Deloitte.

Architecting the Cloud, part of the On Cloud Podcast

Mike Kavis, Managing Director, Deloitte Consulting LLP

Title:	**Chaos engineering: stress testing the cloud**
Description:	As cloud architectures have grown more complex and distributed, failures and outages have become an all-too-frequent occurrence—which can erode executive confidence in the value of cloud computing. However, architecting the cloud is vastly different from architecting traditional, monolithic systems, and it's maddeningly difficult to predict, and plan for, all the possibilities for failures. In this episode of the podcast, Mike Kavis and guest, Kolton Andrus of Gremlin, discuss the emerging discipline of chaos engineering and how it's used to stress-test cloud-based systems in production—which can increase uptime, reduce risk, and accelerate time-to-value of cloud-deployments.
Duration:	**00:26:51**

Operator:
The views, thoughts, and opinions expressed by speakers or guests on this podcast belong solely to them, and do not necessarily reflect those of the hosts, the moderators, or Deloitte. Welcome to Architecting the Cloud, part of the On Cloud Podcast, where we get real about Cloud Technology: what works, what doesn't and why. Now here is your host Mike Kavis.

Mike Kavis:
Hey, everyone. Welcome back to Architecting the Cloud Podcast, where we get real about cloud technology. We discuss what's new in the cloud, how to use it, why we want to use it. We talk with people in the field actually doing the work. And speaking of people doing the work, I'm here today with Kolton Andrus, the CEO and founder of Gremlin, the world's first failure-as-a-service platform, helping companies avoid outages and build more resilient systems. I should probably introduce myself. I'm Mike Kavis, your host and cloud architect over at Deloitte. So, Kolton, welcome aboard and tell us a little bit about yourself, your background and why you guys started Gremlin.

Kolton Andrus:
Thanks, and thanks for the opportunity to be here. Yeah, my background, I started as an engineer on the Amazon retail website availability team, and it was our job to make sure Amazon was up and that people could buy things. And when I arrived, we were doing what I would call your classic SRE approach.

After things had broken, how quickly could we diagnose them and fix them and get things back up and running? Amazon has high goals and high standards, and, so, to meet those high availability goals, to get into the 4-9s world, we realized we had to prevent outages from ever occurring. And that was really the genesis of this idea that's now known as chaos engineering. Ten years ago, it was just good reliability practice.

So, we had a lot of success there. I built tooling. We had hundreds of teams use this approach at Amazon. DynamoDB did some important testing around network partitions and found and fixed some important bugs before they launched. After my tour of duty was up at Amazon, I went and joined a company called Netflix. They had been dabbling in this space and spoken about it publicly in some blog posts around Chaos Monkey, and, so, I thought, great, this'll be an awesome opportunity to learn and go deeper. And when I arrived, what I found was they had a great cultural approach, but their tools were young. And, so, I had an opportunity to build a new platform for them, something that was a little safer, a little bit more precise.

And we went through the same journey that we went at Amazon where we went and worked with teams, we helped them do this testing and uncover things, and, ultimately, we got another nine of reliability, and me and the teams underneath me were paged 25 percent less year over year. So, having seen this approach be needed and necessary at Amazon and Netflix and provide a lot of value, we felt like this is something the industry as a whole is going to need as they move to the cloud, as they move to microservices, as they embrace containers. And, so, we founded Gremlin four years ago to build the best failure-as-a-service chaos engineering toolkit possible.

Mike Kavis:
That's an awesome story. I didn't realize your background from there, so I appreciate all the nines you provided. I appreciate the work that goes into that. I think what's interesting here is, as companies are starting to get pretty mature in the cloud, from, probably not on an operations standpoint, but definitely from the building, and it's a lot of the net-new apps. A lot of companies aren't used to distributed architectures for many, many things that can fail. So, what I find is, they move to the cloud and all of a sudden, they get a lot of reliability issues and then the leadership in the company start poo-pooing the cloud, but it's not really a cloud problem. It's the fact that we need a different approach because the architecture's different, and hence, you know, failure-as-a-service. So, describe to us exactly what you mean when you say failure-as-a-service and why is it so important now to test in production, which a lot of these experiments are in these days.

Kolton Andrus:
Yeah, I totally agree with your observations. People have moved to the cloud, but they're learning how to operate in the cloud. And you hit on a key point there. We're building distributed systems. At the same time, we're moving to the cloud, we're embracing these more distributed approaches. Well, we've just thrown the network in-between everything, and one of the great programming fallacies is that the network is reliable. And, so, that forces us to think differently about how we handle failure. I'm a big fan of microservices and a lot of the approaches we've taken, but it comes at a cost. It comes with these trade-offs.

And, so, Gremlin's failure-as-a-service, you know, we're about building the tools that make it easier for people to reproduce real world scenarios. So, for example, a couple years back, there was a big S3 outage, and a lot of people found out that they had a dependency on S3, even if they were putting config, or things that they thought were non-critical, a lot of the internet was hit that day. Well, that's a classic example of something you need to test for in a distributed system. You have an important third-party service that you're making network calls that your system relies on. What happens if that service gets slow? What happens if that service fails? And, so, Gremlin, we're not going to go ask Amazon if they can break S3 for everybody. That's not a very scalable solution.

And, so, we have tooling, we have these agents, these gremlins you install on your host, and they're able to safely and securely cause these kind of failures. So, for example, we can drop all traffic to S3 to simulate an outage from them and understand the knock-on effect to our system. But we have a wide variety of failure modes. There are things like, you know, what about our resource constraints? What if our CPU is pegged? What if there's a memory leak? What if a disk fills up? Disks filling up is one of my favorite ones to pick on because it sounds like the most simple failure, and yet every company I've worked for has had a public-facing outage relating to a disk filling up.

What happens when time changes on a box, whether that's a certificate expires or daylight savings? There's some really tricky scenarios that happen seldomly, and I've certainly seen a lot of outages on those fronts. You know, you want to test things like your database failover, or your Kafka primary failover, and ensure that that process works correctly. You want to test things like region evacuation or zone evacuation to ensure – or data center evacuation. We built a tool that's focused on the cloud, but it works on the bare metal, and, so, at Gremlin, regardless of where people are at, we want to meet them there. We want to give them the tools to test the failures that are going to happen at scale in their environment.

Mike Kavis:
Yeah, the whole cert expiring, I had a horrible experience with that. So, before my consulting days, I left the corporate world, did a startup back in 2007-08-ish or something. And we got bought like in 2011, what happened is they had a different cert company they used than us, and being from the applications side, I never really thought about what happens if a cert expires. So, a cert expires and of course it took us hours to figure out it was the cert, but then we had it – because they switched companies, the cert provider changed. So, meanwhile the system's down, the UI's down and I'm like, "Oh my God!" So, then we went and made sure that would never happen again. My question to you is, you know, a lot of these experiments are for things that we know. How do we do tests for things that we don't know are going to fail?

Kolton Andrus:
Yeah, and that's one of the interesting parts of this space and of chaos engineering because there's a class of pretty well-known outage scenarios that a lot of people experience. And for people beginning and just getting started, they should start with those scenarios. They should test those and make sure they're able to handle them. But what about the exploratory things? What about the things that might be seldom, but when they happen could have a huge impact? And the truth is, my belief is you want engineers to have the tools to answer those questions in advance. You want to make it easy for people to do the right thing so that it's not a huge time commitment. One of the things we see a lot is kind of the contention between product and reliability. Everyone wants to ship new features and get things out the door. Product managers are always about deadlines and making sure things get shipped. But you know, if we ship it and it breaks, it's kind of all for naught. And, so, if we get engineers thinking about this earlier and earlier and testing these things in advance, one, they're going to architect better software. They're going to have thought through some of these edge cases early on when they can make an impact, when they can change their design decision. And then they're going to be curious and they're going to know that when they have a question like this, they can go

answer it. And they can answer it in five or ten minutes, as opposed to a day or two. And at that point, you can start knocking things out much quicker and get more comfortable with your system and how it fits.

Mike Kavis:
I do a lot of work with highly regulated companies like banks, and just talking about causing failures in production, can't even have that conversation. So, what I see is a lot of people start kind of talking maturity curves, but they start doing these things in test, start proving the value there. What approaches do you see or how do you approach customers who, you know, the developers, the operators, they get it, but management's like, "You're not breaking things in production." How do you get over that hurdle?

Kolton Andrus:
Yeah, I think, you know, first of all, it's perfectly fine to test in dev or staging. That's a great place to start. What I would say is, this approach is about mitigating risk. We want to take a small risk to prevent a much larger risk, and, so, we're always weighing that tradeoff. So, for example, if we can begin by testing in staging and find something that would have been a wide-scale production outage, we've put none of our customers at risk and we've prevented a large impact. That's a huge win. But there are ways that when we move to production, or when we move to shared environments, we can mitigate that risk with this concept of the blast radius. So, whenever you're going to run an experiment, you want to ask yourself, "What could the side effects be, what could the knock-on effects be?" And you want to be measuring those and be thoughtful of those. And then the first experiment you want – you run, you want to run the smallest experiment that will teach you something. So, if you can go into production and run against a single host or, a single user and learn and find things, then you're not putting most of your customers at risk and you're providing that value.

Now, it's valuable to test both the small scale and the large scale. At the small scale, you're testing, "Did we handle this exception, did we handle this edge case, does our code do the right thing?" It's a little bit more like unit test. But when you're testing the large scale, you need to test, "Do I back off of a downstream system when they're underwater? Do I shed traffic when I start to get underwater? Are my timeouts and my thread pools, my security grids, are those things configured correctly so that when things go awry the right behavior happens?" It's funny you mention the banks and the financial institutions. While they are heavily regulated, they have a huge incentive to do this well. The industries where there's a very clear correlation to lost revenue and downtime are some of my best and earliest customers, and e-commerce and finance are high on that list. And what I'll say is I've been impressed by the financial institutions and their forward-leaning outlook in this area. Many of them are doing testing in production and many of them are investing heavily in these efforts. And I'm super glad to see that.

Mike Kavis:
I saw the same thing from cloud adoption when public cloud was king and private was emerging, we kept saying the banks will be the last ones, and they were the first ones. And it wasn't easy, and there's still a lot of political battles internally, but they're kind of blazing the trail on a lot of spaces in cloud. So, I see that. You talked about simulating the user. My question is – this would have been helpful for me way back in the day – is you know, we had users simulated in production running all the time so we could catch production failures. Could I turn off S3 for a user?

Kolton Andrus:
Yeah, so that's one of the exciting things. So, at Amazon, I built this infrastructure-focused product that was around hosts failing, around disks filling up, around traffic being lost. At Netflix, what I built was something called application-level fault injection. So, imagine taking a function and adding a cross-cutting concern, an aspect, an annotation and then being able to inject failure or delay into that function. Now, what's neat is often companies have user IDs, device IDs, that meta information about the request. And, so, we built it in such a way that we could – that was the precision, that we could target a single user. In fact, whenever we did this testing and production, the first user I tested was Kolton@Netflix, and I would see if it failed or handled it well. And if that worked well, then I was able to put customers at risk. And then we would move to, yeah, maybe it's Xbox customers, maybe it's 0.1 percent of production traffic. And that ability to have that fine-grain control allowed us to dial it up at whatever increment we wanted from zero to a hundred.

Mike Kavis:
That's why I like doing podcasts. I just learned something. So, that's awesome because we had these – I guess they call them synthetic or syndicated users. I think they're called synthetic tests, right, where we would – our business was digital coupons, so we would have fake users out there and we were basically testing all the APIs all day long, but we had to write a lot of code to do that. This is a great feature. So, for companies who are afraid of testing in production, if you could shrink the scope to testing artificial IDs in production, I think that can go a long way to getting them as their first step for doing stuff in production.

Kolton Andrus:
You know, we've thought a lot about, again, how do we make it easy for people to do the right things. And yeah, there's concerns about doing this kind of testing and production around safety, around the side effects, around security. And this is one place where I'm going to brag a little bit and I'm going to push on open source. All the open source projects out there in this space don't really have a concept of safety. They don't really have an undo or a cleanup that, you know, "Oh crap" button, to say it nicely. And, so, you know, we built that in from day one, and that's a key part of things. You have a core metric that fires, you know, maybe orders per minute, we can halt that attack based on t hat so that we're not endangering customers. Similarly, security's immensely important to us. You know, we actually just finished our SOC 2 Type 2, which is a little ahead of the curve for a company our size. We do intrusion detection and pen testing and all of this work to invest in security to make sure that this testing can be safe and secure, and I'm proud that it meets the standards of these large public companies and these financial institutions with stringent requirements.

Mike Kavis:
That's the only way to get in those institutions, so yeah, that's definitely a business-driven feature there. Yeah, I get that. So, what do you think are some of the biggest misperceptions about chaos engineering or testing?

Kolton Andrus:
That's a great question. The first one is that it has to be done randomly. People think that the chaos part of chaos engineering means that you must randomly break things. I think there's a time and a place to randomly break things. Netflix used that as a social influence where it could happen to you for real, and, so, they were going to make it happen to you more often, you know, in the simulated environment so that you had to prepare. But if you think about all the types of failures in testing and production, you probably don't want to be randomly dropping network traffic or randomly turning off databases

in your production environment. It mitigates kind of that safety and it increases risk. And, so, to me, I think we'd almost be better served if we flipped the words around. Really what we're doing is we're engineering for chaos. We're taking the engineering discipline that we've developed, and we're applying it to tame that chaos that's inherent in our system. And the outcome is we would love a boring predictable system that just works.

Mike Kavis:
Yeah, I like that, flipping that around. And is this something that could be tied into the CI/CD process so like checking code, it builds, it deploys, and it runs a bunch of chaos tests?

Kolton Andrus:
Yes, and that's actually where our more mature customers find a lot of value. In the beginning, you're doing what I would call exploratory testing. You're understanding the system, maybe you're reproducing some of the recent outages you had, you're getting a grasp on things and getting in control of it. But once you understand it and you understand what the right behavior is, you want to write a set of tests that verify and validate that. And, ideally, they're running in your deployment pipeline, and if something has changed and you've introduced a new risk or vulnerability, you're halting a deploy on that until it's fixed.

The other thing I'll mention here, you know, I think QA has changed a lot in the last ten or fifteen years. Just as we've seen this move from kind of ops being a throw-it-over-the-wall function to being integrated into development, you know, that's a lot of what the CI movement was, that engineers are running unit tests and it's part of the process earlier. Well, similarly, you know, unit tests do a great job testing what I would call build-time validation, you know, these unit tests and the things that we know for that package and that bit of code. But I think there's a huge gap when it comes to run-time validation. And again, it's these things around timeouts, around thread pools, around security groups, around, you know, graceful degradation. And, so, the ability to go out and really understand those helps us, again, not just to get in front of the curve, but to make sure that the code that we're operating does what we expect.

Mike Kavis:
Yeah, I actually had my first tester on the podcast a couple weeks ago, Angie Jones. We talked about that, because there's a misperception that testers go away because developers are automating test scripts, but as you said, that's kind of the unit test, so where have you seen most of the work on the chaos experiments? Has it been with testing groups? Has it been with SRE? Is it engineering? Who's doing these tests?

Kolton Andrus:
Yeah, I think the early adopters are the people feeling the pain. So, they're the SREs or the ops folks that get paged. Now, at Amazon and Netflix, that was one and the same with the engineers, and, so, I think as we see this trend toward you build it, you own it, you operate it, you will see more and more engineers that have skin in the game, that feel this pain, and that care about a solution here. But I think it's broader than that. Certain failures can be handled at the infrastructure level; certain failures can be handled at the platform level. But there's a whole set of failures that only the application can handle. You know, I could make a call to two different services—maybe a recommendation service and an identity service, or maybe a recommendation service, a digital rights management service. If that recommendation service fails, you know, maybe I can serve some cash results. I've got some fallback. You know, it might not be that big of a deal. Let's make sure the customer can do what they want. But if the digital rights management call fails, you know, we may be legally obligated to fail that request and to not serve that content. Now, from a technical perspective, those are both – you know, those both may look the same, but from a business perspective, that's one of the subtleties that I think is important to call out.

Mike Kavis:
I think this all just drives how importance of DevOps and closer collaboration because how can you design the right test if you don't know the business, right? If you don't know the architecture, the application, the business? And it's just another reason why we just need to continue shifting all this domain expertise left so the product team consisting of a product engineer and have all the right skill sets as opposed to our legacy, this silo, that silo. I can't imagine this working well if we stand up a chaos team next to the ops team next to the network team, next to the security team. I just can't imagine this being designed well. I don't know what your thoughts are.

Kolton Andrus:
I agree with you 100 percent. I'm a big fan of The Phoenix Project as a book, and it highlights the pain. And they even speak to chaos engineering being part of a solution in there to help people get through this. But to me, yeah, the engineers need that business context. They need to understand what's important to the business and what the goals we're trying to accomplish are. I've been proud. Our Director of Product has been giving a couple of talks recently from the perspective of product management about why this should be a first-class feature and it should be given the proper respect. If you're designing your features to gracefully degrade and to handle failure, you're building a much more robust product with a much better customer experience.

And I think that's probably the next set of battles or persuasion that needs to happen. We have the engineers, the ops folks, and the testers onboard because they see the value and they feel the pain. But a lot of my work is how do we get the product owners and the directors and the leadership to recognize the value? And it's there. It's about saving teams time; it's about the not getting woken up; it's about helping their teams get more done. But in a world where everything's broken, everything's on fire, and there's not enough time, sometimes it's hard to paint the picture for people that if you can just fix one small thing a day, you'll dig yourself out and be in a much better spot.

Mike Kavis:
Yeah, that's an area I harp on a lot. I try to convince people that DevOps is more than dev and ops, and you really have to change everything in the value chain, including product management. So, if product management isn't part of the DevOps solution, then they're going to continually prioritize only new stuff, and no matter how great you get at collaborating between dev and ops and security, you still never get time to work on old stuff. So, we also need to rethink product and product needs to think – product owners need to share the responsibility—the "ilities" of uptime, liability, scalability. They own a prize. Like if you own a car, you don't have to not worry about security, right. You don't have to not worry about safety. You own it all. Software should start thinking. I don't know if you hear anyone else talking like that.

Kolton Andrus:
You're preaching to the choir here. I agree 100 percent. Well, look, the analogy I'd draw there for the product people, you know, it's kind of like your own personal health. You know, that may not be the biggest priority in your life. Oftentimes we're all struggling to work out, or to eat healthy, or to improve

ourselves, but sometimes it's just it falls to the wayside until something major happens, until you have some health event in your life, and then that is the only thing that matters, and it is the number one priority for you. Well, it's the same with our software products. All the features in the world are the most exciting things until they get to production and don't work. And then none of them matter. The only thing that matters is if the system can be available and serve customers. And, so, just to re-emphasize, I totally agree. This is a first-class problem and it's something that the entire company should rally around and care about.

Mike Kavis:
Great conversation, yeah. Great conversation. I wish we could talk for hours on this we'll have to have a beer someplace and talk in more detail. But where can we find your guys' content? I know you have a bunch of folks I follow, are speaking all over the place. I know you have a blog out there. Where can we find all this content on kind of emerging topic that a lot of people know, but I think when I talk to enterprises, only at the fringes?

Kolton Andrus:
Yeah, and that's, you know, a lot of our mission. Our mission at Gremlin is to help build a more reliable internet, and one of the ways that we're trying to accomplish that is by teaching people what we've learned, helping them understand the pain we've felt and really the hard-fought practitioner knowledge of how to solve this. And, so, join our community, so gremlin.com/community. First of all, there's a huge set of tutorials and articles, getting started, delves into some of the SRE world and the cultural aspects of influence. We have a public Slack. It's focused on chaos engineering, not on Gremlin, and anyone who wants to learn more is welcome to join us there. I believe it's gremlin.com/slack. And then we're hosting a conference at the end of this month. It's our second year doing Chaos Com. We sold out last year. The feedback was amazing. We held it at the Alamo Draft House in San Francisco.

This year we have the Regency ballroom. It's more than doubling in size. We've got a lot of big name sponsors and speakers that are coming, and we're quite excited. One of the things we want to do is how do we effectively teach people, and, so, these boot camps, these conferences, these talks, you know, focused on the practice, focused on the value. Last year's Chaos Com was kind of focused on why would you do this, and this year we're focused on how. And, so, we're trying to get as many practitioners that are solving this problem for real in the room to share their pain and to share what they've learned so that those newcomers and those that are interested can save a little bit of that pain and accelerate their own journey.

Mike Kavis:
Oh, I wish I could be there. It's tough for us East-Coasters. But I'll catch one of these eventually. So that's our episode of Architecting the Cloud today with Kolton Andrus. Kolton, what's your Twitter handle, by the way, so people can find you?

Kolton Andrus:
@KoltonAndrus, K-O-L-T-O-N-A-N-D-R-U-S.

Mike Kavis:
All right find him there, and I'm at @MadGreek65. Find me there and go to www.deloittecloudpodcast.com where you can find this podcast and others from my colleague, David Linthicum, who also podcasts on here. Just go to Deloitte On Cloud Podcast on iTunes or wherever you get your podcasts. I'm your host, Mike Kavis. Great talking to you today. We'll see you next time on Architecting the Cloud.

Operator:
Thank you for listening to Architecting the Cloud, part of the On Cloud Podcast with Mike Kavis. Connect with Mike on Twitter, LinkedIn and visit the Deloitte On Cloud blog at www.deloitte.com/us/deloitte-on-cloud-blog. Be sure to rate and review the show on your favorite podcast app.

Visit the On Cloud library

www.deloitte.com/us/cloud-podcast

About Deloitte

The views, thoughts, and opinions expressed by speakers or guests on this podcast belong solely to them and do not necessarily reflect those of the hosts, the moderators, or Deloitte.

As used in this podcast, "Deloitte" means Deloitte Consulting LLP, a subsidiary of Deloitte LLP. Please see www.deloitte.com/us/about for a detailed description of our legal structure. Certain services may not be available to attest clients under the rules and regulations of public accounting.
Please see www.deloitte.com/about to learn more about our global network of member firms. Copyright © 2019 Deloitte Development LLC. All rights reserved.