EPISODE 518

[INTRODUCTION]

[0:00:00.1] JM: Over 12 years of engineering, Box has developed a complex architecture of services. Whenever a user uploads a file to Box, that upload might cause five or six different services to react to the event of the file being uploaded. Each of these services is managed by a set of servers, and managing all of these different servers is a challenge.

Sam Ghods is the cofounder and services architect of Box. In 2014, Sam was surveying the landscape of different resource managers deciding which tool should be the underlying scheduler for deploying services at Box. He chose Kubernetes because it was based on Google's internal Borg scheduling system. For years, engineering teams at companies like Facebook and Twitter had built their own internal scheduling systems modeled after Borg. So when Kubernetes arrived, it provided an out-of-the-box tool for managing infrastructure, like Google would. That's why Sam chose Kubernetes over other options.

In today's episode, Sam describes how Box began its migration to Kubernetes and what the company has learned along the way. It's a great case study for people who are looking at migrating their own systems to Kubernetes, and if you're interested in finding all of our older shows about Kubernetes, which we've done 20 or 30 of at this point, you can download the Software Engineering Daily for iOS or android. It has all of our old episodes.

With that, let's get to this episode of Software Engineering Daily.

[SPONSOR MESSAGE]

[0:01:37.8] JM: Azure Container Service simplifies the deployment, management and operations of Kubernetes. Eliminate the complicated planning and deployment of fully orchestrated containerized applications with Kubernetes. You can quickly provision clusters to be up and running in no time while simplifying your monitoring and cluster management through auto upgrades and a built-in operations console. Avoid being locked into any one vendor or resource. You can continue to work with the tools that you already know, such as Helm and move applications to any Kubernetes deployment.

Integrate with your choice of container registry, including Azure container registry. Also, quickly and efficiently scale to maximize your resource utilization without having to take your applications off-line. Isolate your application from infrastructure failures and transparently scale the underlying infrastructure to meet growing demands, all while increasing the security, reliability and availability of critical business workloads with Azure.

Check out the Azure Container Service at aka.ms/acs. That's aka.ms/acs, and the link is in the show notes.

Thank you to Azure Container Service for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:03:04.2] JM: Sam Ghods is a services architect at Box and was one of the founders of the company. Sam, welcome to Software Engineering Daily.

[0:03:11.0] SG: Thanks very much. Thanks for having me.

[0:03:12.5] JM: I want to start at high level of how things work at Box, and then we'll get into a discussion of Kubernetes. So I think most people know what Box is. It's a file storage and management service. Most people listening have probably used it. Go as deep as you want, but what happens when I upload a file to Box? When I am just a user and I upload a file, what's happening on the backend?

[0:03:37.3] SG: Sure. So at a high level, our company started before public cloud back in 2005. So we run and manage most of our infrastructure. We have to migrate some stuff the public cloud, like AWS and a little bit on GCP integer, but predominantly we ran infrastructure. So for the most part, when someone uploads a file, it's going to reach one of our data centers, hit one of our load balancer, one of our frontend load balancers and eventually get routed to basically a mesh of JVM-based micro-services predominantly, so Java and Scala as a mix of different micro-services.

Then those also interact with what we've had for quite some time, which is a PHP monolith, which is now running on HHVM, and the file eventually makes its way, streams to our file storage servers, which are basically large storage array fronted with nginx, and then also a backup copy is sent to AWS as well. The details to that configuration are based on what the customer wants or what consumers uploading the file or things like that. Over time, it's more and more depending on the geography as well, the region that the customer is uploading the file.

[0:04:45.4] JM: That's awesome. There's a ton of stuff there we can dive into. You mentioned all these different services that are going to be touched, and then eventually the monolith — Box was not only started in time when public cloud was not a thing, but it was started when the LAMP stack was the thing, so Linux, Apache, MySQL, PHP, and most companies that started with some sort of monolith, people talk about micro-services, but micro-services are often in addition to the model. It's not like the model gets completely broken down and taken away. It's like the micro-services spring up to the side. So how many services — Do you have any idea how many different services get touched when I do that file upload? Is a lot?

[0:05:27.7] SG: Yeah. I mean, what you said is absolutely corrected. It's really, really hard to replace a monolith with micro-services, and oftentimes it ends up just being impossible due to the pace of development, basically, outpacing the ability to rewrite it, because so many person-years of engineering development has gone in. It's just hard to keep up and still be a productive business. If we wanted to stop developing only of functionality, then it may be feasible, but as a running business, it's hard.

So the best you can do is really focus on allowing new development to be a micro-services first and foremost, and then secondly rewriting or replacing the core really critical pieces, the really important ones and moving those. The monolith almost never in long-standing web companies disappears. It's very, very rare. It's more that you're just trying to minimize the impact of it [inaudible 0:06:20.3].

But to answer your question, for an upload, probably just 3 to 4 micro-services, plus the monolith, and that is changing over time as we decouple things and add more features. There're things like service that handles the encryption and decryption, and then there's a service that actually is responsible for storing the file, and then the monolith still handles some of the core business logic for actually adding the file to your account.

[0:06:45.2] JM: Box first deployed Kubernetes back in 2014, and I think the amount of decoupling is enough of a reason why you would want to deploy Kubernetes, but why did you become such an early adopter, because I think around that time there were a number of different orchestration tools. It sounded like you were already starting to migrate towards Docker and you were just looking for a way to manage all of these Docker containers. Why did you become an early adopter of Kubernetes?

[0:07:15.1] SG: Yeah, great question. So I think there’s kind of two questions in there. One is why did we go to containerization? So early on in kind of containerizations, like more broad development. The second, once we’ve decided on containerization, why did we go with Kubernetes, right? So in the first one, for why we went with containerization, is, again, a lot of the infrastructure had grown up as bare-metal and largely the entire lifecycle of the infrastructure managed by us. Again, the reason was, early on, public cloud wasn’t a thing, and so we started building out our infrastructure. Really, for the first 5 to 7 years of the company, I would say until just a few years ago, public cloud was still almost a nonstarter with a lot of our customers, especially the ones that were the bigger ones and the ones that were really allowing us to grow and have a really big impact.

I mean, our customers just to give you an idea, I mean, they spread across financial services all the way through to governments, and so with that kind of customer base, like really only recently with a lot of development on the encryption pieces of it and key management and things like that has public cloud really become an acceptable solution for us to host our infrastructure in order for our customers to actually switch off to their private infrastructure and move on to a platform like Box.

So as a result of that, like it's very hard for us to take advantage of a lot of public cloud, and so we have a lot of this bare-metal structure. So really, any effort we put towards making our developers more productive, our developers have to be able to hit interface that is agnostic to the infrastructure underneath it. So that's been our main challenge and goal, is that while we may be able to have gotten similar, albeit still not as good, but still similar productivity benefits and cost efficiencies and things by just wholesale moving to public cloud, such as AWS. For, one that would've meant that our developers would have had to have dual interfaces of both our bare-metal instructor and public cloud, which differs quite significantly no matter almost what your bare-metal infrastructure looks like.

So basically that would've been required if we were going to just wholesale move to cloud. So instead, we started to look towards what kind of infrastructure will allow us to abstract a way for developers whatever is going on underneath, whether it's bare-metal of our infrastructure, our own virtualization stack, AWS or than other public clouds, like GCP, Azure and so on. That was really what was the key for us to be like strategic long-term, was that that was really a requirement for us to have our developers and the people building micro-services be agnostic to these underlying layers. That’s why we went towards the route of containerization. Does that kind of make sense in my front?

[0:09:56.2] JM: Absolutely. From what I hear you saying, it was about the fact that you wanted a clean API where you had infrastructure as an API, infrastructure as a code basically, whether or not that infrastructure was in the cloud.

[0:10:09.8] SG: Exactly, exactly. AWS, it is a huge step forward for infrastructure, but at the end of day, it's extremely proprietary. I mean, it locks you in almost from every angle, and as time goes, they are continuing to largely do that.

[0:10:25.3] JM: Because of the APIs.

[0:10:26.0] SG: Because the APIs, exactly. That's really — When you're a developer writing a micro service, there’s a dozen things you have to do to get your micro-service live between deployment, authorization, load-balancing, all sorts of server configuration, artifact depository, there are all these things, and if all those things are tightly coupled to the AWS API, one, just as a business, having zero ability to easily migrate or have different public cloud providers compete for your services is a huge liability as a business. But then also, if there's ever any reason that like whether it's customer focused or productivity focused our cost focused, that you want to use either other public cloud services or, again, your own infrastructure for certain use cases, then that’s almost next to impossible to do when you're coupled to one public cloud services proprietary API.

So that's why we went with containerization. Now the question was, “Okay. If we’re going to do containerization,” which allows you to kind of have this abstraction over whatever the containers are running on, which is really nice, so now the question is, “Well, what — Whether a software platform or a service or a thing that we want to do, to have this kind of layer in between developers and the containers running themselves?” We can call it a PaaS layer. To some extent, you’d call it an infrastructure as a service player depending — I think there’s a lot of — You could argue which pieces go into what layers, but fundamentally something between developers and infrastructure right.

So we looked at a number of things. I think we actually started looking at doing this in early 2014 pretty much right when Cocker came out or started to gain steam. Back then, there was really nothing out there. So we looked little bit at Messo, and the main reason we’ve decided not to do Mesis was because, from our perspective, they were targeting too low of an abstraction down the stack.

Mesos, fundamentally as a software platform, is a framework for developing cloud frameworks. Mesos itself doesn't really provide any user-facing functionality. They're just a bunch of components and APIs for you to build your own user-facing functionality, like your own PaaS layer, for example. So there were a number of competing ones, one was Twitter’s, which is called Aurora. Another one is Marathon, which is promoted more by Mesosphere. Just something like a very fragmented space, and it didn’t allow for the kind of innovation and consolidation of effort to hit one uniform API for infrastructure.

[0:12:58.2] JM: Those were the schedulers that were built on top of Mesos, right?

[0:13:01.1] SG: Exactly, exactly, and it really seemed to us to be like kind of self-defeating, because the whole point of this is to be able to say, “I want to run this container and hit some API that is now universal, so that everyone can kind of consolidate efforts around this new layer.

That's why it’s better than AWS and the other public providers, right. It's now an open universal platform that people can target.

So we looked at some of the Mesos options — We were actually closed to kind of writing our own with this model in mind until, I think, it was Docker Con 2014 when Kubrnetes was announced, and we were like, “Oh! Well, okay. First of all, a lot of our mental model Box of how this thing should look was informed by how most other companies in the valley’s mental model of how it should look,” which was all informed by Borg. At the end of the day, Borg really — Which for people who don’t know, is Google’s internal container orchestration platform that powers, I've heard, anywhere between 90% and 99% of all of Google.

Borg is the largest sophisticated, most thorough, most proven containerization framework in the world, the only problem is it's entirely close-source and will probably never be open-source because of the sheer amount of coupling it has to Google's internal infrastructure. There’s just no way.

So basically what the Kubernetes announcement was saying was — And so by the way though, because Borg was never going to be open-source, Facebook, like basically a bunch of engineers from Google more or less went to Facebook and said, “Hey, you know what? Borg is the best way to run large-scale infrastructure, but we’re going to rebuild it here,” and that's how Tupperware largely came to be, which is Facebook's clone of Borg, more or less. Then Twitter basically did the same thing. Now, Twitter did it in a much more open-source fashion. They’d based it off of Mesos, which I believe was kind of either started or are primarily funded by Berkeley, and then they adopted that and they build Aurora on top, but they were working very, very, very uphill, because Docker not a thing. Mesos, like the other big disadvantage besides it being a framework for framework, the other big challenge with Mesos was it was created before Docker. So now they’ve largely overcome that, but at the time, like there is like a lot of complexity, just sheer amount of complexity between Mesos not really being sure if it owned the containerization layer or not, and not really being sure if it owned the end user developer-facing layer or not.

So Mesos just seemed like this kind of chaotic, just not really because of bad engineering or anything. It was just like in the — It was too early kind of for its time and wasn't able to take advantage of these other opponents. So now you have a situation in 2014 where, literally, many members of the team of Borg itself, Borg and Omega, Omega, which was a kind of a successor to Borg, many of those core team members who have probably more experience than any other containerization work on the planet are now working together to rewrite Borg from scratch in a completely open-source way in a fairly modern cloud-focused language, which is basically Go, and they're all focused on developing and delivering this thing, and we were, "Well, okay, that's exactly what the number one thing we would ever want." Really, what intuitively just makes sense is where the future of this kind of infrastructure is going to go.

[0:16:21.8] JM: Even though you saw it in its raw early state in 2014.

[0:16:25.5] SG: Yeah. I mean, we — I think our first version we deployed was like 0, 7 which was about as early as humanly possible, basically just ran Docker containers, and that was about it. It did a little bit more, but for the most part, that was about it.

So yeah, even though it was very early, we just figured, "This is what we want." We definitely won't be able to outpace development, but if we just treated it as a project that — You could imagine something like we kind of built in-house, but has just this huge booster pack from outside, then we'll just grow along with the project, and it just made sense to start investing in that paradigm shift.

[SPONSOR MESSAGE]

[0:17:10.1] JM: DigitalOcean is a reliable, easy-to-use cloud provider. I've used DigitalOcean for years whenever I want to get an application off the ground quickly, and I've always loved the focus on user experience, the great documentation and the simple user-interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU-optimized droplets, perfect for highly active front-end servers or CICD workloads, and running on the cloud can get expensive, which is why

DigitalOcean makes it easy to choose the right size instance, and the prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily, and as a bonus to our listeners, you will get $100 in credit to use over 60 days. That's a lot of money to experiment with. You can make $100 go pretty far on DigitalOcean. You can use the credit for hosting or infrastructure and that includes load balancers, object storage. DigitalOcean Spaces is a great new product that provides object storage and, of course, computation.

Get your free $100 credit at do.co/sedaily, and thanks to DigitalOcean for being a sponsor. The cofounder of DigitalOcean, Moisey Uretsky, was one of the first people I interviewed and his interview is really inspirational for me, so I've always thought of DigitalOcean is a pretty inspirational company.

So, thank you, DigitalOcean.

[INTERVIEW CONTINUED]

[0:19:16.8] JM: Did you examine Docker Swarm at all in this process?

[0:19:20.2] SG: So Swarm came out ever later, I think.

[0:19:23.3] JM: Are those post-Kubernetes.

[0:19:24.5] SG: Yeah. It may have had some very nascent version, but as far as like the Swarm kind of model that we know today, it definitely came out after Kubernetes. So from our perspective — And also like Swarm was pretty challenging, because Docker was taking this approach of, “Well, for a number of years, Docker of like, “Well, we’re just going to bundle it all in and it’s going to be this one project,” and instead having this kind of clean layer between Docker and then whatever Kubernetes was, it made a lot more sense to us. But then also like the people working on Kubernetes were the people who had built what us and a lot of other people in the valley really considered to be the gold standard for how this stuff should work. That I think was the number one — You can't say that like Kubernetes has ever been like a massively superior technically to Swarm, I don't think. I personally have been very fond of the architectural decisions Kubernetes has made, but I think objectively the number one thing you could say is that the team on the Kubernetes side is incredibly strong.

On top of that, not only do you have Google behind it, but also Red Had very shortly after. I think within the year of Kubernetes is launched, basically that their entire PaaS strategy as a company on Kubernetes. So not only do you have Google and their team behind this, but now you have Red Hat and the tremendous amount of talent over there fully focused on building their PaaS layer, which is —

[0:20:50.4] JM: OpenShift.

[0:20:51.1] SG: Yeah. Completely around and 100% dedicating it to Kubernetes, and already you could tell they had a very good relationship. Red Hat was up-streaming everything they could into the Kubernetes project. Then shortly after that you had CoreOS rally behind it and really start to embrace it, and within a year or two later, CoreOS kind of — I don't want to say abandon Fleet, because Fleet still exists, I believe, but really focused on Kubernetes as the solution for orchestration. So you have this incredible momentum around this project that really kind of blew everything else out of the water.

[0:21:21.6] JM: Yeah. It sure has been interesting to watch all of these infrastructure companies to have to really change strategy, like businesswise. I mean, I report on this stuff is much as I can and I have a whole lot of trouble understanding how exactly to think from a business point of view, if you're Mesosphere or CoreOS or Docker. Basically, any of these infrastructure providers whose business gets turned on its head once Kubernetes starts taking off like a rocket ship. How the heck are you supposed to adjust?

[0:21:52.3] SG: Yeah, it's pretty chaotic. I mean, we just saw company after company announce that they were either adopting or rewriting to adopt. Yeah, I think he was a profoundly kind of disruptive force and has forced a lot of companies just be like, "Okay. Well, it really doesn't make sense for us to do what Kubernetes is doing," because Kubernetes really resisted the temptation to tightly couple to the GCP. They knew that their strength was the agnostic nature of a platform.

Now, at the end of the day, obviously works well with GCP. They’re not going to treat it like a second-class citizen, but they did continually, and they continued to, focus on Kubernetes as an agnostic substrate.

[0:22:38.7] JM: Yes, as a nice little jujitsu there. But getting back to Box — So you deployed it at 0.7. It was maybe a little rocky compared to where it is today, but you were making a bet, much like OpenShift made an even bigger bet. I think it was probably, from your point of view with your level of experience, it was pretty easy to see the writing on the wall that this was going to be a good project. It was going to get to a place where it was sturdy, if it wasn't sturdy from the beginning. So how aggressively did you start throwing services on to it?

[0:23:10.5] SG: It took a while to get started. One reason was because the resources we had allocated this team still had a tremendous amount of existing infrastructure maintain. So it took us some time to — Like, first, we had to just shore up our existing stuff and really get it stable and not have it be an acceptable state where we had four or five engineers required to maintain it, but instead get it down to like one engineering or even, ideally, half an engineer required to maintain it. So we did that for a while, and that also gave us some time to spend one or two engineers on filling in some of the gaps. Like I spent like almost a year helping work on a lot of the kubectl apply and declarative configuration stuff, because that was really going to be a key factor in making this stuff ultimately better long-term than our existing infrastructure and making it highly declarative.

Just a brief review of like declarative versus imperative, an imperative infrastructure is one where you make changes by pushing buttons and pulling levers, basically, or running commands. Then the state, the results in state really is just whatever exists. The advantages, this makes it very intuitive and easy to kind of push buttons and take things through pipelines and deploy and launch things, but then you don't really know what your current state is and if you need to launch a new environment from scratch that replicates for environment, it can be kind of chaotic to get it to that state and really confusing in like, “Okay. Well, we thought that if we just deploy service A to the current environment and then to the new environment, it would work, but actually the new environment didn't have these other dependent services and all these kind of stuff.”

A declarative configuration is one where your entire state of your infrastructure at any given time is represented ideally by code, ideally in a repository, some sort of version control system, ideally git, for example. So then when you make commits to git, that is kind of — Those changes are pushed out automatically from the declarative config.

[0:25:06.7] JM: By the way, who was doing this? Do you have a platform engineering team that was just in charge of reformatting the infrastructure or were you just doing this on your weekends when you aren't busy or what exactly —

[0:25:17.2] SG: Yeah, we pulled together what internally called the Skynet team. It is basically just trying to make —

[0:25:21.8] JM: Okay. I love that name.

[0:25:24.0] SG: Yeah, just trying to make it very aspirational for how intelligent we wanted to make our infrastructure. But that was very much a goal. It was basically as much of our dev ops group as you could call it, and we just pulled whatever resources that we could get from various teams. Then with those resources really came in some of the legacy infrastructure which we had to shore up first and stuff. Yeah.

[0:25:44.3] JM: And the rollout process, was there a particular service that you still like — I remember the first couple shows I did about micro-services several years ago, everybody would talk about — The example of Netflix putting its job board as the first to service that they stripped off and put — Oh, actually I think that was Netflix's migration to the cloud. Like the first thing that they put into the cloud, because they didn’t trust the cloud, was the job board, which has nothing to do with Netflix's core competency, but you want the job board to be up. So it’s a nice mixture of you want it to be up, but it's not a big deal if it goes down. Did you do something like that, like put the job board up or like the Box blog on the Kubernetes instance that was 0.7?

[0:26:25.5] SG: Yeah, exactly. Actually ours was even more trivial, because of how new this infrastructure was. So the first thing we up was a demon that had no interface that simply hit box.com and made sure it was up.

[0:26:41.2] JM: Okay. Health check.

[0:26:43.5] SG: Yeah, and it was literally called Public API Health Checker, and you could not come up with a more trivial service, but it really was powerful as like a proof of concept of like this thing runs stuff and you can push it and you can make a change the code and it deploys and it runs and it creates logs and you can check the logs and see that it actually works. For us, it was also a big deal that it was a change driven by a developer through a fairly developer friendly interface that ultimately went out to production, because, again, as a company we have such high and restrictive uptime acquirements. We have such, such critical workflows going through our platform for customers that like uptime is so vital. So our production environment is very heavily protected. And so to actually have something that was very developer friendly and actually modifying production in our production infrastructure was very exciting.

[0:27:35.0] JM: And what were the next couple services? How did you start to build momentum and start to say, "Okay. We're going to really start to put some production stuff on here."

[0:27:43.9] SG: Yeah, I mean, I'll be honestly. It wasn't easy, and to this day — Let's see, we probably have 15% to 20%, I think, of infrastructure on Kubernetes. So we're still not anywhere close to the vast majority. Picking up, it's moving quicker every quarter. The trend lines are in the correct direction, but this stuff — Like a significant organization, it's genuinely hard. I mean, it's only so difficult to like launch a new product or something on this stuff, but the downside of a new product is who knows if anyone's ever going to use it? How much impact are you having? Things like that, right? But if you're taking an existing multibillion-dollar business and trying to move it on to here, the good thing is everything you're doing on day one is massive impact, but it's like you have to contend with, I mean, at this point, eight or nine years of largely crafty infrastructure.

Because what happens is you start to build this stuff and you have some manual steps in there to like deploy servers or like deploy more bare-metal infrastructure, and then those couple of manual steps turn into a few more manual steps and then you're like, "Okay. Well, [inaudible 0:28:47.3] stuff of the stuff," but the problem is there are a lot of coupling in there. So you want to automate one piece of it, but then that there's other downstream steps that depend on that piece, and eventually if you're growing too quickly, if your demands are exceeding your engineering capacity and all these stuff and you need to keep the business running and pushing out new features and all these things to beat the competition, at a certain point like you end up with this kind of very tightly coupled and not very modern set of infrastructure.

So a lot of our struggle, to answer your questions to like move stuff on the Kubernetes, was like there's a lot of tension in the organization of, "Should we be fixing our existing stuff or migrating to the new paradigm?" And that was not and is rarely an easy answer, especially for an organization that's smaller. Like we had maybe 300 engineers or so as we were doing this. I think if you had on the order of thousands — Like Google when they did Borg, had I think far, far more engineers than the scale that we were at when we're trying to do this transition.

So 300 engineers and supporting tens of thousands of customers, you only have so much capacity. So the biggest struggle in moving services on was less about the technical challenges. Those were pretty straightforward. It's was just like, "Well, we need better load-balancing infrastructure. Let's build that. We only need better provisioning of our SSL certificates in production. Let's build a component for that." And I can talk about the details about how Kubernetes made that really nice to build [inaudible 0:30:07.0] infrastructure. But at the end of the day, the real challenge of moving this stuff is like, "Well, you have to invest in the current. You can't completely ignore the current pain and things that are breaking or breaking down, and you can't totally ignore the future." So balancing that investment and effort, not just in our team but across the organization, was the continual challenge.

[0:30:27.8] JM: Okay. So you're talking about two different things here. So one is that it's hard to get your disparate teams to adopt an infrastructure shifting technology like Kubernetes. The other thing you were talking about is something with changing the load balancers out and the deployment process. What exactly were you referring to there? Or SSL certificates.

[0:30:46.3] SG: Okay. On day one, when you launch this thing, there's a lot of gaps. So one example is — So like the first thing we had was the ability to write code into a git repository, write a Docker file, build an image, push that image to a repository, get some basic testing on it with Jenkins and then release it on into production.

Now one of the challenges is you can release the code on to production, but like that code has a lot of dependencies. One of those dependencies is some course, like configuration platform that we had called app.conf , which needed to be available to the service. Now that app.conf, we didn't have a way on day one of pulling that down and exposing it to the service, but we also didn't want to wait until we had that dependency and many other dependencies kind of solve before we even get any production use case.

So what we first did was we actually just mounted the application configuration, the global one, into the service and we coupled basically to the machine, the core machine itself, and we deployed it to the machine using our standard mechanisms. So that one example. Does that makes sense?

[0:31:54.6] JM: Yeah. I think something I should have asked you earlier is do you have one big Kubernetes cluster that's meant for the whole company or our different teams deploying different clusters?

[0:32:04.8] SG: So the answer to your question is we have one per data center, and in our nonproduction data center, we have one for development and one for staging. But it was, and even to this day, still continues to be some degree of debate on that front. So it definitely exists in one state today, but I wouldn't say it's a clear cut answer. Especially if you're in the public cloud, you could argue that one cluster kind of per team can make sense, but you definitely get away better utilization and some deficiencies on the management side if you just do one cluster.

One of the challenges Kubernetes — They're quickly adding more and more features in this regard, but they don't quite have all of the primitives you would want to be able to really do very good management of multitenant clusters, right? So there is enough in there to do it, and we do it. We have multiple teams in one cluster and we largely segment using name spaces, but it's not a super smooth process.

[0:32:59.7] JM: Okay. The reason I say that, the reason I brought up that point was just because I think you are explaining the stuff that you were doing to set up Kubernetes so that other teams could on- board with it, first of all, and then that was sort of the stage one that you had, and then the much harder stage two is going and evangelizing Kubernetes and perhaps teaching other teams how to use it so that they can migrate their services to Kubernetes.

[0:33:25.6] SG: Yeah. The good was like our existing infrastructure was painful enough that we didn't have too much of a fight with most teams. The big thing was that these teams needed features in the platform. They're like, "Oh! We use app.conf, or we need SSL certificates dynamically provisioned when the service gets deployed," and like each one of these things is more features that our team needed to build.

So, for example — And the best example is when we deploy the service, we automatically need a load balancer configured to be able to hit it. So that was one of our biggest things that we have to work on, because load-balancing is largely free in the public cloud, but on bare-metal it's a nontrivial problem. If you have a mesh of micro-services load-balancing to them and having them discover each other is still kind of — I mean, it's getting better rapidly with things like Envoy and Istio and these other projects, but especially two years ago, it was very, very early. And so we had to custom build our own frontend proxy based off of HAProxy that will dynamically send request to the right services based on the DNS name or the port, or things like that.

So that was something that did not launch with the Kubernets platform internally, so our initial use cases were entirely focused on demonized use cases. So I wouldn't say it was so much the evangelism that was big of an issue. It's so much is that like we actually needed the full feature set of things that the platform needed to be able to dynamically launch and run one's micro-services.

[SPONSOR MESSAGE]

[0:34:57.1] 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 CONTINUED]

[0:36:48.9] JM: And since you mentioned some of the tooling, namely Istio, Istio and Envoy I supposed, what are some of the ways in which the tooling has improved to make it easier for teams within Box to move their infrastructure on to Kubernetes? For example, that load-balancing service discovery, etc. We've done a bunch of shows about Istio and other service meshes that sort of give you this out-of-the-box, the service proxying, the load-balancing service discovery, these things that you basically want for every service. I guess of the early days of Kubernetes, you really didn't have that. So you had to figure out a way to wire your older load-balancing technology. You got to figure out a way to connect that to your new Kubernetes infrastructure.

[0:37:33.2] SG: Actually, the connection ended up being somewhat independent, but basically — Yeah, a couple years ago, a lot of the stuff, like Envoy and Istio didn't exist. So what we ended up doing was we actually kind of built it from scratch ourselves using stuff like SmartStack and the existing kind of tools and libraries that was there to interface with the Kubernetes API.

Then, again, as per your point as far as interfacing between — That was a consideration with SmartStack as well, where we were able to do client-side load-balancing between not just the stuff inside Kubernetes, but also the stuff outside, and I think we'll continue to evaluate that space. I feel like right now service discovery and I think more commonly it's called like service mesh at this point when it includes things like not just discovery, but retries and logging and tracing and all those kinds of things all inside of kind of the proxy in between services. That field or that area, I guess, right now is I feel like going through what containerization was three years ago, for example, because — And it's like it's going through a bit of the Renaissance, and I think Envoy, especially, is kind of leading the way on that front.

[0:38:42.2] JM: And do you think it's winner take all?

[0:38:44.9] SG: I don't know. I liken a lot of these things to the Linux kernel. It's not a winner-take-all, but it's really winner take most and there's going to be tons of slight little variations, same way there is like different Linux distributions, but at the end of the day, having a single kernel that a lot of people are targeting is pretty powerful. So I think Kubernetes and potentially Envoy or at least like the interfaces need to be uniform and I think the implementations will get swapped out. So I think that's a very open question. I've seen the Kubernetes team engage in countless, really, great discussions around what it means to be Kubernetes, what it means to run a Kubernetes cluster, what conformance means. Like these are really important questions that I think it's really great that Kubernetes like team and CNCF broader organization care about working on the stuff right now versus being overly draconian and resulting in really bad hard forks, or the inverse being incredibly lax about it and allowing for like the Kubernetes name to essentially mean nothing. Like it's a very fine line to walk.

So I think it's an interesting question. I think it's going to evolve, and each layer of the stackable will have a different set of expectations on what is core and what is essentially swappable.

[0:39:58.4] JM: Yeah, I feel like this is the same question that's being discussed at the serverless API layer, like will there be a consistent serverless API so that you can switch between different serverless providers. We'll see. It's pretty hard for us to predict, I guess.

[0:40:15.4] SG: That is definitely some of my hesitation around serverless. I think there's no doubt it has very bright future and a potential in that front. I mean, yeah like, if you can ignore servers, why can't you ignore even the containers? Why don't you just run code? I think there's a lot of promise, but like if you go down the route of just totally hundred percent coupling your stuff to AWS services, it's not great. We're going to end up in the same spot that we have been with AWS where you're just completely bound to one provider and you're at the mercy of whatever their infrastructure does.

[0:40:48.9] JM: So you've said that a couple of times, what's your take on the Amazon approach to manage Kubernetes?

[0:40:55.4] SG: To be honest, I haven't deeply looked at it. I'm not super familiar with it, because for us, there is —

[0:41:02.7] JM: Yeah, I guess it's not really relevant.

[0:41:04.1] SG: Yeah. I mean, it could be one day. Like I think the dream has always been that ultimately our company, us as an engineering organization, will just be able to spin out of Kubernetes compliant API and hit that to run stuff. Like we would love to not care about anything underneath that. That's the dream. I just think the reality, we have to chip away at it piece by piece, because like we have strict compliance requirements around what runs on our servers and what software packages are running and how they are configured that support our code.

So we can't just like say, "Well, just give us an image that works and we'll be good with that." Like we have patching requirements, we have configuration requirements for the kind of workloads that we run. So I would say that's the number one reason, that we aren't as closely looking at managed Kubernetes packages today, but like I said, I think between — I think like CoreOS has aspirations to be able to provide Kubernetes APIs ultimately or like one your installation for you, things like that. So I think it's very promising, but those are just very early kind of for us.

[0:42:16.7] JM: Do you have Kubernetes managing your low-level storage layer, because you're storing these files on your own managed infrastructure, that's hardware that's in a co-lo somewhere. Is Kubernetes managing that stuff?

[0:42:32.8] SG: Not today. Kubernetes doesn't manage anything with local state or, I mean, long-lasting local durable state, like databases or storage. I bug one of our lead DBAs every day about like, "Hey! It's about time to move those databases in Kubernetes," and then he laughs at me, and then we move on.

I mean, I think we're still probably a year or two away from that. I know some companies have successfully containerized their databases, like I think Square has, but that's a little different than actually running in stuff like Kubernetes. I think, like every day, Kubernetes gets more and more sophisticated at being able to run varying workloads. Like I know being one of the recent versions, they launched — They released the ability to have local state that's not required to be like a remote EBS mount, like actual local state and keeping that sticky [inaudible 0:43:20.3]. I think stuff like those is very, very, very exciting, because I really think our database infrastructure could benefit as well.

The file storage infrastructure, I mean, honestly is just like hard drives with nginx in front. Like it just changes so infrequently that there's not a lot of benefits to having something really dynamic like Kubernetes running it. I still think one day make it makes sense, but for now it's just not a priority.

[0:43:41.9] JM: So I guess differences and what we're talking about here is you have a database that keeps track of maybe where these different files are in the data center, then you have a file system that manages the files that are actually on those disks. Is that right?

[0:43:57.1] SG: Yeah, that's exactly right. Yeah, sorry to not be clear about that. Yeah, our databases I'm talking about are like MySQL databases basically that store all the applications, say, like users, filenames, folders, like the structure the file is on, and then we have a bunch of file storage servers that are only know about blobs of bytes, and those are fronted by HTTP servers.

[0:44:17.4] SG: So I guess maybe you could talk a little bit about engineering at Box from your point of view. So I talked to your colleague, Jeff, recently and actually just an hour ago and it's pretty a interesting conversation, but maybe you could talk about the engineering organization from the point of view of somebody who's helping to migrate it to Kubernetes. So how do you expect the movement towards Kubernetes or the homogenization towards Kubernetes? Like let's talk about the bright future, maybe 2 to 3 years down the line when Kubernetes is widely deployed at the company. How do you see that improving the resource utilization as well as the engineering productivity?

[0:44:57.8] SG: I would say probably the biggest focus for us has always been on productivity with this project. I think resource utilization is interesting, especially for certain companies and use cases, but for us, our servers are reasonably well utilized, I guess and this definitely — And virtualization solves a huge chunk of that. So if we just really care about cost efficiency, I think virtualization is not just as good, but almost sufficient. So I would say productivity has always been the main focus of our Kubernetes stuff, and that I'm very excited about the future of that, because like it's not just about I am a micro-service developer and I want to write my service and get it deployed quickly and easily. It's more that if I'm an infrastructure developer or an operations engineer or a dev ops engineer and I want to make my workflow or the management of the infrastructure more broadly more efficient, the fewer interfaces I have to target and to maintain and manage, the easier my job is going to be, to the upkeep of the health, the building new features, like better monitoring or better deployment or debugging or logging or whatever it is, the fewer interfaces I have to manager or to program to or the support, the easier that's going to be and the faster I am going to be able to build infrastructure.

But even better and even more importantly and the most violent thing I think about all these, is this is the first time that these kinds of interfaces, these deployment and micro-service management interfaces are fully open-source and universal. That is, to me, going to be the biggest impact of Kubernetes in this broader kind of movement, is that like never before you've been able to take another company's like service discovery infrastructure or their log-tailing infrastructure or their — Like really name anything, monitoring or alerting or whatever it is and just run it on your infrastructure, because almost certainly it's going to be tightly coupled to a dozen different internal micro-services that are likely proprietary to your system. There's always going to be some gotcha, whether it's, "Oh! Actually that works only with our authorization platform or only with our deployment service or whatever it is," but now you have the set of APIs that actually is powerful enough to fully deploy and run and manage other micro-service. Does that make sense?

[0:47:25.5] JM: Oh yeah. I mean, this is where I saw in the palpable levels of excitement at Cube-Con where I went recently. There're network effects to Kubernetes.

[0:47:35.4] SG: Right. That's the biggest impact. I think many developers can identify in Silicon Valley having gone from a Google or Facebook or Twitter having such an incredible wealth of tools to run and maintain and debug service, and like a complete like just feeling incredibly frustrated going to another company, especially a smart company and feeling like you don't have anywhere near that level of infrastructure. Now, like for the first time, the opportunity exists to build something like that and then have it be not only deployed and utilized in other companies, but also built on and contributed to and advanced.

[0:48:11.9] JM: I did a show with Brendan Burns recently, and he said this thing that pretty much shocked me that I've been asking most of the people I've interviewed about Kubernetes since then, which is the idea that when you have Kubernetes widely deployed, basically, when every company is running Kubernetes as their infrastructure management tool, you could have a world where I could be a random indie hacker that's building my own software and selling it and I could sell a one-time purchase binary, like and $99 piece of software that just runs on Kubernetes and it could be a return to the days of the independent software vendors. What do you think about that? If you think that's preposterous. Maybe you have some other big ideas about how Kubernetes might change the way the infrastructure ecosystem works?

[0:48:59.9] SG: I think that's totally right. Like if you developed monitoring infrastructure, you pretty much have to run it yourself, but if you wanted to do like — You have to offer it as a SaaS service, because even the idea of like allowing people to run it locally on their own infrastructure is incredibly difficult to reason about, because what are you going to couple to? Like, okay, let's say you say, "Okay. You can run this in AWS yourself," but then what if I'm not running AWS or what if all my infrastructure is in Azure. What if I run it on my infrastructure? Then you have to also support those platforms, and each one of these conferences has dozens of these little nuances, like ELB is not the same thing as Google load-balancing, is not the same thing as Microsoft load-balancing. They behave differently. They have different guarantees, expectations, requirements, configuration, all these kind of things, and that's just one piece. That's just a load-balancing. Don't even get me started on state or like a backend store or things like that.

So the ability to like build software now and couple it with a — You could call it a cloud manifests that programmatically instructs your infrastructure how to fully run some piece of software, not just as some RPM package.

[0:50:11.6] JM: Helm chart.

[0:50:12.2] SG: Yet, exactly. Like a Helm chart or something like that. That's a totally different ballgame and being able to develop and deploy software. So I think that's a very exciting area.

[0:50:20.0] JM: Okay. I know we're running up against time. I just had a few kind of crazy questions. You've been at Box basically since the beginning. Do you have any crazy stories about managing the Box infrastructure just from either early days our middays or recently, because I mean it's a big enough infrastructure where you deal with some tail scenarios.

[0:50:41.3] SG: Gees! I mean, countless. Yeah, I mean, off the top of my head — I don't know, race conditions are just the worst thing in the entire world. The number of hours of my life, expecially early on in Box that I've dealt with, especially like we were definitely pretty pioneering early on on like a multitenant public cloud infrastructure, especially for enterprise software where it really stricts guarantees on like, "Your file system can have two files of the same name in it." You just can't do that, because of who knows how many issues that's going to cause.

So I mean our hardest problem by far were just like being up until 3, 4, 5 AM after some code release and going forward and backwards and seeing some issue and just being like — I mean, almost everything ended up being recoverable, because you have a lot of layers in there that protect you from losing data. Like you basically never lose data, but like having weird things happen that you need to rollback — I mean, gees! Race conditions are just really, really, really bad, and any time you can avoid concurrency, like just do it and just make things concurrent, really, as necessary. That’s why I really love like the Go language and the decisions they made in there where it's like you're not doing callbacks or async by default. You’re really allowing the runtime to deal with that for concurrency, and then there’s that famous talk about concurrency versus parallelism. Like when you actually need things running in parallel, it lets you do that with Go routines. Sorry, that wasn’t like a coherent anecdote. I just have all these flashbacks of PTSD of like multiple share nothing like PHP processes all trying to update the database and then really fun things can happen, and diagnosing and resolving those things. It’s just like you can go to days where you're just lost and confused and it's all worrying about. Then that like incredible rush of relief of actually figuring it out and then resolving it and pushing it fixed is pretty great.

[0:52:40.1] JM: Oh yeah. Well, I'm sure the brain has maybe removed some of those troubling memory specifics from the trauma. Okay. So last question; I have watched a lot of random videos and podcasts and stuff from the CEO of Box, Aaron Levie. He's a pretty charismatic guy. I think that plays quite a big role in how successful Box has been. I recommend anybody who's listening to this to check out some of the talks or whatever he's given. Is there anything specific about leadership that you've learned from him?

[0:53:15.3] SG: I think the biggest thing is just like he sets a very high bar and it e can definitely be frustrating at times, but he's like really inspirational on that front where he really just like cares deeply about doing the right thing well and it's coupled with this ability to not be attached to the outcome for like ego perspective.

So the like will just be in these like really, really violent disagreements, and then if a data shifts or it shows that one way is more clear or then the decision will change and like he’ll change his mind and will move on, and it's kind of incredible and really rewarding to work in that kind of environment. So, I mean, this combination is engineering, I guess, like [inaudible 0:54:05.0] or something, which is like strong opinions lightly held, and I think that's really vital to like pushing for what you believe in, but then the moment like what you believe in doesn't make sense any more, then you have to change and you have to shift. So seeing him implement that over and over again has been really great and really instructive.

[0:54:20.9] JM: Okay. We'll leave it there. Sam, thanks for coming on Software Engineering Daily. It's been great talking you. I really appreciate all the anecdotes about Kubernetes and race conditions. Really good stuff. Thanks.

[0:54:31.2] SG: My pleasure. Thanks for having me.

[END OF INTERVIEW]

[0:54:36.5] JM: If you are building a product for software engineers or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an email, jeff@softwareengineeringdaily.com if you're interested.

With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers. I know that the listeners of Software Engineering Daily are great engineers because I talked to them all the time. I hear from CTOs, CEOs, directors of engineering who listen to the show regularly. I also hear about many newer hungry software engineers who are looking to level up quickly and prove themselves, and to find out more about sponsoring the show, you can send me an email or tell your marketing director to send me an email, jeff@softwareengineering.com.

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

Thank you.

[END]