EPISODE 853

[INTRODUCTION]

[0:00:00.3] JM: Kubernetes has made distributed systems easier to deploy and manage. As Kubernetes has become reliable, engineers have started to look for higher level abstractions that we can define on top of Kubernetes. An operator is a method of packaging, deploying and managing a Kubernetes application. Operators are useful for spinning up distributed systems, such as Kafka, Redis or MongoDB. These data systems are complicated. They're stateful applications with lots of failure domains. The operator framework enables a developer to deploy one of these complicated applications with less fear of the system crashing, or entering an erroneous state.

Rob Szumski is an engineer at Red Hat and he joins the show to discuss Kubernetes and the operator pattern and his time at CoreOS, which was acquired by Red Hat.

I'll be attending a few conferences in the near future. I'm attending Datadog Dash July 16th and 17th in New York City, and the Open Core Summit September 19th and 20th in San Francisco. Also, we are accepting sponsorship proposals if companies are interested in sponsoring Software Engineering Daily and reaching our audience of 200,000 plus developers. We actually don't have great numbers on how many listeners there are, but we know it's a lot. If you're interested, you can e-mail me at jeff@softwareengineeringdaily.com and we can discuss sponsorship.

Let's get on to today's show.

[SPONSOR MESSAGE]

[0:01:45.5] 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]

[0:02:42.2] JM: Rob Szumski, welcome to Software Engineering Daily.

[0:02:44.5] RS: Hey, nice to be here.

[0:02:46.1] JM: You were at CoreOS for four and a half years before the company joined Red Hat. When you joined CoreOS in 2013 long before the acquisition, what was the infrastructure landscape like?

[0:02:59.7] RS: It was interesting, because – so we set out to reinvent the way that Linux was run. This is before containers, so VMs were a really big thing. People building pet VMs. There wasn't a lot of large-scale infrastructure to support running a thousand VMs. Folks like Facebook and others were doing this.

Yeah. I mean, you had cloud APIs and things like that. Folks, even Terraform I don't even think was around then or was very new. There wasn't a lot of this infrastructure, so we were part of the wave building that. Containers were a part of that. Re-architecting Linux was part of that as well.

[0:03:29.9] JM: What problems was CoreOS seeking to solve?

[0:03:32.8] RS: It was running large-scale Linux infrastructure. Our founders – I also came from Rackspace, where they were running technically a very large distributed system running web scale monitoring. They realized that managing all their Linux servers – I mean, this was a cloud company, keep in mind. The right tooling wasn't there the Linux side of it. We had some of these APIs to programmatically boot VMs and things like that, but we didn't have the capabilities to know what is the state of all my thousand servers that might be across seven different GOs. Are they patched? What kernel versions am I running? How do I manage this fleet?

We had tools like chef and puppet, but those were more convergent style technologies. Am I orphanage off snowflakes here and there that are now no longer tracked in my system? That was the landscape that you found yourself in. We are trying to solve that of how do you run Linux at very high-scale?

We backed into containers and Kubernetes and distributed systems through that lens of I've got a thousand of these things, how do I keep them upgraded? Okay, well that we got to re-architect Linux to do over-the-air style upgrades, is really the only way to do it at scale. What do I need? Well, I need a way to move these applications from machine-to-machine, so we can take them down and reboot them to update them.

Okay, well containers provide a great encapsulation of an app and its dependency, so I can move those around. Now I need some logic for where do I move them to, which one of these thousand machines. We backed into Kubernetes and other orchestrators like that.

[0:04:58.3] JM: That's a good description of how the technology changes were evolving at the time. Can you go a little bit deeper on how the strategy within CoreOS changed, like the business strategy?

[0:05:13.8] RS: What's interesting is the business strategy really didn't change throughout the entire company run, as well as the technology strategy at a very high level, which was to sell to companies that had this problem. The beauty of it was the market was growing and growing and growing, because almost every company had this problem. Everyone's running web services, everyone in your industry is running web services getting smarter with machine learning, moving things to the web.

If you weren't doing it, you were about to get disrupted. You see this in everything from transportation and things like Uber and Lyft to real estate, to e-commerce, to same-day delivery and things like that. All these things need APIs, they have machine learning behind them, they have all this stuff needs to run somewhere. We were catering to those types of organizations. Now that's every bank, every insurance company, every government agency, etc.
[0:06:03.3] JM: Why did Kubernetes win the container orchestration wars?

[0:06:06.6] RS: I think for two things; one, I mean, Google's stewardship in the beginning was really, really powerful. They had the technical chops to make people sit down and pay attention to it. They also ran the project really well. I mean, I don't know if this comes from any of the ways that they organize internally, or just they just hit on a nice open source way to do this, which was there's these special interest groups that look after every part of Kubernetes and they're driven by design documents and they have sign offs and they have community meetings. It's a very well-run thing. Even if a lot of those people in the beginning were Googlers, or we were taking off concepts that were tried and vetted at Google, but then now it's taken off into its own thing using that same process.

Other ecosystems were very top-down driven by either a single commercial organization. I don't know as much about the history of Cloud Foundry. I think they tried to do some of this as well. I don't know much about why, if we consider that failed or not.

[0:07:05.6] JM: Yeah. I mean, in the Cloud Foundry case, I don't know that story much either, but what Kubernetes did right, one thing they did right at least was that the whole donation to the CNCF thing, which didn't entirely subtract Google from having influence over the project, but at least was a nice signal. It's in a signal to the community. This thing is fully donated.

For people who don't know much about open source, like what open source means, there's a big difference between Tensorflow, which is open source. As far as I know, that repository is maintained and owned by Google, versus Kubernetes, which Google donated to the Linux Foundation. It has this democratized element to it. When you hit when you have this really big tent, like Linux Foundation oversight, it goes a long way to building trust in the community, as opposed to the Cloud Foundry model, where I think it's more – it still is more in the purview of

Pivotal, which has its pros and cons. What lessons did the container orchestration wars teach you about technology adoption?

[0:08:16.2] RS: I used to play in the OpenStack realm as well at Rackspace.

[0:08:19.5] JM: Oh, wow. That's another historical analog.

[0:08:23.7] RS: Yeah. I was at Rackspace prior to the birth of OpenStack and saw that get birthed there. Rackspace is one of the original two with NASA in releasing that open source code. What I look at when you look at the adoption and the lifecycle of an OpenStack, the community and ecosystem versus Kubernetes and containers, it was the right abstraction level for the end-users, which was the software engineers that are actually building all the web scale services that we just talked about.

If you look at getting a raw VM, or getting a load balancer is great in the OpenStack world, but it's not the right level of abstraction for what you're trying to do. That doesn't help you ship your predictive type-ahead search that you need to put in your mobile app, or it doesn't help you predict transaction volumes or something like that. It's just a means to an end. Kubernetes actually gives you those primitives.

I need to scale out this web service really, really quickly. You're not dealing with booting machines and wiring things up. You've got service discovery at your disposal. If you need to rotate a secret for example, there's no tooling for doing that in OpenStack or with a VM. You're inventing all that yourself. Versus now, you're a little bit closer down to your application where you have tools for doing secret management, for example.

It makes developers more productive, but also has a bunch of great benefits for your infrastructure teams. Whereas OpenStack or anything that is not going to get all the way down to that core end-user is not going to be as successful. Does that make sense?

[0:09:46.4] JM: Yeah, interesting. How would you calcify that into a more abstract lesson?

[0:09:53.6] RS: More abstractly – I guess, it's all about delivering value to the folks that not are – I'll use the decision-makers in air quotes, because it's not the folks with the dollars, but the folks that are making technology decisions. Is it making their lives easier?

[0:10:08.7] JM: Why didn't OpenStack do that?

[0:10:11.3] RS: Because, I don't think it was the right set of tools for the software engineers and the teams that were running infrastructure on those VMs. They still had to build all this tooling. It got them one step down the road, but they needed to be four steps down the road thing.

[0:10:24.5] JM: Who was OpenStack for?

[0:10:26.1] RS: Honestly, I think it was for folks trying to compete with Amazon and building large-scale infrastructure, but people that were building clouds. If you were one of these very large enterprises that was going to have a fleet of a bunch of bare metal servers that you were going to turn into your private cloud, OpenStack was for you, but it wasn't for your software engineers at the end of the day.

[0:10:46.1] JM: I see. What would happen is a central IT person would say, "This solves our problems." Then they would say, "Okay, software engineers go implement my OpenStack strategy." The software engineers would do it, but not fall in love with it to the same extent that people fall in love with Kubernetes.

[0:11:07.6] RS: Yeah. It would be something that they would be mandated or forced to use. It's better than opening a JIRA ticket to get a VM. It's better to hit an API, so you're one step closer. They don't give you all those primitives for like, "Okay, now I've got to deploy some software around here. How do I build that software? How do I package it into an artifact that ends up on this VM and then how do I do X, Y or Z about the lifecycle of that?" That's like, "Well, you can make another VM via this API?" You're like, "Well, that's not what I need."

[0:11:36.5] JM: Right, right, right. Unfortunately, the four-steps-ahead solution was AWS at the time, which was an open source. Then we had to just continue waiting to really get the open source version of the leverage that you get out of AWS.

[0:11:54.6] RS: Yeah. Great point of that is, “Hey, I need a database.” That’s like, “Oh, sweet. Here's a VM. You can install a database on that.” You're like, “Well, that's not what I need.” That’s yeah, gets you one step forward, versus Amazon is like, “Oh, here is a AHA fully secured, production-ready MySQL. Is that what you want?” You’re like, “Yes, perfect.” “Here’s the host name and the password.”

[0:12:13.3] JM: Yes. Okay. The I need a database request is going to be a great – that's a great preface for what we're going to talk about with operators a little bit later. Getting a little bit more into how you think about building products for engineers. You studied industrial design in college. Much of your work has been around the design of technical products, products for software engineers. Give me the one to two-minute condensed master class of designing products for software engineers.

[0:12:47.3] RS: Sure. What I learned in my industrial design background is really this idea of design thinking. It's a way of solving problems and it's a pretty – it's methodical, but it doesn't have to be something that you have to enumerate every single step and do it perfectly every time. It's all about doing research and understanding the problems of the different classes of users that you're trying to meet the needs of. Then fitting a void of in that problem space and then growing out from there.

If you're going to produce a new electronic component or whatever, something like the iPod fit this niche that people had this void in their lives. Then you can design around that. The actual appearance of it is great and the functionality of it is great, but nowadays everything has a service component to it. When you're designing Spotify, it's not just like, “Oh, how does the player look? Oh, they've got this dark UI. That's cool.” It's actually like, “Oh, how do these magical discover playlists work? That's a service component to it.” The experience is constantly changing, and so it's that idea that we have this holistic experience when you're designing something, whether it's a web interface, an iPhone, a CLI, an API, all these things matter because they grow over time.

[0:13:57.9] JM: What's the most common anti-pattern you see among products designed for software engineers?

[0:14:03.4] RS: I would say either breaking API contracts, or revving your version such that like, “Oh, semantically we increase the version, so we didn't break our contract, but introducing a lot of toil for folks where they have to mess with your product.” Basically, what you're asking for is to insert yourself into somebody's development loop in their product shipping. You want your tool to provide a ton of value for them. If you're also adding a ton of friction, “Oh, we changed this, we renamed this, we changed our pricing scheme. Oh, you now authenticate like this, or like, oh, we deprecated that. We have this new thing. It works a little bit differently, but it's 80% there.” Anything that you do those types of things, because it's easier for you as the engineer producing that tool, you're introducing all that friction to your customer.

I see that happen time and time again. Actually going back to the OpenStack thing, this is where Google does a great job in Kubernetes and we didn't in OpenStack is all those projects were versioned differently. They had different backwards and forwards compatibility guarantees. They upgraded, either didn't at all, or provided a crappy migration script, or some of them would have a world-class migration, but it was all different so you couldn't depend on it. That's something that Kubernetes does really well in terms of how you upgrade it. Every CLI out there as well, if you've ever had something that broker dropped flag silently is a really big problem.

[0:15:20.3] JM: The buying of software has changed. It's gone from the days of a CSO, or a CIO buying a big solution and ordaining that the software engineers implement it. I mean, that still takes place in many cases. There's clearly a shift towards a bottoms-up mentality, where the software engineers are the ones who are selecting the technologies and there's a gradual uptake within a company. Then eventually, the central IT says, “Okay, well all of our developers are buying into this. Pretty easy to just go negotiate a contract, because developers are going to use it.” How has that changed the design of software engineering tools?

[0:16:04.7] RS: I think it's raised the user experience bar, is the most overarching thing that you'll discover. From the docs about your product, even if it's API-driven, like the Stripe experience is always heralded as being really, really great. You now need to live up to that for the next thing. If you're going to introduce some video transcoding service, for example, it's got to be clear. What it gets me? How do I implement it? What is it going to look like in the future? Why am I betting on this? Then can I actually plug it into my thing, whatever workflow that I'm implementing? I think that bar is now really, really, really high, which is good. Everybody's lives get easier, but that just means that these crappier products aren't going to make it just because they don't live up.

[0:16:48.6] JM: What are the problems in usability that Kubernetes users still have today?

[0:16:54.9] RS: I think it's a number of things that – it's a pretty broad platform at this point. Understanding all of the knobs that you have to tweak to tune your applications, or tune their eviction behavior, or anything that's about the lifecycle of your application, there's this – and functionality is constantly being added. Understanding all that and just keeping in mind what's coming, what's stable, what's in the beta, that type of thing.

Then shipping, containerizing your software is still a hurdle for a lot of folks. If you're on more the bleeding edge, that seems like, "Oh, we solved that three years ago." Folks are still getting into how do I adapt my software for this new world? Folks are very used to just manually configuring a single server and leaving it running forever; that just doesn't fly in this system. It's breaking those cultural norms, I would say is still a big challenge.

[0:17:50.0] JM: We've done a couple shows on "stateful Kubernetes applications." I think a stateful application, you can generally define as something you attach storage to, like container attached storage, maybe a persistent queue, like a Kafka, or a Redis. Tell me if you agree with me or not on the definition of a stateful application and give me your perspective on how hard is it to run stateful applications today.

[0:18:23.1] RS: Yeah, I think you started to describe – I'll just invent a term here, like lower level stateful applications, which is basically you're taking a either a pod in Kube land, or a VM somewhere else and just attaching a block storage device to it. Then it's like, "Oh, this is really good for a simple PostgreSQL database or something." Or store your data there and that actually ends up on some remote block storage somewhere.

Then there I think there's higher-level stateful applications, which are what you started to describe as the messaging queues. These are replicated databases, things that might not depend on disk per se, but storing things in RAM and doing synchronization in some other mechanism have to app layer. Both of those –

[0:19:05.0] JM: Right. Kafka does keep stuff in RAM, doesn't it? Like UX, okay.

[0:19:07.7] RS: Things like etcd for example, primarily store things in RAM. It gets flushed to disk on the right, so you have that durability. Mostly, what you're doing is holding it in memory, just things like Prometheus, or holding that stuff in memory as well.

[0:19:21.8] JM: Does etcd – it maintains three different copies in RAM for each piece of data, and then when a write is only completed if it's on three different copies in the RAM?

[0:19:33.6] RS: It's write-ahead log. Three, five, seven, nine are the typical numbers that you'd run those in. It basically keeps all the data in RAM, but it flushes it to disk. That only when it flushes itself to disk, it comes back saying that that disk write actually happened. Is it then sent to all the other followers in the cluster saying, "Hey, this was committed." This is so that if the process goes down, you're not losing your quorum, or when it comes back up, you can – the write-ahead log needs to stay intact.

This is one of the interesting things about etcd, is especially and this interacts with everybody because everyone's running Kube, is etcd is only as fast as the slowest core members. If you think about your path to – you have a network request going over into the etcd cluster over a network. It needs to be ingested by etcd, written to disk. That disk might actually be a remote block storage device, that write needs to come back and then be transmitted to all the other followers, once again, over that cloud network for it to be written successfully.

Typically, if you're all in the cloud this happens very quickly and it's all good. 1 or 2 milliseconds, depending, 10 milliseconds. We see a lot of customers, especially of OpenShift that want to run what we call stretch clusters, which is having nodes in two different complete zone – not even just availability zones, but data centers in maybe in New Zealand and Australia, for example.

Now if you've got your etcds, if your quorum is on one side of that, it's really fast. If one of those nodes fails and your quorum now stretches, etcd becomes really, really slow, because it's only as fast as the slowest quorum member, or as the fastest on the slow side of the quorum. That's an interesting thing where these types of databases have all this logic at the application layer that are impacted when it's not just like, "Oh, attach a block storage device to it and call it done."

[0:21:21.9] JM: There's a lot that can go wrong. With something like an etcd, there is a lot that can go wrong. If you were trying to run etcd on Kubernetes, there's stuff that can go wrong. People do run etcd on Kubernetes, right? Which is weird to think about, right? Because you need a Kubernetes – you need etcd to run Kubernetes in the first place. In addition to that, people will run other instances of etcd on Kubernetes.

[0:21:45.4] RS: Yeah. I've heard that some of the cloud providers actually have whole Kube clusters that only run control planes for other Kubernetes clusters. You might have 5,000 etcd databases on one cluster and 5,000 schedulers and things like that, which is interesting.

[0:22:01.6] JM: Is there some economy of scale to co-locating all of your etcd clusters as a cloud provider?

[0:22:09.6] RS: No, because it actually is probably tough on your disk performance, because these things are flushing writes to disk all the time. They need to manage these applications just like anything else. They need APIs for life cycling them. They need them to get rescheduled when the node fails and all that stuff just like you do. think that's why they do it, is tooling that they're very familiar with.

[0:22:29.7] JM: Okay. Let's ease our way towards the operator conversation. Let's say I have a Kubernetes cluster. I want to run a distributed systems tool on it. Maybe I want to run Kafka on it. Maybe I want to run an etcd on it. What primitives does Kubernetes give to me that might make it easier for me to run something like a Kafka?

[0:22:54.3] RS: Yes, you've got all the objects that are in Kube at your disposal. You think of that as your toolbox. You've got stateful sets for doing some of the very simple staple workloads, like we talked about. Stateeful sets basically give you a run this pod and then wherever that pod goes, attach this storage to it. If the pod moves from node to node, move the storage from node to node with it. That gives you a durable place to write some data, for example. You've got load balancing primitives for doing service discovery and load balancing across all those nodes in your cluster and you've got some auto scaling primitives, you've got secret rotation and config rotation and management. All these things, you can start to construct these pretty complex distributed systems. You just need something to glue it all together, basically.

[0:23:38.0] JM: What is a Kubernetes operator?

[0:23:40.4] RS: Kubernetes operator is taking the expertise of running a Kafka. All the expertise that it takes, the knowledge of the community that builds Kafka to install it, fail it over, to scale it up, scale it down. Is building that into a piece of software that uses that toolkit I just described, all the Kubernetes primitives, to make that happen specifically on Kubernetes.

To run Kafka, you might need to start these different tiers of services and then have them discover each other, elect a leader and start replicating some data. Maybe you want to back up some data somewhere else. All those things can use that Kubernetes toolkit to do and then you need something to orchestrate that and that is the operator.

[SPONSOR MESSAGE]

[0:24:28.4] JM: Buildkite is a CICD platform for running scalable and secure continuous integration pipelines. Buildkite helps you keep your builds fast and reliable, even as they grow large. Buildkite’s web UI and APIs are fully managed, well-documented and backed by great support and SLAs.

Teams can easily set up and maintain their own build pipelines and get help directly from Buildkite support. Build configurations are checked into source control and it works with github and github enterprise, GitLab and Slack workflows. There's also support for Webhooks, GraphQL and plugins, letting you extend Buildkite in new ways.

The Buildkite agent is open source, written and Go and you run it within your infrastructure. It's under your control, so you can be sure that the source code and the secrets don't leave your infrastructure. There's an AWS cloud formation stack to get you started and it auto scales from zero to hundreds of agents. Or you can deploy it to a Kubernetes cluster, a cloud provider, bare metal hardware, or a cluster of Mac OS machines.

Visit buildkite.com/sedaily to learn more and see how Shopify used Buildkite as they scaled from 300 to 1,200 engineers. They migrated between cloud providers and they kept their build times under 5 minutes. Check it out at buildkite.com/sedaily.

Thanks to Buildkite for being a new sponsor of Software Engineering Daily. It's always nice to see new CICD platforms, such as Buildkite.

[INTERVIEW CONTINUED]

[0:26:25.7] JM: What problems does the operator pattern solve? I see it is almost like a two-sided thing. It solves a problem for the vendor, the application vendor. If I am selling Kafka to somebody, like if I'm Confluent and I'm selling you Kafka, you're going to need to install it, or we're going to need to help you install it. It also helps the application developer who is installing one of these things.

Whether we're talking about the person who is packaging up an application to help people – to help it get distributed, or the people who are actually installing one of these higher-level pieces like Kafka, it's very helpful. That's the problems it solves. Actually, let's talk in a little more detail about what it takes to create one of these things. Let's say I'm Confluent, I want to offer a Kafka distribution. What goes into specifying that, or building that operator?

[0:27:29.8] RS: One of the key things about distributed systems in general is you need to get your state from a central source and this is where Kubernetes builds on etcd. What operators do is use the Kubernetes API for their source of central state. You basically, just like all the Kubernetes components work under the hood, you have this idea of a desired state and the actual state. The either human operator inputs some desired state for this is a production Kafka, so scale it out this way and keep it HA or whatever. Then the operator is actually going to go make that happen.

What it’s going to do is translate your very high-level desires into a bunch of Kubernetes objects and desired state that's going to submit to the Kubernetes API. Then the Kubernetes API is going to make it happen. On initial install, nothing is going to exist, and so it's like, “Oh, I need to go make these stateful sets, these deployments, make these secrets wired up to these pods, etc.”

What you're doing there is programming that reconciliation loop between actual and desired state. If you see that one of your pods went away because the node failed, for example your operator says, “Hey, desired state doesn't match the actual state. What do we need to do here? Oh, I need to go make a new pod somewhere else. Let me go schedule a new pod out.”

Where this starts to get above Kubernetes primitives is a deployment will do that for you, or a stateful set will do that for you. What if you need to rebalance data during that? “Oh, our shards are uneven now, let me go re-even those out.” That's the type of logic that you can put into the operator.

[0:28:59.8] JM: When I think about why people like Kubernetes, a lot of it has to do with the declarative format. I can declare how many nodes I want at all times and declare a bunch of other things. When you're describing setting up a Kafka cluster and all the things that I want to set up around a Kafka cluster using the operator pattern, can I still use declarative syntax, or do I need to include some imperative logic?

[0:29:30.5] RS: It's still all declarative, just like, these are Kubernetes objects at the end of the day. What we do is build on the extension mechanism in Kubernetes, which is the custom resource definition, CRD. You're still interacting with the same Kube API with Kube Cuddle, or any other tooling that you like and you're submitting objects that are now, instead of a deployment or a stateful set, the object type is a Kafka cluster, or a Kafka topic. It's still very declarative. That is you're inputting the desired state and then the operators then comparing that input with the current state of the cluster.

[0:30:02.9] JM: Okay. Could we go a little bit deeper? I know you're probably not Kafka, like a total expert, but a little bit deeper into what – let's say, I would need to set up in order to build an operator for Kafka, or maybe if there's something you're more familiar with, like Redis or PostgreSQL. I want to better understand what I need to do to configure this thing.

[0:30:23.1] RS: Yeah, maybe let's step back for a second and think about if you picture this operator as your best employee, your best SRE you've ever seen, your SRE, your hero, your ops hero, and just picture that person when you're installing one of these pieces of software, we'll get back to building in a second. When you're installing it, having somebody over your shoulder, you're in a config file and you're setting some parameters and it's like, “Oh, no, no. That one and that one conflict with each other. You can't do that.” You're like, “Oh, thanks ops person. You're great.”

Then when you're scaling something on it's like, “Hey, hey, hey. Remember to do that thing before you do that.” It's having this this hero that's always, watching protecting you from yourself. That's because if you're not an expert at this piece of software for example, you just can't know all the primitives and iterations about do this before that and all these caveats.

The operator has that logic built-in from the experts of that community; if you're a Kafka committer, or a person that works on Cassandra, or somebody in the Tensorflow community, those types of things. Those communities then build that operator. They need to express all those rules, if you will, whatever would be in a wiki page, or a run book, or something like that, into their operator.

If you think about all the steps that need to happen to either elect a new master of a PostgreSQL cluster, or set up that replication in the first place and do your – or your MySQL Galera, any of that, it's that type of stuff that you need to bake into the operator. You need to think through, given nothing to something how do I install this and set it up correctly. Then what are the really important parameters that needed to be checking constantly to make sure that they don't regress, either through a system failure, or a human is trying to come in and mess with something that they shouldn't, that the operator can go back and correct that behavior. That's why I mentioned, they're looking over your shoulder, they're always watching for the correct configuration values and things like that.

[0:32:04.4] JM: There's a tool within the Kubernetes ecosystem called a helm chart, which is for doing, I believe installation pack – it's a package manager for Kubernetes?

[0:32:15.9] RS: Yeah, it is basically a templating mechanism that you basically process these templates with a set of input and then out pops a bunch of Kubernetes manifest, that are then submitted on to the cluster. An operator can do all of that behavior and more. The problem that I see in helm is that there's not something – there's not this desired state loop that's going on, and so it's like a one-and-done submission.

There's a human in the loop, where if you trigger that via Jenkins or something like that. Once that's done, there's nothing looking for those objects. Whereas, an operator is constantly running. It's a long-running process. For example, you can build an operator out of a helm chart, but we actually have this process that's constantly running, looking for is this in the correct state? Is this in the correct state, and making changes accordingly? Then you can get of course, more advanced from this helm chart into very advanced things, like the Kafkas and the PostgreSQLs.

[0:33:05.7] JM: What's the runtime model for operators? You're talking about your system is going to get monitored, right? Your Kafka cluster that you have an operator for is getting monitored somehow, so the operator system that is built on top of Kubernetes is able to run some reconciliation loop and it also has insight into what's going on in the Kafka cluster so they can fix things. Tell me the runtime model for the operator system.

[0:33:34.9] RS: Yeah. You have one operator that's running on the cluster and these can either watch a specific namespace, or the entire cluster for all these Kafka objects and discover them that are the clusters. Then that is watching the Kube API for all of the objects that that operator has started up and is managing. That's one way. You can just use your Kubernetes toolkit and just Kubernetes APIs and get very far down the road.

What you can do is we have this concept of a maturity model for operators. When you get further into the maturity scale, you can actually start instrumenting your Kafka pods and knowing exactly the state of that application itself. Do you might know the number of topics and they're the depth of the messages in them and you might ought to start auto-tuning the cluster based on that, or you might use Prometheus to monitor some of the resource utilization and other requests per second of some of these topics and then acting on that data. That's where these start to get really, really powerful. That's all very much dynamic runtime stuff. It's just reacting to what the cluster is doing, whether those are a human tweaking values and scaling it up, or just more traffic coming in and changing it accordingly.

[0:34:45.7] JM: The scale-up example, can we go a little bit deeper into that? Let's say at a high level, there are more users using my website, right? They're logging in, they're using more sessions, so I need to scale up my application. That includes increasing my application servers, and maybe scaling up the database. If I've got a Kafka cluster that's monitoring my user events and stuff and the user write traffic increases rapidly, my Kafka cluster is going to need to scale up. How is the operator recognizing that my Kafka cluster needs to scale up and then what's happening?

[0:35:25.1] RS: Yeah, so working back from there's going to be a trigger event of some sort and whether that is either requests per second coming in, or if you've instrumented your application and it's got a very specific memory mapping of like, “Hey, I've tripped over this threshold, something needs to happen.” You've got a sharded session store. “Oh, we got more sessions coming in, or we had a bunch of new users create accounts, so we need to now reshard this thing.” “Okay, we've recalculate all the shards and we know that, oh, okay we need to boot a new shard and we're going to move a bunch of data over.”

You start. You create that pod, “Oh, we need this rate limiting proxy in front of it, so I know to put one of those in too. Okay, now let's register that with service discovery, so that we can start getting traffic into our new shard here.” Then health checks come up and okay, our new shard’s online, let's start doing the actual data movement to move the data over from the existing sessions. Then oh, look, now we've calculated that when we do a backup of this in an hour, we're actually going to go over the size of our persistent volume where we've been storing our backups. Okay, so let's now resize that volume so that we're going to be able to fit that in.

It's a cascading set of things that could happen when one of these events, these thresholds cross, and you need to start doing a bunch of stuff. Now the operator can do all this manually, or I mean, automatically so you're not doing this manually. You would never know that this even happened if you have the world's best operator, for example. These are things like, the system I described is roughly similar to a project from Google called The Test, which is highly scalable MySQL. They basically do have a series of proxies that direct you to shards of MySQL and have a ways for scaling it out. They have an operator that they built that does this. These become pretty complex really quickly, but this thing runs MySQL at the scale of YouTube. It's pretty high-scale.

[0:37:12.4] JM: Yeah. Yeah, we did a few shows about that. That model that Vitess takes to MySQL, that's broadly applicable to stateful applications, or stateful things, stateful modules, like Kafka.

[0:37:27.6] RS: Yeah, stateful distributed systems. Yeah.

[0:37:29.5] JM: Wow, interesting. Okay, so listeners to this show can listen back to the Vitess episode. I probably cut to the chase a little bit too quickly here, because I think we skipped over the real value of the operator pattern to the average developer, which from my point of view is it gets us closer to ironically a world where we don't need to think about Kubernetes at all, right?

I was able to build an application eight years ago and I wasn't thinking about Kubernetes. I want to go back to that world, right? I want my database. I want my business logic in my Node.js application. I don't want to think about container orchestration. Explain why the operator pattern gets us closer to that world.

[0:38:19.3] RS: It's all about self-service at the end of the day. About that cloud-like experience that folks love, hence wide AWS and other things have taken off ginormously. It's that I want to start a new prototype and maybe we used a relational database, but we want to swap in a NoSQL for a prototype. Let me just see how this is going to work. It's the idea that you can just go quickly get a database without being an expert and let's say, Couchbase. Be off to running prototyping it and be done with it an hour or two, or a day, or a week, or whatever. Then come back and say, “Hey, we're ready to do this thing.” It makes sense that frictionless experience is what we're going after. Not just, “Oh, let me go find this blog post about how to do Couchbase. Okay, I understand that. I'm going to go dig into the configuration files. Okay, that's great. Now let me go set up a load balancing and write all these Kubernetes UML objects and blah, blah, blah, blah, blah.”

The operator can do all that complexity for you. You just say, “Hey, I want an HA Couchbase cluster. Put it in my namespace, and so it eats against my cluster quota and let's go to town.” The operator can do that if your admin says, “Hey, everybody on this cluster can have Couchbase clusters.” Then you can do it in this self-service manner just by using the Kubernetes API; create a new object that is Couchbase cluster and then boom, you get it. It's increasing time to market, or time to prototype in this example versus having to –

[0:39:37.4] JM: Decreasing.

[0:39:38.4] RS: Or decreasing, excuse me. You don't want to have to fight the system to get the resources that you need to do your job.

[0:39:45.1] JM: What's my experience for deploying an operator? If I wanted to play a database, or deploy a Kafka cluster there?

[0:39:53.8] RS: Yeah, so if the operator is already installed, which is typically more of an admin task, if you're just an end-user engineer, then you – like in OpenShift for example, we have great UI for browsing the capabilities that an admin is provided to you. This might be Kafka clusters, Mongo clusters, Couchbase, PostgreSQL. You can go in and find all those. Then it's just one Kube YAML object that is this high-level config that you pass in and then the operator knows to – if you don't set a value to default it to something smart and things like that.

You just say, “Oh, for a Couchbase, give me a cluster size 3. Then here's the secret for the default account that I want to use stored as a Kubernetes secret,” then you're off and running. Of course, you can tune all these, like bucket settings and stuff like that if you want to later on. That's the power of the operator. You don't need to be an expert in Couchbase. Obviously, you need to know how to use it and connect it to your application, but you don't need to care about how those components discover each other when they're booted, or which ones need to be restarted first when a software upgrade happens, that type of thing.

[0:40:49.9] JM: Red Hat created this thing called operator hub, right? This is a place where people can browse these operator YAML files, right? Who are writing those operators? Is it like, Confluent goes and writes a Kafka operator, or PostgreSQL, goes and writes a PostgreSQL operator, Couchbase goes and write to Couchbase operator. Where are those operators coming from?

[0:41:13.5] RS: It's a mix. What we're going for with the operator concept and operator hub IO is to go to the experts. We want to get a Mongo operator from MongoDB Inc., because they are the experts. It doesn't have to be a commercial company. We want to go to the Tensorflow community for a Tensorflow operator, because they're the experts. All that operational expertise that it takes to run and install and scale and failover those pieces of software, we just want to go to the expert.

In the software engineering realm, this typically ends up being backed by a commercial company. Red Hat also wants to have certified operators, so that our enterprise customers can have somebody to turn to when they want support on your Redis cluster, for example. That doesn't mean that there's not an open source alternative as well. We have a path for all kinds of communities. Then we have a bunch of products and open source that we do as well at Red Hat internally that are operators.

[0:42:06.8] JM: What's been the adoption for operators at this point?

[0:42:10.8] RS: It's been really powerful. If you were at KubeCon last week, or you've been at previous KubeCons, folks are talking about it and it's – we've reached this phase in the adoption of Kubernetes, where we've got our stateless apps down, we've got our stateful applications that are pretty simplistic with just stateful sets, like we talked about earlier. Now we're into this realm of running pretty complex things on top of it. Your banks and e-commerce shops and stuff now are pretty complicated.

You've got these doing rate limiting and quality control things. You've got machine learning on an audit logs. You've got your scale-up web tier. You've got maybe a NoSQL caching tier, like Redis. Then you've got your super-persistent PostgreSQL or Oracle databases maybe, if you hate yourself. All these things are really complex and they've got teams running each one of them. Nobody can know the operational complexity of running all of that, especially when you need to run it in 10 different GOs and then eight different staging environments. You've got a user acceptance environment and you need to stand up all these different stacks. Then your slacking people, “Hey, what's the latest version of your thing and where do I get your bash script to install it or something?”

It's nuts where you can just say, “Hey, here's our operator and here's where they're published. You can get a local copy of our front-end stack, the latest copy, here's the stable one, here's the beta one, etc.” You can start to construct these really complex applications. It might just be 10 operators at the end of the day, that type of thing.

[0:43:39.9] JM: That's great, because it sounds like there is not – sometimes in the tech industry, you get inertia, right? You get look, I've been describing why I've been standing up my Kafka cluster manually for a long time. I don't want to get involved with this operator thing. It disrupts my inertia. That's not happening as much?

[0:44:02.7] RS: I mean, it's hard to say because there's a whole bunch of Kafka clusters out there. The nice thing is that at the end of the day, you're still talking into Kafka. You're not changing – you’re changing your deployment paradigm, but not your consumption paradigm. Then hopefully at the same time, unlocking more self-service for your engineers. You're freeing up their time from fighting, getting this stuff deployed and then as well as day-two operations. Now they don't need to be an expert in digging through all this stuff at 3:00 a.m. when they've got some outage on the coffee cluster, because the operator can go in and knows how to tweak those settings to bring it back up to production-ready, things like that.

[0:44:39.1] JM: What are the shortcomings of operators today? There any bugs, or usability issues?

[0:44:48.1] RS: I think it's just wrapping your head around this new model of the world where you're sourcing your state from the Kubernetes cluster and you're interacting with these applications at a higher level, just like you would a cloud service versus at the individual VM level that somebody might be used to. I just SSH into this thing and drop into a SQL command prompt and start screwing around with the database, or tweaking some of its configuration.

You're entering from a higher level through the operator, so that you're protecting yourself and preventing two people from making the same change at once and that type of thing. It's a little bit different model of working, but just you can't screw around with your RDS database at Amazon, you can't SSH into those machines and start poking around. I think it's a new model of the world that people are coming around to pretty rapidly.

[0:45:35.8] JM: Wait, I'm sorry. Maybe I misunderstood the last point. I set up Kafka with my operators, you're saying I should just never access those clusters directly, or –

[0:45:47.8] RS: Well, it's giving a more rigor and making programmatic changes, versus one-off manual changes to components that might have ripple effects through the system that you're not even aware of. Having a higher level piece of software that is looking out for those changes basically. It's not that you can't reconfigure things, for example.

[0:46:07.6] JM: Any interesting operator case studies? We talked about Couchbase, or Redis or Kafka. Anybody who's developed an operator that you've talked to at length about their experience crafting it and getting it deployed to operator hub and consumed by the public?

[0:46:26.1] RS: Yeah. We had a community event right before KubeCon a week or two ago. I had chaired an operator panel and folks were talking about building their operators. One of the questions I asked was what did you understand about your software by building this operator? The idea that you need to – maybe you had some hard-coded configuration values that you wanted to make more tunable, and so you fix that problem and brought it up to a higher level.

One group shared, they actually found that their leader election bug, or their code had a bug in it, where it didn't work correctly in all different edge cases. Because they started – they were thinking through how do they implement that in the operator and discovered this. I think that's interesting. It's forcing you to tease apart your application hierarchy and architecture a little bit through the process of this, which I thought was interesting. That's a success story on the building side of it is finding some bugs on the implementation side, or running one of these operators.

We like to point at one of our old CoreOS customers was wanted to run Prometheus and that was their blessed tool for running container native monitoring, because it just understood what was going on in their Kube clusters really well. This is an organization that lets folks make their own technology decisions and everybody started just coalescing on Prometheus. What they did is install the Prometheus operator on their clusters, and so each team can run their own Prometheus cluster and change how much RAM it's being used and how long it's going to keep its metrics and different replication settings and stuff like that, but they all use it through the operator.

They were running I think three 350, 380 different Prometheus operators, Prometheus clusters via operator for all their different environments and teams and this, that, the other development environments, which is nuts. Show me somebody that can run a database tier, run 300, 400 of these things with basically no humans involved. These folks were providing very high-level config in getting these production ready Prometheus clusters out, with no ongoing maintenance tax, or understanding exactly how Prometheus works under the hood. They're not following the get commits and the github activity and things like that. They're just getting these ready-made Prometheus clusters.

[SPONSOR MESSAGE]

[0:48:40.8] JM: Software Engineering Daily is a media company and we run on WordPress, just like lots of other media companies. Although it's not just media companies that run on WordPress, I know of many organizations that manage multiple WordPress sites. It can be hard to manage all of these sites efficiently.

Pantheon is a platform for hosting and managing your WordPress and Drupal sites. Pantheon makes it easier to build, manage and optimize your websites. Go to pantheon.io/sedaily to see how you can use Pantheon.

Pantheon makes it easier to manage your WordPress and Drupal websites with scalable infrastructure, a fast CDN and security features, such as disaster recovery. Pantheon gives you automated workflows for managing dev, test and production deployments. Pantheon provides easy integrations with github, CircleCI, JIRA and more. If you have a WordPress, or a Drupal website, check out pantheon.io/sedaily.

Thanks to Pantheon for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:50:03.3] JM: If I have an operator and a bunch of users have deployed it and I want to update my software, let's say I'm updating the database, like I'm a database vendor, I publish a database – I've published an operator for my database, my users have deployed it. How do I push out an update to them?

[0:50:24.3] RS: This is where something like operator hub is really key; it gives you a conduit to publish new versions of your operator and either have it discovered on a one-by-one basis, or actually push that down to the clusters themselves and say, "Hey, you've installed and subscribed to a stream of updates from MongoDB about the Mongo operator." This Mongo operator has entered in this contract, such that it knows how to do rolling updates of the application itself. What that looks like is Mongo itself builds in all the logic to go from version 1.1.1 to 1.1.2 for example, or maybe even a major version 1.2.0. Package that in an operator and you install that on the cluster and the operator knows how to read the existing databases that you have and say, "Hey, I understand this new version of Mongo, now let me transition these databases."

If it needs to install a new component and wire it up, it knows how to do that and uses the Kubernetes toolkit that it has to do that. Maybe it needs to shut down or deprecate a different component, so it knows how to do that. Maybe deploy the new one before you deprecate the old one. All that logic can be in there. Then take a backup before, or start replicating data, or change these environment variables whatever it is.

The rich part of the operator is you have this place to do all that knowledge. Something like helm doesn't know how to do that. It's just going to blindly create some Kubernetes manifest, but it doesn't really know how to introspect what's there, what do I know about specifically what it takes to upgrade Mongo and then make that happen.

[0:51:53.1] JM: You and I were both at KubeCon this month, May, early May 2019, or mid-May, I guess. Give me your reflections on the ecosystem circa summer 2019. What was your mind changed about, or what new insights did you have from the conference?

[0:52:13.9] RS: I think we're seeing most vendors now have fully embraced Kubernetes. I mean, I think people have been saying for a while that Kube has won the orchestration wars. You're seeing almost every vendor, Oracle's out there talking about how you can run their software in containers. That's when you know you've reached not the long tail maybe, but you're towards heavy and to over the curve of adopters. I think that's really great. Folks are now seeing that you can run basically any software under the sun on Kubernetes containers at the end of the day, or just Linux processes running on a Linux host. Now we've got Windows hosts.

I think the ecosystem is so broad now that if you're going to targeting next-generation infrastructure, your go-to-market is basically on Kubernetes. I think it's cool to see that wave actually happening. This is one of the things that we talk about with our operator partners is if you want to embrace the entire Kubernetes ecosystem and have a great experience for your product, then operator is the best way to install upgrade, manage failover your piece of software, and then you can run on any Kubernetes provider out there, which is really, really powerful.

[0:53:21.5] JM: Well, that's true. Although in many deployments, people have some lock into a particular cloud provider, like IM privileges. Well, I guess if you –

[0:53:33.3] RS: If you're only on the cluster, you're not even interacting with IM.

[0:53:37.1] JM: Wow.

[0:53:38.1] RS: This is the beauty of using Kubernetes as your toolkit. It's not to say that you can't integrate with external services if you want, but if you are just using Kube internally, then you aren't locked into anything. This is the beauty of an operator, is if you want to go to Confluent and purchase their product, you can use it on this cloud provider, you can use it on VMware, you can use it on bare metal, it works the exact same.

[0:53:59.5] JM: They can give you a proprietary enterprise distribution, right? If they want to have –

[0:54:03.6] RS: Yeah, absolutely.

[0:54:04.6] JM: If they want to have a closed source binary, that's pretty cool. Let's say I want to deploy – I do want to deploy most of my resources in a cloud agnostic way, but maybe I want to use Amazon for SageMaker, or whatever. I want to have my application and database located on AWS data center, so that there's low-latency between SageMaker and my applications. Are there well-defined patterns for how to do those integrations? Integrations between this, because what you're describing is the dream, right? The cloud agnostic deployment of resources. How would I interface between those resources and the cloud-specific resources?

[0:54:55.3] RS: It depends what it is. If you think about I'm unlocking a swath of these is just like a hostname and a username and password away. You can have headless services in Kubernetes that point out of the cluster somewhere and you can have a secret that stores credential information. Then you can start talking to a huge swath of these, whether it is a database from Azure, or a database from Amazon, for example and your application knows nothing about it. You're using Kube primitives still.

Also, your applications can know how to very specifically go talk to those resources, if that is something that you want to do. We've seen folks have operators that go orchestrate third party DNS APIs, or reconfigure hardware load balancers and things like that. I think there's a ton of choices there for how you want to integrate. One of the concepts that we are currently pushing forward in our operator special interest group is the idea of service bindings, which start getting into what you're talking about.

If you're familiar with the open service broker API, there's this concept of you've got a front-end and a back-end, and they need to share a credential to talk to each other. That is called a binding. We want to bring that into the operator world, where you can have two operators working together to accomplish a common task. Where you see this is pick some random off-the-shelf piece of software that might just say, "Hey, I need a relational database to work."

This is where you would go file a ticket with your DBAs and go get a PostgreSQL cluster or something like that. Now it could be either go do that, or, "Oh, I actually know how this PostgreSQL operator works," and use the operator to go fulfill that. That's I think the new model that we'll get into is here's my piece of software, I require Redis database and a PostgreSQL database. You have these two operators installed already. Boom, I know how to go get databases. They're up and running in 10 milliseconds, that thing.

[0:56:41.2] JM: Does AWS take part in the operator sig at all, or who's in there?

[0:56:49.7] RS: It's a bunch of random folks from some of the partners we've talked about, like Redis and Mongo and Couchbase, and some folks from IBM and a smattering of smaller companies. Amazon is definitely active in the operator space however, because they have a Amazon service operator.

What this does is really cool, this is talking about – this is taking the interacting with external resources to the extreme. They model an S3 bucket as a custom resource definition that their operator is managing and listening for. When you make a new S3 object, it actually will go on the back-end, go make a real S3 bucket and will –

[0:57:24.2] JM: Oh, my goodness.

[0:57:24.9] RS: - turn stuff back about it. It works with a number of their services, I think 10 or 20 of them.

[0:57:30.0] JM: That's like extending their lambda-friendly, event-driven model to your Kubernetes.

[0:57:34.9] RS: Right into Kube, which is really exciting. Even if you did want a cloud database, you can go make an RDS object, then we'll go make that RDS database for you and then return information back about it. This is its name. This is how you address it, etc. What this allows you to do is just work off a Kube API, but then integrate with the cloud where it makes sense. Hopefully, we'll see other cloud providers building this type of functionality in, so that you are just using – your Kube R back is writing to the Kube audit log and then you're getting cloud resources out at the other end.

[0:58:04.7] JM: Coming back to your reflections on the technology industry, what do you think of Google managing to do this jiu-jitsu of introducing Kubernetes into the world, then seeing the dynamics between AWS? Basically, now it's AWS – I think of AWS as the – I don't know the Avengers ecosystem too well, but I think there's some gigantic, titanic alien or something and then there's the Avengers and they're all – they're led by Captain America. You've got Google leading the Avengers and it's all the other people and just marshalling versus AWS. AWS is so friendly with the ecosystem in certain ways. It's certainly entertaining for me to watch. What have been your reflections on that competitive dynamics?

[0:58:55.6] RS: I think what's great about the Kube community in general is it's showing us how we can all cooperate on open source and still put forward maybe other strategies that we have, or other business goals that we have, but still cooperate together.

[0:59:08.6] JM: Even if you're AWS.

[0:59:09.9] RS: Yeah. I mean, if you think about Google, VMware, Microsoft, AWS, all have their hooks into their platforms for running Kubernetes on their infrastructure. You make ELBs on Amazon, you could add an Azure load balancer on Azure, etc. It's really a powerful way to plugin, but then also feel you have the correct amount of tooling that is agnostic in that a hybrid cloud scenario, where you actually do want to use a Kube API, not an Amazon API or an Azure API to get your applications that would –

[0:59:37.8] JM: Which is what we all want as engineers.

[0:59:40.2] RS: I think that's interesting seeing that happen. There's all these open source licensing shenanigans that are happening right now with different databases and things like that.

[0:59:49.0] JM: You got a perspective on that?

[0:59:51.4] RS: Just that the Kubernetes is showing us how we all can work together and everybody can make money and get their needs met without having to resort to being really hostile. That comes back to the project is run really well. It has a neutral governance of the CNCF and why that's really important there. There's a place for these companies to go to if they have a disagreement to start figuring it out with the community as part of that as well. It's not just these two companies talking behind closed doors and things like that.

I think you need to go to market with somebody that is not going to clobber your business at the end of the day, if you're one of these database companies, or one of these open source communities that are just going to see Amazon reap all the profits off of your hard work is a pretty hard thing for them to stomach.

[1:00:34.4] JM: Oh, come on. Do you really feel that way, or is that you speaking as a Red Hat employee?

[1:00:38.8] RS: I mean, I think you're seeing it. I think these companies are – if they don't become profitable, then they're going to go under. Though it's like, what happens to that community that they built? Do they go start working for Amazon now? That wouldn't be that bad, I guess, but as long as Amazon continued that community.

[1:00:53.4] JM: Sure, but their indignation. They're so indignant. They're so, "Oh, this is not – and what Amazon is doing is not in the spirit of open source." I don't like that argument, because we don't have morality of what open source is, right? Open source is just open source. They're making these arguments that for with righteous indignation. It's just, come on.

When you read these blog posts of the open source companies and they're saying that Amazon is eating up our business model. It's like, well, okay, that's a business problem. It's not a morality problem.

[1:01:34.5] RS: Fair. I don't think it's necessarily – it's not a morality problem, but we need to figure out how to make the long-term health of our space in general, not just be dominated by one or two cloud players if they're reaping all the profits in the ecosystem.

[1:01:48.4] JM: Is that the fault of the cloud players, or the fault of the ISVs who raised money in a certain capital structure and placed all the wood behind the arrow of their single open source project?

[1:02:00.7] RS: Yeah. I mean, that's where you need – it's a pivoting into either services, or hosted offerings or other things like that. Yeah.

[1:02:07.5] JM: Absolutely. Another business, something. That's what irritates me about it is if I was a developer at one of those companies, I'm like, “Really, management? You're going into the licensing business instead of the innovation business?”

Anyway, a final question. CoreOS was acquired by Red Hat. What have you learned about acquisitions?

[1:02:30.9] RS: It's interesting. We've –

[1:02:32.3] JM: Then right now, it’s acquired by IBM.

[1:02:34.1] RS: We actually have a current acquisition that is ongoing. I think one interesting thing that I've seen at Red Hat about the IBM acquisition is just that everyone's eyes opened up to what that experience is like. Red Hat has acquired a number of different companies, Ansible and 3scale and a bunch of others.

Now that they've been acquired, they understand what they did to those companies where you just wake up one day and your whole world has been flipped upside down. Typically, maybe not in a bad way, but it's an exciting thing. Now you're a part of this larger organization and you didn't really have a choice in that. All the Red Hatters I think are now experiencing that and know what that's like, which is interesting.

I've been really happy with the integration of CoreOS into Red Hat. Super proud of our team, because all of our technology is basically forming the basis of OpenShift 4 and a lot of the other strategy rolling out of some of the business units in Red Hat. Container Linux was our container-focused Linux distribution that our namesake used to be called CoreOS. That now lives on in a new operating system from Red Hat, a new commercially supported one that's on the pedestal with RHEL and Fedora, which is pretty awesome.

Then all the technology that we worked on in Tectonic, which was our Kubernetes distribution, forms the basis of OpenShift 4. All the all the things we had about large-scale machine management and driving management of those machines from the Kubernetes cluster now forms OpenShift, which is a ginormous business.

[1:03:59.7] JM: Wow. So validating.

[1:04:01.3] RS: Yeah. I'm pretty proud of the team for all that. Then a bunch of our open source work, etcd –

[1:04:05.8] JM: Wait, a lot of stuff from the lessons from Tectonic and all that stuff in CoreOS, you got to basically bring that to life at scale with OpenShift 4?

[1:04:16.9] RS: Yeah, yeah. OpenShift is a extremely critical division – IBM's CEO, Ginni was at Red Hat Summit and mentioned OpenShift a number of times.

[1:04:26.4] JM: By the way, talking about an open source company that was able to find a second product and move into the innovation business, or move further into the innovation business, rather than the licensing business, Red Hat is a perfect example.

[1:04:41.2] RS: It's our expertise in Linux. We always say that containers are Linux at the end of the day. We're just doing that in a clustered manner. Folks have been doing very large-scale rail deployments for a very long time. This is bringing that networking and orchestration layer to it that we found that we needed with the container Linux and CoreOS machines, like we talked about in the very beginning. Managing Linux at scale is really hard and need to be architected a little bit. That is the OS that we produce that we use inside of OpenShift for OpenShift 4.

[1:05:09.8] JM: Nice work. It must be really validate. I mean, it's interesting, and we'll wrap-up in a sec, but it's interesting the Kubernetes ecosystem, what's cool about it is the collaboration and the coopetition. One side of that is acquisitions. Acquisition is a nice form of officiating some collaboration. It's like, you see the collaboration in these special interest groups, or user groups, or forums, or whatever. When the collaboration gets tight enough, you could just have these acquisitions and other forms of consolidation.

[1:05:44.2] RS: I think that was the exciting thing and why that was the best place for CoreOS is that we already worked with all these engineers in the upstream community. They contributed some to etcd and did a lot of scale testing on etcd, because OpenShift are some of the largest clusters out there Kubernetes-wise. When we moved from V2 to V3 of the etcd API and moved to a GRPC from an HTTP API, a lot of that was driven by scale. That was input from Red Hat and Google on how do we get Kubernetes to the next level of having 10,000 nodes in a cluster, etcd is going to start being the bottleneck there.

We collaborated with them on a lot of the type of effort and plugging that in into successive versions of Kubernetes. We worked with all those folks as well on the OCI spec and some of the work that we were doing around our earlier spec called appC. Our CTO, Brandon Phillips was the primary driver behind the scenes of all that OCI work. A lot of organizations took the credit at the end-result level, but he was the one driving that spec and it was a spec that was driven from CoreOS's announcement of our rocket container runtime that really kicked that whole thing off.

[1:06:47.5] JM: Rob, it's been great talking to you. Thanks for coming on the show.

[1:06:49.3] RS: Absolutely. It was fun.

[END OF INTERVIEW]

[1:06:54.6] JM: DigitalOcean is a simple, developer-friendly cloud platform. DigitalOcean is optimized to make managing and scaling applications easy, with an intuitive API, multiple storage options, integrated firewalls, load balancers and more. With predictable pricing and flexible configurations and world-class customer support, you'll get access to all the infrastructure services you need to grow.

DigitalOcean is simple. If you don't need the complexity of the complex cloud providers, try out DigitalOcean with their simple interface and their great customer support. Plus they've got 2,000 plus tutorials to help you stay up to date with the latest open source software and languages and frameworks.

You can get started on DigitalOcean for free at do.co/sedaily. One thing that makes DigitalOcean special is they're really interested in long-term developer productivity. I remember one particular example of this when I found a tutorial on DigitalOcean about how to get started on a different cloud provider. I thought that really stood for a sense of confidence and an attention to just getting developers off the ground faster. They've continued to do that with DigitalOcean today. All their services are easy to use and have simple interfaces.

Try it out at do.co/sedaily. That’s do.co/sedaily. You will get started for free, with some free credits. Thanks to DigitalOcean for being a sponsor of Software Engineering Daily.

[END]