EPISODE 1065

[INTRODUCTION]

[00:00:00] AF: Dropbox is a consumer storage product with petabytes of data. Dropbox was originally started on the cloud backed by S3. Once there was a high enough volume of data, Dropbox created its own data centers, designing hardware for the express purpose of storing user files. Over the last 13 years, Dropbox's infrastructure has developed hardware, software, networking, data center infrastructure and operational procedures that make the cloud storage product best in class.

Andrew Fong has been an engineer at Dropbox for 8 years. He joins the show to talk about how the Dropbox engineering organization has changed over that period of time and what he's doing at the company today.

If you enjoy this show and find it useful, you can help us out by subscribing. You can become a paid subscriber at softwaredaily.com/subscribe, and it's $10 a month or $100 a year, and you get access to all of our old episodes without ads. That's over 1,300 episodes and there's lots of content in there on anything that you're learning right now. Also, at softwaredaily.com, you can find question and answer and all kinds of other content relating to this episode which can help you augment the knowledge that you're going to learn from today's show.

[SPONSOR MESSAGE]

[00:01:26] AF: When a large percentage of the population goes into quarantine, the dynamics of internet traffic change. Some companies need to scale down quickly in order to save money, while other companies like streamers and ecommerce retailers are scrambling to keep up with unprecedented demand. CockroachDB is a distributed SQL database that makes it simple to build resilient, scalable applications quickly. CockroachDB is Postgres compatible giving the same familiar SQL interface database developers have used for years.

But unlike older databases, scaling with CockroachDB is handled within the database itself so you don't need to manage shards from your own client application, and because the data is distributed, you won't lose data if a machine or a data center goes down. CockroachDB is resilient and adaptable to any environment. You can host it on-prem, you can run it in a hybrid cloud and you can even deploy it across multiple clouds. Some of the world's largest banks and massive online retailers and popular gaming platforms and developers from companies of all sizes trust CockroachDB with their critical data. Sign up for a free 30-day trial and get a free T-shirt at cockroachlabs.com/sedaily. That's cockroachlabs.com/sedaily.

[INTERVIEW]

[00:02:56] AF: Andrew Fong, welcome to the show.

[00:02:58] AF: Hi! Nice to meet you, Jeffrey.

[00:03:00] AF: You joined Dropbox in 2012. Are there any canonical engineering problems, the themes that you are working on in 2012 that just still keep cropping up today?

[00:03:13] AF: I would say the things that we still continuously work on, storage is a big aspect. I wouldn't say it's a problem. I think it's a core competency that we think about as a company, and so we continually invest in storage. Ever since I started, that's been a big focus area. Other areas tend to be we're always focused on reliability. I would say a problem that every company faces as you grow in size, there's always the next integration of what is reliability look like. Those are probably the two big ones that stick out to me just in the roles I've played at Dropbox. The other one is probably developer efficiency. Just how do you bring onboard developers? How do you get them to be maximally effective in the shortest period of time?

[00:03:48] AF: How has storage been reinvented overtime at Dropbox?

[00:03:52] AF: If I think back to when I first joined Dropbox, we were doing primarily storage on S3. We separate our storage in two different pieces. One is metadata, which is the attributes about files. Think about this as the access times, the file sizes, etc., the file names and then the contents of the files or blocks. On the database side, that's where metadata is stored. That's traditionally been at Dropbox, and then we had had the block storage side in S3. What we had done over the last – We've done a migration from S3 under our own in-house storage called

Magic Pocket, and there's lot of blog post about this that we've done over the years, but we had to build a storage system from scratch. We built a block storage system and we've continuously reinvented that. Last year, last 18 months, we've launched things like SMR, first to market with SMR, and we built solely a fully integrated storage stack over the last 7-1/2 years.

[00:04:46] AF: The other reoccurring theme was reliability you said?

[00:04:50] AF: Yeah, reliability is another big portion. As you start – I joined when there're probably about 130 people in the company. The company is much larger now. Engineering team is much larger. As you build more and more systems and they have these complex interdependencies, you continually have to reinvent how you're thinking reliability and the parts of the reliability that matter for the customer, the parts of reliability that matter for internal systems as well.

[00:05:13] AF: What was the earliest outage that you remember that maybe changed your perspective on that reliability?

[00:05:21] AF: We had an outage. I think there's a blog post about this as well probably 2013, 2012. Probably about 18 months after I joined. It's a fairly large outage and our perspective, we were always focused on reliability and durability but our perspective just really shifted into that it has to be part of the culture of engineering from end-to-end. It can't be something that you think about after you've launched a product, and this is something that – I've been at a fair number of large companies and it's something that I think on every system that you launch, you sort of go through this journey, and we went through it. We had a fairly large outage and we ended up taking a full step back and saying, "Okay, we have to –" Everything in even head count planning, when I think about head count planning, I think about the number of people dedicated to reliability or durability of systems. Making sure that the metrics are weaved entirely end-to-end through from we think about a system to one that's launched and then sort of the follow up after a system is out there in production as well.

[00:06:14] AF: You were a part of the famous migration off of the cloud. What was your role in that project?

[00:06:20] AF: At that point at Dropbox, was one of the SRE or the SRE managers, the site reliability manager. I was managing the teams that were building a lot of the tools to stand up our own data centers. You can think of this as machine management. The teams that were handling the black box monitoring. When we build Magic Pocket, we had a – Call it a red team and a blue team and you can say the software engineering team was the blue team and they were building a system which was going to store this data and then we had a red team which is effectively the SREs that knew how the architecture was set up, but they built these validation systems that were able to end-to-end validate for data loss without having to not introspecting the code itself, but by looking at it from a blackbox perspective. For every place we stored a block, we wanted to validate that that block was stored and the history of the block was preserved. I own the teams that were building those black box monitoring systems effectively.

[00:07:12] AF: What goes into a black box monitoring design?

[00:07:17] AF: You can think of it as almost form the user's perspective. If the block is not there, then something is wrong. If you start from that premise that if I want to go find this block in a system, I should be able to find it at any point in time. They would test for that purely using the APIs that are available within the system not looking at the internals of the system design.

If there is a get, they would issue a get and see if the block was returned at the top layer and then so forth on every layer of the stack. If there's a magic pocket frontend, they would validate the magic pocket frontend the block was available. Then the next layer down, they would look at what we call an OSD, an online storage device, whether the OSD actually had the block available. Then we'd look at the database to verify that the hash was there.

That gave us a way of validating that every single system was going to give the output that we were expecting all the way up and down the stack. We would never use an internal API call. We'd only use the external because that would be what the user's perception of what was happening was going to be. You think of it as a black box. We knew nothing about this system aside from you can issue these API calls and you validate whether or not it's working or not.

[00:08:20] AF: All those different layers that happen through a read to the storage system, can you go through those a little more slowly? Just from the outside looking in, a user making a read to Dropbox, they're looking for – I'm not exactly sure what would be on a single block, but they're looking for a file for example. They're trying to read a file. What are those different layers of the stack?

[00:08:49] AF: This is going to be testing my memory now on this. I'll try to do this from a pretty high-level architecture, because there're a lot of different pieces to this. If I back up all the way to how Dropbox works, when we have a file, that file is split into 4 megabyte blocks. Those blocks are stored in our block storage system. If you have a QuickTime movie or a movie file, that file is divided by four effectively. So we get the number of blocks and then those blocks are put into this system.

There's a frontend that accept those blocks. That goes and it writes that out to storage device. That storage device has a database associated with it which maintains where these blocks are stored within the various storage devices we have, and we have tens of thousands of these storage devices, and then each storage device manages some hard drives, and hard drives ultimately were the block ends up. You want to test to the hard drive layer, at the storage layer and then at the frontend layer.

I am definitely not doing this justice, and I would say that we have – There's an exceedingly long article we've written on how Magic Pocket works, and it's complicated enough that I would say it's worth looking at the architecture diagram that's online to sort of dive through these layers, but in essence there's probably three main components that I think about this in terms of there's a frontend, there's a middleware layer that arbitrates where the blocks are going to be stored, and then there's a ultimately a hard drive that it ends up data at rest at. There is some metadata that's associated with where all these things are within the system at any given point in time. You want to validate that each of these systems view of the world is correct, and that's sort of what the black box monitoring does.

[00:10:22] AF: Tell me how Dropbox uses the cloud today. After that famous migration off of S3, if people don't know, that Dropbox was started built entirely off of S3 as the storage layer and overtime migrated to its own data center infrastructure I believe in co-los, not your own data centers. How does Dropbox use the cloud today?

[00:10:47] AF: We actually still have a fairly large Amazon footprint. We do our international block storage on S3. If you're doing GDPR for example, that's actually done inside of Frankford. We have storage locations there. We have storage locations in Asia that we're leveraging S3 for. One of the design principles when we thought about building a storage system is that we want to have the flexibility of storing blocks in whatever storage system makes sense for us. It should be a business decision, not a technical decision around where blocks are stored.

When we think about this, a lot of our reasons had to do with economies of scale and being good at doing storage as it's a core business competency, but we also want to preserve the flexibility of saying I want to route blocks to a different location if I need to. There's a layer inside of Magic Pocket which effectively says, "Okay. Where should this user's data go? Does it need to be stored in Europe? Okay. We're going to put it in a block storage system there," and S3 in a super simplistic view of this world, right? Like this is not technically correct in anyway, but you can think of it as the ultimate resting place, whether it's a hard drive or S3, it doesn't really matter to the storage system. It's just dealing with a block and has a pointer to the location at which that block is stored. You can drop it in effectively any storage system that we know how to write to.

[00:11:59] AF: Got it. We kind of talked through the black box perspective of the read, and I guess a write is somewhat similar. It sounds like you have may be become a little less intimately familiar with that layer of the infrastructure. Over time, have you moved into more of a management role or are there different parts of the infrastructure that you're concerned with? When did you start to raise up in the level of abstraction?

[00:12:29] AF: My journey at Dropbox, I joined as a site reliability engineer in 2012. I started managing a small team of engineers probably 12 months in, maybe 9 to 12 months in. From there, started managing and built out an S3 reorganization. Took on some additional team storage being one of them at the time. The last time I was sort of that layer of the stack looking at the intimate technical details probably 5, 6 years ago.

From there, managed a bunch of the infrastructure teams. Went and started doing a super small team of some of our more principal engineers and worked on rewriting RPC systems. Worked on with them deploying a new lock service, and then went to developer productivity in the platform layer for about nine months, rebuilt some systems there. Then came back and started managing all of infrastructure, which is about infrastructure at Dropbox is everything from supply chain vendor management, data center network, storage systems, all the release processes for backends, all the reliability engineering teams. I started doing that about probably 18 months ago and started moving to that role. I've been all over the place in the stack at Dropbox. Storage is one of my responsibilities now, but it's one of N. Infrastructure is roughly the about 190, 200 people right now.

[00:13:45] AF: What kinds of technical issues do you find yourself going deeper on?

[00:13:52] AF: I probably know enough to be dangerous in any of these areas, which is a blessing and a curse as a manager in a lot of ways. The last set of things or rather the current set of things that I've been going pretty deep technically on have been more around reliability in the last 12 months. I spent a bunch of time there mostly because as we've grown as an organization, the system complexity has increased. I haven't been diving necessarily into system A, B or C, but more – I was like, "How do we think about reliability is a problem statement for Dropbox, and then how do we build in the right guardrails, and how do we build in the right processes so that it is sustainable as supposed to I think what a lot of organizations see?" There're a lot of heroics, where you have your super senior engineer that can dive in and fix the problem, but that's not scalable as you grow, right?

I've spent a lot of time on sort of that problem space, and then the rest of it has been spent a lot more around sort of how do we make sure we're getting a set of business outcomes on the efficiency side that we need to dive into? So just understandings sort of what levers we have around efficiencies? What levers we have around reliability? That's been primarily the space I've been spending the last probably 12 months on.

[SPONSOR MESSAGE]

[00:15:06] AF: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW CONTINUED]

[00:16:55] AF: What does increased reliability mean for Dropbox at this point? I feel like Dropbox has been working on reliability for so long that so many of the edge cases are hammered out. What kinds of edge cases are you working on now? Is it a different type of reliability that you're discussing?

[00:17:14] AF: I will use my – I know there's people that talk about reliability have different opinions of sort of this. The internal rubric that I use tends to be durability, availability, performance just because if I think of it from a simplistic view that if it's not there, meaning durability, then someone's going to care a lot about it, right? Our journey started there.

Availability would be, "Okay. Is the system up and can I get to the data that I'm looking for?" You've stored it, but like can I get to it? Then like can I get to it from a performant way? That could be anything from website performance, to client performance, to sort of the mobile performance, right? We have – And this is a project I was not involved in from a technical perspective, but reliability sort of is adjacent to it.

We rolled out a new sync engine. We had a blog post about this as well in the last couple days. That new sync engine truly changes how we have to think about reliability, because we knew all the edge cases of the old one and we knew how to think about and reason about the old sync engine. But when we rolled out this new sync engine, that changes the systems that interact. There are some performance characteristics that are different. You're going to learn more about it and it's the right decision technically, because it allows us to actually have more developers work on the core product areas by having a simplified interface to sync.

Now, the downside of that when you think about this from a reliability perspective is that, yes, there is 12 years of prior art and knowledge that are wrapped up in the existing sync engine and as you replace it, you still have to rebuild some of this. Now the benefit is the new sync engine has this much simpler set of characteristics. So it's easy to reason about, but we still have to go through some portion of this journey around reliability.

They built new frontends for the sync engine to interface with, "Okay. What other backend performance characteristics that they're going to see? How do you scale them? Do they scale differently with different properties than the previous sync engines backend scaled with?"

What I think about reliability, it's a continuous journey. I like to tell my teams that one bad month of reliability is nine months of work for you down the road. It's one of these things that if you're not always on top of and you ever have a regression in, you're going to pay the price for this for some period of time in the future.

[00:19:12] AF: I read that sync blog post, and this was near and dear to my heart because I have Dropbox on desktop, and as somebody who does a lot of content about software engineering, I can think about this problem, and I know sync is not easy to design because it's – You got this client device that's got lots of power. You've got infinite server infrastructure, but you want to be economical about it, and there is a real open question as to what you should be doing on the client. What you should be doing on the server. How often you should be doing each of those things?

I mean, for people who don't know, sync is like if I have this dedicated Dropbox folder on my computer, I drag a file to it. That file gets synced with the cloud. It sounds kind of simple, but if you think about in actuality, well, the client I guess has to be monitoring the folder all the time or the server has to be polling the client all the time. I mean, just tell me little bit about some of the design problems or design questions that you have to approach when designing sync infrastructure.

[00:20:18] AF: This is a little bit out of my area of expertise. Client-side infrastructure on sync is definitely one of the most complicated service areas we have at Dropbox. Now, a lot of the problems that you're going to run into with sync, and I will raise the altitude of this a little bit from like just to let's think about the number of Dopbox users in the world is hundreds of millions. That means that you have some order of magnitude of number of devices that you have to support for this.

Now, let's play that forward to how many people that are listening upgrade their Windows operating system on a yearly basis or have the same number of point releases out there? Now, I think I'm a backend infrastructure person, so I had the luxury of saying, "Thou shall have one operating system in production. Thou shall use one monitoring system," right? It's very easy to make these grand decrees as a backend person.

Now, when you think about designing sync, you have to do this for a set of clients in the world. When I say clients, not end-users, but just laptops, desktops, mobile devices that are on N-number of versions of Windows, N-number of versions of Mac OS ,N-number of versions of Android, or Linux, or iOS. Now, if you take that problem statement alone, you can say it's very easy to design a protocol that would maybe work for one of these, right? Many software engineers have designed backend protocols that they can just say, "Okay, this is the only protocol we use in the server-side." On the desktop side, it becomes very hard, because these operating systems each have their own quirk.

Every time you want to design a sync engine, you have to also think about and reason about how the operating system works as well because it plays a huge amount of impact into how you're notified that a file changed. When a file changed? Weather it even thinks about files if you think about iOS and Android. Those are just – Even before you enter the sync world of like let me get a file from point A to point B and design a protocol, you have to recognize that that problem statement means that you have to design a system that can reason about upwards of – Right now, I'm just trying to think of the top of my head. My vision of operating systems is definitely more than 10 and probably approaching probably 25, 30 that we have to think about from a testing perspective.

Forget about the protocol. Just think about what are you going to do? How do you test this? Because you deploy Dropbox client to all these hundreds and millions devices and it has to work on Windows with a language pack that is for German, a language pack that's for Japanese, a language pack that's for Chinese, right? Each of those has different characteristics. The operating systems have their own quirks which allow bugs to come into it, which means that sometimes like when you click save maybe it doesn't work exactly the same way as it does. You would think it does, but a lot of times it doesn't work exactly the same way.

The protocol has to take into account all of these things and all these edge cases. Even before we think about designing a protocol, we have to think about what the problem space we're worrying about from a testing perspective. That's sort of like my thoughts around sync at a high-level. The problem statement is very hard. It's not just about how do I move file A from point A to point B? A lot of people say, "Why is this not just rsync?" Well, the reason is because like when you're during rsync, you have pretty much a POSIX compatible file system on both sides, like it's number of variables or it's just significantly lower than the number of variables you have to support when you're trying to do sync in the wild across like multiple operating systems.

[00:23:39] AF: But do you know much about the server infrastructure? Can you use the same backend system to talk to all these different clients?

[00:23:47] AF: We have designed a protocol. You can think of it as a library that's on the client-side now, and that client-side library, it has a standard interface to the backend. I'm not intimately familiar with the internals of the protocol. I think the blog post probably goes in a much more detail around that. That's actually on the client-side of our engineering teams. They handle building the client-side, because you have to know so much about the operating system side just to build this this type of infrastructure. The backend side is actually significantly simpler in that regard, because we have standard server frameworks that we're using everywhere else we can use there and we just have a protocol layer that's on top of that.

The internals of the protocol documented in the blog post, I would say that from my perspective, I think when you think about sync, it's really this challenge of building this distributed system in the wild where I think a lot of backend engineers, myself included when I first joined Dropbox, take for granted. So the ecosystem you get when you think about building inside of backend, you just can say no to so many things and reduce the number of variables that you have, but you can't do that when you actually do this type of software on a desktop. Even iOS and Android is significantly simpler because the upgrade paths are more forced from the vendor, but like that's not the case in Mac OS and it's not the case on Windows.

[00:24:53] AF: Since you've spent significant time in the storage area, I'd like to talk a little bit more about that. Can you tell me about how a file gets broken up and distributed on Dropbox today?

[00:25:09] AF: The client will chunk it for you, and then we take that, we store that file. We'll take it on a frontend and we will store it into a Magic Pocket zone. That zone will then replicate it asynchronously to a second site. That's the super high level of this. There's nothing super fancy about how that's all happening. The complexity is really the scale in keeping it simple in such a way that like you can do this at scale. We then have some heuristics and we have a secondary system called MPY where for some portion of this data we can actually move it to a colder set of storage. We can basically tradeoff a network read for disk read, where we're going to read this over the wire as supposed to read it from a disk. So we can then get a little bit more efficiency out of the system, but effectively the principles are all the same. There's nothing there that's different. It's just a matter of like trading off for network speed.

[00:25:59] AF: Why do you only replicate it once? I would think you'd need to replicate it twice.

[00:26:03] AF: This is replicated multiple times. I'm boiling this down to like at a very, very basic, basic level. There is a block and we store in multiple places. How many times do we store that in multiple places? That becomes a tradeoff around how do you want to reconstruct it? Do you want to reconstruct it over the network or do you want to reconstruct it locally within the data center? There's a lot of different ways we can tune this system from that perspective. I am just simplifying this down to like the very basic level. Block comes in, we store it on an online storage device, we replicate it elsewhere. We do keep multiple copies of it.

[00:26:35] AF: What's the size of a block?

[00:26:39] AF: The system today is – Well, the client-side is chunked into 4 megabytes, which we found is a pretty reasonable size for us to use.

[00:26:46] AF: Do you remember how you arrived at that beautiful number?

[00:26:50] AF: I would actually have to ask [inaudible 00:26:52] that question, because that was actually picked as the – We have never found the need to tune that. We've actually looked at it in the past, but they picked 4 megabytes as the very first iteration of the client, and it has worked for us ever since.

[00:27:07] AF: All right.

[00:27:07] AF: We've definitely looked at it when we started doing Magic Pocket and we found that it roughly worked, because anything less than 4 megabytes is the size of, and so most files in the wild are not actually larger than 4 megabytes. When you think about a Word document, they're not going to be 4 megabytes. There's like – I don't member the date off the top of top my head, but there's a distribution of like file sizes.

Four megabyte generally works for the network speeds we see. Generally works with how we store the blocks on disk in Magic Pocket. We haven't found a need really to tune that and it's always worthwhile revisiting, but it hasn't been something that's like it's causes any performance reliability issues for us.

[00:27:43] AF: Okay. Can you tell me about the observability infrastructure at the storage tier? How are you logging across the storage tier and how are you monitoring it?

[00:27:54] AF: Okay. I will take a step back a little bit. The way we monitor a storage tier is the same way we monitor everything else. We think about – One of the things we really believe on the backend side is that engineering is about constraints, right? With this set of constraints, you have to be able to produce a business outcome or a technology outcome for the business. One of the things we know is that we are going to be about – People as a bounding function, right? You don't have infinite people. You have to think about how do you want to spend your resources on building out these types of systems.

One of the things we arrived at is that we really have a principle of like we would like to have one system that serves a single-purpose, right? We don't have N-number of monitoring systems. So we actually build a single monitoring system that all backend services use, whether it's storage systems, whether it's our middleware layer, whether it's our identity system. They all use the same monitoring system, and that's actually predicated because we have a single RPC system as well. Storage system uses the same RPC system that everyone else uses and that allows us to get some efficiencies that are probably not abundantly seeable, but if I have a single monitoring system and a single RPC system, every dashboard I create can be the same for every team because every team cares about RPC latency, RPC gets, RPC puts, right? All this becomes a single button click anytime I turn up a new system.

We actually don't build a specific monitoring system for the storage systems. They actually leverage the exact same monitoring system that everyone else uses. All the data is centralized on a monitoring team. Now, things about storage systems that are interesting to monitor, you have to monitor lots of hard drives. You have to monitor all the various services that are running there. You want to know things like disk freeze, disk writes. You want to know the network latency. You want to know on-block reads, on-block writes. It's all the very standard set of things that you'd want to know in any system.

Now, what makes a lot of the power that we get is that you can get this drive, because we have centralized monitoring system, where they don't have to go and know all the intimate things about the RPC system, because the RPC team actually wrote the monitoring code that allows you to get that data. They have to like – They have to know how storage works, right? They don't have to know how RPC's work. They don't have to know actually how hard drives work, because the hardware engineering team will actually go and instrument hard drives and it all ends up in the same place. Every team gets hard drive monitoring the same way a storage team will get hard drive monitoring. This all ends up being. So you get these economies of scale where everyone is sharing the same data. Everyone has the same interface. You don't actually have to – And this goes to developer efficiency. You don't have to have a cognitive overhead of learning a new monitoring system when you change teams. You don't have to learn a new build system either.

What we think about this is like can we have one system that can be leveraged across multiple teams sop those eight people that work on monitoring can actually be leveraged across the storage team, across the traffic team, across the network team without us having to go repeat this over and over and over again?

[00:30:46] AF: Do you remember any particular instances in the storage area where it was really hard to debug something?

[00:30:56] AF: Let's see, debugging on storage. We were going through one the other day actually. We were seeing – I can tell you the problem statement. This technical solution is definitely above my technical intuition because it's going to be on the hardware side. We were seeing effectively a single hard drive lock up the entirety of a chassis. If a hard drive went offline, the chassis would lock up. Now the interesting thing is these chassis have 100 hard drives in them.

You don't lose 1/100th of the capacity of the server. You lose 100% of the server because there is a bug in the firmware of the controller card that when the a drive would fail in a certain way, the entire chassis would go offline. Now, that becomes incredibly hard to debug as you can imagine because there is no system to go debug, the thing his crashed.

Linux is hard locked-up and it's not even necessarily Linux that's locked-up. The hardware itself is locked-up. Now you are in a world where you have to start to look at, "Okay, you can do mitigations across this." You can say, "Are there any leading indicators of why these hard drives are locking up?" Now that usually requires us to go back to the vendor because there is so much data coming out of these hard drives. If you just look at smart output or you look at any of the logging data that the firmware is going to give you, it's Greek in a lot of ways, right? This is not something that there's a manual online that you can go read about. This is going to be you have to go find the guy that wrote the firmware for this device and like ask for an interpretation of what's happening.

We'll typically work with the vendor to go help us through that process. Then we find out, "Okay. That's how you sort of – You can start to triage this, right?" Then we can start to do a prediction of, "Okay. What's the probability of this hard drive going – Of a hard drive going offline based on these leading indicators? Can we do anything like proactively fail the drive in a different way so that the whole chassis doesn't go down?" Because it's only when it's failing in a very specific way does the whole thing lock up. Can we just take the drive offline? Can we just send a data center tech to pull the drive an hour before we know it's going to crash?

A lot of that goes into like we just have to look at the data and we have to collect all the smart stats. We have to then process it, right? It's like a lot of pattern matching. It's a lot of like, "Okay, we solve this type of this log line event come out across the last hundred times this failed, and so some set of these are going to have to fail over time before we collect enough data to understand it."

But then on the flip side, once you have some of that, you can be proactive about this and say, "Okay. I'm going to fail the fellow drives proactively or I'm going to pull the drive proactively," which then cuts your failure rate down from 100% of the chassis to 1/100th of the chassis, again, as the systems like designed to work through. That's an example of like one we've had to debug. Really, it comes down to just instrumentation of the hard drives and getting as much data as possible off of it looking for patterns in the data. A lot of it is manual. I can tell you, there is no system that – We don't have a system today, and I would assume most people don't have a system to comb through this and like find the – A lot of this is hex codes and things like that that are coming out of these drives, and finding those, like, "Okay. This happened. This happened. This happened, and this is probably the reason why it failed," and that we have to partner with the hardware vendors on just because they're the ones with the expertise and they of the ones that have may have seen this in other customers as well where we may not be the first, but they may know what sort of the leading indicators are, because it may take us six months to figure those out. There's a big partnership between us and the vendors on that side.

[SPONSOR MESSAGE]

[00:34:25] AF: As a company grows, the software infrastructure becomes a large complex distributed system. Without standardized applications or security policies, it can become difficult to oversee all the vulnerabilities that might exist across all of your physical machines, virtual machines, containers and cloud services. ExtraHop is a cloud-native security company that detects threats across your hybrid infrastructure. ExtraHop has vulnerability detection running up and down your networking stock from L2 to L7 and it helps you spot, investigate and respond to anomalous behavior using more than 100 machine learning models.

At extrahop.com/cloud, you can learn about how ExtraHop delivers cloud-native network detection and response. ExtraHop will help you find misconfigurations and blind spots in your infrastructure and stay in compliance. Understand your identity and access management payloads to look for credential harvesting and brute force attacks and automate the security settings of your cloud provider integrations. Visit extrahop.com/cloud to find out how ExtraHop can help you secure your enterprise.

Thank you to ExtraHop for being a sponsor of Software Engineering Daily, if you want to check out ExtraHop and support the show, go to extrahop.com/cloud.

[INTERVIEW CONTINUED]

[00:36:00] AF: As Dropbox moved off the cloud, you moved into co-los, and I realize you still have a cloud presence at this point, but one interesting question about the co-los is you have to determine where to put them geographically. You have to determine your requirements for geographic placement and then you have edge infrastructure as well, which I guess you also have CDNs that you’re using aside from those co-los I assume in the edge infrastructure?

[00:36:32] AF: Yes, we have three different types of way of serving traffic at a high-level. We have data centers, then we have pops, and that we have some CDNs that we leverage as well. There’re three different layers of this. We have publications globally. We have data centers domestically, and then we leverage CDns for some of the content that makes more sense from a CDN perspective. Think about sort of JavaScript and things like that. I think we do client distribution as well through some of the CDNs where the global reach is actually maybe significant – Not significantly more than our edge network, but like reaches like last mile a little bit better than our edge network would reach.

[00:37:07] AF: Tell me little bit about the geographic placement there. How do you determine where to put these co-los?

[00:37:15] AF: From a data center perspective, when we first started doing Magic Pocket, we looked at having data centers roughly – We want to have three copies of the data. We needed to have three different geographic locations. We wanted to look at everything from – Latency as a big part of it. Network connectivity that's available is a big part of it. Power cost is a big part of it. When you're building this out, and here's a sort of interesting tidbit of things that we've thought about were, "Okay, if I'm going to pick a location and in the process of building this out we're going to be doing it in the winter. Do we want to do it in Chicago?" The answer is probably no. From like a business perspective, you're going to have to deal with like snow delays. You're going to deal with all of that. That was one of the things we thought about, was like, "Okay. What will it take to actually do this in the time? Sort of where we contextually from like in the year as well?" It's one of the inputs. It's not the only input, right? Those are some of them.

We actually have a checklist. It's probably four or five pages of sort of like what are the attributes we're looking for in a data center? Because it's both geography, like that's one cut. Then within the geography, do you have the constraints on the building support, the types of things you're looking for? For example, for us, a storage rack, currently it probably weighs much more than this. I remember we did an all-hands, maybe five or six years ago. I think the number we came up was like 2200 pounds, 2500 pounds. It's like a little bit more than a PRIOS. I think it was the analogy we used is like the weight of a single-storage rack. Well, some raised floors don't support that. That makes a difference, right?

Then you have to pick a provided that's actually going to be able to provide you a raised floor that actually has the right characteristics or doesn't have a raised floored in its overhead? These are the sort of things that go into it. It's like not just geography. We can do some cuts around geography and say, "Okay. We don't want to be in the path of a natural disaster. Let's pick a geo that's outside of the Bay Area ultimately for some blocks. Let's pick a geography that's central to the country so that gets you sort of in the Midwest. Let's pick something on the East Coast." You can cut it that way. That's like a quick first cut. Then there was this, "Okay, what are the actual characteristics of the facilities that we need to have?" Because that drove so much more of sort of the decision-making long-term, because from a network perspective, you can find some usually pretty good cities and then you sort of what are you picking from a facility inside of that city? It's sort of a roundabout answer, but it's a multivariable problem for us.

[00:39:39] AF: Got it. Can you tell me about networking? I'd like to learn about networking between data centers as well as networking within a single data center.

[00:39:52] AF: Okay. Between data centers. At a high-level, we have an edge network. This is roughly our POPs, and you can think of this as points of presence around the world. This is last mile, or not really last mile, but like last mile for us to the user, so like we're going to do handoffs to other providers and take on transit. That allows us to reduce latency because we can do effectively long-haul connections that are persistent for us so we don't to pay the startup cost over and over again. That's from client's perspective, we have that. Works just like a typical CDN would work in a lot of ways.

Then what we have on top of that is our data centers internally to the US. We have some data centers there. Now, we've had multiple iterations of internal data center fabric. The original fabric was sort of a two-post, almost typical sort of startup-y sort of feel where everything back to the core. We have since evolved into a clos fabric. Networking team has deployed that. I think in all of our new facilities, that's been out for probably the last 18, 24 months. Gives us – Maybe even longer, maybe 36 months. This gives us a much higher throughput between servers. You're effectively able to utilize more of the cross-sectional bandwidth between server-to-server, try to state that altitude for this. I don't know how much detail you want to have on the networking side. I don't know what the listener base looks like on that side.

[00:41:07] AF: Fair enough. What about DNS? How do you handle DNS?

[00:41:10] AF: DNS for us is handled in multiple ways. We have internal DNS. We have external DNS. External DNS, we – I know this is sort of an open-ended question. I'll say we want to select DNS provider that allows us pretty good redundancy and ability to failover our sites in a quick manner, right? There has to be some characteristics. We care about that.

Internal DNS is actually – We actually use service discovery for almost everything internally. It's less of a concern. So we're not actually – We have a fairly robust – I mean, you have to have run an internal DNS infrastructure. It has to bootstrap everything. But ultimately when our services internally are talking to each other, we're switching over to a service discovery-based architecture there.

Anymore – I'm just trying to pick at what you're looking for from an external DS. It's a fairly – DS for us is not anything I'd say as a special secret sauce. It's going to be we want to pick something that's reliable, something that gives us the ability to failover back and forth between DNS providers as well. We look at this thing like we won't always have two. We want a backup in case the primary one is down. All of those things play into it. I would say it's not necessarily something that I would say is like a – It's a fairly standard setup in our world.

[00:42:21] AF: Totally. Yeah, it makes sense. If I am an engineer deploying a service within Dropbox, what's my workflow? Is there an internal platform? Is there like a platform engineering setup? Am I just using AWS? What's going on there?

[00:42:37] AF: This I can probably – I'll go a little bit more technical detail on this. A long, long time ago, we made a decision that we did not want an engineer at Dropbox to know whether they are deploying on top of our cloud or any other public cloud. The reason for that was that from my perspective, engineers are here at the company to produce product. They're here to produce business value for an end-user. So we want to abstract this like workflow notion of like I need to understand whether I am on a Dropbox computer, versus an AWS computer, versus a GCP computer. It doesn't – At the end of the day, they're deploying some code and they want to see a user use it. That's their goal.

We made this decision not to have any type of infrastructure or any type of abstraction that required you to reason about this in any sort of meaningful way. If I start at the base level, machine management. Machine management has a very simple workflow. You can be install – You can go through some workflow, this lifecycle of, "Am I installed in a data center? Am I in repair? Have I been repaired? Am I like being reimaged?" There's like a stateful system that you can try and traverse as a system.

Then on top of that we said, "Okay. We have to build some cluster management. Dropbox has been around long enough that we'd started on our cluster management software well before Kubernetes was like mature. I think Mesos had some traction, but not a lot. We actually just said, "Look, our bounding constraint actually was good to be memory and network. It's not about CPUs." Bin packing didn't get us a lot of value. So we built some pretty simple abstractions, and you'll find a theme in Dropbox engineering around simplicity. We still basically utilize 100% of the memory, 100% of the network before you utilize 100% of the CPU. That's not something we can bin pack anyway. Okay, what are we going to do? We're going to build a system that lets us manage machines very easily for our end-user and we will build a way that when you deploying in this, you can just move a service from machine A to machine B and you'll never have to know about it assuming you use our core libraries.

That was sort of like the intent. Now, the way this all get operationalized, and I mentioned before that we think about only having one-of versus like N-of types of systems. We try to limit the number of programming languages we have. In production, really, you have 2-1/2, 3, maybe 4 if you count Java. We have Go, Python, a little bit of Rust and a little bit of Java. We make very big investments in our core libraries inside of Python and inside of Go and then we provide deployment systems and platforms that when you build something, you get an artifact. That artifact is the same – Is generated by a single build system for all of Dropbox. That build system generates the artifact. The artifact goes into a deployment pipeline. The deployment pipeline goes out to one of these machines, which is this machine lifecycle, which is managed by this cluster management software. From an end-user perspective, you have this one dashboard which says, "Okay, I input my artifact hash, and then I push deploy," and you can wire it up to have various types of canaries, various types of – We call them states. Maybe you want a staging environment, then you want a canary environment, then you want the partial push to production, the full push production. But it's all the same. I can go from team A, to team B, to team C, and my workflow is exactly the same for deployment.

[00:45:44] AF: It's a great summary. Can you tell me more about internal tools at Dropbox? With that emphasis on simplicity, what are the situations where you – Whatever you could've taken off-the-shelf wasn't sufficient and you needed to build something internally.

[00:46:02] AF: We actually tried it take a lot of things off-the-shelf. A good example of this, probably the biggest case that comes to my mind, the build system we use is called Bazel. Baszel is an open source system that Google uses, effectively a fork of their internal system. We actually is Bazel for our build system for all server-side artifacts. Now, Bazel out of the box works. It lacks a lot of things you would need at scale.

An example of this is we have to wire up caching for it and intermediate build steps for it. What this means is that if I'm going to push build, Bazel can either build it itself or it could look into remote cache to see if this artifact or if this graph has been built before effectively. What we've done is we actually had to build all the caching infrastructure for that. We had to build all the package management backend infrastructure for that.

When I hit build at Dropbox, the build file will specify what re all the intermediate targets, etc., etc., and those intermediate targets will get cashed out in a production system for caching build artifacts. Now, the next person, let's see, you go build something. What will happen is, is that we'll see if any of those intermediate things have changed. If nothing has changed, we'll just pull the cache version. We actually don't have to do a compilation of that again. That allows a developer to have a much faster build time on server-side artifacts.

Now, the real sort of power behind this is that when that's wired up to the build, to the CI systems, every time someone submits a code, a code change, the CI system does the full integration. It does the build, right? It cashes the artifacts. Now you have a cache that's continuously being populated by the CI system so that the next person that wants to go do a build doesn't necessarily have to do a full build anymore. They can pull an incremental out of the cache system that CI populated. So now let's say you're building world and you only change one file and you only really have to do the build for the thing that you impacted and everything else is just pulled off the network. That gives a much, much faster development cycle time when you may have some large artifacts that have to be built. That's an example of where we took something off-the-shelf. We have to add – It's very clear in the documentation. Yeah, it supports caching. None of this exists. So build some caching infrastructure behind it. For me that's the like one prime example that comes to my mind that's like super simple to like explain at a pretty easy level.

[00:48:21] AF: Okay. At this point, you're VP of infrastructure, which means that you're at the intersection of management and engineering. What's the hardest lesson that you've had to learn about management?

[00:48:34] AF: I would say – This is probably the same answer a lot of managers would give. Giving up control is a hard lesson to learn and making sure that you can effectively give the right level of agency and autonomy to the organization at large, especially given that I started in infrastructure when there was I think 12, 15 people, and now it's like 200+ people, around 200 people. I have home a lot of historical state, right? So it's like very hard for me sometimes to say, "Okay. Yeah. I know how that used to work, or I know – I have some implicit assumptions about what's happening here," that overtime either they're wrong or better people have now taken the reins and like understand it more completely, or the system just doesn't exist anymore and my assumption was incorrect to start with. That for me was always just giving up that control and like letting the team go with it, but at the same time putting in the right guard rails to make sure like the right outcomes are occurring.

[00:49:30] AF: How his the company changed operationally since COVID?

[00:49:36] AF: The biggest operational changes are very, very small from the engineering perspective. We didn't do anything crazy on limiting releases or anything sort of like that for that front. What we tried to do is say like we want to have as much business as usual on the engineering side.

Now, operationally, a big part of my role also is supply chain. That world has changed drastically. The director of supply chain pinged me I think January 25th and he goes, "Just to let you know, 2020 is not to be a normal year. This is going to be crazy," and this is January 25th, mind you. I was like, "Okay. Ali, what's going on?" and he's like, "Well, I'm pretty sure Wuhan is going to shut down, and like just to let you know, there is second tier suppliers that we use that come out of some of the facilities in Asia and like we're already starting to see delays. We're going to have to go into full sort of like wartime mode to like keep track of all of this."

A big part of like what they do is they interface with a lot of our ODMs, OEMs they make sure it's sort of like all of our rack deliveries, all the server builds, all the next generation equipment is like it's all in pipeline of materials, right? They had to go effectively into a wartime mode, in war room mode of, "Okay. We're normally sourcing first tier suppliers. We're dealing with down to sort of commodities." I have a commodities management team that's managing sort of DRAM, SSDs, hard drives, chips, CPUs, right? They're handling that level of commodities.

Now, this new world that we're living in post-COVID, that's not where they have to stop. They actually have to think about, "Okay, power supply, or sheet-metal," or like rails that are going to go into a rack that they normally take for granted that's going to come from the supplier. They actually have to look and make sure that all of that supply pipeline is actually there.

The last two, three months for me is actually been spending a lot of time in that world, and I am like super thankful to our partners that are out in the world that actually help make all this happen, because it is definitely supply chain – Global supply chain is like not what it was 6, 12 months ago, and that's actually been a big focus of mine during COVID. It's just making sure we have contingency plans in the software side if there are capacity shortages on hardware availability, and these are things that even the cloud providers will see. This is like macroeconomics. This is not going to be sort of just because you're a certain size. Actually, you will be impacted by this. Literally, everybody is impacted by this.

We've been spending a lot of time know making sure that's available, and a big part of it is like the upfront work you have to do with the relationships with the suppliers and spending the time with them to make sure they understand the business that you have. That's the post-COVID world, and infrastructure has been a big portion on supply chain side.

[00:52:13] AF: That's amazing. Do you feel like it's feasible for everybody to overcome the supply chain constraints, or does it feel like the noose is getting tighter?

[00:52:22] AF:I feel very good about like where we are today. There's nothing in the Dropbox world ecosystem that I worry about on the supply chain side right now for us. We have line of sight for everything probably the next 12 to 18 months. It feels pretty good. I say this today talking to you. Given COVID and the realities of the world and how it's changing, will that answer change tomorrow? Maybe. What we tend to do is look at this week over week and say, "Okay. What's the week over week supply look like? What's the commodity markets look like?" The teams are managing this basically 24 by 7 when you think about we're on the West Coast, but you're having to manage the supply base in Asia. They're coming online when everyone here is going to bed. It's a continuous dialogue and we just have to stay on top of it. There's literally no other way to do this that I know of on the supply chain side.

[00:53:03] AF: Andrew, thanks a lot for coming on the show. It's been really fun talking to you.

[00:53:06] AF: Thank you. It was great to be here.

[END OF INTERVIEW]

[00:53:17] AF: JFrog Container Registry is a comprehensive registry that supports Docker containers and Helm chart repositories for your Kubernetes deployments. It supports not only local storage for your artifacts, but also proxying remote registries and repositories and virtual repositories to simplify configuration.

Use any number of Docker registries over the same backend providing quality gates and promotion between those different environments. Use JFrog Container Registry to search the custom metadata of the repositories. You can find out more about JFrog Container Registry by visiting softwareengineeringdaily.com/jfrog. That's softwareengineering.com/jfrog.

[END]