EPISODE 963

[INTRODUCTION]

[0:00:00.3] JM: A cloud provider gives a developer low-cost compute infrastructure on-demand. Cloud providers can be divided up into two categories; layer 1 cloud providers and layer 2 cloud providers. A layer 1 cloud provider, such as Amazon Web Services owns server hardware and sells compute infrastructure as a commodity. A layer 2 cloud provider purchases compute infrastructure from a layer one provider and builds a high-quality developer experience on top of that compute infrastructure.

Heroku was the first layer 2 cloud provider. Heroku's first business was to provide a high-quality developer experience and low-cost containerization infrastructure on top of Amazon's EC2 virtual machine infrastructure. Heroku has since added features for continuous integration, relational databases, caches and queuing.

Building a layer 2 cloud provider is a very different challenge than building a layer 1 cloud provider. A layer 1 provider must focus on low-level problems, such as hardware infrastructure and virtualization. This does not leave much time for focusing on developer experience. A layer 1 cloud provider must be able to serve every type of potential software customer, but a layer 2 provider can provide a streamlined experience.

Mark Turner is an engineer at Heroku. He joins the show to discuss the architecture and engineering of a layer 2 cloud provider. Heroku is built on top of Amazon Web Services and the core compute infrastructure is built on top of a pool of EC2 machines that are continually scheduled with applications that users create on Heroku. This was a great discussion of the technical engineering challenges and the design of Heroku.

Full-disclosure, Heroku is a sponsor of Software Engineering Daily. I want to mention that we are hiring a content writer and also an operations lead. Both of these roles are working part-time closely with me and Erica. They're great positions if you're looking to learn more about software engineering, or how our business runs. The content writing position is going to write about fairly technical software engineering subjects and the operations lead will help us lead our business in

a more well-organized fashion and will help us spot opportunities that we can improve the efficiency of Software Engineering Daily.

If you're interested in either of these positions, the content writer or the operations lead, send me an e-mail, jeff@softwareengineeringdaily.com. Don't be shy. If you think you'd be a reasonable fit, please reach out. Also, this episode will air on December 4th and that means that this same day, we're going to be at AWS re: Invent and for the rest of the week and we're planning a re: Invent meetup on December 4th. If you are interested in attending that meetup, just go to the link in the show notes and click on it to RSVP. I'd love to see you there and let's get on with this episode.

[SPONSOR MESSAGE]

[0:03:27.1] JM: If you are a SaaS or software vendor looking to modernize your application distribution to gain more enterprise adoption, check out replicated.com. Replicated provides tools to deliver your Kubernetes-based application to enterprise customers as a modern on-prem private instance.

That means your customers will be able to install and update your application just about anywhere; bare-metal servers, in a cloud VPC, GovCloud in their own Kubernetes cluster, vSphere. This is a secure way that your customers can use your application without ever having to send data outside of their control. Instead of your customers sending their data to you, you send your application to your customer.

Now this might sound difficult and maybe you're not used to it, because you're a SaaS vendor, you're a software vendor. Replicated promises that recent advancements from tools like Kubernetes make it far easier than before. The Replicated tools can help vendors operationalize and scale this process. The Replicated tools are already trusted by noteworthy customers like HashiCorp, CircleCI, Snyk and many others. As a result, over 45 of the Fortune 100 already have an application deployed via Replicated in their infrastructure. That's a strong sign of adoption.

Go to replicated.com for a 30-day trial of the full Replicated platform. You can also listen to an interview with Grant Miller, the CEO of Replicated that we did a while ago. Thank you to Replicated for being a sponsor of Software Engineering Daily. You can check it out for yourself at replicated.com and get a free 30-day trial.

[INTERVIEW]

[0:05:35.2] JM: Mark Turner, welcome to Software Engineering Daily.

[0:05:37.2] MT: Thanks for having me.

[0:05:38.6] JM: You work at Heroku and Heroku is a cloud provider. Cloud providers are quite a large space. If we think about the design of a cloud provider, there are different trade-offs that different cloud providers make. These come in terms of costs, in terms of developer experience, in terms of the operations that the cloud provider expects from the average developer, describe the different trade-offs spectrums that cloud providers can make in designing a developer experience?

[0:06:13.3] MT: Man, that's a big question. Mostly because Heroku through our entire lifetime has almost always focused on getting web apps onto the Internet. The trade-offs that we and other cloud providers might decide to make in surface, or in service of the end-goal we're going for can be quite dramatic. When I look at the – or quite dramatically different. For me when I look at – I have to answer the question from a Heroku perspective honestly, because that guiding principle of that web app host is one that helps us clarify the mission a lot.

When we describe the trade-offs that we make as cloud providers, I think it's solely – not solely, but the biggest factor that feeds into that is what service that that service provider is offering. Heroku again like I was saying, we focus on getting a web app on the Internet. Our bread and butter has been Ruby apps for a long time, but we really support a ton of languages. It's really squarely that web app focus. What does that mean for us? That means that we're focused on TLS and exposure of web technologies. There's legacy migration stuff we don't have to touch often.

There's been spectrums of the computing world that we haven't had to support as well, like Windows for example, because our main customer base has classically been using POSIX-based, Linux, Mac OS-based systems.

For us, our main guiding principle has been servicing that web app, that really fits in that 12-factor model. I don't mean to bring up 12 factor quite so quickly. For us, it really does squarely define what describes an application suitable for Heroku.
I think that when you talk about shipping software to the Internet, 12-factor as a description of what that software should do and should be has worked well when we think about what offerings Heroku should offer our customers.

[0:08:24.5] JM: Shortly after Heroku came to market and it became pretty popular, there was a variety of attempts at an open source Heroku experience. I think the software community as a whole did iterate towards that vision and got there with Kubernetes, but even Kubernetes is something that's pretty different than a Heroku-like experience. How do you see the cloud provider evolution? When you think back over the last 10 – Heroku I think was started, it was – was it 10 years ago? I think it was something like that.

[0:09:06.8] MT: It's been over 10 years now. Yeah.

[0:09:08.5] JM: How has the cloud provider space evolved over the last decade?

[0:09:12.9] MT: Man, that's huge. Back when we started, all that existed was EC2 classic. A few disparate companies out there trying to scale up their dynamic bare-metal hosting services out there making it possible for companies like us to put stuff on the Internet and Heroku has always been on the cloud. We've always been an EC2 customer.

For us, the way that the cloud has changed is really mostly that the scale of the infrastructure providers has increased dramatically and they keep creeping up the stack as far as what services they offer, that we are able to leverage and take advantage of to power that the PaaS we build. I think that's generally ubiquitous for everybody approaching cloud stuff today. Everything that companies like AWS offer as the wide disparate services that they provide

means that you really get a giant kit of solutions to problems that you don't even know you have sometimes.

When we started approaching shipping software into the cloud, that really didn't exist that kind of selection of available, “off-the-shelf,” that available choice of tools that you don't have to build yourself wasn't there. For Heroku, we ended up building a lot of the cloud management systems that you don't have to do a lot of today. Things that we take advantage of that are ubiquitous, things like Ansible and Chef, things like Terraform.

There's probably a chance that Heroku at one point in our history wrote a bespoke, or heavily contributed to, or was a heavy user of some open source library, or package that did equivalent features of those things. Over time, the biggest thing that's happened for us in relation to cloud providers and especially in the industry I think is that it's just become easier and easier and easier and easier to use cloud providers. That story is the same story for Heroku as it is for anyone today.

[0:11:25.3] JM: I find interesting is in the layer 2 cloud provider space, which is it's pretty small, it surprises me how few layer 2 providers there are. I mean, there have been a few more over the years, but Heroku came out and then there were really not a whole lot of other layer 2 cloud providers that reached prominence for a while, which is interesting. Do you have any beliefs around why that is? Why have there been so few layer 2 cloud providers?

[0:11:54.1] MT: I think the reason for that, the reason for the lack of diaspora of these providers out there, is that it's a really hard business to operate, both from an operations perspective and from a cost perspective and from margin generation perspective. Living at the gap between the infrastructure providers and that opinionated experience that customers come to us for is surprisingly hard to make a really successful business out of financially.

Heroku has done really well doing that. I think that that's been a core tenant to our success a bit there as we focused on the business side of it and have for a long time. The secondary part is the technology side. Part of Heroku having been around the cloud in “forever,” is that we've had to generationally deal with a few growth changes in our product. From the very start that we've

always been a multi-tenant. I think that's where stuff gets hard for people. That multi-tenancy line is super difficult for a lot of stuff to jump.

Especially in open source communities, especially with the adoption of Kubernetes and the explosive growth of that, the isolated barriers, isolated boundaries of that layer 2 behavior that Heroku and other providers have really thrived in is becoming easier and easier and easier to do. I really do think that the operation of a multi-tenant layer 2 provider is really hard, really complex and really difficult.

[0:13:36.8] JM: Let's talk about that in more detail. First of all, I think by multi-tenancy, you mean the fact that most of what you are building on top of are raw EC2 instances, which are virtual machines, I believe? Those are virtual machines that are spun up on Amazon infrastructure. You're breaking up those virtual machines into containers, specifically Heroku containers, which are called Dynos. Then you have multiple different Dynos operating on the same EC2 instance. Those Dynos may be from the same customer, they may be from different customers. One Dyno might be gigantic, it might be a resource hog. Another Dyno might be totally small.

Then figuring out how to distribute these tenants in the right way, so that they're not conflicting with each other and then what do you do if a machine goes down? It's a huge bundle of problems.

[0:14:37.5] MT: Exactly. Yup. You hit the nail on the head there with capacity and resource allocation. That's a huge one. We might have customers that have workloads, where each single process uses 14 gigs of memory. At the same time, they might have something that where each process uses 2 megabytes and they want each of those things to schedule and boot up instantaneously. That's the orchestration layer problems we deal with that makes it hard.

Then it's the isolation and security boundaries between all of that stuff that also makes it hard, and auditing and patching and maintaining those boundaries is the hard part. Then you factor in layering on those workflows that power that Heroku experience, that those containers encapsulate down at the bottom of it; it all adds up into I think just a hard system to build.

[0:15:25.4] JM: You've been at the company for more than five years, so you have seen the rise of containerization becoming popular in terms of Docker. Obviously, containers were widely used long before that. You saw the rise of Amazon ECS, for example, and the rise of Mesos, Apache Mesos, and then eventually Kubernetes. How did Heroku and the infrastructure strategy within Heroku respond to those major infrastructure changes across the industry? By the way, I mean, that in terms of did you refactor the infrastructure at all to take advantage of those things?

[0:16:12.1] MT: I would be fibbing to you if I said that things like that didn't factor into structural changes that have happened. I can tell you that for things like ECS and Mesos, that's where the influence ends for us a lot of times is it's an inspirational thing, not often something that we necessarily adopt directly and use. That's not true all the time.

There have been many cases over my years here, where we have completely replaced some bespoke custom thing with something from the provider for sure. For example on our private spaces product, one of the design intentions we went into when we built that was how much more AWS could we use to power the isolated experience that we're trying to build. We ended up reaching into some AWS systems, like SWF, Kinesis, Dynamo, a bunch of stuff to try to rethink what the Heroku experience might be if we were to assemble it, not just with AWS components, but with a modern take of a few years ago, but a modern take of what could we do today if we did it fresh.

That was a lot of grand rewrite, but a big stab at an intentional single tendency version of our product. It was a snapshot of the recent approach we've gone to again and again and again, examples being Dynamo. Dynamo is a thing Heroku adopted rather early on in its lifecycle, because it solved huge problems for us. S3 has always been around for us as well and it would be a disservice to not mention S3. It's probably the most reliable thing Amazon's ever built and it's wonderful. We use S3 for everything.

Yeah. I mean, I know every time if it fits and works, we will – we often rather pay for something from AWS than keep something bespoke and custom, or even open-source internally up and running.

[0:18:14.7] JM: Did Kubernetes and the grassroots volume of people who have joined in the Kubernetes community, has that affected your infrastructure strategy?

[0:18:26.5] MT: Oh, for sure. I think that the interesting thing about Kubernetes that is a concrete problem that Heroku has is that back to – we mentioned that we've been in the cloud forever, we've all dealt with the same infrastructure abstraction problems that anyone out there is dealing with. The same orchestration of underlying infrastructure resources that powers the experience we're trying to build, like Kubernetes squarely solves.

For years, we've been playing with and trying to leverage Kubernetes as much as we can and where it makes sense for sure. I think that especially when you look at some of the behind the scenes infrastructure orchestration work that Heroku does that again, Kubernetes does offer some interesting off-the-shelf solutions for stuff that that we've written and we have leveraged that for sure.

It does influence us, because part of the explosion of people working on infrastructure orchestration-style things, especially as they're trying to power PaaS-like application experiences within the Kubernetes ecosystem, it means that and especially as a company like Heroku, we get to potentially leverage some of that stuff to level up and power our platform better, or integrate in more interesting ways, so things the OSB API are really interesting things where Heroku has a direct lineage in where some of those ideas came from. It's really interesting for say as to offer some of our data services via that way sometime in the future maybe. There's just really potential stuff where the Kubernetes ecosystem exploding has made some of our lives easier. We definitely leverage it when we can.

[0:20:10.3] JM: I had a conversation with somebody at Heroku, I think it was a couple years ago. It was around whether or not it would make sense for Heroku to have a Kubernetes offering. You obviously don't have to discuss that. I imagine it is an open question, or an interesting question because we're in this phase right now where a lot of companies are deploying their own Kubernetes clusters for reasons that I don't know, I think they may regret in the future. I mean, I could be totally wrong, but I feel there is a bit of a –

[0:20:54.5] MT: Jumping on the bandwagon?

[0:20:56.8] JM: Jumping on the bandwagon. Like does this really solve a problem that you have? I know there's plenty of companies where it does make sense to operate your own Kubernetes clusters. Uber is probably at a such a scale where they want to be and their economics make sense in terms of running their own Kubernetes clusters. Heroku, obviously it makes sense to run your own Kubernetes clusters in certain context, or maybe you would want to go on Amazon, manage Kubernetes, maybe that would make sense for you.

If you're like, I don't know, like a small insurance company, probably you should not be managing your own Kubernetes cluster. Therefore it opens the door to do you want to even be offering a Kubernetes cluster product, or do you want to just offer the containers and manage the Kubernetes under the hood?

[0:21:43.4] MT: I don't necessarily think that I even think of it as a dichotomy that way. I think that when you look at Heroku and this is just my own 2 cents here, we've always sold you an experience on top of whatever the underlying containerization tech is. Now, Heroku today uses two different container stack, so whether you're using the private space side of things, or our common runtime side, which is the classic Heroku pot everyone's familiar with.

The Heroku side, we use LXE and we have for the longest time. On the private space side, we use runC and that whole open container ecosystem. The interesting thing that we are able to do that because of is that we have our own little image format, the slugs thing that you might be familiar with, if you did docked Heroku, or if you've worked with Heroku before. Slugs are very similar to Docker layers. There's the file system abstraction, or collection of tar-zip files that represent your applications to specific layers on top of a base image that we provide to the systems. That's what your slug is, just the delta between your app and our base image. It's easy for us to shim that slug into whatever underlying container runtime it is.

For us, it's totally reasonable and not easy in the easy work sense, but easy in the conceptualization sense to look at the problem and say, "Could Heroku orchestrate containers running inside Kube somewhere and power its experience, the Heroku experience?" I think that

that's a 100% true. I think that's why a lot of companies are out there today trying to become, or build the Heroku on top of Kubernetes, or be that app workflow provider on top of Kubernetes.

I think that again, similar to how the business is hard to operate, it's the sustained operations and service of that stuff it's hard too. For me, it's I always think there's probably space for Heroku to maybe orchestrate stuff in Kubernetes, or maybe it's always orchestrating stuff in EC2. I also think there's always going to be customers who reach for Kubernetes and want that experience. It's just really hard for me to say where Heroku fits into that tail long-term.

I can't talk about maybe what we would do, or are doing out loud, but we do look at Kubernetes and it's easy for you to imagine this, want you to see maybe consider orchestrating stuff there. That's also at the same time maybe something we don't want to do. Maybe we're okay living in that sweet spot of that 12-factor web app. Yeah, I just think it's interesting future in.

[0:24:29.3] JM: That's exactly the point. See, I have used Heroku for years from personal college projects to a company that I built on Heroku. Well, I mean, arguably two companies that built on Heroku; neither of which were very successful, but just I was comfortable with it. I mean, this gets to the selectivity question. There's a book about Apple that came out recently called *Creative Selection*. A lot of it is just about the Apple creative process.

I think a lot of what has given Heroku some long-term value is it has made that selection process, which the layer 1 cloud providers can't really do that. They can't say no to some new infrastructure trend. You just can’t say no, because then you're just like, you're not the fully fledged buffet of services. You have to say yes to everything, basically. That's how you end up with this gigantic catalogue of services that are hard to navigate through, hard to standardize the integrations for.

On the other hand with Heroku, you have been very selective with just things that have been proven to be desired by the developer community. It's a very interesting product design challenge, because you want to wait for the market to validate a product sector, like continuous integration or something, but you also don't want to wait too long.

[0:25:59.9] MT: Exactly. I think that that's also complicated by especially when you deal with market competitiveness too, right? Like you mentioned, layer 2 providers, but also layer 1 providers creeping up that stack, getting better and better and better. Where we expend our energy in leveling up the platform's capabilities is really important. We pay a lot of attention to how we spend that energy. It's really important for us to make calls that aren't good for us, because there's no way that Heroku is going to directly compete with AWS, or is there a GCP. That's just not the game we play in. We're just not that game.

It's a struggle, right? Because we do have big enterprise product offerings and we do have large customers that have demands for us. Part of that private spaces, product offering that we have now was because of the loud demand that customers had for an isolated boundary from the rest of the multi-tenancy behaviors that they were experiencing, not just on the Heroku platform, but just around stuff they were dealing with in general.

Also, because they like the network boundary that comes with it. That was an interesting thing that customers liked from us too. Yeah, it's like, we were very selective about what we add until we're sure it's what people really want. Even then, it's got to make sense for us. It's got to make sense for the product.

[SPONSOR MESSAGE]

[0:27:24.2] JM: Looking for a job is painful. If you are in software and you have the skill set needed to get a job in technology, it can sometimes seem very strange that it takes so long to find a job that's a good fit for you.

Vettery is an online hiring marketplace that connects highly qualified workers with top companies. Vettery keeps the quality of workers and companies on the platform high, because Vettery vets both workers and companies. Access is exclusive and you can apply to find a job through Vettery by going to veterry.com/sedaily. That's V-E-T-T-E-R-Y.com/sedaily.

Once you're accepted to Vettery, you have access to a modern hiring process. You can set preferences for location, experience level, salary requirements and other parameters, so that you only get job opportunities that appeal to you. No more of those recruiters sending you blind

messages that say they are looking for a java rockstar with 35 years of experience, who's willing to relocate to Antarctica. We all know that there is a better way to find a job.

Check out veterry.com/sedaily and get a $300 signup bonus, if you accept a job through Vettery. Vettery is changing the way people get hired and the way that people hire. Check out veterry.com/sedaily and get a $300 signup bonus if you accept a job through Vettery. That's That's V-E-T-T-E-R-Y.com/sedaily. Thank you to Vettery for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:29:13.2] JM: You've mentioned the private spaces product a couple times. I'd like to drill into that as a case study in how you build products and as well as just a – I think a deeply difficult engineering problem. I mean, you're not an infeasible engineering problem, but just one that has plenty of complexities. The private spaces is an isolated runtime model, in contrast to the multi-tenancy situation that can be exposed to a noisy neighbor issue. How does the private spaces deployment model differ from deploying a multi-tenant in the fully exposed, I can be on your EC2 instance with somebody else model of deploying a Dyno?

[0:30:01.2] MT: Are you asking from the customers, like how you might as a customer experience it? Or are you asking from how we implemented it?

[0:30:06.9] JM: More from the implementation point of view.

[0:30:08.3] MT: Sure. I'm going to talk about the multi-tenant side, because it helps explain the delta between the two really well.

[0:30:12.8] JM: Yeah. Yeah, sure.

[0:30:13.5] MT: On the multi-tenant side, it's exactly like you imagine. We have pools of AWS instances sitting out there. We have this app concept at Heroku. An app contains a collection of process types. Internally, we call those formations. Each formation has a given scale, zero to whatever you like. You can roughly imagine that when you create an app that there's this record

of that app that exists in the system that knows that there's apps a thing. Then when you push code to Heroku and you give us a proc file or instruct us via the API or some other mechanism the formation needs to exist within this app, the way that that is treated on the multi-tenant side of Heroku is we start the container orchestration process.

Right then, we're going to pick a random, or some other scheduling algorithm runtime node to schedule that container on. We're going to execute that decision to put the container on there. Once we got a signal that it's picked up, we're going to start the preparation of routing rules to that given container for whatever domain names are set up for that application. Then eventually, that's going to converge into a good state and that Dyno or container will start servicing requests for that given domain set.

We separate the routing layer from the execution layer, that container layer. The orchestration layer that I mentioned deals with two things; the scheduling of that container onto a node and then also the providing of information to the routing layer that that node is routable. The routing layer has its own behavior the way it deals and handles with that state change. On the orchestration layer, that's the whole kit and caboodle.

On the private space side, because of the isolated nature of this, there's a bit more familiarity and ability and willingness for us to tightly control the instance, counts and things that might exist inside that private space. You can imagine that flow being roughly the same, one in which that formation creation or scale request comes in, but the orchestration components that sit inside that boundary are a lot more familiar with how to interact with the underlying infrastructure directly, instead of depending on outside systems to provide those scaling behaviors.

The big difference is that you could imagine that the flows that we use to deal with interacting with a container fleet change in the private space side, it has instance capacity or other resource capacity concerns within that logic flow that don't exist on our multi-tenant side, just due to the scale at which the two separate – the two things operate.

On the multi-tenant side, there's many thousands of things and the systems that scale react to fleet capacity requests and changes our big systems on into themselves. Just a way that we

approach capacity management and fleet management in response to customer initiating scaling is the big difference between the two. Does that make sense?

[0:33:22.0] JM: Yeah. Can you go into a scale-up example? Let's say I'm Uber and I'm deploying my whatever, my marketplace application to the public space versus the private space. How is the scale-up story going to be different in those two environments?

[0:33:45.7] MT: Sure. In the Heroku, the multi-tenant model, scale-up, so basically the going from zero to one or number to some end number, is going to result in the scheduling of execution runs, assuming there's capacity. The orchestration layer is going to say, “Hey system, do you have enough memory out there for me to shove 64 gigs of work at you? Whatever that might be.” If the if the system believes it can do it, it will attempt to do that schedule and then come back success or fail and then do the router manipulation bit I said before.

On the private space side, that's not the case. On the private space side, we would know based on the scale whether we have the capacity, sorry, that scale request that the customer is making, whether we have the capacity to service the request or not. The customer comes in and says, “Hey, I want to scale this thing up to from zero to 64 processes,” then we're going to be immediately in this process of scaling, or creating 64 compute resources on the EC2 side, or the AWS side, or reusing 62 resources that were there before. There's very quickly that immediate decision, do I need additional resources in the cloud provider to service this thing? If so, start that process.

That exact decision flow doesn't exist on the common space side. There's just this admit of – there's an insufficient capacity problem here. I can't schedule this resource. There's going to be an out-of-band system that responds to the signals and then starts capacity handling. On the private space side, again, that scale-up flow is going to immediately start provisioning AWS resources, or again, reusing stuff that might be there if it's there. More often than not, it's going to be a provisioning of AWS resources.

The biggest difference between the common runtime experience and the private space experience for those customers would be they're going to notice that provision time every time. They're not going to notice it on the common space side, because we can have huge Slack pool

capacities and things like that, but you could imagine that we don't for any given space have 64 sitting idle instances ready to go for that given customer, especially due to the isolation boundary that they run in. In the private space model today, then it would be a pretty slow boot there.

[0:36:07.0] JM: Interesting. If I understand correctly, I remember from a previous show with Heroku that I did, Heroku does have this fascinating engineering problem of you have to keep a pool of instances that you can schedule new containers onto. I don't know if those are reserved instances, or if they're spot instances, or maybe you just have an entire layer that's responsible for grabbing these instances and basically saying, “Hey, Heroku wants those.” Then we're going to do something with them and then you can take them out of the pool.

In any case, you're saying that if for no other reason than the networking boundaries, you can't do that for their private spaces, because with the private spaces, you need a specific type of instance, you need an instance that is network partition in a way that it is friendly with existing private space instances for that particular customer.

[0:37:04.6] MT: Yeah. I think it's important to talk about that. Don't just isolate at the VPC layer. This is present state model for private spaces are such that they're split at the AWC count layer. Right now, each private space is an isolated AWS account and that just adds a whole another layer of resource sharing hurdle that is hardness to deal with.

[0:37:24.7] JM: Wow. Tell me more about the networking requirements for building private spaces.

[0:37:32.9] MT: Sure. Part of the design intent with private spaces where we wanted to offer a sort of – not sort of. A actual HIPAA, PCI and SOX compliance environment for customer workloads. We squarely targeted PCI DSS as the initial compliance regime. We wanted to comply with in that private space initial architecture.

What comes with that is modeling what is or isn't cardholder data. We have a lot of energy that goes into in systems in there that completely isolates the customer’s software and data from our

software and data, because of not wanting to have our systems be able to penetrate, or have exposure to that cardholder data environment that the customer is really responsible for.

That really is just what comes down to it is just really firm boundary, super firm boundary between the customer's data and the importance of it versus our data and it's not important in comparison to the customer's data. What that means topologically and networking wise is you can imagine that there's more sub-nets in our life, more security groups between those sub-nets.

There's huge isolation boundaries between the resources that sit in these spots that are intentionally there to block off communication, all in hopes of making it hard to communicate. That's all coming down to both topology and just other structural differences between the two sides of the platform.

[0:39:12.1] JM: What was the hardest engineering problem that you had to solve where you're building private spaces?

[0:39:16.5] MT: People. Let's see here, hardest engineering problem.

[0:39:21.5] JM: You can go deeper on people. People is interesting. People is very interesting.

[0:39:26.4] MT: I think that Heroku did somewhat of a classic engineering mistake with the private spaces product in which we assembled a group of folks together. We picked and choose good engineers from around the org and blessed them with the decision to go build this new platform product, this new version of Heroku. That instilled a lot of negative sentiment with the people that were "left behind" with the legacy stuff.

That was never really the intent behind what that product initiative was and it never really manifested itself. A lot of the way that we presented and talked about that project internally really gave that sentiment and feeling to people that they were being left behind in this legacy Heroku. It couldn't have been further from the truth. I really think the biggest challenge, again, I know this isn't the engineering challenge, but the biggest challenge in my time with dogwood was overcoming that perception that product was there to replace everything else everyone else

was working on. That was really hard for us, because that was never the goal. It was just again a single isolated version of that cedar product, that common space product that people were familiar with.

[0:40:46.5] JM: Wow. There was a misconception that this was a total – this is the second version, or this is the next version of Heroku, but this is an adjunct product. This is like a totally different product.

[0:41:00.7] MT: Yup. You could imagine the sentiment that it caused internally and that was a really rough time spent. It's not so much you don't dwell on it, but when you just reflect back on those moments, there was just a lot of rough interpersonal stuff that occurred because of the perceived boundaries that created.

[0:41:20.4] JM: I mean, that stuff literally happens at every company. Miscommunications just happen everywhere. I mean, Heroku is a partly remote company, right? Solving these problems with a partially remote engineering team, that's even more novel. We don't we don't really have good patterns for disseminating information in remote organization yet, I don't think.

[0:41:44.3] MT: Yeah, we've been 80% remote or higher the entire time I've been at Heroku. My entire experience as an engineer here has been working with predominantly remote engineers. It's hard to communicate sometimes.

[0:41:59.1] JM: Pros and cons of a remote engineering team, how are you feeling about it?

[0:42:03.2] MT: I have spent the last decade only doing remote jobs like this. For me, it's the best thing I've ever done in my life. Because and I'm a fierce defender of spending time with my children. I have two young kids. Every day, they get off school at 3:00 p.m. I will just piece out of whatever I'm doing and just go hang out with my children every day. I have that benefit and that luxury, because of the way Heroku and Salesforce operates the remote workforce, man. It's really great for me to have that respected boundary and my own personal life that I get because I'm a remote employee.

Getting back to just the work side of things, I also appreciate working with a remote crew that takes an intentional effort into making that team effective. I just really like tearing into the problems that teams experience with that remote angle and seeing how we could do it better. In the five years I've been here, there's been a lot of these remote-ish, remote-related problems we've dealt with and they've all been fun to deal with. It's been really fun. I don't know. It's just been a fun way to live and I really appreciate the personal benefits that I get in my own personal life from being a distributed employee.

[0:43:19.5] JM: Isn't it cool watching this development spread across the working world, like the ability to work remote and the increased adoption? It is a totally new way of doing work. Yes, some people have been doing it for 10 years. It sounds like you were an early adopter and you have some extra savvy, but it's cool seeing it propagate. What are some words of wisdom? What are some learnings reducing those kinds of communication gaps and issues that are just inseparable from the gratuitous benefits of working remotely?

[0:43:58.2] MT: I think I mentioned this word a moment ago, but I think that I have to pair it myself again and say that there's a lot that makes really effective remote teams work that is in parallel or similar to a bunch of other intentional relationship approach stuff. What I mean by that is that if you are able to collectively agree on how you guys work together, or how that group works together, it makes work easier.

What I mean by that is if you guys commit to having to stand up on a Monday and a Friday and you just make that happen and then you also commit to having a conversation, if it's not working anymore and you always make sure that it's a safe space to have those conversations where you have the critical eye on those processes that you're doing, then you never put people into an uncomfortable position where they feel they can't make a change to those processes that makes their life better.

I guess for me, it's approaching the way you work with a bit of flexibility and understanding that the other person that other end of line has just as much of a complex life as you do and as much random shenanigans coming into their life, because of that remote nature that just being a bit flexible is useful.

The other thing that I think comes with especially as you grow your career with an organization and rise up through the ranks, I think it's easy to – it's especially poignant lately, but it's easier to fall into that manager schedule that is really aligned to some business hours in a specific time zone. If you're not super, super careful about defending your working hours, you'll lose them. People will start injecting themselves into times of the day that you don't want. I think it takes really important defensive time management skills to be protective of those moments that you want for yourself, because you need them to survive as a remote employee.

[0:45:59.8] JM: All right, coming back – Those are great words of wisdom. Coming back to the engineering side of things, actually one thing I'm curious about is the product development side of the house. Because we've talked a little bit about just the uniqueness of being in a position of a layer 2 cloud provider that does this process of creative selection, where you're throwing away a lot of product ideas that you could do. Or not throwing them away, just saying we have to say no to these things because the things that we do we're going to have to do really, really well and we want to keep the surface area of the product low enough that it remains approachable. Tell me about the product development process from what you've seen at Heroku.

[0:46:43.6] MT: Sure. We've always had a product organization within Heroku, with product managers and product owners. For the most part, they are the owner. They own the shape, or feel of the way a part of the platform is or feels or functions. You could imagine Heroku has this add-ons marketplace, where you can get a database, or a static IP address, or some other resource attached to your app. We have a product manager who owns the way the whole ecosystem feels.

That person is there to be the decider about what we do and what we change in regards to those features. The idea Genesis where stuff comes in and what we might do really comes from all over, but it is a combination of customer feedback, internal engineer suggestions, random feedback through Slack or e-mail or whatever. Then those end up into the product manager's mind, or roadmap in some way, or list of features that they might want to do.

Then they work with engineering leadership like myself within the various parts of the org to figure out how and if we want to do some of those things. That's how most features end up flowing through Heroku is that exact same flow. Some of the sides we want to do something.

They propose it to a product person, usually maybe they talk to their peers first or something like that. Maybe it's a customer doing that exact same thing and that customer asks their sales person, or that customer asks support, or that customer with in a support case asks an engineer directly, or an engineer gets asked by their friend who runs on Heroku. All of that, we're trying to flow through our product management org.

Over my time at Heroku, it's interesting that I've seen that get better and better and better. I would say that today, more so than ever in my time, we're very good at making sure that product shape and direction-oriented stuff flows through the relevant product owner's mind or area. We still expect and have tons of participation in the iteration and creation of new features, or new ideas, a new PaaS we could go down for the product from the individual folks down on the ground working on it.

[SPONSOR MESSAGE]

[0:49:14.4] JM: Today's sponsor is Datadog, a scalable monitoring and analytics platform that unifies metrics, logs, traces and more. Use Datadog's advanced features to monitor and manage SLO performance in real time. Visualize all your SLOs in one place. Easily search, filter and sort SLOs and share key information with detailed intuitive dashboards. Plus, Datadog automatically calculates and displays your error budget, so that you can see your progress at a glance.

Go to softwareengineeringdaily.com/datadog and sign up for a free 14-day trial and you will also get a complimentary t-shirt from Datadog. Just go to softwareengineeringdaily.com/datadog. Sign up and get that free t-shirt.

[INTERVIEW CONTINUED]

[0:50:13.7] JM: I'd like to get your perspective on the broader space of the cloud technologies today. One area I'm interested in is the history of these open source infrastructure platforms. The things that were around before Kubernetes, you have OpenStack, you have Cloud Foundry. I'm wondering what is different about Kubernetes, if there is something different, if there's something different about the community, or the way that the API surface developed, or just the

timing. I assume you've seen some of those previous infrastructure platforms and I just love to get your perspective on from a historical point of view.

[0:51:00.2] MT: Yeah. This is just my own personal feeling about this. I think that – that's not any way to addendum what I'm saying. I feel like I say a lot about lucky things that way. I think Kubernetes is really – it's a combination of a couple different things, but I think it's timeliness and also that people needed a way to get Docker containers onto the Internet that worked all over the place. If a structure providers, like GCP specifically really saw the leveraged power that there was in making sure that there was a ubiquitous interface for describing what that infrastructure might be.

I think that in combination of timing, both in just the way the Heroku – not Heroku, but the Kubernetes product just ended up in computing, but in concert with cloud providers wanting to find a way to compete with each other better meant that Kube provided a really good interface for us. That comes from the perspective that it's exactly that for us at Heroku. Kubernetes looks an awful lot like the perfect way to abstract away that infrastructure bootstrapping problem we have and some of the baseline infrastructure management problems we have.

For me when I look at Kubernetes, I see the layer 1 providers betting that if they get more companies targeting Kube, that the transition story between your cloud and my cloud is simpler, but we're going to keep you sticky based on the value add stuff, like the rest of the tool chain AWS provides, or the rest of the tool chain Azure provides, or GCP, or whoever it is. I believe again that it's squarely in the customer growth and attraction perspective for tier 1, or layer 1 providers to have a pretty good Kube offering. That's where I don't think it necessarily so much sense for Heroku. Again, I think if you look at it from a way to leverage getting our stuff into a cloud, it makes sense, but maybe not for Heroku.

[0:53:05.9] JM: Yeah. I mean, I'm like, “Heroku, please do not make me interface too late. Just keep me from having to do that, please.” It was such a brilliant jiu-jitsu move by Google to get into the cloud space via releasing Kubernetes. It’s the thing that just makes me feel very happy.

[0:53:27.7] MT: You talked about Cloud Foundry and OpenStack before and they all had those same Chef. There were no good deployment stories for things. There were no good ways of

doing that. There's big companies out there that sold you complicated Chef systems for managing your OpenStack install. Those are never good. Nowadays, you can get a managed Kube cluster from Google and go a pretty long distance. That was the story just couldn't do a couple years ago.

[0:53:58.4] JM: I almost feel – I may be not qualified to say this, but I always feel it was necessary to go through those growing pains, through those different generations of first, the OpenStack where we learned as an industry okay, there was some politics, there's some political problems between the companies.

Okay, we're going to need a better community. Then Cloud Foundry was great, but it was very much territory that was entirely in the purview of one, or just a few companies. Then the Chef and Puppet and Ansible, that whole generation was this stuff is too imperative. We're scripting everything and it's in a bunch of different scripts and it's too – it doesn't fit the architecture that we need.

Then finally with all these learnings, we could take us in then Docker. Docker was just timely. Finally with all these learnings, we could say okay, let's put this into a single platform and it's Kubernetes.

[0:54:50.0] MT: I think they did a couple things right from the beginning. They abstracted away things like load balancers as a generic interface from the very start. It was really easy as an infrastructure provider to look at that and say, “Oh, man. I could slot behind my AcmeCloud LB and service this YAML request from this customer.” Again, it's a really interesting computing thing.

[0:55:12.3] JM: Do you have a sense of how advanced, or production-ready the operator-based deployments of these database solutions, or Kafka – Have you worked with operators much?

[0:55:27.2] MT: Yeah. We’ve definitely – we poke at competitors. Part of the operator approach for provisioning resources that appeals to us in one angle that we like is again, that isolation boundary one. You could imagine that an operator approach to managing data resources within

a cluster boundary is amazingly the exact thing we would want. The one of the interesting wrenches for Heroku is that oftentimes, we want to make something ephemeral.

For myself, that almost sometimes includes that cluster and oftentimes I want those data resources to live well outside of the lifecycle of that individual Kube cluster. That's where things like relying on operators for databases just rubs me personally the wrong way. I just think that some bootstrapping cluster life cycle stuff around that doesn't necessarily align with how ephemeral I think you should treat Kubernetes sometimes.

[0:56:25.9] JM: That's profound. It's actually a really good point, because people are starting out about, “Oh, your Kubernetes cluster. It’s camel. Not pets.” It’s like, “Well, uh.” If you start to put an operator in there, then I don't know. Maybe you have some fault-tolerant, maybe you have a Kubernetes cluster that's – you can do the Kubernetes federation thing, like one of your Kubernetes clusters is the super-sensitive, always up Kubernetes cluster and the other ones are the cattle. Would that work?

[0:56:53.4] MT: For sure. Perhaps. I also think that this will get better and better and better. Also, I just want to be clear. I don't think that any – the true story with Kubernetes from the start is that every time these operational entire application lifecycle problem becomes a problem, a solution comes out there.

Even things like the operator. Just imagine of popular operator that orchestrates AWS databases on your behalf. There's ways for you to make those resources live beyond that cluster boundary just by the way you orchestrate the creation of those resources within the provider. With judicious careful creation of those resources, you can do it as well. In that case, I think the operators are fine. I just think you got to be careful. Anything, you got to be careful.

[0:57:42.6] JM: The concept of service mesh, is that appealing for you, or for a company like Heroku?

[0:57:47.4] MT: It's complicated, because I think that there are use cases where service meshes make sense for some workloads, but not necessarily for our own internal use. Almost always, our customers want some service mesh style behavior between their services and processes

running on the platform. We are often in a situation where we are considering, or reasoning about, or working on ways in which we might offer service mesh style behaviors attached to Heroku's container system, but nothing's really stuck, especially when you take a look at the private space isolation boundary. Almost all the customers that want the service mesh are almost also willing to pay for the private space and just get the isolation boundary and just do inter-process communication that way.

We offer service targeting and stuff like that that really alleviates some of the other benefits you get from some off-the-shelf, or just some service mesh concepts. I don't know. It's been tough for us to find a good fit within Heroku. That's just our Heroku hat. Speaking more probably about Salesforce, there has been definite advantages of service mesh approaches to some services we might offer.

[0:59:11.9] JM: Okay. Just a few more questions to wrap up, do you have any predictions that you haven't discussed yet about how the Kubernetes ecosystem will evolve?

[0:59:20.7] MT: I personally think it's going to keep getting easier and easier and easier to run applications on Kubernetes in the same way as using heroki mean applications. As a developer, I have WordPress. I think it's going to be really simple to go from downloading WordPress on wordpress.com, to getting a bespoke deployment on some Kube somewhere rather simply. I get all kinds of signals from that all over the place that that –

[0:59:50.7] JM: I certainly hope. I'm spending way too much on my WordPress host. I'm spending way, way, way, way, way too much.

[0:59:57.0] MT: I think everyone's aiming – not everyone is aiming for that, but I feel I that deployment story to Kube is really, really solved simply, it's going to be interesting. Really interesting, I think.

[1:00:09.3] JM: Yeah. WordPress and GitLab are the best examples of can you do this in two seconds yet? I'm waiting.

[1:00:18.7] MT: I think Kubernetes is so big. I also think that you're going to see if you're attached to CNCF stuff this isn't surprising, but I think you're going to see an explosion or adoption of the cloud provider-admin interface stuff, I think. I think part of OSB –

[1:00:34.2] JM: Oh, like build your own cloud provider stuff?

[1:00:36.3] MT: Not build your own cloud provider, but the OSB admin API stuff is –

[1:00:39.6] JM: That's open service broker.

[1:00:41.2] MT: Exactly. Dealing with the administrative side of what the Kubernetes service catalog might interact with is a poor story at the moment. I think part of leveling up say things like an operator provisioning services, but more specifically cluster local things trying to provision stuff via OSB, I think that you're going to see more providers say integrate better with Kubernetes quickly once some of the provider API stuff is better. Doing things like a credential rotation for a data store from a provider's perspective in an arbitrary Kubernetes cluster is a hard problem today.

[1:01:16.9] JM: Okay. That's a great answer. All right, last question. How will the cloud provider market look different in five years?

[1:01:22.9] MT: If the last five years are any hint about what the next five are going to be, I think it's like I mentioned before –

[1:01:27.6] JM: Who knows?

[1:01:28.7] MT: Who knows? I think the one for sure guarantee is you're going to continue to see AWS, GCP and Azure chase checkmarks.

[1:01:38.3] JM: Right.

[1:01:39.0] MT: I've mentioned this again a couple times, but I think about it as creeping up the stack a bit. You talk about layer 1, layer 2. For me, the one is layer, one provider is a slowly,

slowly morphing into layer 2. That puts them into depend how you look at it, it's yet just another feature addendum to the layer 1 portfolio services they offer. I think that you're going to start seeing more and more of that stack creep up the stack that we haven't seen.

I especially think because you saw how quickly folks jumped on lambda, like lambda and server list ignited the minds of all of our friends around the world. Again, I think that as those providers start coming up with actual solutions to problems that are useful all the time, people are going to start using them all the time.

[1:02:29.2] JM: Mark, it's been a pleasure talking. Really good conversation.

[1:02:32.3] MT: Thanks for having me.

[END OF INTERVIEW]

[1:02:42.5] JM: As a programmer, you think in objects. With MongoDB, so does your database. MongoDB is the most popular document-based database built for modern application developers in the cloud era.

Millions of developers use MongoDB to power the world's most innovative products and services, from cryptocurrency to online gaming, IoT and more. Try out MongoDB today with Atlas, the global cloud database service that runs on AWS, Azure and Google Cloud. Configure, deploy and connect to your database in just a few minutes.

Check it out at MongoDB.com/Atlas. That's MongoDB.com/Atlas. Thank you to MongoDB for being a sponsor of Software Engineering Daily.

[END]