The reason I recount this history as I have in a few past episodes, and I'll continue to in the next couple of weeks, is to underscore that there were a few years where there was a lot of debate around the best container orchestration system to use.

Some people preferred swarm, some preferred Mesos and some preferred Kubernetes, and because of this lack of standardization for several years, the community of developers who were building the infrastructure in this space were put in a tough position. Should you pick a specific orchestration framework and go all in building only tools for that one orchestration framework? Should you try to build tools to be compatible with all three frameworks in attempt to satisfy the Kubernetes users, the Mesos users and the swarm users, or should you just sit out altogether and build nothing and wait for one of these container orchestration systems to win?

There was this fracturing of the community, because there were these different orchestration systems. This fracturing led to healthy debates, but it also slowed down innovation, because different developers were moving in different directions. Today, the container community has centralized. Kubernetes has become the most popular container orchestration framework.

With the community centralizing on Kubernetes, developers are able to comfortably bet big on open source projects that are built around Kubernetes like Istio, Conduit, Rook, Fluentd and Helm and each of these are going to be topics that we're going to cover in the next few weeks, and the centralization on Kubernetes also makes it easier to build enterprise companies, because these enterprise companies are no longer trying to think about which container orchestration system to support. So if you wanted to start a monitoring company tomorrow and you wanted to monitor a container orchestration system, you don't have to choose between monitoring all of the different container orchestration systems. You know that you should focus on Kubernetes customers.

There's also this wide array of Kubernetes as a service providers that are offering a highly available runtime and there's a variety of companies that are offering observability tools to make it easier to debug Kubernetes distributed systems problems. So there's all of these people that are rowing in the same direction for Kubernetes and it makes this a really exciting space to cover right now.

But despite all of these advances, despite everybody focusing on Kubernetes now, Kubernetes is less usable than it should be. So what does that mean? Well, it still feels like operating a distributed system when you're using Kubernetes and hopefully someday operating a Kubernetes cluster won't feel like a complicated distributed system. It'll feel like operating your laptop computer. To get there, we need improvements in Kubernetes usability.

Today's guest, Joe Bed, was one of the original Kubernetes creators and he's a founder of Heptio, which is a company that provides Kubernetes tools and services for enterprises, and I caught up with Joe at Cube-Con 2017, actually Cube-Con Cloud Native Con 2017, and he told me about where Kubernetes is today, where it's going and what he's building it Heptio and also how we can make these things more usable.

Full disclosure, Heptio is a sponsor of Software Engineering Daily, and for the next two weeks we’re going to cover exclusively the world of Kubernetes and the surrounding projects. Kubernetes is a project that's likely to have as much impact as Linux. Whether you're an expert in Kubernetes right now or you're just starting out, we've got lots of episodes to fit your learning curve. We've done a bunch of episodes in the past. You can find those past episodes in our software engineering daily app for iOS or for android, and then if you're an expert, then you'll find the next few weeks of episodes quite interesting, because we’re going to really dive into a lot of the different projects and I hope you like it. Happy 2018, and thanks for listening to Software Engineering Daily.

[SPONSOR MESSAGE]

[0:05:39.3] 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 of container registry, including Azure container registry. Also, quickly and efficiently scale to maximize your resource utilization without having to take your applications off-line. 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. That’s aka.ms/acs, and the link is in the show notes. Thank you to Azure Container Service for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:07:05.5] JM: Joe Beda is the cofounder and CTO of Heptio. Joe, welcome back to Software Engineering Daily.

[0:07:11.2] JB: Thank you for having me. It's exciting to be here again.

[0:07:13.5] JM: Yes. The state of the market, Kubernetes is probably less usable than it should be. This is something that you've touched on in some of your talks. What are the issues that companies have as they're trying to adopt Kubernetes?

[0:07:30.7] JB: I think it's not just Kubernetes. I think the entire cloud native landscape, and I don't know if the viewers have seen that chart that the CNCF puts out of all the different companies, all the different technologies. Yeah, it's called the cloud native computing landscape. That is really fascinating from a sort of innovation point of view. There are so many interesting things going on. Folks are trying out so many different ideas.

From the point of view of a customer, that thing is incredibly intimidating. So there's just so many choices to be made, there are so many different options and I think the biggest problem we have right now is not the problem of getting any one particular technology working at any one particular way. It's figuring out what the right choices are right for you, and the frustrating part is that there is no one right answer, because one of the things that makes this stuff so popular and makes it so successful is its flexibility, but that flexibility comes at the cost of having all those choices that users have to make.

[0:08:31.6] JM: Are these difficult choices that exhibit themselves upon the first stage of deployment? Like when I'm first on day one figuring out how to deploy my Kubernetes cluster or they are more things that are operationally difficult where I have lots of subjective decisions like which service proxy do I choose or which distributed tracing framework do I choose, or is it all of these?

[0:08:54.6] JB: I think the getting started experience, that day one experience that's like, "Hey, let me use Kubernetes." I think we've done a good job of providing a lot of options, whether it'd be mini cube or GKE or AKS or what have you. That makes it super easy to kick the tires of

Kubernetes and start learning about it. I think that's a critical thing just to get folks interested so they start seeing the hint of the value proposition before they invest more time.

I think a lot of the a lot of the frustration and a lot of the confusion comes when you start layering on all the ancillary concepts and systems or when you want to start taking this stuff to production and you start asking the hard questions about, “Okay, is the stuff that I used to get up and running going to be appropriate as I start scaling out for larger and larger projects?”

[0:09:41.8] JM: The choices they are going to make are going to depend on the persona of the Kubernetes user you are. You explored these different personas this morning. If I remember correctly, it was application operators, platform operators, application developers and cluster developers. Those are the four personas?

[0:09:59.0] JB: Yeah. I think I had a 2 x 2 matrix and I think on one edge it's the application, it's the developer versus operator and on the other end it's folks working at the cluster and service level providing platforms versus folks working at the application level. I think that there's a special set of concerns for each of these types of users. Some of these users don't actually applied to the folks getting started, but are critical as we start looking at how do we expand and how do we make sure that we continue to have a vibrant ecosystem. The folks working on cluster and service development in terms of providing platform for other folks to build on top of, that's generally a pretty advanced persona. Those are the folks who are looking to use this world to essentially create leverage for other developers. That's usually not something that somebody starts with.

I think most folks start with that application operator application developer. I have my app that's pre-existing or maybe I'm thinking about writing a new app. How do I use a cluster? A lot of folks will also then also say, “How do I administer a cluster?” which puts him a little bit in that cluster operations role then also.

[0:11:06.0] JM: For people who are a little confused maybe, let’s talk about those personas a little bit more. Application operator, this is like I run a bank and Kubernetes came out and I decide to migrate my bank to Kubernetes. I am an application operator. Is that correct?

[0:11:26.5] JB: Yeah. The developer is writing code. The operator is figuring out how do I actually deliver stability and process and predictability about how that code runs in production? I think in some ways this is a little bit of a new concept for folks, because pre-Kubernetes, pre-containers, a lot of times we have this idea of dev ops, and sort of the body tools for dev ops would be traditional configuration management, like Puppet, or Chef, or Salt, or Ansible. In those, really, you have somebody who's worrying about not just deploying your application, but also managing the underlying system, and so it ends up being a blended role of systems operation at a single host level but also application deployments.

What we’re seeing with the advent of these cloud native systems is that there's a new line being driven. There's a new level of specialization where you can provide a cluster API that takes care of a lot of those systems operations thing and that can be done by a separate group, a separate team, a separate company, Kubernetes as a service, and then there's the question of how do you actually deploy applications on top of it?

Now a lot of times what we'll see is that the application developer then will also take care of actually there application deployment, but they can do that without having to worry about the nitty-gritty details of managing systems. So we’re redefining some of the lines and creating new alliances between these different roles.

[0:12:55.3] JM: So once I have that's Kubernetes deployment, I've got to the day one experience. Maybe it's on EKS. Maybe I've rolled my own Kubernetes deployment and I am picking between all the difference — Who is the person that’s picking between all the different stuff? Who is the person who's making the decision between the different distributed tracing tools and the different service proxying tools? When you're talking about this like reframing of the different roles in an organization, is there somebody specifically who is doing that? Is it a developer? Is it a person who is classified as operations? Because I've been going to these talks and I see you have to configure so many different tools to get really good observability across a Kubernetes cluster. If you want to get distributed tracing and a good logging framework, a good monitoring framework, metrics with Prometheus or something, who is setting up all these stuff?

[0:13:50.1] JB: That's a really good question and I think the answer is it depends. I think that there are situations where this can be set up in a sort of dial tone, like ambient way across a cluster. So you can may be by a cluster service from the cloud providers or from somebody else and over time we're going to see it's going to be easier and easier to get more of these the systems preinstalled that will deliver value out of the gate across the entire cluster.

[0:14:17.1] JM: Right, because I was, as [inaudible 0:14:18.2] was saying in his AWS talk earlier and like it was just funny because a talk about AWS and they're like, "Okay. So we've got managed services for like half of this and the other half you still have to go and get an open source tool off-the-shelf."

[0:14:31.7] JB: Yeah, but sometimes it's going to be sort of an ambient thing that applies to anything that's running in that cluster. You can do logging at the node level where it's sucking standard in and standard out, I mean standard out and standard error off the container and then putting that into a logging system, or you could run it at the application level where it'd be an application ops type of thing where maybe you write your log files to a known directory and then you're running something like Fluentd not on the node, but inside the pod to be able to collect those logs and actually push them someplace.

There are cases where this is provided as an ambient service across a cluster, in which case it would become the role of the cluster operator and then there are cases where this is something that the application operator would actually play a part of. I think as we start seeing things like service meshes come into play, a lot of times those things are very much — They're deployed in a sort of ambient manner across the cluster as a service that might apply to multiple application teams. But I think there're plenty of places like Prometheus, for example. If you wanted to application observability or application monitoring, you could use a shared Prometheus instance that actually might apply to lots of applications, in which case, that thing would be run by a common team perhaps and it would be part of that cluster operations role or you could run your own personal version of Prometheus just for your application, in which case now that actually becomes a single-purpose thing for that particular application and it starts being part of that application operations role. I think we're still figuring out where do people expect different pieces of this to land.

[SPONSOR MESSAGE]

[0:16:16.5] JM: This episode of Software Engineering Daily is sponsored by Datadog. Datadog integrates seamlessly with more than 200 technologies, including Kubernetes and Docker so you can monitor your entire container cluster in one place. Datadog's new live container view provides insights into your container's health, resource consumption and deployment in real time. Filter to a specific Docker image or drill down by Kubernetes service to get fine-grained visibility into your container infrastructure.

Start monitoring your container workload today with a 14-day free trial and Datadog will send you a free t-shirt. Go to softwareengineeringdaily.com/datadog to try it out. That's softwareengineeringdaily.com/datadog to try it out and get a free t-shirt.

Thank you, Datadog.

[INTERVIEW CONTINUED]

[0:17:14.8] JM: How are you seeing it manifest at the companies that you talked to that are deploying Kubernetes? Do they have somebody that specifically does operational concerns, like the person who is deploying Prometheus and other monitoring and logging tools, the person who is deploying Fluentd or do all the developers just jump between deploying this kind of operational logging and metrics and stuff and building an application?

[0:17:41.9] JB: Honestly, it's been it's been all over the map, and I think we saw this play out in some of the keynotes over this morning. We had Kelsey get up at first, and he was like, "Hey, use something like GKE. Let's deploy a lot of clusters that are single application clusters. Why not GKEs being managed for you?"

In that case, if there's some sort of service that you need that's not provided, like GKE, like let's say application monitoring, you can use the hosted service from Google or you could run your own version of something like Prometheus. It won't be a shared service because you have small clusters, and then you had Clayton from Red Hat talking about how Red Hat runs a very large hosted open shift installation where folks can come up and actually get a namespace that they can operate in. That's a situation where you have a very large shared cluster across lots of users.

So I think there's this tension in there, there's a disagreement right now in terms of where the best practices are whether you should have lots of small clusters which are single-purpose and you start treating clusters as cattle, versus do you have a larger and larger clusters and we build more and more multi-tenancy into those clusters such that you can actually have a single well-managed cluster with a lot of services that can be shared between a lot of different application groups, and different people bring different assumptions to that and we've seen that play out in terms of what enterprises are looking for.

I was talking to one company at Reinvent. They have 200 clusters, because across for every team they have a dev staging and prod. You start multiplying that out, you can very quickly have a lot of clusters and then you have other companies, and this is really the sort of Google Borg model where you have much larger, chunkier, multitenant-ish type of clusters that gets shared between a lot of different groups. Then there's questions of how hard is your multi-tenancy? Are you in a high trust environment where you kind of let these things see each other and interact, or are you in a low trust environment where you really want to create very strong walls between the different tenants?

This is part of that sort of incredible adaptability that Kubernetes has. That also creates a lot of confusion, because which of those is right for you? I don't know. It really depends on your situation and your needs and how you're viewing the operation’s tasks.

[0:19:57.5] JM: And it's so confusing at this point that we don't have like patterns to describe it.

[0:20:01.6] JB: I think we’re starting to see the patterns, but which of those patterns? I don't think we've done a good job of codifying those patterns, they haven’ been described well. We don't have a decision matrix actually sort of like, “Oh, in this case, this makes sense for you. In that case, that makes sense for you.” We don't have any of that quite yet.

[0:20:16.5] JM: Okay, interesting. You are running Heptio and Heptio has got several different projects, and the projects are aimed at improving the usability of Kubernetes right now. One of the issues in the Kubernetes usability conversation is management of configuration. So in Kubernetes management of config is done with YAML files, and you were talking earlier about a project called ksonnet, which is as I understood a higher level configuration system. So basically it's a more usable way that you write your configuration, ksonnet, and it gets compiled to YAML?

[0:21:06.1] JB: Yeah, it gets translated into the core sort of API primitives. You can render it out as YAML. I didn't demo that this morning, but it's one of the functions of ksonnet. Yeah.

[0:21:13.9] JM: Okay. Describe the motivation for that.

[0:21:16.5] JB: So I think a lot of times we see users start out, and part of that day one quick introduction to Kubernetes, they'll use something like cube control run and they'll pass in a Docker image and then 30 seconds later they'll have something up and running, and that's a great experience. I think we all aim to get folks hooked on technology by giving them that quick dopamine hit of like, "Wow! Look at me. I'm a hero. I did something great."

The problem that we have in the Kubernetes community is that, fundamentally, that cube control, like imperative model of running stuff is a dead-end. There's features that are not available there and it becomes a nightmare to manage over the long term, because your state — The true sort of like place of record for what you want to be running in your cluster is your cluster itself. It's not repeatable. It's hard to actually manage that across a bunch of different environments and clusters.

The next transition is to actually start managing YAML files and that transition from cube control run, look at how easy this is, to YAML, is a very painful transition. We don't necessarily have the right documentation for the YAML. We don't have all the right examples. That format was written from the get go to be an API sort of machine-readable type of thing. Yet folks are still authoring it by hand.

The next step that folks take is they generally start doing some sort of automation and templating on top of that. You do some [inaudible 0:22:42.6] and AUC and make files and bash and you sort of create this own sort of like pile of automation for doing some specialization for generating this stuff, and eventually they spent a lot of time managing their configuration instead of actually solving their problems. What we wanted to do with ksonnet was provide an experience that feels almost like that cube control run, but yet lead you down a path that is long-term going to be maintainable.

So one of the ideas here, and this is something that Alexis Richardson from Weaveworks has been talking about a lot, is he coined the term or at least he's using the term git ops, right? The idea here is that we move from being the place of record being this ephemeral-ish state that might be sitting in your cluster to being something that sitting in source control that you can code or view that you can manage as code. Part of this sort of configuration is code type of thing, and then you can have automated systems that trigger off of that to do the deployments and to drive that stuff. We wanted to do something that felt as easy as cube control run, but also lead you down a path where you can start doing some of this git ops stuff.

[0:23:48.3] JM: Can you describe like what these files look like? What is a ksonnet file look like? How does it differ from YAML file?

[0:23:54.8] JB: Ksonnet is built on top of this language called Jsonnet, and Jsonnet borrows a bunch of ideas from an internal language inside of Google called BCL, board configuration language, and that languages is — It's frankly controversial inside of Google. It's worked well for Google for 10 years. It's not perfect, but my take on it is that like if we can get €10 out of a technology, that's not bad. So I think there's a lot of good ideas —

[0:24:21.9] JM: Most of the controversy?

[0:24:23.2] JB: It comes around with what I would say when I look at it — And it is one of those things where I think a lot of folks have a visceral feel for it, and it's hard to actually put it into words what the problem is. In my mind, it's a relatively flexible tool, but there's not a lot of structure for how you use it. I think there's an analogy between like, say, JavaScript in say 2000 versus JavaScript today. It's fundamentally the same language, but JavaScript today, because we have tools, because we have patterns, because we have frameworks, we've taken that unstructured raw language and we've wrestled it into shape with the stack of tools that we have applied to it.

The other week — This was for the Kubernetes up and running book that Kelsey and Brandon and I wrote. I wrote an example web server for it that you could use it, called [inaudible 0:25:15.4], and it ended up being a React app and it was my first time doing JavaScript development from when I was on the IE team back in like, I don't know, year 2000. It’s fascinating to me to see that transition of how far JavaScript has come, and I think BCL inside of Google is as unstructured in some ways as JavaScript was back in 2000.

One of the things that I think we can do with ksonnet is that we can we can take something that is unstructured, but we can establish some good patterns, some good tools, some good best practices for how you actually use that and then help it scale to solve some of those sort of visceral problems and sort of spaghetti code type of things that folks saw happen with BCL over time. That’s a fairly long analogy. Hopefully that makes sense.

[0:26:03.0] JM: Okay. If I understand the analogy correctly, you're comparing it to JavaScript in the sense that JavaScript 10 years ago, maybe even five years ago, kind of spaghetti-ish less best practices that have been standardized, and today JavaScript is still loose, but the way that people should code JavaScript is more well-understood. So if somebody makes a poll request to your JavaScript repo, you have a better understanding of when you can reject it and say, “Hey, this is bad formatting,” or whatever, bad code smells and whatnot. Similarly, with ksonnet, you're saying that today, if you're just describing your configuration to your Kubernetes cluster with YAML, a lot of different subjective decisions that you can be making, and with ksonnet you have an opportunity to build a more of a standardized system, more —

[0:26:54.9] JB: Yeah, sort of establish some pattern, some tools that really help you scale this stuff out in a way that can be shared between multiple folks.

[0:27:02.9] JM: Got it.

[0:27:03.0] JB: It's kind of a tortured analogy.

[0:27:05.3] JM: Yeah. No, I understand. So another project that you started at Heptio is Sonobuoy, which it runs a set of Kubernetes tests to generate a report of a Kubernetes deployment. This is a general diagnostic health test that can help expose bugs that are hard to find. How hard is it to debug problems in a Kubernetes deployment, like subtle health problems?

[0:27:30.3] JB: It can be difficult, and I think there's a lot of things that can go wrong and some of this comes down to the diversity of ways that you can install Kubernetes. It's easy to get a cluster that looks at first blush like it's a well operating cluster, but there might be some subtle misconfigurations that are hard to tease out.

As a company, as we look to how do we want to support Kuberentes, because one of our things is that we want to help folks be successful with Kubernetes regardless of where they're running it. We wanted to have a way to qualify a cluster and say, "Hey, is this cluster in good shape?" We built Sonobuoy as a way where you can drop this in there. It can run a bunch of diagnostics and then we can look at that to say, "Hey, is this thing configured in a sane, workable way?"

The first thing we did with that though was run some of the standard end-to-end conformance tests that Kubernetes has been developing over time. We patched it up and made those things easy to run, and so now as the CNCF works on the Kubernetes certified cluster or what is — I'm probably getting the name wrong, but there are certification of Kubernetes clusters. That program is built on the work that we did with Sonobuoy. So pretty much everyone of those clusters ran Sonobuoy to collect a report. That became proof that, "Hey, I have a certified cluster," and over time we're going to be extending Sonobuoy to not just do this sort of official standard conformance tests, but also do some more extended tests that we can help folks spot problems before they become critical for them.

[0:29:03.0] JM: Kubernetes is used for very critical infrastructure, and when the Kubernetes cluster goes down, it can be a disaster, and Kubernetes is a system that builds in some fault tolerance, but you might still want to configure it across multiple availability zones or multiple clouds to be truly resilient. What are the things that can lead to a disaster in a Kubernetes cluster?

[0:29:30.7] JB: Well, there's two things that I want to talk about there. I think there is — Again, this is another one of those that it depends. Sometimes it makes sense to run a larger cluster that may spread across multiple availability zones and other times it may make sense to run multiple clusters and then coordinate what software is running on each of those clusters individually. There are costs and challenges and advantages to each of those.

One of the things that I think one of those decisions that users have to make is what is the trade-offs that I want to make in terms of efficiency, ease-of-use of having one larger cluster perhaps versus some of the availability upsides I can get from running more isolated and independent clusters, because if a cluster fails, what's the blast radius. How much does it take down? If you're running a cluster per zone, for instance, and one of those clusters fails, you actually take down the other zones. If you're running a cluster across zones and that cluster fails, then you essentially have taken out your whole region. So that's a trade-off that we want users to think carefully about also.

In terms of what can take down a cluster, the most common thing would be misconfiguration, and I think across any sort of system that folks are operating, fat fingering an administration command, getting a decimal point wrong or something like that in the configuration file, those are the types of things that are going to dominate any of your uptime metrics. I think a lot of folks think about hardware failure, but you’re going to see 10 times as many software configuration failures due to administration than you will see hardware failures. That's going to dominate everything that you're going to see.

In terms of the types of things that can actually take down a cluster, some of these would be — In some ways it's hard to take down a cluster. The biggest thing that will take down a cluster will be your backing database for Kubernetes hitting issues. This would be etcd, that will lead to the worst behavior. You generally only see some of that stuff at scale. It's much, much better with etcd 3. It scales a lot better, but that's where you're going to see problems, and there certain types of things that you can do that will overload etcd more than others.

I think this is something that's actually interesting as people start building extended systems on top of Kubernetes, and it's instructive to look at like open shift is built on top of sort of the Kubernetes base components. It's a little bit of a specialization, because they've grafted a bunch of open shift specific code in top of it, but the way that open shift works is that they put up a whole lot more stuff in etcd than your typical Kubernetes cluster. So a lot of the scaling issues that you can hit with etcd, a lot of the stability issues, the Red Hat folks with open shift have actually blazed that trail, because they run, say, CICD pipelines where they put the results from every run into etcd itself. So that actually ends up stressing out etcd quite a bit.

[0:32:25.5] JM: When you look at that design — When I look at the Kubernetes design, one thing I find interesting is that etcd is really important, but it's actually pretty loosely coupled from the Kubernetes system, right?

[0:32:40.0] JB: I think theoretically, over time, we might have the ability to swap out etcd and put something else in there. That's not a priority for anybody in the community right now. It'd be great if we could do that. It's just a matter of doing the work and testing it and nobody has stepped up to do that. I think we'd like to see that happen eventually. It's just not something that's there right now. So it's more tightly coupled than we'd like right now I think I would say.

[0:33:03.1] JM: Okay. So when you look at the design decisions of an open shift, which is a platform built on top of Kubernetes, and you see them building more rich functionality into the interaction between etcd and the rest of the platform to have a more — I guess a more highly available cluster out of the box basically?

[0:33:22.5] JB: They want to provide more extended capabilities and they're re-leveraging that etcd to do things to offer new capabilities on top of Kubernetes. They're using that same etcd not just for the core Kubernetes stuff, but also for, say, like I said, CICD pipelines that might be running as part of open shift.

I think this relates back to the extensibility primitives that Kubernetes has. As you start extending Kubernetes, so we have these things called custom resource definition, and it's a way that you can create new types of things, new types of nouns in the Kubernetes world. Incredibly powerful, because you can use the same Kubernetes patterns to say, "Hey, create me a database by writing a Kubernetes object," and that's a custom resource definition, and then you can have a piece of code that goes off and creates that database for you. That's a really interesting capability to be able to reuse the Kubernetes machinery to extend it, but as you do that, if you're not careful, you can start overloading some of the capabilities of Kubernetes itself.

One of the things that were going to see over time is what are the best practices for extending Kubernetes? How do we make sure that you don't overload the Kubernetes cluster by piling too much new, good, interesting stuff on top of it and how do we start creating sort of isolation so that those things can run truly on top of it? That's something that as we create more and more extensibility points, it's going to be interesting to see how do we do that in a way where we don't overload the core functionality, the core parts of Kubernetes.

[SPONSOR MESSAGE]

[0:35:05.3] JM: The octopus, a sea creature known for its intelligence and flexibility. Octopus Deploy, a friendly deployment automation tool for deploying applications like .NET apps, Java apps and more. Ask any developer and they'll tell you that it's never fun pushing code at 5 p.m. on a Friday and then crossing your fingers hoping for the best. We've all been there. We've all done that, and that's where Octopus Deploy comes into the picture.

Octopus Deploy is a friendly deployment automation tool taking over where your build or CI server ends. Use Octopus to promote releases on prem or to the cloud. Octopus integrates with your existing build pipeline, TFS and VSTS, Bamboo, Team City and Jenkins. It integrates with AWS, Azure and on-prem environments. You can reliably and repeatedly deploy your .NET and Java apps and more. If you can package it, Octopus can deploy it.

It's quick and easy to install and you can just go to octopus.com to trial Octopus free for 45 days. That's octopus.com, O-C-T-O-P-U-S.com.

[INTERVIEW CONTINUED]

[0:36:38.2] JM: Are you describing the fact that if you were running open shift, that open shift is running on top of Kubernetes cluster that has become less flexible in a sense?

[0:36:51.1] JB: I'm using open shift as an example. I don't overplay that. They're doing a lot of great work. I guess what I'm trying to say is that Kubernetes has a core set of concepts, pods, services, replica sets, deployments, that type of stuff and it has a bunch of code that deals with those things. Every time you deal with one of those things, it's going to hit interact with etcd through the API server, okay?

Open shift extends that with things like build and other types of templates and stuff like that. So they have a lot more sort of nouns and primitives and resources that they've actually extended Kubernetes to have those things with, and in doing so it puts more stress on that fundamental on the etcd database, because they're doing more with the same database than what you'll see in a typical Kubernetes cluster. That's not just isolated to Red Hat and open shift. We're going to see those same problems crop up with things like custom resource definitions and operators, and so that's one of those best practices and one of those things that over time we're going to have to figure out what's the best way to actually scale a cluster, add capabilities without putting all your eggs in one etcd basket.

[0:38:03.6] JM: When you let the platforms that people can build on top of Kubernetes, are you thinking like somebody who would build a serverless platform on Kubernetes?

[0:38:14.2] JB: There's like multiple. Yeah.

[0:38:15.2] JM: Yes, multiple. What are the other ideas? There's the serverless thing you could build. You could build like a Heroku on top of it. You can build an open shift on top of it. Any other categories of platform —

[0:38:27.4] JB: I would say that almost every service that you see from a cloud provider you could build it as an extension to Kubernetes. You could build an RDS on top of Kubernetes. The operator model is this idea that you create new types of resources that can represent higher-level concepts, like a MySQL database or like an etcd cluster or like a Kafka instance or an elastic search instance. You give a very simple description of that in YAML just like you were describing a deployment, and then there's a piece of code that takes that and then renders that down into lower level Kubernetes primitives and then actively manages those.

When you look at something like RDS, and this was something that, Craig, my cofounder, brought up during the keynote. Any sort of star as a service is you take a piece of technology, you create a provisioning system on top of it and then you provide an ops team. That ops team more and more over time becomes more and more automated. That's the only way you can scale a cloud. RDS, when we look at something like Amazon, is it's a technology, the database. You have a provisioning system. Right now that's the AWS APIs around RDS, and then there's an ops team with a highly leveraged set of automated tools to be able to deal with that.

There is no reason why we can't see that same thing repeat on top of Kubernetes. Over time, Kubernetes becomes the common substrate, the common framework for doing any type of managed service, and so it could be RDS, it could be Kafka, it could be serverless Lambda type a thing. All of those things make sense to think about how can we offer those higher-level values on top of something like Kubernetes.

[0:40:11.4] JM: Oh, that's a huge vision.

[0:40:13.4] JB: Yeah. So when I say that we're really at the start of people building, that's the stuff that I'm really thinking about there.

[0:40:18.6] JM: Yeah. It's an incredible vision. Okay.

[0:40:21.7] JB: We're going off topic.

[0:40:22.5] JM: No. Yeah, a little bit. I mean we'll have that interview of the Kubernetes open source AWS thing in five years or 10 years.

[0:40:32.1] JB: We'll see how dead on I was.

[0:40:33.2] JM: We talked about disasters a little bit I wanted to use that as a jumping off point to another thing that you worked on at Heptio, which is Heptio Ark, and this allows for disaster recovery of a Kubernetes cluster. So given that we talked about disasters earlier, can you describe some of the flavors of disaster that Heptio Ark helps you recover from and how it does that?

[0:40:55.3] JB: Sure. Heptio Ark came out of one of our first customers. They were asking us, they were running Kubernetes on top of AWS and they were asking about how they could put tags on the EBS volumes that Kubernetes was creating, and I'm like, "Why would you want to do that? Why do you care about the tags?" They're like, "Well, we have the system that automatically backs up our EBS volumes for us." I'm like, "Ha! Nobody's thinking about disaster recovery for Kubernetes." I think there's an opportunity there.

What Ark does is it works within Kubernetes to capture the state of your API, so all that configuration of what's running, what should be running. It snapshots that or least tries as much as possible to snapshot it and will work overtime perhaps to provide ways to get true snapshots out of that, but it gets a snapshot of those objects and then it also coordinates behind-the-scenes, getting snapshots of your underlying block devices, like EBS volumes. It will capture both the state of the cluster along with driving point in time snapshots of those underlying data volumes that your workloads are running with, and then it package that up in a way so that you can name those things and then restore them.

Some of the flexibility that we're working on here and that we want to develop in the future is being able to save these things across the entire cluster, maybe only a part of the cluster or certain set of name spaces. Being able to take these snapshots and move them maybe between regions within the same cloud provider or even across different cloud provider. It's a way to take your states, make sure that you don't lose anything along the way, but also provides a degree of flexibility in terms of being able to use the promise of Kubernetes running in a consistent way and a bunch of different places to be able to literally move those workloads around.

[0:42:43.4] JM: Wow! So snapshotting Kubernetes or snapshotting a state of Kubernetes, does that include — Like do I need to snapshot all of the discs that are mounted to Kubernetes and all of the databases that are mounted to Kubernetes and I need to do all of that at once?

[0:43:03.3] JB: That's what it tries to do. I think we don't have all the primitives to get a true point in time snapshot yet. It's something that we'll be working on, but it's both the configuration state that's embedded in the Kubernetes cluster, essentially the data in etcd, but we don't want to take the etcd stuff directly because a lot of times you don't have access to the etcd database, and restoring that can be fraught.

It's taking all the data that's stored in etcd along with trying to take all of the application volumes that are attached to that cluster also and driving a unified idea of a snapshot across all of those or a backup across all of those.

[0:43:40.7] JM: Wow! This could be like a tool where you could just have a lift and shift of your —

[0:43:45.5] JB: Yeah, lift and shift or disaster recovery, or even in a test environment, being able to take an environment and then duplicate it into, as you want to say, "Hey clone this namespace into this other place."

There's a bunch of places we're having this this flexibility, this immutability will really play out. We're still pretty early on with that but I think we've started seeing a lot of excitement around it and one of the things that we've been talking about here at Cube-Con is that we've been working with Microsoft to make sure that Ark works really well with is Azure.

[0:44:17.1] JM: Fascinating. I want to talk some about service proxy and service mesh. So we've done a number of shows about Envoy and Istio and LinkerD. I'm sure we'll do something about Conduit in the future, but basically the model for this service proxy and service mesh thing is on each of your pods in Kubernetes you might stand up a sidecar container that sits alongside your application container, and the envoy service proxy is something that all the communications between services get shuttled between. So if you've got an application container sitting in your pod, it's going to hit your envoy proxy in the same pod before it hits another envoy proxy in another pod which will communicate with another application container. So all the communications between services are proxied through Envoy proxies, and then the Envoy sidecar also can aggregate information back to a centralized service mash.

First of all, did I get any of that wrong?

[0:45:19.7] JB: Yeah, that seems pretty right. Yeah, I think in my mind, and I'm not the expert here, but I think it's worthwhile to call out what are the benefits that users see when they deploy these things. I might be missing something. I mean this is not the stuff that I'm living and breathing every day right now. I think one of the big things that you get out of this is security, those tunnels between the different envoys can be secure and that's taking care of for you. So to the workload, it looks like they're talking plain old TCP or GRPC or HDP between each other, but under the covers, those connections between those things are authenticated, encrypted and there's an opportunity to depending on how deep you look inside of that, those packets to start doing certain levels of authorization. Who can actually call what, when and where, make sure that you know who's actually doing the calling. That security overlay is really important to folks.

The next thing is observability, right? If you deploy one of these meshes, it has some level of insight in terms of the protocols that you're running. It can attract metrics for you without having to change your application at all, and then expose those metrics so that you can aggregate them and actually get some really good insight in terms of what's happening inside of your cluster.

The next thing is routing. There's this idea that, “Hey, I'm service A. I want to talk to service B,” but maybe for like 10% of my traffic I want to talk to B prime, or maybe if there's this bit set on this particular request, I wanted to take a right turn and go to Joe's version of B. That dynamic per request routing ends up being a really interesting feature of these service meshes that can be used for all sorts of different things.

Then finally, just in general, there is this idea of policy where we have these new places where we can put all sorts of decision-making in a unified way that we haven't been able to before. That's a lot to take on. It's a lot of value being delivered, but we’re still pretty early in terms of folks adopting an understanding and wrapping their heads around everything a service mesh can do.

[0:47:26.2] JM: Yes, and another project within Heptio is Heptio Contour, which lets Envoy be used as yet another thing as a load balancer. Are we bundling too much technology into Envoy? Why wouldn't we just have more sidecars, or does it make sense just to have everything in this one —

[0:47:45.4] JB: I would not put contour into the same bucket with the rest of these service meshes. One of the other things that Istio does is it does ingress. It interfaces the service mesh to stuff that's outside of the service mesh, because you have this mesh. The question is like, “Okay, that's great. Everything can talk to each other inside the mesh. How do you actually bridge that to the outside world?”

Istio has its own idea of ingress and egress from the mesh. That being said, I think that envoy is a really flexible, really interesting piece of technology that has value both inside of a service mash, but also as a regular old load balancer. We’re using it as a regular old load balancer so that you can — And it's well suited to that in a Kubernetes world, because it's fundamentally built to be API-driven. That's the way that it was built and used at Lyft both as a mesh and also as an incoming HTTP load balancer. So we’re using it that way.

[0:48:42.1] JM: Yeah.

[0:48:42.4] JB: It's a really good fit and we’re trying to make it work well with the existing patterns inside of Kubernetes. So you can use Contour to do load-balancing for stuff coming into your cluster even if you don't care what a service mesh is, right?

Then over time, there's a lot of opportunities to make it more flexible so that it can actually start bridging, not just load-balancing inside of a single cluster or perhaps across clusters or perhaps things that are on cluster and off cluster and that some of the stuff that we’re looking at here, is how can we actually start viewing that software-based load-balancing tier as a thing in and of itself outside of the service mesh outside of the cluster. That's a little bit further ahead.

[0:49:22.6] JM: Okay. All right. Well, because I didn’t really understand the last part.

[0:49:26.8] JB: Okay. Right now the idea is that you have the cluster. It's a thing. Data comes into the cluster. You need a way to actually find the right services for that data coming into the cluster. But a lot of times when folks have more complicated set ups or they’re in the position where they’re transitioning to a cluster, they want to have traffic come in to their data center and to their production environment and they want some of it going to cluster A. They want some of it going to cluster B. They want some of it going to the set of virtual machines that aren't on a Kubernetes cluster at all. When I say, “How do we start thinking about load-balancing as a general problem outside of just Kubernetes? That's I think where there's going to be some interesting work going on in the future.

[0:50:08.3] JM: Can Envoy currently route stuff to stuff that is not on Kubernetes?

[0:50:15.3] JB: Oh, yeah. In fact, Lyft developed Envoy without a Kubernetes cluster in sight. These idea, they work well together, but you don't have to use them together. You can do micro services without Kubernetes, right? Of course, right? You can do service meshes without Kubernetes. There's a whole bunch of independent innovation going on, but we're finding that these things do work well together.

[0:50:40.5] JM: So I was watching some of your videos. You got some pretty interesting video series. I like the Friday, like you sit down in front of your computer and just code on Kubernetes for an hour, like that actually pretty useful. I forgot the name of that.

[0:50:57.7] JB: TGI Kubernetes.

[0:50:58.8] JM: TGI Kubernetes. Okay, yeah.

[0:51:00.0] JB: It's on the Heptio channel on YouTube.

[0:51:01.3] JM: Right. I enjoy those. There was one of them where you were talking about — I think you are talking about like applications of sidecar. So we just talked about one application of sidecar, which is like, "Yeah, deploy and Envoy sidecar to all of your different service pods." I heard you give another example of like when you could use sidecars in the same pod to coordinate updating a search index, like a search index at Google.

[0:51:25.3] JB: Yeah, this is the classic use of Google.

[0:51:26.6] JM: Give that example. It's really interesting.

[0:51:28.4] JB: Generally, like when you're doing search, you have the thing that's evaluating your index, and that your serving job. Then you have the thing that's updating the index, and if you look at more traditional things like Elasticsearch, that ends up being the same thing is both serving the index and actually ingesting new data. It’s a very dynamic system. That's not always the case.

Sometimes you have a separate, sort of like maybe you have a Hadoop cluster that's building a static index and then you want to deploy that. This could also apply in the machine learning space. You have something serving up a machine learning model, which is really a bunch of data files that are sitting on disk that you load up. Okay, now you have something that you've built a new model that's new and improved. How do you deploy that?

You could have the same thing both taking on the task of serving the thing, but also fetching and validating and preparing the next model that you want to roll over to. You could instead ordinate these things by having a data loader container that's running in the same pod as your data serving container and have those things coordinate either through maybe a shared memory section or through a bunch of files on disk, so the loader loads some stuff, it puts it on local disk, and then once in a while the server says, “Hey look, I have a new thing on disk. Let me start loading that from disk in the background.”

So you've decoupled sort of the tricky issue of how do I actually figure out what the next thing to load is from the thing that's actually reading that thing up and running. T that's a pattern that we’ll see happen at Google quite a bit.

[0:53:06.6] JM: Are there more patterns of when and how to use sidecars, or is this one of these things that is not yet been very standardized, but it's definitely useful?

[0:53:15.1] JB: Yeah. I think logging is another example. So we talked about sometimes logging is a service provided by the cluster itself that you write standard out, standard error and then magic happens and it shows up in some sort of logging system, or you could run something like Fluent D, for example, as a sidecar in your pod, create some connection between your workload in Fluentd. Sometimes you drop some files on disk and then Fluentd will tail those things and then upload them someplace. So that's two things, running a sidecar, coordinating over some Linux primitive, like say a disk volume.

Other places — One of the patterns that we saw at Google is that as you become more and more of a polyglot world where you're dealing with lots of different languages, you and up with a library problem where I want to do something cool, but I want to do it across N-different languages. So now I find I have to maintain N-different libraries to be able to do that. That's really painful. There's a whole bunch of places where you see this stuff happening. I think logging would be one of those. You could imagine instead of having Fluentd, you have a logging library that you could link in to N-different languages that could do it directly, but instead we've taken that thing that logically might be a library and we've broken it out into a sidecar.

Any time where you have a library that's difficult to integrate, there might be a possibility to break that out into a helper sidecar, and that's eventually essentially where service meshes actually came in from the Google's point of view. It used to be that there were a set of standard libraries that you would link into every Google job. Over time, those became difficult to maintain and there were a whole bunch of efforts internally to start extracting those into essentially what we call sidecars outside of Google.

[0:55:01.8] JM: Does that ever become a resource consumption problem when you just start to stuff a bunch of sidecars over the same —

[0:55:09.0] JB: Yes.

[0:55:09.6] JM: Okay.

[0:55:11.4] JB: But I think one of the things that you'll see, like this was in that the keno yesterday, was the Fluentd folks started talking about how they did an alternative implementation. I don't know if they'd considered a rewrite of Fluentd called Fluentbit, which is built to be run as a sidecar. Has a lower memory footprint. Similarly, the Linkerd folks have done their own reimplementation of Linkerd, or at least the parts of Linkerd that are critical for service mesh in Rust and they've gone that down to, say, like — I don't know, like a 1 megabyte footprint or something like that for a very simple proxy in their service mesh as part of conduit.

I think adapting systems that weren't built to be run at sidecars tend to blow things up, but over time, as those things become popular, people are taking a very critical eye to make sure that they don't actually become too much of a resource hog.

[0:56:03.3] JM: Yeah. But in any case, when you get a bunch of different containers in the same pod, there is this question of prioritization, like resource scheduling. Like which of the containers do you prioritize resources for? Are we still in the early days of how those different containers get prioritized?

[0:56:24.8] JB: Yeah, I think so. I'm not an expert here. I think the part of Kubernetes, this would be the sig-node stuff. There's a lot of really interesting stuff going on there. What I can say though is that the resource model for Kubernetes is really one of you can set for in a per container basis. You can set both request and a limit, and so the request is, "Here's how much of a resource I need." The limit is, "Well, if it's available. let me use up to X."

What you can do, generally, is that if you have something that's latency critical for serving, you make sure that the limit and that the request end up being the same. So that means that you don't get any surprises. You always get the same amount of resources, but if you have something that's more as catch as catch can type a stuff where you have that flexibility, you might say, "Hey, I need at least X amount of CPU, but I'm willing to take as much CPU as you can give me."

There are some flexibility in terms of how you specify what your resource requirements are. Over time, I think there's room to make that stuff even more sophisticated, but I'm not totally right in on what the sort of latest and greatest thinking is there.

[0:57:32.4] JM: Okay. Understood. You wrote a book recently, Kubernetes Up and Running. Did you learn to explain Kubernetes better when writing that book?

[0:57:43.0] JB: I did.

[0:57:44.8] JM: What are people confused about that you learned explain better?

[0:57:47.5] JB: I think that the chapter that I think is really — And I wrote this one, and writing a book with a couple co-authors is fascinating, because it becomes a game of chicken of like assigning chapters and see who writes it first. That was an interesting experience in and of itself. The chapter that I actually enjoyed writing the most was the one on services and service discovery, and the first time I wrote that, I took a very principled, like build it up approach, "Hey, service is really this, but then let's layer on this capability and let's layer on this other capability."

It's a fairly complex topic inside of Kubernetes, and one of the feedback that I got from some of the early readers is that it was the wrong order to do that stuff, really. I started with a very concept first mentality, but I ended up reordering that entire chapter, so I started with like, "Here's how it works. Well, now let's dig a little deeper. Here's how works. Now let's dig a little deeper." Really, sort of use case first type of thing instead of a concept first type of thing. I think that chapter turned out pretty well in terms of explaining the different sort of facets of how services work in Kubernetes.

[0:58:57.1] JM: We last spoke about your time at Google Cloud. that was like a year and half ago. Kubernetes has changed a lot in the last year and half.

[0:59:05.8] JB: Yes.

[0:59:07.9] JM: Do you have any analog to the way that the Kubernetes community is growing or is this a completely new type of ecosystem?

[0:59:18.0] JB: I haven't seen anything like it before. I mean, the fascinating thing for me is that I'm relatively new to open source, believe it or not. I think my career early on was that Microsoft — I tried to be more open and sort of more direct in terms of talking to developers. I was like one of the first interviews on channel 9 when I was at Microsoft back in the day talking about Avalon and Windows presentation foundation and that stuff.

I enjoyed sort of that more outward facing, human face on how to relate to developers similar stuff when I was building up some of the early work around Google cloud and GCE. But moving to the world of not just talking in an open way, but actually being open about the community about the code has been a really, really invigorating experience for me. I'm not sure if I'm the best person to say whether it's truly unique. I can say that it's taken me by surprise. I'm humbled by the community. I think it's amazing. Like a lot of times when people ask me to sign the book, I'll say, "Thank you for being part of the Kubernetes community," because I really do consider all 4,000 people here Cube-Con part of that community and I think it's just really an amazing experience. So I don't know how unique it is. It so hard to judge that stuff from the inside, you get blinded by it, but II don't want to take it for granted for sure.

[1:00:37.2] JM: Okay. We've been talking about Heptio pretty obliquely. Heptio is a company. If you have these open source projects, but you are not just open source projects, you have training. What's the overall vision for — You've raised a lot of money, so obviously there's a big vision, and you've described a very big 5 to 10 year vision of where Kubernetes is going. I find it very interesting where you're starting today with these open source building blocks and some training. Are you just kind of trying to figure out what's going on?

[1:01:07.7] JB: No. We have a plan.

[1:01:09.1] JM: You have a plan. Okay.

[1:01:10.2] JB: They don't write you a check like that without some semblance of a plan. A lot of times when I explain it to people, we're doing three things with Heptio, and I think a lot of times people see one but not the other and they get a little bit confused. I think you know over the next year we're going to do a much better job of balancing how we talk about that, but those three things are helping customers be successful with Kubernetes right now, and that's services, support and training. So if you need somebody to help you get Kubernetes deployed in figure out how it works within your organization, with your applications, come talk to me, we can help you out.

One of the big reasons we're doing that is that we want to have a sort of high touch interaction with customers to understand what are the real problems that they're facing. We don't want to solve the problems that we think people have. We want to solve the problems that we verified that they have. So that's a good place, good way for us to really roll up our sleeves and get in the trenches and understand how are people using the technology today and what are the problems that they're having.

The second thing we're doing is we want to support the open source community, and this includes both working upstream, making sure that we're good citizens in the Kubernetes project itself, but then also creating new projects in the ecosystem, like Ark, and Sonobuoy, and ksonnet, and Contour.

The motivations behind doing that is — Well, number one, our company is intertwined with this community in a really big way, and so we want to make sure that the community in Kubernetes is successful over the long term. As a company, you can't practice just value extraction. You have to give back, and it's just a matter of good business and making sure that you're responsible for helping to have things have longevity.

It's also a good way for us to build credibility with developers, because we're showing up. We want to be doing the work. We want to be contributing, and I think folks notice that when you're actually doing that.

The third thing we're doing is that, and this is the stuff that we haven't been talking about as much, is when you start looking at Kubernetes from being a sort of like small company, dev-ops departmental level project to something that starts stretching across an organization, there is new things you have to do. There're new concerns. There' re new integrations that need to happen. So I think Kubernetes, the open source Kubernetes right now works well when operating at that relatively small scale. There're new challenges when you start looking at it across the organization, and that's where we're going to be focusing a lot of our effort around building product. Can go into too much detail right now, but we think that large companies are going to have unique needs when it comes to Kubernetes, and that's a place where we can differentiate and build some product.

[1:03:54.9] JM: Okay. That doesn't sound like a position where you're going to be competing with the major — Well, I mean you might be competing with major cloud providers, but it's a little bit different —Okay. I guess I don't know what you're building yet, but —

[1:04:09.2] JB: Well, I can tell you that our goal out the gate is I didn't find myself going head-to-head against Amazon or Google or Microsoft or all three. We want to add value regardless of how or where you run Kubernetes, and I think it's served us well so far. So when the Amazon announcement happened last week, number one, I don't think we weren't surprised. I don’t think anybody was surprised. We’ve done some talking to them before, but even when we found out, we’re like, “Okay. That's great,” but it also didn't threaten us from a business point of view. We’re like, “Okay. That's great.” Now, Kubernetes is that much more legitimized and that actually creates more opportunities for us, because the more folks running Kubernetes, the more there are going to be ways for us to add value and create and play into that ecosystem that's building on top of Kubernetes.” So that's what we’re really interested in.

[1:05:03.3] JM: I won’t keep you much longer, because I know you got to go.

[1:05:05.3] JB: No worries.

[1:05:05.3] JM: I was just at SpringOne Platform, which is like a Pivotal conference. One thing I learned at that conference that I didn't know was like how Cloud Foundry makes Pivotal a lot of money, and the way that it works is, if I understand correctly, is if you are a — Let’s say you’re a bank and you want to deploy to Cloud Foundry, and let's say you deploy on Amazon or Azure or Google, regardless of where you deploy that Cloud Foundry, you’re usually getting — I don’t know about usually, but you're often getting support from Pivotal.

I found it interesting, because this is a business model where Pivotal has managed to have a platform where they can make money off of who — Any cloud provider that's deployed. Do you think there's room for like a Kubernetes player like that?

[1:05:59.5] JB: I'm not sure that that same business model will work for Kubernetes. I think what we've seen is that because Kubernetes has such a vibrant, open, multivendor, multi-company community, there is no one dominant player in the space that can monetize like that.

I'm not super embedded into the Spring or the Cloud Foundry world, but it's very clear that Pivotal is the is the main driving force across all of those. Whereas I think if you look at the CNCF, if you look at Kubernetes, it’s much more evenly spread across so many different companies.

[1:06:36.6] JM: Okay. All right. Well, Joe Beda, thanks for going back on Software Engineering Daily.

[1:06:39.7] JB: Thank you so much for having me. It's always a pleasure.

[1:06:42.1] JM: Okay! Great.

[END OF INTERVIEW]

[1:06:45.2] JM: If you are building a product for software engineers or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an email, jeff@softwareengineeringdaily.com if you're interested. With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers.

I know that the listeners of Software Engineering Daily are great engineers because I talked to them all the time. I hear from CTOs, CEOs, directors of engineering who listen to the show regularly. I also hear about many newer hungry software engineers who are looking to level up quickly and prove themselves, and to find out more about sponsoring the show, you can send me an email or tell your marketing director to send me an email, jeff@softwareengineering.com.

If you're listening to the show, thank you so much for supporting it through your audienceship. That is quite enough, but if you're interested in taking your support of the show to the next level, then look at sponsoring the show through your company. So send me an email at jeff@softwareengineeringdaily.com.

Thank you

[END]