EPISODE 558

[INTRODUCTION]

[0:00:00.3] JM: Before you begin today, we have an announcement. We've launched Software Daily, which is a place to post your software projects and discuss them with other people. On Software Daily, you could find collaborators and feedback for your software project. If you have an open source application or a side project you've been tinkering with or an academic computer science paper that you want to get feedback on, then come the softwaredaily.com and post your project.

Software Daily is about cool projects and new ideas and creativity, and if your project is especially interesting, we'll send you Software Engineering Daily hoodie, or a t-shirt or even have you on the podcast to talk about what you're building. I've been posting some of my own side projects on Software Daily and I'd love to see what you are working on. So check it out at softwaredaily.com.

With that, we'll get to today's episode, which is with Tammy Butow. She's worked at DigitalOcean and Dropbox where she built out infrastructure and managed engineering teams and at both of these companies the customer base is at a massive scale. At Dropbox, Tammy worked on the database that holds metadata used by Dropbox users to access their files, and to call this metadata system simply a database is actually an understatement. It's a multitiered system of caches and databases and this metadata is extremely sensitive. It's metadata that tells you where the objects across Dropbox are located. So it has to be highly available.

To encourage that reliability, that availability, Tammy helped institute chaos engineering, inducing random failures across the Dropbox infrastructure and making sure that the Dropbox systems could automatically respond to those failures. If you're unfamiliar with the topic of chaos engineering, we've covered it in two previous episodes of Software Engineering Daily. You can find those episodes in the show notes.

Tammy now works at gremlin, a company that does chaos engineering as a service, and in this show we talked about her experiences at Dropbox and how to institute chaos engineering

across databases. We also explored how her work at Gremlin, which is a smaller startup, compares to Dropbox and DigitalOcean, which are large companies. It was great to talk to Tammy and I hope to do it again in the future.

[SPONSOR MESSAGE]

[0:02:41.1] JM: If you are on call and you get paged at 2 a.m., are you sure you have all the data you need at your fingertips? Are you worried that you're going to be surprised by things that you missed, errors or even security vulnerabilities because you don't have the right visibility into your application? You shouldn't be worried. You have worked hard to build an amazing modern application for your customers. You've been worrying over the details and dotting every I and crossing every T. You deserve an analytics tool that was built to those same standards, an analytics tool that will be there for you when you needed the most.

Sumo Logic is a cloud native machine data analytics service that helps you run and secure your modern application. If you are feeling the pain of managing your own log, event and performance metrics data, check out sumologic.com/sedaily.

Even if you have your tools already, it's worth checking out Sumo Logic and seeing if you can leverage your data even more effectively with real-time dashboards and monitoring and improved observability. To improve the uptime of your application and keep your day-to-day run time more secure, check out sumologic.com/sedaily for a free 30-day trial of Sumo Logic.

Find out how Sumo Logic can improve your productivity and your application observability whenever you run your applications. That's sumologic.com/sedaily.

Thank you to Sumo Logic for being a sponsor of software Engineering Daily.

[INTERVIEW]

[0:04:26.0] JM: Timmy Butow, you are an engineer at Gremlin. Welcome to Software Engineering Daily.

[0:04:28.8] TB: Thanks so much. Thanks for having me.

[0:04:30.6] JM: Before we get into Gremlin, let's talk little bit about your previous work. You were at Dropbox as well as DigitalOcean and these are companies that are at such a tremendous scale that low probability failure cases become pretty likely. Do you have any anecdotes from crazy outages or failure scenarios that you encountered while you're at Dropbox or DigitalOcean?

[0:04:55.3] TB: Yeah. So, I mean, you're going to have things break all the time when you work on large-scale infrastructure. At DigitalOcean, there are over 10 data centers around the world and so many customers that are using the infrastructure that you are providing to your customers. So you just see failure every single day, whether it's some sort of failure on the infrastructure side, like a hardware failure, a firmware failure, some sort of kernel related issue or a networking issue, or it could be something that the customers maybe set something up wrong or maybe there was a new feature, but they didn't configure it correctly and then some sort of issue happened.

I mean, I just saw so many big things happen while I was working at both companies. I think like in terms of an anecdote, one of the interesting things was we had a very big outdate where quite frankly sometimes your data center might go down. There could be some sort of issue or half the data center goes down and you need to be ready for that, and a lot of the time if you are offering people multiple data centers, then you'll hope that they would build their infrastructure in a way that they could fail over to a different data center if something goes wrong, but then that is quite hot to be able to build that out to do all the load-balancing. Often, when people start, they don't do that. It makes me think a lot about things like if you're starting a new company now, like start with failover in mind instead of just starting in one region.

One of the interesting stories was back when I was working at DigitalOcean really early on, I was one of the early employees joining in the 60s when there wasn't that many of us and we could all fit into this one tiny office. A lot of people work remotely, but I moved from Australia to New York, and I got to work with Mark IMbriaco who was our VP of tech ops and we had a really big outage for one of the popular data centers. Some of them are more popular than others based on the location, and I just think like the way that we handled it was really good. We work

really fast to get everything back up and running quickly, but then we also communicated with the customers really well. That's a big focus at DigitalOcean is showing love to customers, and because the customers the engineering like me, like I always really enjoy that. I love working on a product where your customers are engineers as well. To me, it's great.

We did a lot of things to try and make it better for them be you know that it's already really hard time, so we make sure that they understand the [inaudible 0:07:18.5], they're really clear. So we're like sending the messages and sending them updates so they don't have to write to us. I think that's really important. To me, I think a lot about that as you should send out push notifications when there are outages. You shouldn't like expect or have people worried or expect that they need to write to you. They shouldn't have to. Really, you should just be pushing things out just like you do on a mobile phone. If you like want to tell someone something, you sent a push notification. I think about it like that.

The other thing we did try make it better for people was we gave them free credit. So that's something that you can do if you're running a company. If you have some sort of outage, you can try and make it better for people. You're never going to get that downtime back, but you can try and give them something to make a little bit better. So we gave them free credit for hosting in the future and that was quite good. Yeah, as engineers, they really appreciated that.

[0:08:10.6] JM: At Dropbox, you were part of the migration from the cloud to Dropbox's own data centers, and I read about this. I did a podcast episode about it and I thought it was one of the more interesting migrations that I've talked about on the show and had the privilege to read about. Can you talk little bit about the experience of — For those who don't know, Dropbox started off as basically a layer of usability over S3, and over time as the service became more popular it became clear that by moving off of S3 as the core backing data store to Dropbox's own data centers, the cost savings could be immense. Maybe you could talk a bit about the migration there.

[0:09:01.2] TB: Yeah, sure. Yeah, it's a really interesting story. When I joined Dropbox a few years ago now, like two and a half years ago, and that was Magic Pocket, which is the block storage system that had been built within Dropbox to move off S3 and then move into Dropbox's

own infrastructure. Magic Pocket was built by a few engineers. One was James Cowling. He's also Australian, and he's done a really great podcast with you that I really enjoyed.

Yeah, he joined from MIT, like straight off doing his Ph.D. He joined Dropbox and he was friends with the founders, Drew and Arash, and so he tells a lot of stories about that, but he thought he could join Dropbox, build block storage for Dropbox and try and make it super reliable and make it very durable so that we would like not lose data, that's like the big focus, that customers could trust it and that would work really well, be very performant and also have very, very low on-call. There was a lot of goals in building Magic Pocket, and when he built it, he originally did it in Python, then it didn't work out so great in Python, so then it got rewritten into Go, and then it did work really work.

I was there when Magic Pocket had been built and it was running, but then we had to do the migration to move all the data from S3 into Magic Pocket. At the time I joined Dropbox and I was the SRE manager for the database's team at the start when I very first joined, and the database's team looks after all of the metadata. That's like several thousand database machines, like big beefy machines that are looking after all of the metadata, customer data, file information, all of these types of things. Also some databases that look after details to store the blocks and locate the blocks.

[0:10:45.6] JM: By the way, when you say metadata, Dropbox is essentially a giant file system, but in order to have a file system, you need metadata around those files and the locations and things like search indexes. So that's an incredibly important piece of infrastructure.

[0:11:03.3] TB: Yes, exactly. Without that then you would have the block, but you wouldn't be able to was actually get access to them and they wouldn't be attached to a person's name or the company. So that is really important, and it has to be super durable, very reliable. You want really great availability. Those two pieces are very core.

There's also another piece that's really cool to Dropbox, which is called server file journal, SFJ, which is like a journaling system and that has a lot of machines as well and quite a bit team. But, to me, when we did the migration, it was really interesting because you are — It's not just

migrating the blocks, but it's putting a heavy load on the metadata databases when you're doing. So we had to work really closely to do the migration, but it was very smooth.

There's an engineer on the Dropbox Magic Pocket team. He was there from the beginning. He's an SRE and his name Scott MacFiggen. Before Dropbox, he was at Facebook and he was one of the early engineers at Facebook. For me, it was an awesome opportunity to get to work with Scott. I learned so much from him about how to build a very, very reliable system from the beginning and make sure that it did have very low on-call and very low manual intervention that you needed to take. There's really not much manual operations that you're for Magic Pocket at all, and just the on-call is so low, you hardly ever get paged.

So working with him to do the migration, everything was smooth and it was awesome. Yeah, I mean, there's not too much more to say about that, because it actually went really, really well. To me, showing me that having a great team, having everyone work together, everyone being on the same page and just being super aligned, that's what is important.

[0:12:43.9] JM: How do you get that smoothness? Is it about rehearsal? Is it about creating an extremely detailed run book? Is it about having lots of meetings? Is it about doing it slowly and incrementally? What are the principles that allow you to get smoothness despite a very intimidating — At least from a reporter's point of view, very intimidating migration?

[0:13:08.9] TB: Yeah. There're a lot of things that you needed to do to make something smooth, and I think that over my time now, I've been looking for like over 10 years now in technology, working in different types of companies, working on large-scale infrastructure for many, many customers, and it is all about not going too fast. You don't ever slow, but you need to have the right pace. I like to think about it, if you're running a race, like you want to have the right pace so that you can keep going, because it's a marathon often. It really often is not a sprint. You want it to be a marathon and you want everyone to be able to sustain the journey and not get burned out and be able to understand what's happening, be able to put themselves forward in the best way possible so that they can play their role.

The other thing is having really clearly defined roles and responsibilities for who needs to do what. Also making sure that you have very good 24/7 on-call rotations. I think like there are a lot

of things that I've learned over years about what you need to do and there really are so many things like, say for example, with your on-call rotations, you need to make sure that those are ready, that everybody knows what's happening, when, what are the critical things that are happening that day.

For me, that's just about having very good communications, and I actually think Slack is very, very good for these types of things. So you can post automatically in Slack. We make a lot of — Yeah, post a lot of automated messages in there to let people know what's going on. Also, some communication around this is what we're planning to do today. This is what we expect to see. If something goes different, make sure to come into this channel and let everybody know what's happening. That way you really keep everyone aligned and then at the end of the day, recapping, because at Dropbox there's also a team that works in Europe, in Dublin. Some of the SREs are over there. So then you switchover in the nighttime and it's follow the sun, very similar to Google and other companies. So then you need to be able to transfer the knowledge over to make sure that they can hand that on.

The other thing that is important too, when you have your on-call rotations, I really firmly believe in having a primary. So somebody who is on-call that gets paged, but then also making it really easy for them to escalate if necessary. So then having a secondary on the rotation, and that secondary being present and available at all times. It's not that it's just that one person that needs to get paged and fix the issue if something goes wrong. They always should feel like they have a backup that they can trust, they can rely on. Then it should just be like a one click to escalate to the secondary, and then if that doesn't work, then one will click to escalate to the entire team so that everybody will jump on and resolve the issue.

I think actually having those layers where you can actually go, "Okay. I'm doing this right now. I think it's going to go well. We've planned it out. We have are rehearsed it. We've practiced it. We feel very confident in what we're doing," but then you also know that if something goes wrong, you're there to fix that. You have an idea of how to fix it. You have your run books. You have your scripts. You've got your tooling. So a lot of automated remediation tooling is in place at Dropbox, like tons. A lot of the time, you're not manually fixing something. It's more that there's automated remediation that will kick in to catch something that goes wrong, but you still want to be able to page somebody just in case.

A lot of the time the pages at Dropbox are warning pages, which I do think is important, because if something is unexpected, then it's good to throw up a warning page. Ed especially if you don't have that many pages happening all the time because you've really tightly managed your [inaudible 0:16:37.0] and the thresholds that you have, then you're going to get paged for things that are — They're not normal and you don't expect them and they're warnings, but you want to get them before they turn into an outage. So that's what I'd say about that. Just like attain that you can really work together well, you can trust everybody else. You know that you can always escalate. You never have to feel like it's all on me. You always should feel like there are other people there to support you. I think that's what's really important.

[SPONSOR MESSAGE]

[0:17:11.0] JM: Engineer chaos. Don't let chaos engineer you. When systems fail, you need to be in control. VictorOps is the incident management tool that you need. VictorOPs provides a comprehensive view of your system by integrating with the large number of monitoring, alerting and communication tools that your team already uses. It's your one stop shop for diagnosing an incident, escalating and alerting the proper engineers, collaborating during the firefight and resolving the problem. Through detailed notes, logs, run books and graphs, you can see the information you need in real time and also create detailed post incident reviews after the fact.

Head to victorops.com/sedaily. That's victorops.com/sedaily to see how VictorOps can help you. Go to victorops. com/sedaily. Be victorious with VictorOps.

[INTERVIEW CONTINUED]

[0:18:23.8] JM: We're going to get into chaos engineering eventually, because you are now working at Gremlin, which has a chaos engineering as a service. For those who don't know, chaos engineering is the induction of failures in your system deliberately in order to test the system's response to those failures, and this chaos engineering practice was simmering, slowly growing in popularity around the time that you were at Dropbox and working on that database infrastructure, the core database infrastructure that supported the accessibility of those files the we all use on Dropbox.

I know chaos engineering is obviously important if you just have a product where you want it to stay up and running. You've got a product, things are stable and so it's like, “All right. Let's keep it stable. Let's make sure things are going well, and we'll occasionally test this stability via chaos engineering.”

Do you also want to do chaos engineering when you're making this sort of giant migration from like the Dropbox S migration away from S3? Did you institute chaos engineering even in the midst of something that was like pretty much deliberately chaotic?

[0:19:46.3] TB: Yeah. So that's great question, Jeff, and is something that I really do believe in a lot. I've learned a lot from doing this over the years, a few different types of migrations. That was a really massive migration. Also, while I was at Dropbox, so I started being the SRE manager for the database's team, then I also became the SRE manager for Magic Pocket. So I was leading both teams, and that was a very good way to be able to transfer knowledge across the teams and also have a few SREs from the database's team actually go and work on Magic Pocket later too to share some of the knowledge around and just help our teams build up the knowledge between the two teams. I think that was very important.

We actually became untamed, the storage team, after a while too. I just think for chaos engineering, like what I started to do at Dropbox when I joined was do something called disaster recovery test, and that is related to chaos engineering, and I've been running disaster recovery test since 2009. So I started running them when I was working out in the National Australia Bank, and we have to run them, because it was part of your compliance. When you work at a bank, you need to be able to prove that you can fail over to be able to hold your banking license. Then there are regulators that actually come and check to confirm that you can actually failover.

What really happens, it's a very big exercise and it takes a long time. You have a representative from every service team across the whole bank. Your bank is pretty bit. There are thousands of engineering at most banks. Often, banks can be one of the largest employees of engineers in a city. That's definitely the case in Australia, and I worked at that bank for six years as an engineer, working on a few different teams. But the first team I started on was mortgage broking, and mortgage broking, that's very critical system. You need to work very quickly. You

need to make sure that mortgages get processed accurately and fast, because the thing there is if you're doing mortgage broking and you don't process someone's mortgage, then what can happen is they actually lose the ability to purchase the home. So they go to an auction, they put in their offer for what they want then they try and process the mortgage. If it doesn't go through within a certain amount of hours, then they lose it and maybe they go to somebody else and they're able to process their mortgage with some other provider or maybe they just lose the house and then they're very unhappy.

The regulators understand this, because it can be quite painful. If you've gone to an auction and you found your dream home and you've been looking for months and then you don't get it just because a system was down. So it was my responsibility to made sure that that system was up and running all the time. One of the things we did that was very helpful was disaster recovery testing, but there are some problems with disaster recovery testing. One of the things is it's a very long procedure. It takes a long time to organize to get everybody ready to do it and you have to do a lot of planning in advance.

What happens is you actually get people to go to a different building. We would actually leave the main office, because we're also testing the idea of what happens if the whole office is not accessible? Nobody can work from within the office and we all need to work outside of the office from either a different location or from home remotely. So we would actually go to a different unlocked location. It's a secret data center with its whole own set of machines. It was all bear metal at the time back then in 2009, and we will gradually failover every service and make sure that we could say, "Okay. Everything is still processing. Everything is still good." If it wasn't, then we would make a checklist of what we need to fix before the next disaster recovery test.

Really, you only run a big disaster recovery exercise like that once a quarter, and that's not enough. So what I believe is that you should be doing chaos engineering really daily at least, but to get to that point, you have to have a lot of things in place to be able to have this idea of continuous chaos and injecting failure every day, because you can imagine how many things change in a quarter,. and if you're only doing your disaster recovery test once a quarter and there are thousands of engineering making changes all the time, a lot of people think that banks wouldn't ship code very frequently, but actually do, and it's not just something that I ship once a month, I ship once a week, but they often can shift daily changes. It really depends on what

service area you're working on, but I worked on mortgage broking, foreign-exchange, internet banking, security engineering.

I learned a lot from that, and then I went to work after that in tech companies where you're also working on large-scale infrastructure and have a lot of customers, but you don't have the compliance requirements and the regulators coming and saying, "Okay. You need to make sure to prove to us every quarter that you've done this to keep your license."

So then it actually is the responsibility, I think, of SREs, site reliability engineering and production engineering, to take that on and say, "Okay. I want to make sure that I can say my service is reliable, my services is durable," and be able to actually say that to your customers and, yeah, just [inaudible 0:24:43.7] that, that you need to do that for your company and for your customers. I think that that's a very big responsibility, but it's very important.

To me, that's where chaos engineering really comes in, because you can actually make it something that you're doing all the time. You can inject failure every day. You can inject it multiple times a day, and that's something that we did at Dropbox when I started. We would do failure injection or disaster recovery test once a week, and then we decided, "All right. We're doing fine at doing them once a week. It's a great exercise. We would all get together as a team and do it similar to a game day."

Then after that we would go, "Okay. Let's do it even more." So then we start to do it twice a week, and it was really run within the service teams. Then after that, a lot of the other teams were saying, "They wanted to do some disaster recovery testing as well." So then we started to do chaos days, which was once a week. We would do some chaos engineering activities for pretty much all of the teams across the infrastructure. So that was a really nice progression there, but I still think chaos engineering is a very new field and there's so much room to grow and explore and also a lot of opportunity to be a pioneer in this space right now. Yeah. So I'm excited about it.

[0:25:56.6] JM: After doing a few shows with Colton about chaos engineering, I feel like I have a decent grasp about how chaos engineering works in the context of chaos engineering services, like stateless services that are just accepting requests from some other service or from

a front-end. We think of these things as maybe micro services that might have the 12-factor principles associated with them. But I have less of an idea of how to do chaos engineering on a database, and you can have unpredictable failures that occur within a database where, for example, bad data gets written and it completely corrupts your database. So that can be a tremendous problem.

It sounds pretty annoying to roll back unless you have well-defined rollback procedures and you know how to do that without disrupting service to the users. Is there a salient difference between chaos engineering services and chaos engineering databases?

[0:27:06.7] TB: Yeah. Also, it's like a very interesting topic and this is something I'm really passionate about, because I was leading the database's team. Also, I've just always love databases. So I've really always loved databases since probably early 2,000 — I don't know, maybe '99, 2000. I started to work with MySQL and I just always really loved the idea of a database. Also, my mom, she worked as a database administrator when I was in high school and I used to help her write queries for her job, which is pretty funny. But I always loved database.

[0:27:40.7] JM: By the way, that actually sounds really fun.

[0:27:42.6] TB: [inaudible 0:27:43.6] really fun. [inaudible 0:27:44.9] real production experience, like she would bring me into her office, and she was working in the university, so they had a lot of data that they were looking after. It was a lot of fun. They let me come in there.

Databases to me have always been something that I've been really curious about, I wanted to learn a lot about. It's this interesting thing where like a database is so important and so critical that it can be quite scary, because you don't want to do something wrong and you have to be very careful and you need to have a really good plan for what you're going to do. Like you said, you need to be able to feel confident that you can recover from any type of failure that occurs.

So when I went to work at Dropbox, when I started there, there are some amazing engineers who've worked at Dropbox over the years, and that before that they'd worked at other places, like YouTube, and then they are now working on the databases at Dropbox. One of the things

there is there's very good automated remediation for the MySQL infrastructure. By this, I mean, you would have a primary with two replicas and then also another pseudo-primary with two more replicas, and that's what the setup was like, but then there's also two sets of backups; local backups and also remote backups, which I think is very important to have two sets of backups.

Then a lot of automation around clones, promotions. If anything fails, there is actually a web UI called [inaudible 0:29:06.4] manager, which is the database manager, and it has logging for all of the automation that's happening. You can also kickoff emergency promotions from within this UI, which is really cool, and there's also a CLI tool as well. But it means that everybody across the company actually has the ability to see what's going on with the database. Is everything healthy right now? What backups are in progress? What promotions are in progress? What clones are in progress. Are there any sets where there's a primary with only one replica right now and it's missing a replica and it's coming back because the clone is in progress?

Or you want to look at things like how long does it take to run a clone? How long does it take for it to come back in? How long does it take to run a promotion? You can imagine that these things change based on the time of day, based on the network, based on the actual day of the week? But the thing that you can learn from doing chaos engineering is, with chaos engineering, you're actually watching it in real time. So if you do something like kill a replica, which is a very good exercise to do for chaos engineering, then in real time your whole team can watch and say, "All right. I kill this replica. Now an extra clone is being kicked off. I'm then going to have it come back into that cluster." Then you'll be able to say, "How long does that whole process take to any issues happen along the way?" And you can collect a lot of data because you have those machines and you can capture them, because one of the things with automation is that a lot of the time, if some sort of issue happens with a machine, you're going to throw it out of your fleet, because you want it to go back through to a hardware installer and come back out in very good shape. You don't want to just leave it in there if there's some sort of issue. It's just better to get rid of it and bring it in when it's fresh and clean and ready to be used again. I think that's a really good mindset to have.

A lot of the time, people talk about it as just not having your host or your service be your pets and giving them special names, but just making it easy to get rid of them, and that's what chaos engineering helps you to really think about when you're doing it on databases.

The other thing you can do — Because a replica — Shutting down a replica is — Like that's a pretty good exercise to do for chaos engineering, and it's not too scary, because you have another replica available and you should be automatically kicking off clones. The other thing that you can do is you can shut down a backup and then make sure that you're able to have your other set of backups there and also capture the additional backups that you lost. Make sure that that's automated too.

The other thing you can do is you want to be able to check how does your restore process work? If you're doing backup restores, which you should be too, and what happens if a backup fails during the backup restore process? That's another thing that you can do with chaos engineering. You can also do something that — I think it sort of starts as the level there where it's not as damaging to do something like that. Then the next level up is looking at doing disaster recovery on a primary. Actually, say, first, if you kill the primary, that's a pseudo-primary and those are — A cluster that's sitting there and there's replication happening that they're available if you need to be able to bring them over, but you kill that one and then saying, "Okay. Yup, my replica was able to be promoted, and then a new clone kicked off and I was able to get my cluster back into good shape again," and that'll happen fine.

If any type of issue happens, then you will be watching and you'd be able to know. "Okay, this issue happened. Now I need to fix that problem." If it happens in the middle of the night when nobody is on-call, but if nobody is awake and then it happens and they have to get paged, there's always that minute or so where they're just hopping on their computer and then seeing what's happening.

I think chaos engineering is like you're there right in that moment watching everything and you know what's happening, because you're the one that actually injected the failure. That's a much better place to do, rather than coming in, having to look at the logs and understanding what automation has happened previous to this moment and then what's actually failed and what needs to happen next. You're just there watching it. Actually, it's very, very good on-call training

chaos engineering. It makes your team being much better at working together and dealing with situations like that.

Then the next layer up is actually killing your primary, which is in your production cluster, and then you're testing, "Yup! That's normal. That's probably going to happen once in a while," especially if you have thousands of service. Your primary will die, because there'll be a hardware issue or there'll be some type of problem with your server. So then — Or maybe some sort of — You've got some sort of alert and then it automatically removed it from the fleet because there was some type of problem.

Then you want to be able to test that the replication has worked well, that the promotion happens as expected. Yeah, that's why it really makes sense to me, because I mean you can't expect that if you've got, say, a thousand primary, is that all of them are going to be healthy and stay up and running all the time, because it's not just going to happen. They're definitely going to fail and chaos engineering gives you the ability to actually control that failure.

[SPONSOR MESSAGE]

[0:34:17.8] JM: Users have come to expect real-time. They crave alerts that their payment is received. They crave little cars zooming around on the map. They crave locking their doors at home when they're not at home. There's no need to reinvent the wheel when it comes to making your app real-time. PubNub makes it simple, enabling you to build immersive and interactive experiences on the web, on mobile phones, embedded in the hardware and any other device connected to the internet.

With powerful APIs and a robust global infrastructure, you can stream geo-location data, you can send chat messages, you can turn on sprinklers, or you can rock your baby's crib when they start crying. PubNnub literally powers IoT cribs.

70 SDKs for web, mobile, IoT, and more means that you can start streaming data in real-time without a ton of compatibility headaches, and no need to build your own SDKs from scratch. Lastly, PubNub includes a ton of other real-time features beyond real-time messaging, like presence for online or offline detection, and Access manager to thwart trolls and hackers.

Go to pubnub.com/sedaily to get started. They offer a generous sandbox tier that's free forever until your app takes off, that is. Pubnub.com/sedaily. That's pubnub.com/sedaily. Thank you, PubNub for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:36:01.1] JM: Databases have an abstraction called a write ahead log, which is this append only history of transactions that have occurred across the database. So if you knew a time stamp of when your database got corrupted, you could conceivably look at the write ahead log, if you had the entire write ahead log up until that time stamp and just replay all of those events to get the database up to date and then omit all the events that happened after the corruption.

Similarly, if you had your primary, and your primary got corrupted somehow and you had a backup that was a snapshot of 10 minutes ago and you said, "Okay. Well, five minutes ago — Like 10 minutes ago we took a snapshot. We have a backup. Five minutes ago, we started having database corruption. What we could do is we can just take the first five minutes of that write ahead log and then apply that to that snapshot, that backup database snapshot, and then we can have a database that is correct up until the point of the data corruption."

Is that the typical model for doing recovery on some sort of database corruption? If not, maybe you could walk me through some different ways of doing incident response when you have some kind of database corruption issue.

[0:37:32.0] TB: Yeah. It's interesting. Write ahead log, some companies use that, but there have been big outages related to write ahead logs. This one that's publicly been written about, which was at Uber, where they were using a write ahead log, but then they actually experienced a problem and had an outage that went on for several hours after that issue.

So write ahead logs, I've seen some big problems happened there, but it is this idea that — Exactly what you said, that it should be able to help you, but it depends if you are going without idea of how to build your database infrastructure.

The other ways like if you use MySQL, there's like semi-sync replication, which is very good, and then you've got your been long. So if you're storing your been longs, you're going to be able to get that data back. Then also, I think it really depends on how you want to build your infrastructure, but when I looked at write ahead logs, I'm not sold on write ahead logs myself. I'm not, definitely not. That's just one thing to say, "I wouldn't build it that way."

Especially after reading the problems that have happened that, the [inaudible 0:38:39.4] outage was really massive.

[0:38:41.9] JM: It sure was. We did two shows about it. It just sounded terrible.

[0:38:46.7] TB: Yeah, exactly. So I think that's really tough. I like the idea better of having — Like a lot of people will say to me, "Tammy, why would you pay for two sets of backups?" It's like to me it's just a no brainer. It makes sense to put that extra money in there. Same as it makes sense to have another region where you have replication occurring and you have your pseudo-primary and your pseudo-replicas, and they're over on the other side of the county or something like that. You know what I mean? Instead of just having everything in one data center.

[0:39:16.1] JM: I'm sorry. What is that? Pseudo-primary and pseudo-replica?

[0:39:18.8] TB: These are database machines which you're not actually going to send user traffic to. They're just replication clusters. So they can be used to actually replace your primary, like the ones that you're actually using when you're sending user data to them, but you're not going to be sending traffic over to that pseudo-region. The idea there is that you have these machines that — They're using semi-sync replication on sometimes they may not be, they're using just replication going over there. Then they're going to be very, very close to what you have in your primary region. Then you have your backups, which you can have different types of backups; incremental backups and long backups, and this is a good thing to look into as well when you're doing your database infrastructure, is how do you store your backups, because there are different ways to store backups that make it faster and easier to recover incremental data that may have been lost or some sort of issue has happened.

So if you need to replay your backups, obviously you want to think about how long does it take you to replay your backups? How long does it take you to find the data that you need to find? That's why you would store it in different ways.

The thing there is if you have your cluster, your pseudo-cluster, it's different from just having backups, because they're actually proper machines that are running on your infrastructure and you're able to use those. Yeah, I think that's like a very smart way to do it and I really like it. The other thing you can do is active-active, which is what Slack does. I know I looked into that a while ago, what they were doing there with MySQL. The thing too about Dropboxe is it's not just MySQL. That's a really important thing to know.

A lot of companies would — If they're using MySQL, they just have MySQL. But Dropbox also has something called Edge Store, which sits above the MySQL layer and it's really, really great. It's a distributed data store. It was built in-house. It's written in Go. It has amazing like rate limiting, throttling and a massive team of engineers that are constantly creating new features for Edge Store. Also, it means that you don't have to deal with things like schema changes and all of that sort of stuff. Like that's just not what you need to do when you work on databases at Dropbox, because all of that is actually handled within Edge Store.

So the engineer are using Edge Store. They're not writing directly to the database, and that makes you alive so much easier as looking after the databases, because you don't have to worry about a lot of things, but then there are a lot of nice tools that have been created to help you deal with issues when you're looking after your database, but migrations is always a big one and schema changes, and I think there's a lot of good tools coming out of GitHub, for example, Ghost, which is by Shlomi. That's a really nice one to look at. But I think, for me, when you've got live scale infrastructure, you need to do things in different ways. Then it's really important to look at every layer of your stack and think about how you can make that more reliable, and to me it's all about injecting failure. Don't think that failure is not going to happen. Think that failure is always going to happen, but you want to be the one to control the failure. So you inject continuous controlled chaos. Learn from it and then make your infrastructure even more resilient going forward.

To be able to do that, you need to actually have engineers come on board, a team that believe in reliability. Take care about it. They want to make sure that your system is running as good as it can be. They prioritize that. Then I think this also comes into things like, as a manager, you need to be able to say, “We actually need to get more headcount to make this even more reliable, and these are the reasons why,” and putting together a plan of why you need to do chaos engineering and why you need to pay for double sets of backups and why you need to do all of these things. Why you need to build automated remediation tooling to test the restore process?

I think a lot of the time people don't do that, because they’re quite overwhelmed with maybe their on-call load or other types of issues that they have to deal with. If you take a bit of time to go, “In 2018, we want to focus on fixing these issues, and this is how we’re going to do it,” then put that forward up to your VP level, your C-level, I think that's a really important thing to do.

[0:43:35.7] JM: So now that you’re at Gremlin and you're working on making this kind of chaos engineering more accessible to people, and you look at something like database chaos engineering, how do you build a feature that lets you do database chaos engineering as a service, or do you just sort of do it at a higher level where you're just perturbing a service that communicates a write to the database?

[0:44:02.2] TB: Yes. The way that I would like to recommend that people start is, within Gremlin there are different types of attacks that you can do, state network resource attacks. One of the things that you can do is a shutdown attack, and this was very common from Chaos Monkey, from Netflix. They talked about the idea of just shutting down one instance. The thing with Gremlin is you are able to tag your instances based on the service. Then also any other tags that you'd like to add. So the thing that you can do to do chaos engineering for a database when you're ready for that is to do a shutdown of a replica, and you can do that within Gremlin. It would be very tightly controlled. You can choose the exact hosts. You get logs for that attack happening. You know exactly how long it's taking and then what the outcome of it is. I think that’s the best way to start when you're doing database chaos engineering.

The other thing that we’ve started to do is look at the outages that have happened outside in the real world. Last year I was on call coal when S3 went down. That was a massive outage. I think

[inaudible 0:45:07.3], so the incident manager on call at Dropbox for — I think it went for 5 or 6 hours. It was a really long time. Just sitting at my desk, on my computer, typing furiously, make sure that everything was okay. It was just a really intense day, because as the [inaudible 0:45:23.3], the incident manager on-call for the whole company. So you're the one that's managing that whole incident. Making sure you have the right engineers onboard, the right tech leads.

[0:45:32.3] JM: What do you even do? Did you have a protocol in place for, "If S3 fails, do X."

[0:45:36.5] TB: Yeah, definitely. That whole processes is very tightly managed at Dropbox. So the first important thing there is the [inaudible 0:45:46.9], which is a rotation, a very small rotation of five engineering leaders. That person is responsible for any big catastrophic outages like that, because we had done the migration off S3, so we weren't heavily impacted like other companies were, but there are still some ways that we were impacted.

So the thing that you do as an [inaudible 0:46:06.0], and I've been managing incidents for many years now, but the first thing is you figure out, "All right. We've detected that there is a problem. What are we going to do next?" You need to get the right people on board. We automatically kick off a few things, like automatically create a Slack channel for the incident, automatically page people that we need to page, make sure that the VP of engineering and the CEO knows, the cofounders know what's happening.

I would be responsible for communicating to them the update of what's going on and making sure that we're all in sync. The thing there is it's, really, that you just have to be working so closely together, that you don't have too many people working on at the same time. We just make sure, "All right —" I usually open up a dock, and I already have templates for these. For me, it's very easy. Just open up these doc. I have templates for [inaudible 0:46:55.1] issues that are happening. If it is related to some type of large service like that, [inaudible 0:47:00.3] just fill that in, "Yup. It's this service. This is what's happening. This is the current impact. What are the mitigation steps?" Then I ask the engineers to put in any steps that they need to go through, and if they need any one, need an extra help, so then now request help and then I'll go and find that for them. I might be calling people on the phone, making sure that people get online and are able to go in and help them fix the issues. Then we're able to mitigate really, really fast. Because

once you've got the right people and you know what you need to do, then you can just get stuff fixed and that's what we're able to do there.

The thing that that shows you, that incident is, "Yes. It will break. Massive services will fail and need to be ready for that." Actually, during that outage, we're able to mitigate the issue and recover before the outage even was over, which was really cool. So like our engineering team was able to fix everything within that time, and it all worked promptly for us. We had to had some issues with thumbnail previews, but everything was fine after that.

Then after that, the thing that you can look at using something like Gremlin for is you want to be able to repro that outage. So you want to be able to reproduce it and say, "Okay. If this outage happened again, how would we handle it? How would everything go? Would we be impacted? What would actually happen?"

So within Gremlin, it's really easy. It's like three clicks to reproduce that S3 outage. We've started to in templates, so you can actually have templates that you save, and that's a very, very good thing to do with on-call operations, is any time you have some sort of outage, as part of your action items off to the outage, say, "Let's actually reproduce this."

We say, "Okay. We need to do these action items. There are maybe five things that we need to, 10 things we need to do to make sure that this never impacts us again like it did today, and do all of those action items." Then in two weeks or three weeks or a month, actually reproduce the outage and then prove to yourselves that if it happened again, you wouldn't be impacted. That just makes you feel really great as a team, because you go, "Wow! We have this big outage. It impacted us in a negative way, but we fixed all of these things. We've reproduced the outage and now we're totally fine." That's just a really cold feeling. Just thinking about that makes me smile, because I've done it so many times now and you're like, "Wow! Our team is really awesome."

[0:49:19.8] JM: Given that you've worked at these places, like Dropbox and DigitalOcean and you've had first-hand experience dealing with these outages, like the S3 outage, and then I think about something like the Dyn outage that brought down Netflix and Twitter. How concerned are

you about these single point of failure internet services that we have come to rely on? Do you think things are getting better? Are you concerned at all?

[0:49:50.7] TB: Yeah. I'm definitely concerned. I would always say in my mind like, “Everything will break.” So you have to be able to build your infrastructure in a way knowing that it's going to break and that you’ll have some type of backup mechanism or you've got everything in place that you know that if it does break or when it breaks, because it's probably going to break some time, that you will be ready and everything will be okay.

One of the things that I've decided to think a lot about lately is third-party APIs, but just in general, for all types of different things. We rely on third-party APIs so much more these days. If you think about it in terms of like an ecommerce store or any type of platform where you’re actually making purchases online, so it could be something like DigitalOcean, and then people go through where they make their purchase, maybe they use a few different APIs. They might be an address validation API. They might be using something like a payment gateway; PayPal or Stripe or something like that. Then it could be all other types of third party APIs which are being used during that purchase.

So it's really interesting, because what happens if one of those fails? Is it critical? What’s your backback? Do you let the person continue if that fails? Because if it's something like the payment, like say if Stripe fails, which I remember there was a big Stripe outage a few years ago where two-thirds of the API request failed, then what do you do in that case? That can be very hard because you’re going to not be able to process those payments.

The other thing is what is it’s address validation? Do you just say, “Okay. Address validation is totally failed.” Do you have a backup for that? A backup service if your service fails or do you just let people make the purchase and don't validate their address and then validate it after? I think these are the things that we need to start thinking about more in terms of, “I’ve built my infrastructure like this. Best case, every single API works. Worst case, every single API fails.” Then make sure that you actually are building your infrastructure in a way that it is more resilient and reliable.

[0:51:52.6] JM: But you have to admit the fact that we are going to mourn more third-party vendors, probably decreases the chance of correlated failures. Whereas if we are managing these all ourselves or if we were going entirely to AWS, for example, probably there would be more correlation among those failures.

[0:52:13.2] TB: Yeah, that's an interesting thing too when you think about something like database backups. If you have all of your infrastructure in AWS and all of your backups in AWS, then you're all in one place. All your eggs are in one basket, but then you can think about, "Okay. Actually, maybe I want to have another set or backups that either they're local backups in your own data center, or maybe they're in GCP or in Azure, something like that, but I mean it is good that these days you can easily create a GCP account or an Azure account and you don't have to just have everything in AWS. It's very easy to do that and you don't have to go and like sign an agreement at a data center, in a colo space and like buy all your hardware. So that is really great. It makes it easier to handle the failure and then it's just we're at a point now where when you inject the failure using something like Gremlin and doing chaos engineering, you can actually test that everything else that you've built to handle that failure actually walks. So I'm excited about where we're at right now.

[0:53:14.2] JM: Yeah, I love seeing these different cloud providers all mature in parallel with each other and they all get their core competencies. They're starting to develop divergent services, and then you have this Kubernetes phenomenon where now we have a common layer of infrastructure between the cloud providers, which makes things much more copacetic. It's not closely related to what we're talking about, but do you have any thoughts on that? How is Kubernetes going to change the way infrastructure is managed?

[0:53:46.1] TB: Yeah. Actually, I've been working with Kubernetes a lot over the last few years. I started using it back in early 2015, and that's because I was at DigitalOcean and they asked us to think about providing Kubernetes for DigitalOcean customer and at the same time we were also providing CoreOS. It was when they were both very big at the same time. That's when I first got to meet Kelsey. I was visiting Docker and I met Kelsey Hightower there. Got to talk to him about Kubernetes, and since then I've always been working with Kubernetes. I actually teach a chaos engineering boot camp where I spin up a Kubernetes cluster, and I've done this at Velocity and also I did it at SRE Con just last week, and I give everyone who comes along,

usually there's like 200, 250 engineers who come along. I give them their primary with two nodes for their Kubernetes cluster, and then I deploy a demo microservices app on there and I give them some open source chaos engineering experiments to run, and then sometimes I'll also give them a Gremlin agent to run, because the way that Gremlin works is you have an agent on your host. You can run it directly on the host or in a container and then you can trigger attacks using the UI or the API or CLI, and I'll show them how to run a few small attacks, like a CPU attack using my open source tooling, which is on my GitHub. Then you can actually see how that impacts your Kubernetes cluster. CPU attack, [inaudible 0:55:09.9] too much of an issue.

There's another attack, which is network corruption, and that one you're going to see a really big issue. Also, if you do some network latency attacks, something like that, you will see how that failure injection will impact your cluster. I think it's a really good exercise, because the great thing about Kubernetes and all the cloud infrastructure providers that are out there now is you can just spin up a cluster. In five minutes, I've written up a tutorial which I've shared on the Gremlin community about how to create that Kubernetes cluster with the demo microservices up. So you can actually spin it up and then start to do some chaos engineering and learn from our real-world experience. I think that's what it's all about. Like these days you can do that. So like, let's do it. Let's learn more. Let's actually figure out what those failure modes are. Let's try and make our software more resilient.

[0:56:02.2] JM: I'm pretty interested in what it was like to transition from a company like DigitalOcean or Dropbox, which have larger developed markets. You have a large customer base to a more nascent market, like Gremlin, where people know that they want chaos engineering eventually at this point, but the company is fairly new. People are still not completely comfortable with the process of chaos engineering. What are some of the transitionary going from a large, well-developed market style company to a more nascent market?

[0:56:46.4] TB: Yeah. When I joined Gremlin, I actually spoke to them a few years ago, the founders; Kolton and [inaudible 0:56:51.4] and they'd said to me, "Oh! It would be great for you to join us one day," and that was when it was just the two of them and they were building Gremlin out. They just left Netflix, Amazon, Salesforce where they built these chaos engineering

platforms before. I joined a few years after that as employee 9, and it's definitely the smallest company that I've worked at.

When I joined, I was really excited to be able to join early on though, so I can help build the product out and make sure that it works really well for engineers and that we're providing really great value and helping people build really reliable systems based on what I've learned in past. But it's definitely a very different experience, because, like you said, chaos engineering, it is very new. There are some people who've been doing this type of work for the last few years, but the idea of chaos engineering where it's continuous chaos, it's automated. Really, you should be injecting failure constantly. Actually, it shouldn't be causing an issue because your infrastructure is able to handle it. It should be actually more rare that the chaos causes an issue and actually pages you and you need to fix something. That's a really new area, and I think it's exciting that we get to be the pioneers in that space and I really enjoyed working with early customers who have come on board that are doing really exciting things in the space of chaos engineering.

Yeah, to me, working at a smaller company, it's really like the startup struggle. We have a very small office. There's not many of us. I spent my first few weeks working remotely from Australia, but I just think it's an exciting opportunity and I'm really glad to be on board.

[0:58:24.7] JM: Okay. Timmy Butow, thanks for coming on Software Engineering Daily. It's been great talking.

[0:58:28.6] TB: Thank you very much. Thanks for having me, Jeff.

[END OF INTERVIEW]

[0:58:34.5] JM: LifeRamp is one of the fastest growing companies in data connectivity in the Bay Area and they're looking for senior level talent to join their team. LiveRamp helps the world's largest brands activate their data to improve customer interactions on any channel or device. The infrastructure is at a tremendous scale. A 500 billion node identity graph generated from over a thousand data sources running a 85 petabyte Hadoop cluster and application servers that process over 20 billion HTTP requests per day.

The LiveRamp team thrives on mind-bending technical challenges. LiveRamp members value entrepreneurship, humility and constant personal growth. If this sounds like a fit for you, check out softwareengineeringdaily.com/liveramp. That softwareengineeringdaily.com/liveramp.

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

[END]