EPISODE 517

[INTRODUCTION]

[0:00:00.5] JM: When Box got started in 2006, the small engineering team had a lot to learn. Box was one of the earliest cloud storage companies with a product that allowed companies to securely upload files to remote storage. This was two years before Amazon Web Services introduced On Demand Infrastructure As a Service, so the Box team managed their own servers, which they learned how to do as they went along.

In the early days, the backup strategy was not so sophisticated. The founders did not know how to properly set up hardware in a co-located data center. The frontend interface was not the most beautiful product, but the product was so useful that eventually it started to catch on. Box's distributed file system became the backbone of many enterprises. Employees began to use it to interact with and share data across organizations. The increase in usage raise the stakes for Box's small engineering team.

If Box's service went down, it could cripple an enterprises productivity, which meant that Box needed to hire experienced engineers to build resilient systems with higher availability. To accommodate the growth and usage, Box needed to predict how much hardware to purchase and how much space in a data center to rent, a process known as capacity planning. As Box went from three engineers to 300, the different areas of the company went from being managed by individuals, to teams, to entire departments with VPs and C-level executives.

Jeff Queisser is an SVP at Box and one of the earliest employees. He joins the show to describe how Box changed as the company scaled. We covered engineering, management, operations and culture. In previous shows we've explored the stories of companies like Slack, DigitalOcean, GIPHY, Uber, Tinder and Spotify. It's always fun to hear how a company works on the inside from engineering the first products to enterprises with millions of users.

To find all of our episodes about how companies are built, download the Software Engineering Daily app for iOS or android. These apps have all 650 of our episodes in a searchable format. We have recommendations and categories and related links and discussions around the episodes. It's all free and also open-source. If you're interested in getting involved in our open-source community, we have lots of people working on the project, and we do our best to be friendly and inviting to new people coming in looking for their first open-source project. You could find it at github.com/softwareengineeringdaily. We'd love to see you there. We'd love to see you in our Slack.

Let's get on with this episode.

[SPONSOR MESSAGE]

[0:02:59.8] JM: Your company needs to build a new app, but you don't have the spare engineering resources. There are some technical people in your company who have time to build apps, but they're not engineers. They don't know JavaScript or iOS or android, that's where OutSystems comes in. OutSystems is a platform for building low code apps. As an enterprise grows, it needs more and more apps to support different types of customers and internal employee use cases.

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

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

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

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

Thank you, OutSystems.

[INTERVIEW]

[0:04:51.1] JM: Jeff Queisser is a SVP of engineering at Box. Jeff, welcome to Software Engineering Daily.

[0:04:55.8] JQ: Thanks. Great to be here.

[0:04:57.3] JM: It's great to have you. You were a very early employee at Box. You joined in 2006, which was just a year after it got started. Give me a framing of the early technology stack and the early problems that Box had to solve for.

[0:05:13.0] JQ: Yeah. So year was 2006. This was the year of the LAMP Stack or some part of that section. So everything with Linux, Apache, MySQL, PHP. That's what was happening at time, and so that's what we built on top of. We were kind of early in that way and I would say — And we can talk more about this later. We were very early on in the scale journey of Box also. So that worked for a while and of course that continued to scale things up past that point, but the LAMP Stack is what it was.

[0:05:38.4] JM: The first version was just a Box that was sitting there and you were waiting for to fill up, and then you had another server waiting, or what exactly?

[0:05:46.8] JQ: That's not terribly far off. So think — There was no AWS. Also, just as a kind of a reference point.

[0:05:54.4] JM: That's two years away, right?

[0:05:56.7] JQ: Yeah, something like that. Yes. So imagine the era of you go to something like a soft layer and you’re just nearest provisioning bare-metal machines, and so that’s where we restarted. Yeah, the scale numbers were not that big. So it was not terribly far from add another fileserver.

[0:06:11.3] JM: So soft later, that was a way to provision machines remotely, right?

[0:06:16.4] JQ: Yeah. We worked with a couple of different hosting parameters, but that’s just an example of you’re purely operating in a bare-metal world, and so you can use around provisioning tools in general, but they're pretty much delivering you or that model is delivering your just a physical server, which is your server.

[0:06:30.1] JM: Do you have to call them to set it up or can you just — Is there some sense of infrastructure as a service in 2006?

[0:06:38.1] JQ: Yeah. The API was basically a trouble ticket. So that’s not a great API.

[0:06:42.2] JM: Okay. But it’s not terrible. There was something else called like Slice Host or something around that time, right? There was like some semblance of infrastructure as a service, the first ideas of it.

[0:06:54.5] JQ: Exactly. I think the seeds were kind of planted around that. Yeah.

[0:06:58.7] JM: Did you do anything around virtualization around that time?

[0:07:02.4] JQ: Nope. In general, we've been relatively successful in just sort of loading up different workloads. So it actually was a while until we got into a more virtualized environment, just because a web server can crank it. 75% of CPU use didn't — And kind of the same types of utilization you’d get with virtualization.

[0:07:22.3] JM: So do you remember how many customers it took to get to a point where you needed to add additional servers? Did you started to actually systematically think about scalability?

[0:07:32.3] JQ: Yes. So at some point we were kind of clearly outgrowing the hosted model. And so we set up our — We used some co-location space that was here in the Bay Area, and another cofounder, so Sam Ghods and I, we basically taught ourselves how to data center and got a Cisco 6509 kind of big iron switch and a couple racks and a bunch of servers and learned about power engineering and cooling engineering and actually kind of did a cut-over.

[0:08:01.4] JM: Was that hard? Is it hard to learn that stuff?

[0:08:03.7] JQ: Well, it was hard, because also in 2006 there was just nowhere near, whether it was on the technical side or the business side of startups, they were just nowhere near the kind of vast resources that are now out there in the form of everybody's YouTube talks and slide decks and a thousand more people that have actually kind of hit appreciable scale. Yes. So when we're trying to figure out how do you actually build out a data center, we literally wired parts of the patch panels backwards and we had a vendor who came in later and said like, "This is exactly typically standard." So I would say that we definitely — There is a steep learning curve back in 2006 on trying to DIY some of these. I think we ramped up pretty quickly, but at the start, definitely pretty challenging.

[0:08:43.4] JM: Was it typical to DIY or did people tend to hire vendors from day one? If they just had a software product they wanted to build, they knew they had to co-lo it. Didn't most people just hire somebody?

[0:08:56.5] JQ: Yeah, and we were just — I think we were maybe 6 to 12 months away from when we started actually build out more of an operation staff. So that's clearly the right model. We were just getting started and being scrappy. Yeah, it was very long before we had layered in people who are data center experts and Linux [inaudible 0:09:12.8] experts and network engineers that had both the data center outing and the core internet BGP routing, and so we just started to layer that on after and built up a tech-ops org around all of that.

[0:09:23.6] JM: Of course you learned early on something that will become a canonical issue later on a Box. I believe it's still something that is creative challenge today, which is capacity planning, and this is an issue that every company that is a cloud provider or resembles a cloud provider has to deal with. Explain capacity planning is.

[0:09:45.7] JQ: Yeah. So capacity planning is largely what it sounds like. You're running a service. It's super dynamic. You have a large customer base that is billing on top of you, and so you need some amount of ability to have medium and longer-term forecasting to make sure that you literally have the infrastructure in order to support that kind of workload.

[0:10:04.8] JM: So what was hard about the early days of capacity planning?

[0:10:08.5] JQ: Well, some of this is we didn't fully understand the kind of footprints of the services. Some of it was that we were quickly adding functionality so that the actual capacity profiles were changing. Probably the most interesting and important lesson that we learned is that this — We ended up being most successful with the capacity planning when we made it more of a centralized service. I think if you have a team who's out running hard and they're working on building the next thing or maturing something, you can end up with a team that’s pretty saturated, and there is an amount of specialized kind of discipline and skillset and tooling, really, on capacity planning. So probably we had the most material increase in our capabilities there as we centralized that. So now we have a team, what we call the service engineering team, who’s focused on how do we think about our consumption across all the services, how do we think about optimizations, and it's one central team that kind of sits and looks out over that, and that's just been much more successful for us.

[0:10:56.1] JM: Okay. So they know what they should be tracking. They know how to forecast it. At this point I’m sure you have enough data to give you a pretty good idea of how to do —

[0:11:06.9] JQ: We’ve been incredibly predictable. So, yeah.

[0:11:09.4] JM: Nice. That's great. Was there ever a time when you had a spike that you didn't predict and you were like, “Oh, no! What are we going to do?”

[0:11:16.5] JQ: Fortunately we’ve never really wound up in that kind of position. There was a hard drive shortage. It was triggered by a natural disaster, and I think it knocked out. It turned out that there was kind of a supply chain single point of failure for hard drives a couple years ago and there was literally one factor that made the motorized servos at that level of precision. So the entire industry got backed up. So certainly we were kind of in the wave of everyone else who were making sure that we had the right kind of capacity from a pure storage perspective.

[0:11:42.3] JM: Do that mean potentially having to turn off the service to new customers? What would you even do with if that happened?

[0:11:51.2] JQ: Yeah. Fortunately, we run with a very long buffer. So we weren't really sweating bullets, and we had a whole bunch of different contingency plans and alternate providers and kind of PaaS that we could look at. But no, there was never a scenario where we'd do that, turn off the [inaudible 0:12:06.0]. Yup.

[0:12:06.3] JM: That's pretty cool. Did you understand what was needed to do redundancy and replication and multi-availability zone or whatever your level of replication is today? How did you get to that understanding and what was the level of replication of the data in the early days? The whole idea of Box is basically you put a file on the cloud and it doesn't go anywhere. It doesn't disappear. But, of course, that puts the ball in your court. That puts the onus on you. You have to replicate it. You've got to keep it durable, and basically any kind of tail risk scenario.

[0:12:44.5] JQ: Yeah. So we kind of think about the problem in two different domains. One is around some of the system-level metadata, and then the other is around the actual kind of encrypted blob storage. We have different strategies. So we operate with multiple N of the total, where N is the total amount of capacity, bits required to operate the site. So we have facilities in the Bay Area, we have facilities that are outside of the disaster zone that are going to be sort of far enough way and isolated where we're able to run the entire site from.

Security and availability are fundamentally the most important things we do as accompany. So we spent a lot of time thinking about both security and availability. So extensive design reviews, extensive kind of thinking through what ifs and scenarios and doing some adverse testing to make sure we're totally buttoned down on that.

[0:13:27.5] JM: In the early days, did you find distributed systems expert to consult or —

[0:13:34.9] JQ: Yeah, the V.01 of Box in 2006. I think at the very beginning, we were trying to really understand did we have a viable business model here? And so we always had backups, but I would say the level of maturity around that has certainly really increased since the very beginning. Yeah, we started out with a backup system, and as we layered in, as the infrastructure got more sort of sophisticated, as we continued to hire more and more sophisticated people, we've kind of invested more there.

[0:14:00.5] JM: Did they let you know they're like, "Uhm — It's kind of a skeleton crew system you've got here. So maybe add in extra layers of redundancy or put your backups, perhaps, away from Palo Alto," or wherever you were at the time.

[0:14:17.7] JQ: No major surprises as we brought on more people, but you can certainly imagine that scenario, the really senior person who comes in and takes a look around. But no, we didn't wind up there.

[0:14:27.1] JM: So when the cloud came out, did you instantly look at it as an opportunity to give — Because eventually you started using it for a — You can use it for emergency, overflow capacity, but I think the core of your business is such that the economics make sense to really just not use the cloud, because you're just going to really eat up a lot of your margin if you use a cloud. How did the cloud look in the early days, and how is your relationship to the cloud evolved?

[0:14:55.2] JQ: Yeah, that's a really good question. I think when AWS literally launched, I think we looked at it and we thought, "Yeah, this is obvious. This is really spectacular."

The interesting piece of our business is because we serve some of the largest and kind of most complex enterprise customers and organizations around the world, our customers really wanted to be able to get in and understand exactly what's going on and what are the practices, and they wanted to be able to physically visit our data centers, and they are subject to sometimes really high levels of regulatory or compliance scrutiny. That was unique that we are able to solve that problem, and some of the cloud providers were not able to solve that problem.

So I would say there's been a shift. In fact, public cloud is effectively not usable for us when it came out, and I would say in the last couple of years, as pricing pressure and a really robust kind of market around price structure has emerged, I think that's been good. I think as we see more and more cloud providers build track records and step up to the plate on some of the more advanced compliance requirements that's helped and just sort of general acceptance. So we're starting to use more and more cloud, and for us this is not a binary bit that we flip of we have to be all data center or we have to be all cloud and it has to be a religious kind of decision for us. We're making pretty informed decisions about, "If we can be in this region where we're not going to go credibly light up a pair of data centers, and if we can just do that and if our customers feel like this is the right infrastructure for them to run on, that's a really good opportunity for us." So we were absolutely layering in more public cloud, and that's really only been possible in the last couple of years.

[SPONSOR MESSAGE]

[0:16:26.2] 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]

[0:18:33.1] JM: Are you using public cloud for application development, like data engineering tasks or something like BigQuery, for example. Like you wouldn't want to write your own BigQuery, but that's quite useful for doing certain data engineering jobs, which I'm sure is the kind of stuff the Box wants to be doing with some of the higher-level services that you're developing now.

[0:18:56.0] JQ: Yeah. So we're definitely looking at what are the next kind of revs of functionality. A great example of what we're doing in the public cloud is actually we have a product that we announced that our last Box Works called Box Skills, and so think of it as a layer that lets you actually access both Box-provided machine learning skills and also public cloud provided kind of skills. So that might be with IBM, or Amazon, or Microsoft, or Google and they might be experts and have the world's best vision or video kind of analysis and we might be expert in something around the kind of document set. And so we are actually just plugging those indirectly. So we're literally using public cloud in a way that's going to show up as just a better product. So, yeah, a huge opportunity there.

[0:19:40.3] JM: So that's like a Box shim over these higher level cloud provider services that you're talking about.

[0:19:46.6] JQ: Yeah. We'll be able to plug in, and the advantage here is that no one wants to go get locked in to a particular API or do all the development work to go understand how to do this or make it portable, and so the goal for us is let's make it completely plug-in to your existing workflow and the kind of core data product Box that you're already using. Yeah, just making that drop-down sort of simple of like, "When this happens, push this thing out to X and get back the transcription." So we view this as a tool to build directly into workloads and to just make a really interesting technology that's going to otherwise be inaccessible and kind of locked behind the APIs, which is something that we can actually go democratize.

[0:20:23.3] JM: I think is really cool, because, much like in the early days, people took the different aspects of cloud providers, the early aspects of cloud providers, like one of your competitors who shall be not be named just took S3 and layered usability over S3, and that was a whole business. Today, the cloud providers are coming out with these really exotic type of tools, like machine vision or, you mentioned, like transcription, for example. We use a service for making our transcriptions, for making at least the first version of our transcriptions of our podcast episodes. It's Google Speech to Text, and it's not very usable. It's like an API, which is great, but if you were somebody at a company, like a legal company, like a law firm or something and you wanted to take just some piece of audio and you want to drag it to or you just want it right click it and make a transcription of that. The Google API is not built to do that, but that would be a perfect example of what Box would be able to layer on to the Box sort of file system utilities.

[0:21:37.7] JQ: That's exactly right. You could imagine almost every knowledge worker has some sort of use case where it would be great for them to, "I've got a thousand images and I need to classify them, or I have all these — I need to look for certain thing in contracts, or etc." Yeah, we think it's super valuable. We're really real excited about.

[0:21:53.8] JM: Do you have any other use cases that come to mind? I don't know , like doing large-scale analysis? I could imagine highlighting — Like you're looking in a directory and you've got a ton of PDF files. I could imagine highlighting all the text files, all the PDF files and then saying, "Do some sort of thing over them," if I'm just a knowledge worker, and then that thing keys off a map reduce or a spark job or whatever, but I'm just a knowledge workers, so I don't know anything about map reduce or spark.

[0:22:20.4] JQ: Right. Yeah. So we are looking at how do we make this successful the larger and larger data sets. So there is kind of one dimension which would be a one-off kind of use case. There would be something else which is set at the repeatable workflow so that you can just have a business process and when a new audio transcription comes in automatically do this and place it here and send me an email and on and on. Then there are sort of — I think what you're getting at here, which is are there larger scale operations that can be done? Then how does that actually get manifested as far as the interface and the result set, and that's also something we're thinking about.

[0:22:48.9] JM: Very cool. Let's shift into a day-to-day sort of discussion; management, and hiring, and operations and so on. You were there from three engineers to 300 engineers, or however many you have. Do you know how many engineers you have now?

[0:23:08.1] JQ: We're in that kind of zone.

[0:23:09.4] JM: Okay.

[0:23:09.6] JQ: Yeah, probably a little more than that.

[0:23:10.7] JM: So what were the key milestones getting from three engineers to 300 engineers?

[0:23:17.3] JQ: Yeah. I think there are a couple of distinct phases as I think about this. I think phase one in the startup as you have literally all of your engineers are in the same room and you have just complete mind meld. So you need effectively no formalization of any of your communication methods, because everybody is sitting in the same room and has the exact same context. So that's kind of phase one, and it's a really high kind of high throughput, can be really lot of fun and also very chaotic in that phase. But there's just a natural limit of how many engineers can actually kind of operate in that model.

So at some point you break into teams, and so maybe you have a frontend team and a backend team and that's kind of your next logical phase. Then as your complexity grows, that backend team no longer looks like a team, but it looks like I have a messaging infrastructure, I have a database infrastructure, I have a caching infrastructure, and I have a monitoring, I have platform as a service, on and on. Then you kind of get to orgs, and this is where you want to make sure you have some kind of senior leadership who can actually think about the macro-portfolio, everything that we're doing and make sure that we have all the right people in the right kind of positions. I would kind of say this as the progressions I everyone is in a room, to teams, to actually orgs that have much more complexity.

Then at some point, it starts to — A bunch of your process that kind of generally works well starts to break down if you don't actually build it up. We think a lot about operational excellence. If you have a few hundred engineers, how do you make sure that it's clear? What are the dependencies between X and Y? If we do this here, how does that kind of show up over here both architecturally and from a project — General product and project management perspective. So at some point you kind of want to build in lightweight systems that help you continue to scale once you get to the multiple distinct kind org's phase.

[0:24:55.3] JM: At the stage where you've just got everybody in a room, do you have things like KPIs and rules and regulations and stuff or is it just too early to do that kind of step? I mean, this is something — I just have a podcast, but we've got two other people that work with me full time, and I start to think about, "Okay. Should I be like setting goals? Should I be —" Like when do you put in the formalisms?

[0:25:24.0] JQ: Yeah. That's a great question. I think it kind of depends the grade of system that you're putting in place, but I think the concept is you want to have it basically all scales. So the concept of goals, I think supercritical, and a few years ago we did a roll out of OKRs at Box," and it was highly successful and it's very galvanizing.

[0:25:41.1] JM: Objectives and key results.

[0:25:42.6] JQ: That's right. OKRs are a system originally by one of the, I think, CEO of Intel, Andy Grove, and very, very useful. So yeah, I think you want the concept of goals as early as possible. I think, also, your fluidity will be higher. So it's completely fine to say, "There's 10 of us in a room. This is the goal. This is what we think. This week we think is the most important thing we can go execute on," and that can change the following week as long as everyone's rinse crystal clear on that. So I think you want the concept of goals as early as possible. I think you want the concept of KPIs as early as possible, although I think in really small teams, in small environments, you tend to hardly even have the instrumentation to make some of that visible. Ideally, you need to have some discipline early on so that you kind of instill that very early on in your culture, which is we're going to care about the following things. We're going to always pay attention to the user experience. We're going to pay attention to the error rate. We're going to pay attention to the cost per transaction, on and on. I think, conceptually, you want to start with KPIs early, although they won't be terribly sophisticated.

Then some of the larger kind of project and program management where there is one program that we're trying to achieve that involves 15 different scrum teams. That problem, definitely, you can pond down the line, and that shows up later.

[0:26:45.6] JM: What about the issues that developed when layers of management start to develop between yourself and then the engineers at the lower parts of the hierarchy. So you started about saying you went from everybody's in the same room, to you break up into teams, till you break up into entire organizational structures. At each of those breaking points, you have a new hierarchical step in the tree. How do things change around there, and do you have to add in additional rules or do you have to dissolve rules or is it more about the people you hire and about like developing trust and just trusting the people below you will propagate things up properly? Tell me about what you've learned about where you want to be systematic and where you want to just trust in the fact that person A is reporting to person B and it's going to work hunky-dory.

[0:27:47.4] JQ: Yeah, it's a really great question. A couple of things; the one overarching thought would be communication becomes incredibly important. Where it happens organically in the everyone's in the room together kind of phase, it no longer happens free and organically. So communication becomes massively, massively important. So what you really want to —

[0:28:04.6] JM: And you have to instigate it.

[0:28:05.5] JQ: Exactly. So, literally, I think about my personal coms cadence, and I have a document somewhere that literally goes over, "So on this cadence, we do an engineering all hands. On this cadence, I rotate through and have a casual lunch with teams. On this cadence, I do a manager's meeting." Definitely, you want to get really, really intentional about the communications kind of cadence one.

Kind of two overarching things would be context is incredibly valuable. So if you can share with people not specific prescriptions, but if you can share them the context on the business problem or the technical or product problem they we're trying to solve, you enable much more distributed decision-making. You really ossify as an organization when you do it — Or when every decision has to get punted up multiple levels.

So we actually expressly operate on this philosophy, which is that we try and always push decision-making down to the lowest level that makes sense. And so we don't want to centralize a bunch of the decisions where we can actually [inaudible 0:28:59.0] off our context and hire really brilliant people, do periodic check-ins and make sure we're on the right track and generally kind of get out of their way. So that's an overarching kind of approaches, is share a tremendous amount of context about what problem you're trying to solve and why it's an important problem, and then philosophically push decisions down to the lowest level possible. So that's on the kind of coms piece.

You still do need an operating system, and an operating system, like I was saying, on the cadence, you have to think about what are the points of engagement where I make sure things are on track or we have an opportunity to change the strategy or to course corrector, or etc. So that helps to just get very systematic on that piece of it.

[0:29:32.2] JM: What's an example of setting context in how you would push that context down to the edges of the organization?

[0:29:37.6] JQ: Let's say we're building a new a new feature where we think there's a huge opportunity. Rather than, "Here's the spec and let's go build this," I would sit down in our product team will do this and eng managers would do this, eng leadership team, would sit down with the team and set up the, "Here's the problem. We have X-customers that are running into Y thing, or we think we have an opportunity here," and we're hearing this in the field, and I would give real data points. So as concrete as you can be, is massively helpful on context in general. So here are — Literally, here are five customer quotes that said that if we did X, they would find it incredibly valuable, and we think there is a huge opportunity here, and the value prop, the magic moment for this product will be X.

There are probably five different valid routes that you guys as a team need to go think through and bring it back to us and we'll kick it around and we'll have some conversation and be a brainstorm partner, but this is fundamentally the problem we're trying to solve on the opportunity, and if we get to this magic moment, we think we win. That tends to go really well, and that empowers people instead of your prescribing exactly what to do and there's just very little autonomy and interest in just executing something that's already a spec, which is probably the wrong spec, because you didn't have enough context, because you didn't fully understand the problem.

[SPONSOR MESSAGE]

[0:30:49.7] JM: The octopus, a sea creature known for its intelligence and flexibility. Octopus Deploy, a friendly deployment automation tool for deploying applications like .NET apps, Java apps and more. Ask any developer and they'll tell you that it's never fun pushing code at 5 p.m. on a Friday and then crossing your fingers hoping for the best. We've all been there. We've all done that, and that's where Octopus Deploy comes into the picture.

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

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

[INTERVIEW CONTINUED]

[0:32:21.3] JM: I suppose that's going — The success of that setting context and pushing it to the edges of the organization, the success of that is going to be directly correlated to how well you hire, because if hire well, then as the mission propagates through the organizations, it's going to get more refined and improved. If you hire poorly, it's going to get worse and worse and muddier and the spec is going to never calcify into something that is going to be good. It's going to get terrible.

I think that's probably why like big bureaucratic organizations that maybe don't have a great talent tend to get those super hierarchical systems, where the smartest people are at the top and then they develop a super rigorous spec and then just tell people lower down what to do.

[0:33:13.4] JQ: Yeah, I think that sometimes people try and patch for people with process, and no amount of process will ever fully patch the people side. Yeah, you absolutely want to make sure that you have the right kind a team on the field. Yup.

[0:33:27.3] JM: Any suggestions for decentralized communication?

[0:33:30.7] JQ: Decentralized communication. So I'm not sure this will fully answer your question, but the approach I have kind of landed on is you have to go multi-channel. So if you have; this is the strategy in here's what we're doing, here's why we're important, you know a lot of people who are on the ground running and working on some very interesting, very hard problem and they're engrossed in that problem typically.

So if you just send an email that says, "this is the strategy for this year." You're going to — A, that will not I will not have been enough repetition; and B, you will just miss a bunch of people on that channel. So we try and have the same message and really bring it to life on an email, in an all hands, in company all hands and in eng all hands, and then some casual public conversations and having lunch with the team. So if you don't go multichannel, you're missing a huge opportunity on the communication front.

[0:34:13.6] JM: Yeah, that's great. I was at Amazon very briefly and one thing I noticed about working there was how frequently the core — Like, whatever, the 12 core commandments of Amazon were just like reiterated and repeated and we're worked into daily conversation, and it's pretty helpful. I mean, you don't even need to know about something that is going on in the shipping and taxation team, but because you know the 12 core principles, you can have some semblance of an idea of how that team operates. Yeah, they're being frugal.

[0:34:45.9] JQ: Yes, absolutely. Building a shared culture, shared values is a very, very important technique. Yes. Everyone can rally around that.

[0:34:53.2] JM: So given that we've brought an amount of gravity to the idea of hiring, how have you evolved the hiring process at Box? What does it look like today?

[0:35:04.0] JQ: Yeah. So we have gotten really focused on how do we become better predictors? How do we be more efficient in the process so that we're spending time in the right kind of places? How do we make sure that the candidate experience is fantastic? So we've gotten — We never did the how many manholes in Seattle kind of question, but I think we've just done a lot of refinement to make sure we're asking questions that we feel like correlate with actual day-to-day performance, and we share a lot of business context in the process so people can kind of hopefully be pretty pumped up by; this is the mission. Here's the kind of problems that we're trying to solve.

So we've done a lot of tuning of the overall candidate experience and the questions we ask. We've gotten very rigorous and formalized on the, "Here are our people that are approved to ask X, Y, that are basically trained on a given question." It doesn't help if you have a core of 10 or 20 people that are all asking the same question, but grading it in a spectacularly different way.

So we've tried to get much more standardized, and in general we're operating this thing much more like a kind of an overarching system, and in general, have been really happy with — I feel like we're bringing on really, really just spectacular kind of engineers.

[0:36:09.2] JM: What do you think about the trade-off between having a specific team do an entire hiring loop? Like you have a billing team, for example, and they're hiring a new engineer for billing and they just have — And they're desperate for an engineer. They really need it. I've heard about the problem where if you give billing team full autonomy over the hire, if they're desperate, they may compromise on the hiring. So at certain organizations, you have centralized hiring facilities that basically will double check or you'll have one person from another team in every hiring loop so that you don't have that kind of risk. Do you have any checks or imbalances like that?

[0:36:55.6] JQ: Yeah. We always make sure that there are multiple people from other teams that are interviewing, participating in the interview loop, not just make sure that we're maintaining the right bar, but also because there might legitimately be a better fit with another team. So that's a practice.

We watch this very carefully. So we look at all of the interview feedback and kind of random notes, and we make sure that we don't feel like there's a position where we're making the wrong call, and that's something that we look at at a couple different levels as part of the overall kind of check and systems.

I think the root of this question though is I think as a company you have to have a hiring philosophy, and that hiring philosophy can be, "We're going to massively empower managers and we're going to make sure that if we make some mistakes along the way, that we have a path to identifying that," or you can say, "We're going to try and basically — We'll do eight rounds. Talk to 60 people, and we're going to be pretty certain that we think this is the right person," and you could kind of air on that side, and you just fundamentally as a company decide what are you optimizing more; velocity, autonomy total throughput of the system, minimizing the kind of risk of thrash in your overall hiring system. So you have to pick that at sort of the top level. From there, then a bunch of the decisions become really, really self-evident. So maybe you'd say, "We have no risk tolerance for maybe bringing something that's not exactly the right fit. So those can be a committee, and it's going to be incredibly rigorous." Then another company might say, "You know, we're optimizing for throughput, and maybe occasionally we'll get something wrong," and then that's where you'd probably then go add more autonomy.

[0:38:19.4] JM: Okay. Let's talk a little bit about engineering, and I guess — No, actually operations. I want to talk about operations with you. So Box, I think of as a somewhat operationally-intensive company, because you're constantly doing capacity planning, you're rolling out new features, you're updating operating systems. I mean, you manage your own hardware, although I think it's still co-located, right? So I guess that's sort of outsource to the people at the co-lo. Like you can trust that they're taking care of the hardware, or do you actually have hardware engineers?

[0:38:50.9] JQ: Actually, at this point, we have hardware engineers. We’re responsible for the equipment. So all the way down to the electrical. So it is operationally-intensive. Yeah.

[0:38:59.8] JM: Okay. Do you buy data center or do you just rent space on other data centers?

[0:39:04.0] JQ: We’re were renting space, typically.

[0:39:05.3] JM: Okay. All right. Are there pros and cons to that? Like are there ever a time where you're like, “I wish we really owned this real estate and could do more creative things,” or do you just factor that out as a core competency?

[0:39:18.2] JQ: Yeah, I think it's just a question of scale, and so there is a progression, and let's exclude public cloud from this, but I think there's a progression here where you would start out in some managed hosting, like I described in 2006 Box. Then you might kind of graduate to co-location in a retail facility, and then you get to a wholesale facility. So the difference between retail and wholesale is wholesale is basically just the space, then you provide your own electricians and your own contractors that do all the installation, and so that's going to be in the multiple megawatts kind of zone.

Then past that point, you look at; do we actually want to build up our own physical data centers? And certainly there are some scale advantages that can come with that. There is also a lot of complexity in kind of all the real estate, 30-year depreciation timelines, all the construction kind of risk. So it’s just a matter of degradation of what's the appropriate kind of scale fit for a given time.

[0:40:07.8] RC: When you're owning so many different vertical layers in that stack, does it become challenging to make sure that when something goes wrong, the right team is aware that that error has occurred and that information propagates the team, and the team can figure it out in a timely fashion?

[0:40:28.5] JQ: Yeah, it's a great question. So because there are a bunch of different things that are all the kind of integrated systems, we approach each as its own infrastructure layer, and as much as possible we try and have those layers presented as pure APIs with their own — On a common monitoring platform, but with graphs and dashboards that really are specific to that layer.

So at some point it becomes not dissimilar from just a public cloud kind of compute machine, because ever and above that the kind layer, it can actually depend on, “This is the API and the interface,” and we have a team that wakes up every day thinking about making sure that their layer is kind of healthy and performing. So it is not terribly complex. You just build about the right monitoring and the right organizational structures and your design goal has to be, “We’re going to build these with some sort of modularity and API between the technical and organizational layers,” and if you don't do that, then you would wind up so commingled that it would be impossible to kind of manage.

[0:41:18.6] JM: Two years ago there was this Google SRE book that came out and it talked about the new way of — Well, not a new way, but it was basically Google's way of managing operations. This was a pretty popular book. It changed a lot of people's minds about things. Did it change your mentality at all, or had that mentality, the SRE way of doing things, had that already propagated to Box?

[0:41:43.6] JQ: Yeah, I think that had largely already propagated to us. So I think the failure mode that some organizations get into is that tech ops can be consumed purely by firefighting, and if you think of tech ops as a reliability engineering organization in a very literal sense of SRE, we were already aligned on that kind of approach, and I completely think it’s right approach. There is just no need to spend time on toil, which I think was a concept that they blew out to a broader stage and it's worth the investment to reduce your kind of total amount of toil.

I do think that there are distinct skillsets. They can happen to be co-located in a single engineer, but in general there is a distinct skillset around thinking through reliability engineering, and it’s incredibly valuable and to the extent that you can embed reliability engineers and teams, I think you get great outcomes. That's something we try and do.

[0:42:26.9] JM: So you are articulating — The difference between the SRE approach and perhaps pre-SRE operations, is that the SRE's are more proactive. Their goal is to have reliable infrastructure as supposed to fixing the infrastructure when it breaks?

[0:42:46.3] JQ: That's exactly right. Yup, I think that they get included in the design process in a first-class way instead of a very traditional dev and ops, their throat over the wall kind of model where the day before release, the ops team is screaming, because they realize that the software package is going to do X, Y, and Z and it's going to be a disaster. So yes, exactly. It's the shift from being generally proactive, versus generally reactive and feeling like sort of this is one team that's aligning on these common goals. So let's build some really — Something that’s going to blow our customer’s minds and do it in a really reliable way.

[0:43:16.6] JM: Yeah. Well, I’m looking forward to talking to Sam about the Kubernetes migration, because it sounds like that was probably helpful in in assisting the deployment process. So I know you’re running out of time. Why don’t you tell me your product development process as a way to close off. When you have a new feature that Box wants to develop, how does it get designed? How does it get engineered and how the resources get allocated to it?

[0:43:42.4] JQ: Yup. So first of all, from an kind of innovation source perspective. In general, it’s our our product management team's responsibility to drive features and to make sure that we build them in a way that meets what we’re trying to do from a customer market perspective, but we very much drive innovation through the whole business. So some of our best ideas come out of our customer success organization, or from engineers themselves, or from someone on the sales team. So just in general, we're pretty sort of wide aperture of how we think about kicking off an idea, and we just solve for, “Is this an interesting idea? If so, let's get into the product and see if you can make it real.”

On the product side, I think there are categories of features that your customers know that they need and there’s categories of customers across — Categories or features that your customers don't know that they need. In general, we try and think about those separately.

So we’re going to get a lot of — In general, it would be maybe customer-driven, and we’ll get a lot of feedback if there's something that is a customer probably knows what they want. So we have a product manager who can go out in the field and have a bunch of conversations and watch how people are using the product and kind of get to a decent idea from there. There’d be a category of things where customers never asked us before, and maybe they're not in the best position to provide a huge amount of input upfront until we fleshed it out. So that's a case where we're not going to do as much kind of intense market analysis and research, because fundamentally we're working on something that we think is a bit of a longer-range bet or a little further out. So in that area, it would be that kind of approach. We focused more on what is this and what's the aha moment and that the customer value prop and how do build this and less on the, "Let's make sure we have rigor around what we think the opportunity size is," and on and on.

We have a product manager that gets to a crystal clear idea of what we think we want to do and we try and do some small iterations and cheap prototyping or other iterations upfront so see if we can get feedback on something before you go commit to build something. We try and be iterative. As i mentioned on context earlier, we try and saturate the engineering team and the context and not just, "Let's go build this," but here's the problem we're solving and this is a really interesting and impactful problem. Anyways, you have to have to kind of improve this. We want to hear and have a conversation on.

We have a tight communications, so one of our key units, organizational units at Box, is the binding between a product manager and the engineering manager and also typically the program manager. So they spent a lot of time working on this. We get to a spec, we do some prototyping, and then we also make sure that we have, if this is sort of architecturally intensive, make sure that there is a design process for, "Let's think about how this is going to work, how is this going to scale, how are we going to make sure it's resilient and reliable and secure," and then we kind of walk that through a process and get to a point where we're actually often running and building it. It's kind of a long process to distill, but that's the essence of it.

[0:46:17.5] JM: Well, that was a helpful distillation nonetheless. Jeff Queisser, I want to thank you for coming on Software Engineering Daily. It's been really fun talking to you, and I appreciate you making the time.

[0:46:25.4] JQ: Great talking to you too. Thanks.

[END OF INTERVIEW]

[0:46:30.0] 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]