EPISODE 930

[INTRODUCTION]

[0:00:00.3] JM: Databases go offline. Services fail to scale up. Deployment errors can cause an application back-end to get the dust. When an event happens that prevents your company from operating as expected, it is known as an incident. Software teams respond to an incident by issuing a fix. Sometimes that fix returns the software to its ideal state. Other times, the software remains in a degraded state and it takes more fixing to return the software to the place that it should be.

One way that a software team can learn from an incident is through incident reproduction. When an incident is turned into a reproducible system, it becomes a predictable training exercise, rather than a surprising and painful outage.

Tammy Butow is an engineer with Gremlin, a company that makes chaos engineering software. Chaos engineering is the process of creating controlled experiments that simulate outages. Tammy returns to the show to discuss common incident types and how those can be made reproducible for training exercises.

If you're building a software project, post it on FindCollabs. FindCollabs is the company I'm working on. It's a place to find collaborators for your software projects. We integrate with Github and make it easy for you to collaborate with others on your open source projects and find people to work with who have shared interests, so that you can actually build software with other people, rather than building your software by yourself.

FindCollabs is not only for open source software, it's also a great place to collaborate with other people on low code, or no code projects, or find a side project if you're a product manager, or somebody who doesn't like to write code. Check it out at FindCollabs.com.

[SPONSOR MESSAGE]

[0:02:01.9] JM: This episode of Software Engineering Daily is brought to you by Datadog, a full-stack monitoring platform that integrates with over 350 technologies, like Gremlin, PagerDuty, AWS Lambda, Spinnaker and more. With rich visualizations and algorithmic alerts, Datadog can help you monitor the effects of chaos experiments. You can also identify weaknesses and improve the reliability of your systems.

Visit softwareengineeringdaily.com/datadog. to start a free 14-day trial and receive one of Datadog's famously cozy t-shirts. That's softwareengineeringdaily.com/datadog. Thank you to Datadog for being a long-running sponsor of Software Engineering Daily.

[INTERVIEW]

[0:02:56.1] JM: Tammy Butow, welcome back to Software Engineering Daily.

[0:02:58.3] TB: Thanks very much for having me, Jeff.

[0:03:00.3] JM: You've been at Gremlin for almost two years. How is your perspective on chaos engineering evolved since joining the company?

[0:03:09.5] TB: Yeah. I mean, it's definitely changed over the last two years. I think when I first joined Gremlin, I'd come in and I'd been doing chaos engineering the last few years before that at Dropbox, where I'd primarily focused on chaos engineering for very specific services, storage, databases, build, code workflows. To be so focused on those areas, you learn a lot about the detail, like how should you specifically practice chaos engineering on databases.

Joining Gremlin, it gave me the chance to really step back and look at how you can practice chaos engineering in a number of different ways on different cloud providers. I've been able to look at AWS, GCP, Azure, but then also all types of different systems and for different reasons. Yeah, one of the things that I'm most interested right now is how you can use chaos engineering to reproduce incidents. By that, I mean, say for example, looking at some of the biggest incidents that have happened at your company, or even that have impacted the entire industry, like massive DNS outages. There was one with Dyn. Then the really big S3 US-EAST-1 outage that impacted so many companies that we all use and love.

Yeah, just being able to figure out how you can use chaos engineering to reproduce those incidents, to make sure that if they ever happen again that you're ready. That to me has been a really good learning lesson. I've just been able to actually have a lot more time to step back and think about how we as an industry can evolve. That's been very good. Because before, I was so much more focused on in the weeds, specifically on databases and trying to improve MySQL in particular. Now this has been awesome to just get to really think as an industry how do we evolve, not just MySQL specifically for chaos engineering.

[0:05:05.8] JM: Gremlin is a very specific type of infrastructure product. It is nonetheless an infrastructure product. Do you have any general lessons about how to build infrastructure products that you've learned over the last year?

[0:05:22.1] TB: Yeah, sure. I think I've learned a lot. Some of the interesting things that happened early on, when you look at building developer software, developer tools and specifically infrastructure tools, I previously worked at DigitalOcean, so I got to work on cloud infrastructure there and I learned a ton through that experience, which was awesome.

I think the main thing that I learned is there and also at Gremlin is when you're building infrastructure tools, this goes for building it as a business, as a company where you're providing software to people all over the world, or if you're an engineer, that is an infrastructure engineer working inside a company and you're building some tooling, or software, or service for people inside your organization. That's what I was doing at Dropbox. I was building storage systems for other engineers to use at Dropbox, to then service customers.

I think one of the most important things which people don't really spend too much time on but is very important is onboarding of your customers, of your users. That includes installation and making sure that that's really easy. That's actually one of the really important areas to focus on.

I got to actually do a lot of work to improve that while I was at Dropbox, where I spoke about this at GopherCon when I did a talk there a few years ago. We dramatically improved the onboarding experience, because you want your customers who are Gremlin in our case. They're engineers. You want them to be able to love using your product from the first day. You want it to

be super easy to use. You want to be able to help them get up and running. You want everybody to be able to get value out of it as fast as possible and you want to just reduce any issues that they come up against.

Often, we call it developer friction. You want there to be no developer friction. Yeah, I've got to actually do that for most of my career. It's something that I'm really passionate about. I think we could do better as an industry too. I look forward to seeing that evolve over time, because when you do think about yourself as an engineer, you're always building this. If you're an infrastructure engineer, you're always building it for other engineers. The product that you build, if you're inside a company, or if you work at developer tools, you're building it for engineers who are your customers.

Then the other thing I've got to focus on is we have an agent that we deploy on hosts, or inside containers. One of the things when I first joined Gremlin was I was like, all right, I think Kubernetes is just going to get more and more popular. It makes sense to focus on that and containers in general. We actually have the ability to use containers with Gremlin and that was a big thing that we released as part of our product. If you go to the UI, you can see that you can attack specific containers when you're doing your chaos engineering and you can also install Gremlin in a container.

I've actually been a Kubernetes user since 2014. It's been a long time for me. It's been interesting to see that evolve too, to go from everything running directly on the host, to now a lot of companies are using Kubernetes. I'm just think it's going to become more and more popular, especially managed services when we look at Kubernetes.

Then that makes you think too. A lot of people say to me, “Hey, but if I use our managed provider, will everything just work out of the box? Auto-scaling. Reliability. Redundancy. Failover.” Obviously, we know no, it won't just work out of the box. You have to verify it and make sure that it does work, but that's just what you learn as you do these chaos engineering experiments and as you try out new software and you try and build infrastructure services, even when you are using a managed provider.

[0:08:53.9] JM: Yeah, from my experience talking to people, it doesn't take chaos engineering for you to figure out that your Kubernetes installation is a little less awesome than is advertised by the most generous Kubernetes marketing out there.

[0:09:08.0] TB: Yeah. It's an interesting thing that I would love to dive into more. I've recently been reading some failure stories, specifically around Kubernetes, or containers in general and just learning more about what are the challenges that people are coming up against. Since so many people are interested in using it, what are the things that we can do to improve? Because Kubernetes is open source, so we can all contribute back. I mean, that's really exciting to me. I love the idea of diving into those issues and then being able to actually come together as a community, fix those problems and then keep moving forward. That's a big thing that I love to do.

[0:09:46.9] JM: What are the common failure cases, failure scenarios within Kubernetes deployments that you're seeing?

[0:09:52.2] TB: Yeah. One of the ones would definitely be order scaling. I think that's an interesting one. Recently, we created a – we launched a new product called Gremlin Scenarios. The idea there was let's look at common types of outages that have happened in the past and then try and figure out what scenarios can we create to be able to reproduce those outages, so that if they ever happen again, you won't get bitten again. You'll be able to handle it when it strikes again.

One of the things there is we have a failure mode, which is okay, I had a pod fail, or I had a host fail, or I got a lot of traffic and I didn't auto scale correctly. I think that's an interesting thing too, because we've seen ourselves, like when we set up Kubernetes, say on Amazon, on AWS, maybe you think out of the box that auto scaling is going to work, but sometimes it doesn't. Sometimes you need to actually make sure that you're configuring it correctly to meet your needs. It also depends on how you're auto-scaling.

I think often, everyone knows the term auto-scaling, but it's like, let's go to the next level. How does it auto-scale? Does it auto-scale based on CPU spiking? Then how much does it have to

spike by? Is it because it spikes specifically on a node, or is it because you have it spiking across one pod, or multiple pods? There are a lot of those questions.

To me, it's that idea of the model where you ask why five times and you keep going down. Or how. How does it auto scale? How does CPU spike? How do I know that CPU is spiking? How long does it take to auto-scale? Does it scale back down, or am I stuck at this really high cluster that I'm now paying for? That's another thing as well, scaling up, but then scaling back down as traffic adjusts over time.

Yeah, I've seen a lot of interesting things there. Then I saw that recently somebody shared a compilation of Kubernetes failure stories. I've been reading those as well to learn more about it. I honestly think the interesting thing to me is a lot of the issues are the same issues that you would see if you weren't running Kubernetes, which is interesting to me.

Very similar to the issues that you would see if you're running EC2 and you're doing auto scaling there. I think we can actually learn a lot in terms of preventing outages by looking at outages that have happened in the past. Were we able to manage auto scaling correctly before? How do we know that we can manage it correctly now, now that we're using Kubernetes? You need to really verify that.

[0:12:21.2] JM: Do you ever get a feeling that maybe this whole Kubernetes thing is crazy? Maybe way too many people are thinking about this whole re-platforming effort, or is it definitely worth it?

[0:12:31.6] TB: Yeah, I don't know. I used to wonder about it, because I've been working in tech for a long time now, over 10 years and I started in I guess, you would say a very old school industry. I mean, I worked on incidents that involved mainframes in banking. If someone was to say to me, “Hey, Tammy. Do you think that every company should use mainframes?” I'd be like, “No, definitely not.” I do think that it's good for us to evolve and to improve. I just think we need to do it together and we need to share the results. I actually am a big fan of that. I'm a fan of using new technologies and trying to move forward, because I have been so far back in the past and I know what that world was like.

I'm really glad we are where we are now. I think that to get better, it takes time and it takes effort. There's a lot of big companies that are focused on figuring out how we can use Kubernetes. I think that that's a good thing actually. It's good for us to come together and focus. It means that you don't have everybody going off in many, many different directions, which I think would be worse. I actually like the trend that I'm seeing with everyone coming together to try and say how do we fix this together as an industry? How do we make systems more reliable? How do we make systems more scalable? That's why I'm involved myself.

I totally understand where you're coming from, because obviously it is very popular. I mean, I've been involved in Kubernetes since 2014. Now it's almost 2020, so it's actually been a long time and it's really like, I've seen Kubernetes evolve. It was much harder to use back in 2014. It was much more confusing. It's so much better now. That's also the thing. I've come from where I'm like, “Wow. It's night and day. It’s totally different.”

[0:14:19.4] JM: Over that period of time, infrastructure has gotten a lot better. Just using infrastructure, I feel across the web, I feel the web as a whole has gotten a lot better. I don't know for sure, but something about it just to the extent that I can put my finger in the wind and say that this is true. It's something about it, it feels like Kubernetes is to blame for a lot of particular kinds of improvements that have occurred over the last six years.

That said, I do wonder is it for everybody, or is this – maybe is it something that either you're a cloud provider, or you are offering a service that has such a particular type of load that it begins to look like a cloud provider? I don't know, these are very nascent thoughts for me.

[0:15:03.9] TB: I think those are great questions though, awesome questions to ask. Because to me, the reason I like Kubernetes as well is because I came from a database background where you thought a lot about redundancy and reliability. You did want to have yeah, the idea of having a primary and having replicas. Then with Kubernetes, that's what it's really trying to push is this idea that if a pod goes down, then there should be another pod that comes back to replace it, which to me just reminds me of scaling databases. That's what you do.

I like that, we’re thinking about compute like that as well, because we always did think of storage like that, because you just have to. If you lost your storage machines, then you would

lose your data, so it was very bad, whereas if you lost your compute, then maybe your users can't access your website, which is also really bad. Now we're taking it more serious. I still think there's a lot of work to do to be able to get to a point where it is reliable. I mean, I think that that's one of the reasons why people are interested in Kubernetes.

The other thing is I think it depends on the workload. Obviously, not every workload make sense for Kubernetes databases. I don't think that – there's many other products, if you were to run database infrastructure, that work much better than on Kubernetes, right? Because it's more focused on compute workloads. I think you always have to figure out what is my business? What is the core service that we offer to our customers? Does it make sense for us to run it on this type of infrastructure? Also, what is the scale? How many users do you have? Does it meet the needs? There's a lot of work that you need to do there when you're planning it out.

[0:16:39.0] JM: Have you heard much from anybody who's using AWS Fargate, or the Azure container instances, or Amazon ECS? I'm very fascinated by these flavors of Kubernetes that are a little bit more like the Heroku heavily managed experience. To me, these make sense for so many enterprises, which may be in some cases deploying Kubernetes for mysterious reasons, or maybe good reasons that I don't understand.

[0:17:10.1] TB: Yeah. I mean, I myself have got to try out a lot of different platforms. I've recently used Fargate when I – I was trying that out, because I wanted to see what it's like. I used it to create an API. I was pulling data from DynamoDB. I would say it was an awesome experience. It was really good to use. I thought it worked out well for my use case, specifically that use case. Yeah. I mean, I thought that was great. I had a great experience.

I think it's been interesting to see AWS services change over time too. I would say some of them are becoming much more easier to use DynamoDB. Also, I've been using that a lot. We use it at Gremlin. I think it's become really good to use and it has a lot of – people are thinking about it in the right ways when they think about databases and reliability and durability and scaling out Dynamo. I've just seen a lot of improvements happen. It's good to see AWS listening to customers, listening to the community and trying to create these different types of products.

Like you said, Fargate is heavily managed and it's also really easy to use and it's fast. For me to create in my DynamoDB database and then use Fargate to create an API, that was fast. It was hardly any time at all. Maybe under an hour, which is pretty amazing to me. When you think of before how long it would take you to create that infrastructure. To me as an infrastructure engineer, I was like, “Oh, wow. This is fast. Now I can move on to the next thing that I need to do.”

[SPONSOR MESSAGE]

[0:18:44.6] JM: When you listen to Spotify, or read the New York Times, or order lunch on GrubHub, you get a pretty fantastic online experience. That's not an easy thing to pull off, because behind the scenes these businesses have to handle millions of visitors. They have to update their inventory, or the latest news in an instant and ward off the many scary security threats of the Internet. How do they do it? They use Fastly.

Fastly is an edge cloud platform that powers today's best brands, so that their websites and apps are faster, safer and way more scalable. Whether you need to stream live events, handle Black Friday traffic, or simply provide a safe, reliable experience, Fastly can help. Take it for a spin and try it for free by visiting fastly.com/sedaily.

Everybody needs a cloud platform to help you scale your company. Everybody needs a CDN. Check it out by visiting fastly.com/sedaily.

[INTERVIEW CONTINUED]

[0:19:57.6] JM: The chaos conference was fairly recently and you came in to San Francisco for it. I stopped in very briefly and I had a conversation with Kolton, who is the CEO of Gremlin, just at this after-party thing. It's funny, because I've talked to him several times in the past, but it was something about that being at the conference that I started to realize that what you're working on is pretty – it's a total category creation. Not only is it a category creation, the chaos engineering category, but it's a category creation of a very – arguably a painful product. It's hard to adopt. I mean, that's the point is that it is going to make you have a bad time to front-load the bad times, so that the good times will roll more smoothly and more frequently.

It was interesting, because I was talking to him and then somebody from a large tech company came up and was talking to him about advocating for chaos engineering within the organization. That it's really this process of evangelism. There's so much evangelism that's required to convince people that chaos engineering is worth doing.

How is the evangelism going? Are you feeling a sense of understanding in the engineering community about the worth of chaos engineering?

[0:21:19.3] TB: Yeah. I mean, it's interesting. That makes me think of a lot of things. I think as an engineer, whenever you are creating a plan for what you want to deliver in a quarter, or in a year, you need to evangelize what that is internally in your company, anything. If you want to build a new service, if you want to do a migration, if you want to replace the system for another system, you need to try and understand how to get buy-in for what you want to do, then you need to figure out how do I best present this information within my organization? Who are the people that I need to speak with?

Then you need to actually go there. You need to meet with them. You need to present your idea. You need to get their feedback. You need to give them the chance to ask you questions. I mean, I've been doing this for my whole career. Whatever it is that I wanted to build if I had an idea and I have a lot of ideas, so I always need to figure out how do I package this up, how do I present it, how do I get buy-in, how do I get feedback, how do I then actually get started, how do I get head count? There are so many things that come into play there. To me, chaos engineering is the same as that.

If I was thinking like, “All right, I want to –” I mean, when I was at Dropbox, I did a massive migration where we removed legacy MySQL databases and move to a new database that we built, I had to get approval to have 70 engineers across the entire company help me do that big migration, because it required a lot of rewriting of code and a lot of work for backfilling data. I mean, I just think of it in the same way as that.

There are always big things that you want to get done. My advice, I think SREs do this all the time. SREs always have to be advocating for reliability. You have to advocate and say, “Hey, we

need to be measuring the right things." We need to be actually caring about our customers and making sure we don't have a ton of downtime and making sure we don't get tons of support tickets and customers complaining that our product isn't working as expected.

Yeah, I don't know. For me, it comes naturally because I've been doing it for so long, but I'm very passionate about whatever I build and whatever I create. I want it to be of a really good top quality. That's my advice when people ask me how do I do it internally. I'm like, "Look, you have to be a bit brave." That's my first tip. You have to be brave. You have to really back your idea. You have to collect data. I always recommend if you're an engineer and you have an idea for something that you want to do at your organization, it's better to – actually, this is a tip that I learned really early on. Put together a little slide deck, even if you've never made slides before. Just five slides, what is the why, how, what behind your idea and then figure out who you need to present it to.

It's much more effective to actually try and book in some time, or go to open office hours with your CTO and actually take your deck and present your deck in person and get feedback in real-time, instead of just posting in say a Slack channel, or just shooting someone an e-mail, or just shooting someone a DM. That's a really easy thing to do, but it's not as high value and you won't get as much return on investment, because you went on investing much time into that, maybe you've thought about it a lot, but just writing a small sentence doesn't – it really enable you to showcase your idea and all the thought and time you put into thinking about it.

Yeah. I mean, I think it's great. I love to see engineers really advocating for their ideas and what they want to do. I would say as an engineering manager, I love it when engineers come to me and say, "Hey, Tammy. I've collected this data. I identified this problem in the organization. Get some ideas that I have for how you could potentially fix it. Here are three different types of ways we could do it. We'd love to hear your feedback. Do you think this might fit into our roadmap?" That's awesome. It actually hardly ever happens. You'll notice that often really senior engineers will definitely do that, but I think it's great for every engineer to start thinking about doing that. That's how we really move forward and how you provide great value.

[0:25:11.7] JM: When you're encouraging an organization to adopt chaos engineering, or somebody in an organization is trying to get buy-in to adopt chaos engineering, where is the

place to start? Is it the CIO, or the CTO, or is it just an individual engineer within the company? Can it be a bottoms-up thing, or does it have to be top-down?

[0:25:36.2] TB: Whenever somebody comes to me, it could be anybody that could come. Sometimes it's an engineer. They've read about it. Maybe they came to a talk, or they listen to a podcast and they wanted to know more about chaos engineering. Then often, they'll ask me like, “Yeah, how do I get my team excited about this? I think it could be useful.” I'll say, “Okay, what are your top problems that you have right now?” Because to me, I love to focus on value. What is the value you can provide? The best way to provide value is to solve problems that are really painful for people in your organization.

For example, say you just have a lot of problems with on-call. That is a very common issue in our industry, people getting burned out, people working really long hours. I was on-call every second week and I was on a 24/7 schedule. I would be getting paged all hours. I'd have to stay home all weekend long, because I'd need to be able to open up my laptop within a minute or two to be able to get online and fix problems.

I've lived that life, lived it, breathed it, and I know how painful it was. A lot of people want to fix that. Then for example, if that's the big problem that they have at their organization, they got a lot of pain from on-call then you can say, “All right, well you can actually use chaos engineering as a way to improve your on-call,” and I've done it in many ways.

One of the good ways to do it is to use it for on-call training. You actually go, okay, let's sit in a room together. It's not the middle of the night where you're getting paged and you're by yourself at home. Let's actually train everybody up before they go on-call, which is a really new idea for our industry. I randomly just came up with that idea one day when I was at Dropbox, because it didn't make sense to me to just throw people the pager and say, “Good luck.” I just thought it wasn't fair. It didn't set people up for success.

I love the idea of setting everybody else up for success. Yeah, that's one thing that I might say, “Okay, if you have a problem there,” let's figure out how to do on-call training. We can reproduce your common incidents that have happened in the past using chaos engineering. When we're all

in a room together we can start on staging. We don't have to start on production. That's how you can really do it. That's a great way.

If they're really struggling from downtime, from outages, then you can also use chaos engineering for that too. There are a lot of different things that I like to focus on. If people are having a lot of problems with the network and they're not sure how to debug that, then it might be – I might say, “Well, maybe you need to focus more on network chaos engineering, where you're injecting latency, packet loss, maybe black holing traffic.” I think that's really what I would say.

It's not just one thing of everybody across the industry should do this specific thing. I think it's really about figuring out where your problems are in your organization and then figuring out how you can provide value. Then following through, being brave saying, “Hey, this is what I want to do.” I think you need to find other people in your organization to work with you. I always love to go to my VP of engineering, or my CTO to say, “Hey, this is what I think we do to get value. I think we could actually get a lot of value by doing this for three months to start and then let's reevaluate it. Let's do it as an experiment.” That's one of my biggest tips. People seem to really listen when you say that and they're happy to give you a chance. I've never had anyone say no. Not yet.

[0:28:55.8] JM: You mentioned there something about incident reproduction. It makes me wonder, when a company has an incident, is it usually that this incident is something that happens on a regular basis, like the same incident that recurs on a regular basis that may be the same node dying, or the same database error? Or are incidents highly variable?

[0:29:23.2] TB: Yeah. I would say there's actually – incidents can fall into those two buckets. There are some incidents where I will say they are the same incident over and over and over. A lot of people, some people in industry disagree with that, but I will give you an example for myself where I know I just got the same incident over and over.

Say if you're in a small team, maybe you only have three engineers on that team that can be pretty common and that's what your on-call rotation is. Say you're on call. Every night you get paged at 8:00 p.m. because a batch job is hitting your database and it's ending up, causing

performance issues and you're getting paged for it because there's some problem and you identify that yeah, it's just some random batch job that someone set up to hit the database to pull data.

Then you realize, “Okay, well we need to actually figure out do we need this batch job? Can we get rid of it? Who actually set this batch job up?” I've found that has happened a lot to me, where there are different types of batch jobs that are doing that. I'll often say it just happens over and over again, because the batch job would pull at the same time and hammer the database at the same time and it is just the same incident.

As a conclusion to those incidents, often I’ll find that yeah, it was set up by somebody that's no longer even at the organization. You're like, “Well actually, no one's using this anymore. This isn't valuable. We should really turn this off.” I think there's a lot of that happening. If you don't ask why is the database getting hammered at the same time every night? Why am I getting paged for this? Where is this batch job coming from? Why was it set up? Who set it up? Do we even need to run it anymore? Can we get rid of it? Then you're just going to be having that same pain every single night and you're going to have that same incident just repeating over and over and over. That happens a lot.

Same thing with like you said, machines going down and there's no automation to bring them back up. I saw it recently. There was an outage where it was just a host failing for one company and it took them an hour to bring back up the host. That's not very good. That's a really long time, mean time to recovery for just starting off a new host. I don't think that's good either. Then when you look at other incidents that are very rare, they might be – it might be something very unusual that happened and maybe it's only ever happened once in the life of the organization and it never happened again, then that's important to know that as well and you put that into the buckets of incidents that are much less likely to reoccur, but you do really need to separate those out.

I think an important thing to do there is also to name your incidents. This isn't common practice yet in the industry, but I wish it was. Something we did at Dropbox was we would name our incidents. It would be sev zero, lazy walrus. We actually gave it a real name. It was automatically generated using a Python tool that had been built in-house. That was awesome,

because then you can say hey, this incident happened again. Lazy walrus, or slow crow and everyone knows what that incident is. If it never happens again, then you never hear that name again. I think that is really important to track repeat incidents and that's why you name incidents.

[0:32:31.6] JM: Among these two buckets, let's say the incidents that recur on a regular basis versus the ones that seem to be fairly unique, which are the ones that we should try to reproduce?

[0:32:45.8] TB: I would actually say – this is a common question that I get as well. Where do I start when I'm doing my chaos engineering experiments? The first thing that I like to say is first, figure out what your top five most critical systems are for your company. That's important first. Say for example, if your critical service is a DynamoDB database, then you need to go and figure out all right, what outages and incidents have we happened for DynamoDB in the past? Since it's one of our top five critical services, then look at those two buckets and go well, what were the impact of those incidents?

We have this really rare incident. How long did it take us for it to go through and actually get resolved? Do we have action items that came out of it that we still haven't actually closed, that is still open? That's one of the things too. If you have a really rare incident that occurred that is in the rare incident bucket, but it still has open action items and it's been three months, then those are the incidents that really scare me, because maybe it's a sev zero that has open action items that no one actually fixed and there was no follow-up after the postmortem. Then those incidents are ones to talk about again, to make sure that you're okay.

You need to verify that everything has been put in place correctly, to make sure that that doesn't happen again if it was a really big incident and that's where you can use chaos engineering too, to verify that the actions that you took, to make sure that everything was okay, actually worked out well. Then if it's something that's in the bucket of this happens all the time, you definitely need to be reproducing those incidents too, because that's going to help you get rid of them.

I think it's often, people will say, “Should I go for low-hanging fruit, or should I go for the things that are more impactful?” The things that are actually impacting your business. I would say, go for the things that are impacting your business if that's – it's taking up a ton of engineering time,

it's customer pain, it's causing us to lose money, that's how you should be evaluating whether or not you should focus on something and spend time on fixing it.

[0:34:44.0] JM: In the organizations that successfully adopt chaos engineering, how do they find the resources to make time for it? Because most high performing engineering organizations, they're constantly just feeling they're behind on everything. There's so much, so much stuff to do. How do you get the will to prioritize getting chaos engineering going within an organization?

[0:35:13.9] TB: Well, actually now is a great time to start thinking about it, because a lot of people are doing 2020 planning. That's really important. If your company does do yearly planning, then you can say this is something that I think could help us, train up everybody so everyone feels confident when they're on-call, help us reduce outages, help us improve our meantime to detection, meantime to resolution and really put forward the case for why you want to do it, how it's actually going to benefit your company. That's a great thing to say, so to say, "Hey, I want a pitch that we include this in our 2020 planning."

Like I said before, put together a deck and present that to your manager, or your VP, or your CTO. If you can go to open office hours, you can actually get some time with them and you can talk about it. I think that's a great discussion to have. You might also identify – sometimes when you're an engineer in an organization, you see your part of the world, but it's really hard to see other areas. If you chat to your CTO or your VP and say, "Hey, what are the biggest problems that you're noticing across our engineering org?"

They might be seeing – I mean, they probably will be seeing totally different things than what you see, but they also might not know as much detail about what you're working on and the problems that you're being impacted with day-to-day. The amount of people I've spoken to over the last two years who weren't in engineering and didn't know that on-call was a thing that engineers did is very – I was like, "Wow, I can't believe it." People in other teams just don't know that we do on-call. They don't know that we have to go home and get paged and have to resolve incidents and have to not go out on the weekend and have to be doing all these things.

If other departments don't know about that, it's really hard for them to understand that actually, no, I'm spending my entire weekend doing this work and that's going to impact my ability to actually deliver new features. I think we need to have more conversations. We'd say the product teams and the marketing teams and the sales teams and the business side of the company, so they actually understand where our time is going. Often, how we're needing to spend our weekends.

[0:37:20.5] JM: You talked a little bit earlier about the onboarding, getting the onboarding right for people who are trying out an infrastructure product, whether that's an internal tool or an externally-facing tool. What have you found to be keys to make it easier for people to get started with chaos engineering? Or is there one weird trick you've learned over the last year to make it easier for people to get started with this stuff?

[0:37:45.5] TB: Yeah. One of the things I like to say is that for me, the CPU attack is the hello world of chaos engineering. The reason for that is because it's very, very easy to see in your monitoring tooling, I think that that's a good place to start. If you run a CPU attack using Gremlin on a host, or on a container, then you'll be able to actually see that pop up in say, Datadog, or New Relic. SignalFX, whatever monitoring tooling or observability tooling you're using.

That's really great to me, because say for example if you do a shutdown, or a shutdown attack where you shut down a host, or shut down a container, it's less visible and it doesn't show people the actual results as clearly as a CPU attack. Then when you think about CPU attacks, you think like, “Why do I need to think about CPU?” Well, usually that's how auto scaling is configured. It's configured by you get to a certain point in terms of CPU and then say if you're using AWS, it'll kick off your auto scaling rules and then your system will scale. Then that's how you make sure that you're able to service all of your customers’ needs.

Yeah, that's the way that I say to get started is to just first focus on building yourself your own personal demo, so you can just see that all working together. You've got your chaos engineering software, you're able to actually see the result in your monitoring tooling and that should really not take you long to build, but then you're able to show other people and explain it. Then from there, I think the best way to get started is to then focus on what are your top five critical systems, what other systems should you then focus on exploring and then now that we've built

Gremlin scenarios, I'm really excited about the idea of we actually have recommended scenarios.

You can focus on looking at what types of outages or incidents should you reproduce using our recommended scenarios. We have them there for DNS outages, auto-scaling like I said and host failure, known failure. You can go through and actually try each of those out on your own infrastructure and make sure that you're able to handle it. We've built in the ability to record the hypothesis and also your notes and your results and there's a calendar functionality too. You can share those scenarios with your team, you can also create custom scenarios.

I think that's an exciting thing that I've been thinking. I can't wait to see what the community creates in terms of their own custom scenarios and to hear what value they get out of running those.

[SPONSOR MESSAGE]

[0:40:15.2] JM: Cruise is a San Francisco-based company building a fully electric self-driving car service. Building self-driving cars is complex, involving problems up and down the stack, from hardware to software, from navigation to computer vision.

We are at the beginning of the self-driving car industry and Cruise is a leading company in the space. Join the team at Cruise by going to getcruise.com/careers. That's G-E-T-C-R-U-I-S-E.com/careers.

Cruise is a place where you can build on your existing skills while developing new skills and experiences that are pioneering the future of industry. There are opportunities for back-end engineers, front-end developers, machine learning programmers and many more positions. At Cruise, you will be surrounded by talented, driven engineers, all while helping make cities safer and cleaner.

Apply to work at Cruise by going to getcruise.com/careers. That's getcruise.com/careers. Thank you to Cruise for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:41:36.0] JM: The workflow for a scenario is you figure out how to program an incident and then the incident occurs and then you use your monitoring and your logging tools and your tracing tools to figure out what's going on throughout your infrastructure and how your infrastructure is adapting to that. What's the workflow there?

[0:41:56.6] TB: Usually, what I like to say with folks, so I guess it depends on the type of enterprise as well. Often, it starts with I think a good idea is to do a game day, so that's more what the workflow would be.

Often, you have somebody who says, okay, we're going to be able to get value out of doing chaos engineering. Something that we often never do is sit down together in a room, whiteboard out, our architecture and figure out how does this system work, what are the different things we know about this system, what are the gaps that have. Then we actually get together and figure out what scenario makes the most sense for us to run in this game day?

Usually, a game day would run for one to three hours. You might do one scenario, but you might actually go through and do up to three scenarios. That's a really good workflow for getting started, because I think it makes so much more sense to actually get together. Once you've had one person learns how to do chaos engineering, the actual I've installed the agent, I've made sure that it can run, I've made sure that I can see the results in my monitoring tool, then you can go and actually plan out your game day and get started. The best thing is to have four to say, 15 engineers in a room. Doing it together, it's actually very good when you do it as a collaborative exercise. That's a great way to do it.

[0:43:13.9] JM: Let’s talk a little bit more about different kinds of failures. One important distinction that we can make about failures is it's usually a partial failure in my experience, more than a complete failure. When you're talking about systemic failures and partial failures are often harder to diagnose. They're harder to deal with. They may even be harder to reproduce. What's the difference between approaching partial failures and approaching complete failures when you're trying to reproduce incidents?

[0:43:46.3] TB: If it's a complete failure that might be for example your DNS is currently not working. Let's talk about that example for a little bit. Say if you have one DNS provider, say route 53. Then what happens when route 53 isn't working correctly, which that often will happen? It's happened to me before in the past.

I talked to a lot of folks where that happens. Then it's really good practice to have a secondary DNS provider, say for example NS1. What you need to do is make sure that you've configured that correctly. You need to be able to failover from route 53 to an S1 and you need to make sure that you've built that out correctly. You need to also make sure that everyone in your department, or whoever needs to be responsible for that knows how that works.

As you onboard new engineers, you need to make sure that they understand how DNS failover works at your company, especially because this is something that's not very across different companies. If you change companies – I've run, might handle DNS differently, especially because a lot of people don't actually do DNS failover. They might only have one provider. I think that's the example of yeah, sometimes it just won't work and you need to do it. I think a good practice that I did in the past when I was working on magic pocket, the storage system at Dropbox was we would actually run DNS failover exercises every week. Whoever was on call had to do that.

We would purposely inject that failure to make sure that we were ready to be able to handle it, because it is so critical to your system and you need to make sure that DNS is being handled well for everything to operate correctly. If it's a partial failure, there are lots of interesting examples there. I found often, there'll be partial values related to networking.

To me in my career, I've got so much value out of just learning more about networking; understanding networking tools, understanding how the network works at my company, getting to know the network engineers better, understanding what they prioritize, what they're focused on, what changes they're making to the system. Because I've definitely working on a lot of networking outages and those to me yeah, sometimes you'll be like, "Oh, I seem to be having issues, but only for a short amount of time that everything seems fine, then everything's back to being bad again. It's flaky, or just up and down, some problem occurring."

One of the things that I think is valuable there is to actually sit down with the network engineering team and you can do a game day and say look, when you notice some issues, but they don't happen all the time, there's intermittent problems, can we do a game day on network related problems and try and figure out how we can fix it? One of the examples for me that I had as a networking related outage, but it was – it would happen every so often, not all the time and it was really quite hard to solve was because the networking team rolled out QoS, which is quality of service for networking.

It means that they're able to prioritize what traffic they choose. It depends if they've picked your traffic and they're going to give your traffic a lot of priority, or some other traffic in the system. The traffic for me for database, clones was not being prioritized as much. I was having some issues there with networking and database clones, so then that meant I actually had to talk to them and say, “Hey, I'm noticing this problem. I’m not really show what's going on. I've tried to rerun some clones. I've tried to kill clones midway, which is chaos engineering to be able to see what happens when a clone gets killed in the middle and then it has to restart.”

That's a lot of the work that you really need to do when you dive into it, to be able to make improvements and measuring all of those metrics. How long does it take for it to die? How long does it take for it to restart? How long does it take for it to finish? What are the different networking issues you see over time?

Yeah, it ended up that they had been lowering the priority for clones specifically, but then I was able to showcase that data, explain to them when I do these certain types of chaos engineering actions on clones, it actually is a really big negative impact that the traffic's not being prioritized by the network engineering team. They actually did agree to reprioritize it, which was great and meant that we're able to – that's like getting buy-in for me as an example, from the network engineering team, to make sure that I'm able to appropriately service customers and make sure that they're always getting the best experience possible.

Yeah, that's an example there. I would say those outages where it's sometimes it works, sometimes it doesn't work, it seems to not work specifically for this thing, but I'm not really sure. You need to collect a lot more data and then you need to try and figure out who you should work with to move forward and actually make an improvement.

[0:48:29.0] JM: Yeah. You can also have these failures where the failure occurs, you figure out your response, so you respond, maybe you go to a backup and then you have a degraded system, and so you're operating this degraded state for a little bit. Then you recover your system, but you still have a degraded state. You can't get back to a full recovery, because you have made this change to your degraded system. You've made some change to your infrastructure, just like a knee-jerk reaction.

Then you've injured your infrastructure and it becomes difficult to switch back to a normal state. I guess, this is more of a question around incident response, but do you have any suggestions for avoiding getting into these degraded states that are really hard to roll back from?

[0:49:21.6] TB: Yeah, that's a great question. I mean, I think about this so much, because I worked in banking. One of the biggest things in banking is roll back. You need to be able to roll back every single change that goes into production. Really, that's a different way of thinking. Everything that you build has to be built in a way that you could roll it back and it wouldn't be really closely tied to all these other systems, so that you wouldn't be able to actually get it out of production.

You just build your software in a very different way. That's the first thing. I think that's good. I think it's good to build software in a way that you can actually roll back, because when I did start working at startups I was like, “Oh, we should just roll back that change.” It just went out, let's roll it back. Sometimes people would say, “What do you mean roll back? Let's roll forward.” I'm like, “No, no. I don't want a patch in prod. I'm not a fan of patching in prod,” which is what I would call it when you're having an incident and then somebody on the fly says, “I'll just make this extra change.” What happens if that extra change causes more incidents? That's patching in prod. It's often fast if you just take the code that was recently committed and just roll it back.

That's what we did at Dropbox and it was much better to be able to roll back. Always have that state. That's a steady state that you can get back to. Yeah. I mean, I'm not sure why that is still a debate in industry. I'm a big fan of rolling back. The idea of just people having access and writing code and just shipping it without any tests while an incident is happening and there's no time for code review, or they're not going through anything correctly is just really dangerous.

Also, if you were working somewhere like a bank, you just can't do that. You're actually not allowed to do that, because when the regulator's come to review your work, they'll ask why did this incident happen? Why did this incident get worse during the incident? Then they would say, “Oh, this person actually did all these things wrong.” That's definitely not going by the book and that's why the internet got worse. Then you might lose your license and not be able to operate as a business.

That's why I like people who have worked in big enterprises with heavy regulations, have a different type of view of the world. That's definitely impacted my view, but it's work well for me, like working out startups and working enterprise. It's super-fast to roll back if you build your software in a way that you can, so then you can't actually move fast, but not break things in a bad way. You're moving fast, you're doing chaos engineering in a really good way, you're actually thinking through it. It's more doing surgery. You've got all the data upfront, you're doing it in a really thoughtful controlled way, so that you're focused on getting value for customers.

[0:52:01.8] JM: I think I actually saw that point about the degraded systems and the difficulties of that brought up in a talk by Adrian Cockcroft, who he's the industry luminary that people really loved. Really great speaker. He always seems to be a little bit ahead of the curve in terms of, he predicts trends and then they tend to come true because he's worked so long in industry and it's like, he's – he worked at eBay and then worked at Netflix for a really long time and orchestrated their microservices effort. Then now he works at AWS. He's near the top of AWS. He has been talking a lot about chaos engineering recently. Have you had any enlightening conversations with him about chaos engineering? Why he's so interested in it?

[0:52:49.4] TB: Yeah, definitely. Last year, Adrian came and spoke at our Chaos Conference, which was awesome. It was the first conference that we'd held and we ran the second one again recently, which was great. It was really cool to have him do the opening keynote and just talk about how the industry's changed. He talks a lot about why chaos engineering is valuable. It's because systems are actually much more complex now.

Am I seeing a lot more users and then also there's a lot more instance and a lot more outages. We also have the ability and the skills to think about how do I build graceful degradation into my

application. He talks about that a lot. Just being prepared for failure and that's something that everyone at AWS when you speak to them they'll say, “Yeah, you should definitely do chaos engineering.”

Every single person that works there, because they know that sometimes a customer might say, “Oh, I just expected that it would work all the time,” but their computer systems, their machines, machines turn off, machines have to be rebooted, machines have to have security patches applied. There's a lot of things that need to happen, so that's why they're all such big fans. They've seen tons of some of the biggest outages that have occurred across the world and they really want to help everybody prepare for it, so they just really like to focus on that idea of preparing for failure. Yeah, Adrian's awesome.

[0:54:09.0] JM: Oh, he sure is. Yeah. He's super entertaining too. I don't know, he has this way of speaking with a very soft cadence while saying things that are extremely insightful, or in some cases, alarming. It just has this very straight cadence. I don't know. I watch him for rhetorical tips almost. Yeah, so what does the next five years for Gremlin look like?

[0:54:35.5] TB: Yeah. Now that we've released scenarios, I think the next for the rest of this year, I'm really looking forward to seeing how everybody uses that to be able to reproduce incidents and focus on reducing MTDD, reducing MTTR, reducing repeat incidents and being better prepared for outages, feeling they're confident. If an outage happens, I know how to handle it. I think it also helps people work together in a team as engineers, which is great too.

Then we've got some other things that are coming up, but I guess I'm not allowed to share those types of things when you have secret – really cool secret things that you're working on. I can't share too much there, but we've got some really great new features and products that we're working on that are coming out, which is great.

As an industry, what I would love to see is just more folks airing their failure stories, but in a way where it's this is what happened, this is what we've learned about it and this is what we did to actually improve our system. These are the actions that we talked and this is then how it helped us, because I think that's what we really need to do, like not just say this is the failure that

occurred, but what did you do after that failure occurred? What were the action items that you took? Which ones were the most valuable to you? Was it technical? Was it people related?

As an example of it, people related to change you might make. You might say, let's prioritize on-call training as a technical change that you might make. It might be let's fix our auto-scaling rules. I'd really love to just hear more stories there. We also have a chaos engineering Slack, which everyone can join. It's really easy to get do. Just if you go to gremlin.com/slack and it's just a public open community of about three and a half thousand engineers who are in there talking about chaos engineering, incident management, SRE, which is good too. All great things to hear about.

Then the other thing is I'm the co-chair for SRECon West America in 2020, which is going to be in March in the Bay Area. We're going to be focusing a lot on deep work. The idea that it's really important for SREs to be able to focus on deep work and making the time to be able to figure out how does my system really work? What happens when failure occurs? Also, just being able to identify how do you get the time for that deep work? How do you avoid distractions and how do you get buy-in from your organization that deep work is important?

Then we also just want everyone to come together and share what they've learned, because the thing is deep work takes time. If you want to become an expert in a system, then it takes a lot of time. The cool thing is you can then come out of that period of time where you'd really dive deep and learn a ton and you can share it with everyone else. I think that's how we all grow as a community and we move forward.

I saw a really cool talk at SRECon early this year, which was all about logging. It was awesome. It was the best logging talk I've seen in so long. It was really cool, because it was just a super deep dive on logging and I loved it. It was someone who'd been working on just specifically logging actually for 20 years. You can imagine how much they have to teach everybody else. I think that that's great.

I hope over the next five years we actually do that more. We have engineers come together to share what's worked, what hasn't worked, when something went bad, what they did to fix it, how they've then improved over time, that would be really great.

The other thing that I'm seeing which is interesting too is a lot of folks coming straight out of college and going into SRE, which I think is really exciting and that just shows you that there's this big shift. I think it shows that when students are studying at college, they realize that reliability is number one. It's really important. A lot of people say reliability is feature zero. I think seeing that trend from engineers going, “No, they're at college. They're learning about how to build systems,” and then they go, “The first job I want to have is I want to be an SRE and I want to focus on reliability and durability, because that's important.” That's what people really want to have on the Internet these days, when everyone's so connected. I think that's very cool change.

That never happened before. Usually, I'd only met SREs who'd been in industry for I don't know, maybe six to 10 years and then they changed to become an SRE. The other thing that's happening too, which is interesting is say self-driving cars, that's very interesting in terms of chaos engineering. I saw there was self-driving prams recently to carry and transport your kids. I’m like, wow, that's a whole different world.

[0:59:03.8] JM: Sorry, self-driving what?

[0:59:05.1] TB: Prams. A stroller for children, for babies.

[0:59:08.7] JM: Oh, no.

[0:59:09.1] TB: I know. I'm like, “Wow, definitely.” As an engineer, you would want to be making sure that was very reliable. Yeah, I just think reliability is –

[0:59:18.2] JM: I hope that's not made by Boeing, is it?

[0:59:20.4] TB: No, it's not. But oh, my gosh. I think I tweeted it and somebody wrote back and said, “I would not want to be the engineer building that software,” because –

[0:59:28.0] JM: I would not either.

[0:59:29.0] TB: Yes. It's a tough thing. That's a big responsibility. Yeah.

[0:59:33.3] JM: Tammy, thanks for coming on the show. It's been great talking to you once again.

[0:59:36.3] TB: Great to speak with you as well, Jeff. Thanks for having me.

[END OF INTERVIEW]

[0:59:48.1] JM: As a programmer, you think in objects. With MongoDB, so does your database. MongoDB is the most popular document-based database built for modern application developers in the cloud era.

Millions of developers use MongoDB to power the world's most innovative products and services, from cryptocurrency to online gaming, IoT and more. Try out MongoDB today with Atlas, the global cloud database service that runs on AWS, Azure and Google Cloud. Configure, deploy and connect to your database in just a few minutes.
Check it out at MongoDB.com/Atlas. That's MongoDB.com/Atlas. Thank you to MongoDB for being a sponsor of Software Engineering Daily.

[END]