Architecting the Cloud, part of the On Cloud Podcast

Mike Kavis, Managing Director, Deloitte Consulting LLP

Title:	**Culture change, not tech, is the secret to DevOps success**
Description:	Everyone wants to do DevOps and SRE, but often companies hear the buzzwords and start down the path to implementation before they understand what they're doing and how to do it right. In this episode, Mike Kavis and guest, Truist Bank's Sanjeev Sharma—author of *The DevOps Adoption Playbook*—discuss how to implement DevOps and build reliability and resiliency into both infrastructure and architecture. Hint: it's not solely about the tech. Instead, it's about process, platforms and environment, and—most importantly—culture. It's about building a culture of joint responsibility based on value and outcomes. In fact, Sanjeev's opinion is that changing the organization's culture—as well as how you go about it—is a critical key to DevOps success.
Duration:	**00:34:33**

Operator
This podcast is produced by Deloitte. The views and opinions expressed by podcast speakers and guests are solely their own and do not reflect the opinions of Deloitte. This podcast provides general information only and is not intended to constitute advice or services of any kind. For additional information about Deloitte, go to Deloitte.com/about. 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, and welcome back to Architecting the Cloud Podcast, where we get real about cloud technology. We discuss all the hot topics around cloud computing, but most importantly with the people in the field who do the work. I'm Mike Kavis, your host and chief cloud architect over at Deloitte. Today I'm joined by Sanjeev Sharma, head of platform engineering. I've known you for many, many years, way back to the Javits Center and some of those old conferences we used to do, but great to have you on the podcast. Why don't you tell us a little bit about your background, where you come from, where you're at, and we'll get right into our topics today, which is DevOps and SRE.

Sanjeev Sharma:
Well, thanks, Mike. Thanks for having me and it's always wonderful to talk to you. And, yes, we could spend a lot of time talking about all those cloud events we've been to and reminisce about the time when we could actually go to events and meet people in person. So, hopefully that'll come soon again.

So, as you mentioned, Mike, I am the head of platform engineering at Truist. For those of you not familiar with Truist, Truist is the new bank that was created just over a year ago by the merger of BB&T and SunTrust. So, right now, Truist is the sixth-largest bank in the United States, which not very many people have heard of, but once the merger completes, I'm sure a lot of people will hear about it.

So, I'm new to Truist. I joined Truist just around six months ago, so not even a year here, and my background has been for the past decade in the whole software delivery, DevOps, and cloud engineering space. I spent 15 years at IBM in roles along those spaces. I was the first client-facing CTO for DevOps at IBM and I led the cloud architecture practice at IBM for the IBM cloud and public cloud offerings. And after that I worked for a company named Delphix, which was in the – the short form way to describe it would be the DevOps for data space. And now here I am at Truist and I'm responsible for building the next generation platform both for our on-premises environments and also for our public cloud environments.

Mike Kavis:
Cool. Sounds like a fun role. And what you didn't mention, which is one of the topics we're going to talk about, is you've got a pretty good book out there called *The DevOps Adoption Playbook*, first, before I get into that, so what drove you to writing that book? And tell us about that experience.

Sanjeev Sharma:
Well it's a lot of work that went into that book, right? I had written a book before. I wrote the original *DevOps for Dummies* not the commercial version which is available today; that's written by another author. But I wrote the original – this was a free handout which IBM allows you to download from their website. It's like around 80 pages introducing DevOps. I wrote that back in 2013, so I'd kind of gotten the bug of writing a book, and I didn't want to be known for just a *Dummies* book so I thought I would write a full-size book. And as I was working as a DevOps CTO at IBM, I had actually had the opportunity to work with some of the largest companies in the world and some of the most innovative companies in the world, which as IBM clients were adopting DevOps.

This was still the early days of DevOps. Some were very immature, some were mature, and I was trying to document all the lessons learned so I could take them to the next client. And this book essentially became a culmination, kind of a collection of the battle stories, right? And I called it a playbook because I structured it almost like a coach's playbook which a coach of a sports team would figure out, right, here are all the plays my team is ready for based on where I am in the game, who am I playing, which of my players are healthy and on the ground right now, on the field right now, and what the weather conditions are like. Which play shall I run now? So, it's a collection of plays which different organizations run, and I'm quite frankly doing some of that right now in my new role and looking at, okay, this is what's applicable to us where we are, and this is what's not applicable for us or for this particular team because they're not mature yet, right? They need to go through the various steps. So, it was kind of my documentation of everything I'd learned while I was doing the DevOps role at IBM.

Mike Kavis:
And little did you know it'd be the playbook you'd be using in the future. *[Laughter]*

Sanjeev Sharma:
Yes!

Mike Kavis:
You brought your own playbook with you. That's cool.

Sanjeev Sharma:
Yes, and fortunately, it wasn't based on just my opinion. Of course, every book is the author's opinion, is shaded with the author's opinion, but it was based on real-world experiences. And I tried to keep it as technology-agnostic as possible. So, although it's been three years, four years almost since I wrote it or started writing it, the lessons still apply, right, because there's very little reference to particular technologies which would make it obsolete. So, I tried to keep it that way, so even today somebody reads it they're not going, "Wait a minute. Everything he's talking about here is obsolete." Given the speed of change we have in technology, that happens pretty often.

Mike Kavis:
Yeah, and that's a good segue into my first question, because you kind of divided it up, I don't know, for playbooks or categories, but for adopting DevOps there's four areas, right? There's process improvement, there's tools, there's platforms and environments, and there's culture. And as you were talking it's technology-agnostic, I think one of the antipatterns is a lot of companies think of it as technology, as CI/CD, as Jenkins, as tools. And as you say in the book here, that's just one of four areas you need to focus on. So, give us some examples of areas of improvement to focus on in each area. So, there's process improvement. I'm going to skip tools and we're going to go to platform and environments and culture. Just give us some of the kind of playbook things you wrote about in those areas.

Sanjeev Sharma:
I think – you know, and you're spot on there, Mike, where you talk about that distraction people have from technology – by technology, rather, where it becomes – and I recently was talking to somebody and we started talking about their DevOps adoption. And immediately they started talking about, "Well, we adopted this tool and we adopted that tool." And I'm going, "Wait a minute. Wait a minute. Time out, right? You're missing the whole point here, right? Technology is not going to solve it if you haven't fixed the process, and process is not going to solve it if you haven't fixed the culture." And one of the things I talk about in the book is that the biggest challenge to overcome when you're adopting DevOps is cultural inertia, the pushback – and how do you identify cultural inertia? If somebody ever comes to you and says, when you're talking to them about change, about transformation – and this is not specific to DevOps; this is specific to any kind of transformation.

If you hear people say things like, "Oh, but it's not broken, right? Why do we need to fix it?" Or "That part is not my responsibility; you need to go talk to X." Or, "I'm waiting for the other project to change how they operate because until they change, I cannot change." All those are examples of cultural inertia, where the people who are in leadership positions or people who are responsible for the change are not taking the responsibility because they quite frankly don't want to change. They want change to be non-disruptive. They want change to be somebody else's problem. And that to me is the biggest challenge we need to work on, and it requires all three things to happen simultaneously.

Another antipattern I've seen is people saying, "Oh, we are doing this undertaking, this whole effort to fix our processes, and then we'll go build a platform to support those processes." And they're ignoring culture, and they're quite frankly ignoring platform also. And – or vice-versa, where they start building the platform first without fixing culture, or – you know, and nobody tries to fix culture before fixing the other things anyway. Culture is the last thing people try to address.

So, I think the trick, the secret sauce, so to speak, is to figure out a way, and this will vary from team to team, from organization to organization because every culture is different. Every team's maturity is different. Figure out how to do all things at the same time, how to transform the culture and build a culture of joint ownership, right, how to build a culture where everybody thinks and agrees that change is their responsibility, and they should be measured on that impact on change rather than be measured on only what they are responsible for, the same way. So, there are several models there.

One of the things I've been spending a lot of time studying which is kind of what I'm promoting, somebody else's book while you're having me talk about my book, *[laughter]* is the book called *Team Topologies* by [Matthew] Skelton and [Manuel] Pais. And I'm probably butchering their names

Sanjeev Sharma:
But they do a great job about how to fix culture by focusing on team structure, because they go hand in hand. And the boundaries which are created by the way our teams are structured today, whether they're around functional silos or organizational silos or managerial hierarchical silos, they need to be broken first before anything else will have an impact.

Mike Kavis:
Yeah, and one of the biggest challenges with culture change is big bang is hard, right? So, have you seen examples whether it's a product or a business unit, companies just focused on changing the culture of that, like, micro-area? Or is it always bigger than that, where we need to change the whole company's culture?

Sanjeev Sharma:
So, I think you have to do both. You will only have impact in small pockets, but if that impact in small pockets does not occur as a part of a larger transformation, everybody looks at it as an outlier, okay? Let me explain. If there isn't a larger project or larger effort – I shouldn't say project because we don't want culture change to be a project which has a beginning and an ending. It's an ongoing thing. Unless there's a larger effort being sponsored by senior leadership to change the organization's culture, how people operate, how people communicate and collaborate, how people are measured, how success is measured, unless you have that going on in parallel and then have a few pilots with smaller teams or strategic projects to show the success of a new platform or adopting DevOps or adopting even something tactical like CI/CD, the rest of the organization just becomes an observer who's looking at this one team succeed and saying, "Oh, they're outliers. They flew under the radar, or they were given cover by the executives so they could succeed. We can't do that. We are not in the same space."

But if it is a part of the larger transformation, people say, "Okay, we are supposed to look at them, learn from them, learn from their successes, learn from their stumbles, and then figure out how we can implement the change which they are experiencing and they are succeeding with." That connective tissue is to me missing in most larger organizations.

And you asked for an example so let me give you one, right? One organization which I worked with, and I'll stay away from names – they had that, right? They had the pilot project, but the executive who was leading the broader transformation, right, he made sure that the pilot project kept coming back to the entire team, the larger organization and presenting about how they were progressing, including their failures and their stumbles. He had t-shirts printed up which he gave to everybody, right, not just people on his project, which said, "Everybody is responsible for delivering business value to the customer," – very wordy t-shirt. But he gave it to everybody, not just the people on his project. He gave it to other executives. He gave it to the CFO and said, "If you hold up money and our people can't buy software on time, no, you are delaying delivering business value on time."

He gave it to the coffee machine maintenance guy and said, "If my people have to walk to the Starbucks to get coffee and waste 20 minutes, you caused that 20-minute delay of delivering value," right? He gave it to the janitor and said, "If the restroom is down because you are working in it and people have to go to another floor, you wasted time," right? He used to use these kinds of techniques to create that joint sense of ownership, even for people who were very far away from the project. So, everybody was rooting for their success, and also looking at it closely and going, "How can I learn from this? Because I'm next. I'm going to adopt what they're adopting."

Mike Kavis:
Yeah, I have a similar experience where this CTO was leading an effort and his slogan was "Business Value at Startup Speed." And that's the common theme here, is DevOps isn't about tools. It's about value and outcomes. And when leaders understand that, those are the ones who end up onstage at the big conferences talking about success stories, and they often share the lessons learned as well. So, good conversation there. So, the next thing I want to talk about when companies adopt cloud there's this concept of the bubble, right? For some amount of time you're going to actually spend more before you spend less because there's this long learning curve. And you kind of talked about the same concept on productivity. You talked about minimizing the productivity dip when adopting DevOps. So, first explain what that means, and then talk about what are the strategies to minimize that dip?

Sanjeev Sharma:
Yeah, and I'm glad you caught on that and are asking a question around that because that is a misnomer which a lot of people have, right? When especially two vendors come in they'll show all the productivity improvements that are going to happen when you adopt their tool. And what I always make sure people understand is that, there is a graph which shows an image of productivity dip, and that productivity dip has nothing to do with that particular tool or that particular change. Anytime you change, make a change, there will be a productivity dip because people need time to adjust to that time. People need time to learn. People need time to change how they operate. It takes effort to break the cultural inertia of any sort, right? You have a reorg. There's a dip in productivity immediately after a reorg because people need to adjust to the new managers, get to know them, their new teammates.

Heck, you change the name of a project and there's going to be a dip in productivity because people will spend time discussing whether it's a better name or a good name or a worse name. Any change will result in productivity dip, and the larger the transformation, the bigger the productivity dip is. And the goal of the people leading the transformation is to minimize the productivity dip, is to use techniques to make sure you're educating people, to make sure you're doing it in an organized manner and not ripping the Band-Aid off in one go, or minimizing the disruption, right? Not making slogans like, "Move Fast and Break Things," right? That's counterproductive. It's how do you minimize that dip? How do you help your people, help the people who are not involved in the decision-making process, right? These are people who have the hands on keyboard, the people who are actually doing the work, who are being asked to change how they operate, asked to fix something which in their mind was never broken because they were looking at it from a very narrow aperture.

Helping them understand that the new change, which might result in changing how they operate and maybe even result in them doing more work than they were doing before, changing and having them learn something new, having them change who they are working with, all that is part of a bigger plan which is going to pay out for the company at the end, and for them, too, because life will become easier, right? And they will have a chance to do more interesting things or – you've got to make a plan for everybody based on what their goals are. That attention to detail, where you are thinking about how that change impacts each and every person on the team is very critical, and that is where I think a lot of organizations drop the ball.

Mike Kavis:
Yeah, and one of the other things is organizations have only so much capacity to absorb so much change. So, I've seen people trying to get to the future state too fast and bring too much change at once. So, I'll give you an example. There was one company early in their cloud journey and they wanted to build some new stuff, and the architectural diagram had every buzzword you could imagine. There was blockchain, there was AI, there was ML, I mean, you name it. It had everything, and I kind of said, "Well, stop. You haven't even embraced cloud yet. There's a massive amount of change to do that. You haven't embraced DevOps yet. You've got to baby-step this. You can't go do all these things at once because it's so much change, right? You can only absorb so much change at once." So, these journeys, you have to be patient. You don't want to be slow. I don't want to give the wrong impression that you've got to take your time. But you kind of have to pace the amount of change with the capacity of an organization to absorb it. What are your thoughts on that?

Sanjeev Sharma:
Absolutely. Absolutely, right, because change, A, is going to cause productivity dip. B, it is going to change – as we just discussed, is going to change people's roles. But people still have to do what they are required to do, right? Nobody's day job is going away, right? It's not like – you know, in the analogy people have, right, like of transformation is changing the engines of a plane while still flying it. They still have to fly the plane, right, while they're panicking about one engine being removed and replaced with another one. So, the ability to consume change, the ability to absorb change is something very difficult to measure, and it depends on several factors, right? One of – and the most important of which is I believe the maturity of the team. Teams which have worked together with each other and have gone through change cycles in the past as a team, and there is minimal disruption to their existence as a team, right, people they work with, are able to absorb more change than teams where the change is resulting in disruption of the team itself, right? The people are being spread over other teams or people are going in and out of the team.

And then on the other extreme there is also change fatigue. I remember working with an organization back when I was with IBM where they were having a reorg every six months. They were having new executives come in and bring in their new programs of what they wanted to change every six months, every year, right? That was resulting in change fatigue, and every time a change would be announced, right, people were now stepping back, laying back in their chair and saying, "Oh, let me wait. Let me wait to see if this is real or this is just flavor of the month. Let me see who else changes. Let the other team move first. And I'm not going to waste time again on this change if it's just going to last six months."

So, I think that middle ground is very important, that, A, you have talked through the change so the change itself doesn't change over time. And, B, you understand based on the maturity of the team and their ability to absorb change – are you rolling it out too fast, right? Because if that happens, as you mentioned you know, people just get overloaded and burned out and they're just going to move on to other roles or not change at all. They start becoming resistant to change.

Mike Kavis:
Yeah, and the other part of that is you kind of need to build a culture that accepts that change is coming all the time, right? You don't have to take it all at once, but you do a culture change for this one thing, you're not done, right? There's more coming, right? There's all these new ways of developing, new technologies coming. I mean back in the day when we started there was just mainframe, right? There wasn't much change. And then client server came, and that was some change, but you know, we had years to make that change. And then the internet – we had years. Now it's like every six months there's something new, and it's just – you can't take it all in, but you need to build a culture of change, that change is expected.

Sanjeev Sharma:
And I think one of the things DevOps and agile have taught us is the whole – and agile, especially, is doing this in an iterative manner, doing this not as a big bang, massive project which is going to last nine months and every nook and cranny of the project, every task, every work plan and structure is laid out from the beginning, and people are expected to march to it, because we know that doesn't work, right? Even change needs to be done in an agile manner, in an iterative manner where there are timeboxed sprints or whatever agile technique you want to use for the change itself. And you're having retrospectives after every few sprints to say, "Is my change progressing as needed or not?"

So, even a change cannot be done as a big bang project with every work breakdown structure laid out rigidly over an extended period of time, over months. We need to do it in an agile manner where we have timebox sprints or whatever your favorite agile technique is. And have retrospectives where you're looking back and saying, "Okay, am I being successful with this change? Is it moving at the velocity I wanted it to? Or am I overloading my people? Or are people more eager for faster change, right? Can I go faster or can I go slower?" I think doing that in an iterative manner, looking back and saying, "Am I learning?" and learn as you go, and continuously improve your change process management itself is the way to really introduce change and understand where the inertia is and iterate about how you can overcome it.

Mike Kavis:
Yeah, and that's really one of the tenets of DevOps, right, continuous learning, continuous change. So, I'm going to move to SRE, and I don't know if it was a presentation, or maybe it was us being snarky on SRE on Twitter. But I remember you saying something along the lines like, "Look, you are not Google. You don't need SRE, but you do need reliability engineering." And I kind of agree with that, so let's talk about that. When you talk to clients and make those statements – give us an example of that.

Sanjeev Sharma:
Right, and you know, that was a blog post I wrote because at that point I was doing a lot of consulting in the cloud space, and I would run into clients, companies who'd say, "Yeah, come help us adopt SRE." And I would say, "What does SRE mean for you?" I would ask them, and they would point me to the Google book. And I'd be like, "That book's not for you," you know? *[Laughter]* "It's not for you. It's for Google, right? Tell me what about your organization looks like Google. Why did Google think about, why did Google invent, why did Google come up with SRE, right? How do they do SRE?" The principles, yes, you can adopt.

You need reliability and resilience, but how are you going to do it in your datacenter, or in the cloud even, with the applications you have, with the operations team you have, with the high availability requirements and resilience requirements you have, or the architecture you have? It's not possible with taking the SRE book from Google, as is, and saying, "I am going to adopt it," because your applications, half of them probably don't even have a high-availability architecture. They're running on one single set of servers which has single points of failure all over it. What will your SREs do? And a lot of companies, an antipattern was they took the DevOps teams, relabeled them SRE, and said, "Go to town!" That's not SRE either, right? So, –

Mike Kavis:
Well, first they took the Ops team and renamed them the DevOps, and then renamed the DevOps to SRE. *[Laughter]*

Sanjeev Sharma:
That, too, right? Exactly, right? So, what is it? And why do you want to adopt it? Where are your fault points where your Ops team is struggling? So, let's work at it backwards. Let's look at it – why did Google come up with SRE? Well, they felt their operations team were suffering. They were having – the word they use is toil. They were toiling. They were doing repetitive tasks over and over again and they were getting burned out. And they brought in software developers, they brought in software engineers to come in and look at those tasks and say, "Can we automate them? Can we do autodetection and auto-remediation of these low-level tasks? And then I can save my engineers' time, my folks who are operations' time to focus on things which are not business as usual, which are not repetitive, which are outliers, which are things we have never seen before or see rarely and require human intervention?"

But if it is hey, something went down, and it's not just for outages but also for reduction in quality of service. Okay, there's reduction in quality of service; what's happening? Well there's a network bottleneck. Well, can I distribute the traffic? Can I throttle the traffic? Those are tasks which can be automated. There are activities that can be automated, but they can only be automated if you are architected, your network is architected, your applications are architected for that. If you don't have applications running into availability zones, whether they are in the public cloud or in your datacenter, you can't be doing your balancing between two instances of the application because you don't have two instances of the application.

So, I think SRE has to start from first principles. The Google team did a great job defining what those first principles need to be, but then we as an organization, which is not Google, need to look at it and say, "How can I adopt those principles? And which ones of them apply to me? And then how do I implement them for my needs and you know, the architecture which I have adopted?"

Mike Kavis:
Yeah, I agree with that, and it's kind of comical when you have an infrastructure team talking about site reliability engineering that has nothing to do with a website, right? And it's just reliability engineering, right?

Sanjeev Sharma:
Yeah, and at IBM we used to just refer to it as service reliability engineering, not site, right? Because, sure, we have a site, but that's just one part of IBM, right? And those guys already were doing SRE, just didn't call it SRE because the book hadn't come out yet.

Mike Kavis:
So, one thing I think is missing – for some reason I don't think I've ever brought this up before, but what's missing from the reliability engineering conversation – it always seems to be about infrastructure, but what about architecture, right? AndI don't know if you saw the AWS re:Invent, any of the keynotes, but Werner's session, he talked a lot about architecture, and that's where a lot of reliability is as well. You have to architect to be reliable. Yes, people have to make sure the platforms and the infrastructure are resilient and reliable, but what about the software we build, right? What about creating loosely coupled services? What about taking parts of the application and making them independent from others so if they break the entire customer experience doesn't break? What about those things? I'm just not seeing that in any conversations today.

Sanjeev Sharma:
No, I think that conversation is going on there, right? And they're documented also, but I think not enough people are talking about it, right. So, when I go and have an SRE conversation, I talk about there being – and it's not an SRE conversation, it's a reliability and resilience conversation. I talk about looking at it through three lenses. Lens number one is what you're talking about, my application architecture. Is it highly reliable? Is it highly resilient, right? What happens if a service fails, right? I remember having this example of – I'm trying to be careful here so somebody can't figure out who the guilty parties are.

But there was this mobile app where one function in that mobile app for this client I was working with back then would crash the entire mobile app if that service was too slow. Well, that's nothing to do with infrastructure. That's an application architecture flaw which does not allow that application to resolve elegantly the non-availability of one service. The rest of the application should be able to work, right? Let's give an example of an airline application, right? If I'm trying to check into my flight and for some reason the seat map is not loading because of whatever reason, that shouldn't crash my entire check-in process. It should just say, "You cannot check your seat right now. We'll do it when you board. But here you go. You are checked in." Right? That is an architecture design function, what you referred to as loose coupling, where the rest of the application which is calling the seat map can sense to say, "Okay, the seat map is timing out; let's move on," instead of just hanging there forever and messing people up.

The second part is testing for reliability. How do I test? And that's where the whole process around chaos engineering is maturing, where I am going to run experiments to test what happens when something goes wrong. And the third is the responsive part of reliability engineering, which is how do I manage my MTTR, my mean time to restore of the service? And how do I manage to ensure quality of service for that other services which are not impacted if a particular service does get impacted and either has an outage or a degradation in performance?

Mike Kavis:
Yeah, I agree, and one of the things that you were just talking about, like the seat map one, there's this concept – and this isn't new, right? We were doing this back in the mainframe, but it's called perceived performance. And the perceived performance is what the eye of the customer sees. So, in that example your seat map may be seriously broken and it may be taking two seconds to respond instead of 100 milliseconds, but you hide that through the techniques like you've described. So, the one I use a lot is, like when you go look at your packages to see where they are, it's going to tell you the latest status.

Well, if the API that goes out and gets my status is broken, I just show them the last known status. That API is broken but the customer doesn't know that. They just know the last known status, it's in this location. And to be able to architect those things, teams need to be product-focused. They need to be invested in the product and everyone must have shared goals in the product-ilities, right? The availability, and too often – and that's where the silos kill us, right? This person owns tasks. this person owns security. This person owns this. This person owns – everybody should have this. So, you talk about the leader with the t-shirt, that everyone owns the business value and outcomes. I mean, that's the secret sauce to all this in my opinion. Thoughts on that?

Sanjeev Sharma:
Absolutely, right? And I think it is that joint responsibility and eliminating silos, having communication collaboration, having feedback loops, and another factor: having visibility, right? One of the reasons people don't get a sense of ownership is because they have no visibility. Developers don't have visibility into how their application is performing. They don't have visibility into – unless something gets escalated, a ticket gets escalated, and it results in a break fix or a change request or a buck fix which shows up in the developer's dashboard. A lot of developers do not have visibility and they don't consider it their problem when something goes wrong. Well, that's wrong. The other extreme is hey, give developers pagers or whatever – they don't have a pager today, so they get woken up and they'll make sure nothing breaks at 4:00 a.m.

Well, that's also wrong. That's just an extreme way of trying to fix the problem by waking people up in the middle of the night. Why not make sure that everybody has visibility into the information they need and developers have visibility into how what they build is performing, and then they can react to that information. Siloing of information, siloing of data, KPIs, and other performance and other metrics results in decisions being made without having all the information. So, I think that's another aspect which we need to address as organizations, is how do we get the right data visible to the right people, so that they know what is going on? And even when things are well, they know what's going on, and when things go wrong they know what's going on and they can be more proactive to prevent things going south.

Mike Kavis:
Yeah, and everything we just described, if we wrap it all together it sounds like what you need is an engineering culture, right, a culture of building highly-reliable products and services that create business value. And I think we've grown up over the years in, not only technology silos, but product silos within companies and haven't had that mindset in a number of years, and that's the culture change you talk about. It's a drastic change. It takes a lot time to get there, but I think companies need to move towards that.

So, that's it for today – great conversation. Hopefully we can meet in person one of these days and continue it. But tell us where we can find you on Twitter and if there's any content out there we should go look for besides your book. Just recite the name of the book for people and tell us where we can find all your content.

Sanjeev Sharma:
So my book is called *The DevOps Adoption Playbook*, and the best way to reach me would be on Twitter. You can also me on LinkedIn, Sanjeev Sharma at Truist – I think I'm the only Sanjeev Sharma at Truist. But my Twitter ID is @SD_Architect, so SD for software delivery, @SD_Architect, and my blog is SDArchitect.blog, so very easy to find me. And there are links over there to my book. If you go to my blog you'll see the link to my book right over there, straight to Amazon or wherever – you can get it from wherever you want. But it's available, and do follow my blog. The whole "SRE – You Are Not Google" blog post is posted over there, and there's many other blog posts which I would love to chat about. And reach out to me. DM me if you have any questions or have a comment on anything I said here on the blog or here on this podcast or any of my blog posts, or the book for that matter.

Mike Kavis:
Well, appreciate you spending the time today. That's it for today's episode of Architecting the Cloud. To learn more about Deloitte or to read today's show notes, you can head over to www.DeloitteCloudPodcast.com. You can find more podcasts by me and my friend and colleague Dave Linthicum just by searching for Deloitte On Cloud Podcast on iTunes or wherever you get your podcasts. If you want to contact me directly? I'm at MKavis@Deloitte.com. You could always find me on Twitter: @MadGreek65. Thanks for listening and 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.