EPISODE 497

[INTRODUCTION]

[0:00:00.3] JM: In the last four years, CoreOS has been at the center of enterprise adoption of containers, and during that time, Brian Harrington, otherwise known as Red Beard, has seen a lot of deployments. In this episode, Brian discusses the patterns that he is seen among successful Kubernetes deployments and the pitfalls of the less successful. How should you manage configuration? How can you avoid IP address overlap between containers? How should you laud and monitor your Kubernetes cluster and whose responsibility is it to set all that stuff up?

Brian also discusses the motivation for multi-cloud deployments and how to implement multi-cloud Kubernetes. CoreOS offers a distributed systems management tool called Tectonic, which uses Kubernetes for container orchestration. And in a time when there are lots of options to choose from when it comes to managed Kubernetes providers, it was great to hear Brian describes some of the architectural decisions for building Kubernetes into Tectonic.

[SPONSOR MESSAGE]

[0:01:13.0] JM: Your company needs to build a new app, but you don't have the spare engineering resources. There are some technical people in your company who have time to build apps, but they're not engineers. They don't know JavaScript or iOS or android, that's where OutSystems comes in. OutSystems is a platform for building low code apps. As an enterprise grows, it needs more and more apps to support different types of customers and internal employee use cases.

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

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

I love to see new people exposed to software engineering. That's exactly what OutSystems does. OutSystems enables you to quickly build web and mobile applications whether you are an engineer or not.

Check out how to build low code apps by going to outsystems.com/sedaily. Thank you to OutSystems for being a new sponsor of Software Engineering Daily, and you're building something that's really cool and very much needed in the world.

Thank you, OutSystems.

[INTERVIEW]

[0:03:04.0] JM: Brian Harrington is the chief architect at CoreOS. Brian, welcome to Software Engineering Daily.

[0:03:09.0] BH: Thanks. It's great to be here today.

[0:03:10.9] JM: You've been working with Kubernetes for quite a while and I'd like to ask some general questions to get some lessons learned from your experience. Then we'll get into some other discussions on multi-cloud and the future and whatnot. But talking about operating Kubernetes, the way that a Kubernetes cluster runs is defined by configuration. What are some tips that you have on configuration management?

[0:03:39.2] BH: One of the most important things that I always like to lead with is make sure that you put all your configurations in revision control. It's one of those things that for folks who have been kind of getting more into the discipline of kind of software reliability or site reliability engineering as kind of put forth by Google and other companies out here on the West Coast, that doing configurations code and immutable infrastructure is super important, and Kubernetes is a champion in that effect of making sure that you have everything that you — Kind of the end results of your desired state in some type of revision control in text files. It makes everything much, much easier to iterate, kind of track down changes over time, see when things went wrong or when somebody accidentally fat-fingered something. Super, super important.

[0:04:30.3] JM: You're suggesting in your talk that users set up Jenkins to react and monitor, and react — Yeah, to react to these config changes. How would that work?

[0:04:42.8] BH: Personally, I use Jenkins and that's mainly out of muscle memory. Part of having those configurations and revision control means that having some robot who can watch those repositories for you and take action when things change makes it a lot easier to work with. I would like to clear. Use whatever tool you're most comfortable with. I try to stay agnostic of tools. But like I said, for me, Jenkins already has the built-in plug-ins to watch a generic git repo or GitHub of Bitbucket or kind of whatever tools you happen to be using today.

Even coming down the fat of if you just had those files sitting out on some generic web hosts and you were just watching an e-tag to see if things had changed. Having robots do our bidding for the future is definitely the direction in which things are going.

[0:05:35.3] JM: And that direction should be; I make a change to config. I push it to my GitHub repo and my Kubernetes deployment should be reacting to that push that I have made.

[0:05:49.9] BH: That's exactly the case. That's how we do it here at CoreOS today. Within our infrastructure team, we will have changes that are made to repositories, then we have a whole series of tests that are done on those changes. We use tools like Go serverspec, which is just a generic Go implementation of the Ruby tool serverspec, which will link everything. Kind of make sure that in-state is going to be the thing that we actually expect, and we do some things like running Terraform thumps, Terraform plan. Making sure that all of the state changes that we're expressing are going to be sane.

Then we have the actual kind of CD system, which for us — When is a CD, I mean continuous deployment, which for us is Jenkins, have sent us a Slack message saying, "Hey, we've ran all of these tests. This was the outcome of it. Do you want me to apply this?" That allows us to just respond in a very convenient manner to the CD system and say, "Yes, everything looks good.

Go ahead and deploy it," which allows for a lot more collaboration. If you're much more likely to have members team see when things happen and kind of take action to ensure that there is different individuals who are providing that kind of check and balance for each other as well as having the robots make sure that everything looks good.

The beauty is as we push towards these more generic type developer workflows, most developers at this point are familiar with the GitHub poll request model. So it allows us to take that same paradigm that developers are familiar with and move it into the infrastructure and operations space, which really enables different teams to interact a lot more efficiently.

[0:07:46.6] JM: Terraform also can play a role here. Terraform enables, basically, the creation and the changing and the improvement of infrastructure. It's an infrastructure as code tool. How does Terraform fit into a Kubernetes deployment workflow?

[0:08:05.8] BH: For us at CoreOS, we realized that Terraform was increasingly becoming a critical component for a lot of companies that were working on various cloud providers. We embraced it very early on and kind of integrated it tightly with our installation workflow, so that for anybody who is using Tectonic and anybody who already had Terraform in place, it made it super simple to bring up machines independent of the particular cloud provider. I'm able to say, "Take the ideas of like a cloud formation on AWS," where I kind of give a domain-specific language of the desired state of the outcome, except I can do that for many, many different cloud providers. I can say from the same tool, same DSL, "I wish to have a kind of data center set up on packet with the following network kind of configurations and the following machines that are brought, or I wish to have this VPC brought up Amazon in this specific region and this specific availability zone. This is they kind of layout that I want for everything."

It has meant that by having that one tool, we're able to minimize the number of places that people have to interact with to make changes, which has increased the overall efficiency of everybody working in the organization on these sorts of infrastructure build apps.

[0:09:38.0] JM: Let's switch the topic of state management. So some state in Kubernetes is going to be managed by etcd. Other times you're going to be managing your storage in databases and blob storage and object storage. Could you just give us a tour of the different ways of managing state in Kubernetes end where the domain-specific applications with each of those storage technologies, what those applications are?

[0:10:10.2] BH: Sure thing. You called out the important kind of locations for state, but I like to think about breaking it down slightly differently. You have the state of the actual control plane and then the state of the data plane. The state of the control plane is going to be communications and the kind of expected pods that should be running, the actual content that gets represented inside of etcd. On the data plane, that the runtime information about the applications that you're running. That's going to be the specific files that are getting served from an nginx instance or the contents of a PostgreS server. You have to have slightly different processes for kind of managing these and backing them up and ensuring resiliency there.

When you're managing that state for etcd and the API server, a lot of that is kind of handles auto-magically for you, because that's the main value of having that control plane in place. Being able to go through and run backups of etcd where you serialize the data out for disaster recovery is important, but in the grand scheme of things, that's actually the easier piece to work with.

For the data plane, where you need to make sure that you have kind of the contents of that relational database or all of the containers, keeping the manifests of how things are deployed is one piece. A lot of folks at CoreOS tend to use Helm for that. So they'll actually templatize the applications that they want to run into a Helm chart, then from there, depending on whether it's a stateful application or not, they may need to bring with them some type storage, and using the generic concepts of storage classes within Kubernetes allows them to just say, “Okay I need some persistent volume. I don't really care what the underlying mechanism that is managing that storages is, be it EBS, if you were on top of Amazon Web Services or just a premium managed disc if you are on top of Azure.” But being able to just say, “I need five gigs. I don't care what it is. Make it happen,” and then kind of bringing that data with you means that you have this Helm chart that can define the runtime in the application and then you just need to manage whatever that process would be, whether it was a bare-metal machine or whatever it looks like today. That's namely going to be whatever the native tools are for various applications. Being able to use PG Dump or being able to kind of use Curl to instantiate data on that volume to get everything started.

[0:13:16.3] JM: Your describing a usage of Helm chart that I guess I didn't know. I'm not totally familiar with this space, but my understanding of Helm was if you basically creates a very simplified form of deploying something to a Kubernetes clusters. If you want to deploy Cassandra, for example. Cassandra is a multi-node database and Helm gives you a way to just install it with just a few commands was just kind of a big deal, because I don't think we've had a really of distributed systems package manager in most of the times in the past, and you could also do it for deploying something like WordPress. But you were describing it just now as people use Helm to deploy the application that their company is building. Perhaps, for example, if I'm like Zendesk or some other SaaS company and I run on top of Kubernetes and I've got all these different services. Maybe I can deploy all of those services at once across a Kubernetes cluster by putting it all into a Helm chart. Helm chart is the description of the systems that you're deploying on Kubernetes. Can you give more clarification for how people are using Helm to deploy their applications?

[0:14:37.6] BH: Absolutely. Helm, the primitives that are available in it allows you to do a lot of the templating of containers themselves, the kind of pod-specs. A lot of the different primitives that are available within Kubernetes. When you are able to start mapping out the individual components of an application and then relate the kind of dependencies between them within the Helm chart, you're are able to have this kind of much more atomic reference of an application as one single asset that allows you to both revision the individual pieces of it and allow kind of check-pointing at different points in time. So one workflow of that is — For us internally, we have all of the various assets that run coreos.com, account.coreos.com, things like that. So we are able to reference the constituent micro- services for each of those as a, "This is coreos.com version 1.0. This is CoreOS version 1.1."

Then individual developers who are working on those kind of different internal versions of those applications can pull out everything together at one specific time by referencing that. That's actually an important thing to note and has been one point of friction for users of Helm up to this point, is that there's a little bit of, I guess, I can say looseness between semantic versioning of an application, semantic versioning of container definition, the versioning of the Helm chart, the versioning of all those things that get attached together. You run into areas where the Helm chart might be version 1.0.2, but maybe it's referencing six different components that are all getting referenced together and kind of coming up with a way of knowing that you're getting all the pieces that you want and being able to introspect that down to know that two folks are working on the things that they intend to be working on has definitely been a little bit confusing for some users.

[SPONSOR MESSAGE]

[0:17:09.3] JM: Today's podcast is sponsored by Datadog, a cloud scale monitoring platform for infrastructure and applications. In Datadog's new container orchestration, Kubernetes holds a 41% share of Docker environments, a number that's rising fast. As more companies adopt containers and turn to Kubernetes to manage those containers, they need a comprehensive monitoring platform that's built for dynamic, modern infrastructure.

Datadog integrates seamlessly with more than 200 technologies, including Kubernetes and Docker so that you can monitor your entire container infrastructure in one place. With Datadog's new live container view, you can see every containers health, resource consumption and running processes in real-time. See for yourself by starting a free trial and get a free Datadog t-shirt at softwareengineeringdaily.com/datadog. That's softwareengineeringdaily.com/datadog.

Thank you, Datadog.

[INTERVIEW CONTINUED]

[0:18:18.3] JM: So let's go on to talk about networking. Container communicate with each other in Kubernetes. This is networking. What are some common mistakes that people make when they are configuring their networking across Kubernetes? One thing that I run into over and over and over again is folks using the same IP address ranges in every single environment. The beauty of VPC on Amazon is that you can use whatever IP address range that you want. So when you're starting out in development, everything seems hunky-dory. You're using 10.0.0.0/18. I'm using 10.0.0.0/18, and then we go and try to have both of our VPCs intercommunicate, or we try to set up a point-to-point VPN's that we can kind of bridge what is effectively two different data centers. Then we immediately have to start figuring out network address translation in all kinds of things that we should really be hoping to avoid in the second half of the second decade of the 21st-century.

Hence, there's — CoreOs, we've put together some open-source tools to kind of make it easier to work with those things. Evan Tschui, or @tschui on GitHub, wrote a tool called Ciderblocks to help with some of that simplified calculation. It actually just does one very opinionated thing. The AWS solutions architecture team put out some guidelines a couple of years ago at this point on like how to "correctly layout the address space on a VPC so that you had private, public and protected ranges." and this is just one more aspect of kind of if you're in the average case, you don't necessarily. You may not have worked with a lot of networking in the past, and having some of these tools that can go through and do the right things for you with a minimal amount of understanding to make sure that folks do the right thing is always super helpful.

Again, kind of reeling this back in. Make sure that you don't use conflicting IP address ranges. Kind of configure your DNS in such a way that ideally you can even use name server delegation down to your cluster to be able to use things like CoreDNS on the cluster to reflect back what the IP range or what the IP address of individual services is. If you're using tooling like VPCs, it's super, super helpful to be able to configure those point-to-point VPNs so that you can just do vanilla routing.

For us here at CoreOS, we have multiple different cloud providers running with Tectonic clusters in multiple different regions across those cloud providers, and by being able to plan that out correctly. We actually just to layer 3 routed traffic between a lot of the clusters. For developers here in like our office in San Francisco or Berlin or New York, they just reference those clusters by the kind of RFC 191810.addresses and they don't have to do any kind of mapping over the public Internet. Everything is just done very, very cleanly.

[0:21:46.5] JM: My most intimate introduction to the problems with multiple IP address mappings going to the same URL was — I actually made a mistake of — I was moving my podcast feed from one host to another and I did it in a way where there was a period of time where I had the same domain name was mapping to multiple RSS feed IP addresses, and then so I updated the WordPress instance that backed the RSS feed on only one of those IP addresses and it wreaked havoc on my listeners, because around that time I had like 400 or 300 episodes that I had done in the back catalog, and the way that podcast players work for some reason is that if there is enough RSS feed confusion when they request that RSS feed, they just try to download all of the episodes, whereas normally a podcast players is only going to download the most recent three episodes.

To make a long story short, because I made a mistake in my IP address mapping just on the freaking podcast, I ended up dossing my listeners. I had a bunch of listeners who emailed me were like, “Hey, all of a sudden I downloaded 300 MP3 files onto my iPhone, and why did you do this to me. I'm unsubscribing from your show.” It’s like, “Wow! That’s terrible. I guess I'm never moving my podcast off of my host again.” I'm sure it gets significantly worse than that when you're talking about building people's critical infrastructure.

[0:23:28.4] BH: Yeah, at the same time I am a firm believer in — The way that a lot of software development and computing has kind of progressed, is we’ve put up so many guardrails and bumpers that folks don't have the opportunity to just kind of skin their knee. We’re kind of running into this situation where if you trip, you fall off a cliff, and rather be in a situation where you can have a skin in your knee and you learn what you don't know.

To some extent, as these systems become more and more complex, it just means that we have to approach this very intentionally. We have to think about like what is the kind of crawl, walk, run model there, because, like you said, if you're in the boat where you don't know what you don't know, like small things like that shouldn't necessarily cause somebody's phone to be downloading 20 gigabytes worth of data, but I think that's an opportunity also for some things that kind of we figured out in the past, but we still have yet to kind of port to Kubernetes. Things like being able to do better introspection of traffic that’s coming into the cluster to detect some concerns like that.

I mean, not to necessarily blow anything up here, but how many folks who are just running an application on top of Kubernetes would know if somebody was trying to do a brute force attack on logins or anything else. We’re in the situation now where when that application crashes, because it's getting attacked. You just don’t gracefully kind of back up and everything will be good to go.

In the past, at least when that application crashed, you would get alerted, you come in and start taking a look, but I often see with development clusters and folks who are starting out, pods that is just in a crash loop and are restarting thousands of times an hour, and that gets into some of the next steps of kind of making sure that you've got alerting on these sorts of behaviors, making sure that folks are able kind of able to get that telemetry out to take action, because it's great having these automated systems, but sometimes it just results in not being exposed to some of the more basic levels of taking a step back and digging in to what's going on.

[0:26:00.0] JM: Let's talk more about those operational devices that people are going to be setting up. When you're deploying a Kubernetes cluster, there are all these things you might want, like log aggregation, monitoring, alerting. Maybe you want service proxying and service mesh, and I think of setting up these things within Kubernetes as the job of almost like a platform engineer. Who is responsible for setting those things up at an organization? I guess maybe what should be the process for a company when they're deploying Kubernetes, how aggressively should they start to roll out things like log aggregation and service proxying? Because these are all things that, really, they make your Kubernetes cluster run much more smoothly, but of course they take time away from you actually working on application features.

[0:26:58.1] BH: Yeah. All of those functions that were historically part of an infrastructure team or a systems administration team. I'm of the opinion that those sorts of things are still absolutely a hundred percent required on Kubernetes. Now, I think that is also one of the values of having different distributions of Kubernetes, because it gives different development teams on those upstream kind of distributions the ability to differentiate themselves.

I am of the opinion that you should not be deploying one cluster, let alone multiple clusters if you don't have log aggregation or if you haven't thought through things like single sign-on. Those were just kind of table stakes in the past for bringing up an entire environment, and I don't think that Kubernetes is any different.

Now, what I do think is the bigger opportunity there is that as we transition into what I call the realm of the iPhone, you and I were briefly talking at Cube-Con about your copying and pasting from Stack Overflow. One my colleagues here at CoreOS, he's kind of very romantic about the previous days of the spoke systems and the fact that you had to go and handcraft everything and how there's been something lost by the fact that that is knowledge that is less and less come in place.

But I am kind of — There is a twinge of my heartstrings that get tugged every time I think about what it takes to bring up pre-execution environment from scratch or things. I don't think that knowledge is ultimately that important, especially as we are able to build more tooling that solves that. I want to see us get to the era where it's not considered the era of copying and pasting from Stack Overflow, but it's more the era of the iPhone where you turn that device on and it works, and if for some reason it locks up — The average person. I should say. I still do this, but the average person doesn't just hook up a cable, fire up a remote debugger and start digging into it. They should be able to treat it more like just a robot that they kind of grind down the parts, recycle and instantiate again.

Having all of those common core infrastructure pieces deployed every single time, like log aggregation, like a service mesh, like kind of ingress and egress filtering and stuff, is just going to get it to the point where folks know what is coming, batteries included in a cluster. What resources they can expect to be there.

One other kind of important thing there is when we started working on Kubernetes years ago, I was talking to Alex Polvi, our CEO, and he pointed out that we are kind of in this era of bootstrapping the iPhone and how, basically, there is no app store yet, so everybody is going to have to fix all of their applications. The thing that I pointed out to him was that even when the first iPhone came out, there were still certain core applications there. There was dialer, there was notes, there was contacts, and I think that a lot of these things like a log aggregation, while I prefer the kind of you can swap it out for the flavor of that that you prefer, if you want to use Sensu, versus Prometheus and Alert Manager, you can do that.

I think that getting towards that point where we all agree on kind of what those common services are and that they're always going to be included and what the individual services are is more implementation detail is going to be the ultimate experience for developers that makes them much more apt to use the system and much happier about using the system.

[0:31:11.6] JM: Well, there's a lot there. I mean when you talk about the nostalgia for the days of knowing what to do rather than having to look up something on Stack Overflow — I don't know. Going to Stack Overflow and copying and pasting some code, even that to me feels like looking up op-codes in a textbook, which is what people had to do in the past. It's like, "Why do I have to look up something? Why do I have to do this endless Googling and why am I not just engaging with this platform, like the creative pallet that it could?" Because we're in early days, I think.

I mean, I don't know if you, Brendan Burns, his keynote. I actually did not see it either, but I just know the name of it, was basically this is still too hard.

[0:32:03.9] BH: Yeah, that was all the meta-particle stuff, and I agree with Brendan. I firmly believe that you know we need to ensure that this knowledge doesn't die out, because I don't want to get to the point where the contents of the library of Alexandria gets lost, but it is decreasingly important. When you track the overall level of technical understanding that an individual user needed to have to use a computer, that level technical understanding has continued to trend downwards as the number of computers that individuals interact with increases. I do believe that that's the correct direction.

Understanding that op-codes are even saying should not be the kind of table stakes there. Again, I do firmly believe that Brendan's ideas about meta-particle are absolutely the path of the future. There's a few aspects of it where I take a look at it and I scratch my head about it, but at the same time I very firmly believe that I am also, despite the code that I write, I am not a developer. I fundamentally approach things from a different angle.

[0:33:23.1] JM: Right. I think the meta-particle idea is — If I remember correctly, it's like a language level distributed systems primitive where you could have a variable that is — It's kind of like a replicated variable and so it has the durability of a replicated data store, but it's not a toy project and I want to call it a toy project, because I know he has intentions to make it a bigger and better thing. But even if you just look at it as a toy project, it's kind of interesting. What happens when you move replication to — Well, I'm not putting it right. What do you think he's trying to do with that project, where when you move distributed systems concepts into the level of the application developer?

[0:34:13.3] BH: I think the easiest way for me to summarize that is he's just trying to make it something that's a part of the standard library. In the same way that had Ken Thompson known that HTTP or known HTTP would be so ubiquitous or known that every application effectively would be a network application, I think we would've seen very different things in the initial versions of C.

What Brendan is trying to do is take all of those ideas of what we now know are just the things that every application kinds of needs to potentially be aware of and putting that into one standard library so that you can start out with the basic development for your application and then later go, “Oh, I need to be able to shard this application, so let me kind of just go use the functions of that library and say, “Application, now you shard.” Then take that further and go, “Okay. Now I want the shards to be able to each handle quantile of traffic.” These are frankly put there, components or their disciplines that have been learned over years of building these systems and that having scarped and rebuild them. So we’re standing in the shoulders of a lot of giants who have figured all of these out and it’s taking that knowledge of all of those different domain disciplines and putting it into one small contract where we know that all of the functionality of that is always going to be available. So you don't need to pack a toolbox in the back of your car. You can kind of accept that, “Things are going to run reliably enough that we don't need to bring jumper cables with us every time or we don't need to carry an empty gas can.”

[SPONSOR MESSAGE]

[0:36:18.4] JM: If you enjoy Software Engineering Daily, consider becoming a paid subscriber. Subscribers get access to premium episodes as well as ad free content. The premium episodes will be released roughly once a month and they’ll be similar to our recent episode, The Gravity of Kubernetes. If you like that format, subscribe to hear more in the future. We’ve also taken all 650+ of our episodes. We’ve copied them and removed the ads, so paid subscribers will not hear advertisements if they listen to the podcast using the Software Engineering Daily app.

To support the show through your subscription, go to softwaredaily.com and click subscribe. Softwaredaily.com is our new platform that the community has been building in the open. We’d love for you to check it out even if you're not subscribing, but if you are subscribing, you can also listen to premium content and the ad free episodes on the website if you're a paid subscriber. So you can use softwaredaily.com for that.

Whether you pay to subscribe or not, just by listening to the show, you are supporting us. You could also tweet about us or write about us on Facebook or something. That's also additional ways to support us. Again, the minimal amount of supporting us by just listening is just fine with us.

Thank you.

[INTERVIEW CONTINUED]

[0:37:53.7] JM: To talk more about operations, which was a lot of what your talk at Cube-Con was about. You were talking about multi-cloud deployments. Let's just start with a naïve question. Why do people want to deploy Kubernetes to multiple clouds?

[0:38:14.0] BH: For that, I think there's two fundamental reasons. One, I agree with in my heart but I think that it's a little bit flawed. The other I think is very, very legitimate. The legitimate reason is, is that you want to be able to kind of really guarantee high-availability in that sense. That is the kind of rewinding back in the day. You have one data center where you're kind of subletting a cage out of Level 3 or Equinix and then you've got your own data center, and the reason why you're doing that is you want to have enough heterogeneity in your environments to be able to really guarantee that you have resilience against a lot of different types of failures. So that is to say that you know that if you're deploying something to S3 in U.S. east that when a tornado comes through and takes out a data center, that you don't lose availability, because you also have copies of your files sitting in a bucket on top of [inaudible 0:39:17.6] storage.

Now, the area where I think folks are a little bit flawed just because of the technical level of what it takes, is the kind of race to the bottom of the price. So I want to live in a world where the compute truly is commodified so that I can just up and switch everything from one provider to another in a moments' notice. I see that the spot pricing on AWS has now dropped to a point where it's worth moving everything off of some random open stack provider, or conversely that I know that the pricing on packet.net has gotten down far enough that I can consolidate a bunch my resources from AWS and move them across, but the complexity of some of the API changes that need to occur is not something that is just immediately swappable or immediately movable.

Now, that ultimately is where the value of those built-in components, like storage classes within Kubernetes that I was mentioning earlier, start to prove me wrong, and I have to say I am happy to be proved wrong. Like I said, it’s ultimately the place that I want to get to. I want to see the underlying cloud be agnostic from the workload. But for a lot of organizations, they are using something like Amazon SQS, and if you're using Amazon SQS, you now can’t just swap over to another cloud provider.

Now, the value of Kubernetes means that if we can build out various open cloud services and then use those open cloud services atop, now you have gotten back to that point where you're abstracting the storage, you’re abstracting what is providing an AMQP interface, what is providing a relational database.

[0:41:22.1] JM: This is a big deal.

[0:41:23.0] BH: It is. It's very exciting to me.

[0:41:25.9] JM: One thing Brendan said that I've just been thinking about when I interviewed him recently that I’ve just been thinking about since he said it was you could imagine a world where people start to make money off of proprietary binaries that they sell to you that you could deploy on any cloud provider. If you can imagine if all of the different cloud providers are running Kubernetes, and let's say I am a developer and I write a — Let’s just say I write a better WordPress. I write a new WordPress and I sell you the WordPress binary for $99 and then you go and deploy it on whatever cloud you want and you get a license forever and you can just use it forever. That could be a business model.

[0:42:09.6] BH: It absolutely could be. I think it's a huge opportunity for developers as we have increased also with the ubiquity of cross-OS and cross-ship architecture programming languages, like Go lang where you can kind of set your Go arch variable and compile down to arm 64 or Raspberry Pie, arm 7 or just kind of vanilla Intel x86-64 taking all of these various cloud providers now where you can then render out that one binary and run it on that cloud provider in whatever mechanism you want is going to mean that, for developers who have those brilliant ideas want to build those better mousetraps, that is easier for them to get started and ensure that it will run in a consistent way, because once you build the application, you then have to support it, and that's the part that's a lot less fun. If we can make that easier for folks to do as well, we'd hope to see thousands of applications bloom in that field.

[0:43:24.4] JM: I know that our time is running short. I have both of our calendars tolling, but talking a little bit about this Kubernetes as a service providers, when I was at Cube-Con and I saw you there, I was walking among the Expo Hall and I must have seen 15 or 20 different Kubernetes as a service providers. Tectonic is a Kubernetes as a service provider that comes from CoreOS. Maybe you could talk about what are the subjective decisions that you can make when you're architecting a Kubernetes as a service.

[0:44:01.4] BH: Well, the first kind of important one is deciding how much you insist upon owning, you being the service provider, and how much flexibility you're going to provide the customer. It's very easy to take and stand up control plane or run that control plane on behalf of users. If you're doing that, then bringing nodes is super, super simple, but you need to be able to answer the question for your users, like, "What API flags are you going to dictate?" will be run, or on the flipside, are we just going to allow our users to tell us every single API flag that they want to run?

One of the differences between what CoreOS is doing with tectonic is we provide that service model of managing the lifecycle updates of the Kubernetes components and increasingly applications that run atop a cluster as well as the host operating system. But we provide that on the customer's own premise, be that within their data center or within Amazon or TCP, which is definitely a different model than what a lot of the folks at Cube-Con were doing.

I think it is — And will continue to be our differentiator, because from the start, ee have led with this idea that we are going to do things in a very opinionated manner, but we want users to ultimately have the control there, which there is no greater control than running on your own hardware, in your own environment, within your own network configuration. That definitely presents a challenge, because user A decides that they just have to have the BGP, while user B decides that they want everything in an overlay network. You have to be able to give folks the knobs while at the same time knowing and being able to measure exactly where you draw those lines to ensure that folks aren't shooting themselves in the foot or that they are going to break automated updates or things like that.

[0:46:24.9] JM: The CoreOS customer-base, is that predominantly people who are deploying to their own infrastructure?

[0:46:30.9] BH: Yes. That what we try to really focus on. Now, that is to say, we have a lot of users that will deploy both to AWS and to their own data center. In that sense, it is still arguably their infrastructure, because they are kind of defining the various configurations on the host. They're stating you everything's going to use this specific time server, but it is that flexibility regardless of kind of the underlying provider, and that's kind of how we end up working with the customers to provide them the value for using Tectonic versus using other Kubernetes distributions.

[0:47:18.5] JM: To wrap up, I just love to get any interesting business prognostications that you might have, like how is Kubernetes going to change the dynamic between cloud providers? Is it going to allow for the emergence of artisanal cloud providers? What other kinds of business models does they shift towards Kubernetes exposed? Give me your most — Your craziest theories about how business will change because of this shift in infrastructure.

[0:47:48.3] BH: My craziest gets pretty crazy, but kind of taking a stab at this, I have seen a lot of folks who are kind of pushing the ideas of the function as a service model. While it is still extremely early days, I've seen that proved out in parallel with the increase of both popularity in tools like Zapier and If This Then That, but also the open-source kind of equivalence, like Hugin, and then all of the various competitors that are propping up, even all the way up to Microsoft releasing Microsoft Flow.

Having those proprietary engines for being able to start to glue together all the various micro-services is going to be an increasingly popular model. It's tragic that Yahoo Pipes was the first there, and the first is generally not most successful, but as Kubernetes also increases in popularity, it's going to make it easier for folks to kind of even have cross Kubernetes interaction of applications, having things where you're able to send a Slack message and interact with a service from one provider who then triggers some action on a function running on open [inaudible 0:49:20.2] somewhere that then talks to an RSS feed to pull some data down and then push that off to yet another service provider.

I want to see us get to the point of kind of having these greater meshes partially to simplify the amount of work that anyone needs to do. Now if we can do that in an open source manner, that's the part that will really get me truly excited and kind of have every cockle in my heart singing for joy. At the same time, I also want to make sure that folks who have these ideas aren't necessarily burdened by being able to find a sponsor, because ultimately having that good idea used to be all that you needed, and as folks are now competing with bigger and bigger giants, they have a little bit of a tougher road to home.

[0:50:19.6] JM: Agreed. Okay, useful words. Thank you, Brian. Thanks for coming on Software Engineering Daily.

[0:50:24.4] BH: Of course. Thank you, and have a fantastic afternoon.

[END OF INTERVIEW]

[0:50:30.9] JM: GoCD is an open-source continuous delivery server built by ThoughtWorks. GoCD provides continuous delivery out of the box with its built-in pipelines, advanced traceability and value stream visualization. With GoCD you can easily model, orchestrate and visualize complex workflows from end-to-end. GoCD supports modern infrastructure with elastic, on-demand agents and cloud deployments. The plugin ecosystem ensures that GoCD will work well within your own unique environment.

To learn more about GoCD, visit gocd.org/sedaily. That's gocd.org/sedaily. It's free to use and there's professional support and enterprise add-ons that are available from ThoughtWorks. You can find it at gocd.org/sedaily.

If you want to hear more about GoCD and the other projects that ThoughtWorks is working on, listen back to our old episodes with the ThoughtWorks team who have built the product. You can search for ThoughtWorks on Software Engineering Daily.

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

[END]