EPISODE 1419

[INTRODUCTION]

[00:00:00] KP: Scalability, in all its forms, has been an active challenge for software projects since the earliest days of computing. Today, cloud computing services facilitate frictionless access to high-availability systems. But that success can make the challenges of engineering a high-availability system seem easier than they tend to be in practice. Lee Atchison is the author of *Architecting for Scale: How to Maintain High Availability and Manage Risk in the Cloud*. In this interview, we discuss a variety of challenges modern software engineering teams may face and strategies for resolving them.

[INTERVIEW]

[00:00:40] KP: Lee, welcome to Software Engineering Daily.

[00:00:43] LA: Thank you very much. It's great to be here.

[00:00:46] KP: So to kick things off, can you tell me a little bit about where you get your start in technology?

[00:00:51] LA: Sure, yeah, I've been in technology since – Well, since 1987 is when I left the university and entered the technology market. I worked for Hewlett Packard in Cupertino, California back when Silicon Valley was a little bit smaller than it is nowadays. And I stayed with HP for about 10 years doing testing measurement software systems. And then I left and went to my first startup, which went the way that startups usually go. And after a few years with them, I accepted a job with Amazon and moved to Seattle. So throughout that whole process, I went for the Bay Area, to Philadelphia, of all areas, to Dallas, and then finally ended up in the Seattle area, which is where I live now.

And I was with Amazon for about seven years, both in the retail side of the company, as well as in AWS. And I could probably do a couple podcast episodes just on war stories from Amazon. But Amazon was a great company to work for. And then I moved on to New Relic, which was an analytics company that was just – They were fairly small startup company at the time I joined them. But I worked for them for about seven years in a variety of different roles, ending up being kind of an ambassador to talk to their customers, and within the marketing and sales organizations, going around talking to customers and trying to sell as a brand associated with – I ain’t going to do that over again, if you don't mind.

[00:02:19] KP: Oh, yeah, go for it.

[00:02:21] LA: Okay. And finally ended up at New Relic. Spent seven years at New Relic and doing a variety of things there, all the way from a developer role all the way up to, at the end, when my book first came out, I went around on the road with New Relic talking to customers and doing presentations at conferences and things like that. Kind of promoting the New Relic brand, but my book at the same time.

[00:02:44] KP: And is that architecting for scale high-availability for growing applications?

[00:02:50] LA: Yep, that's right. The first edition of that book came out in 2016. And that's when I took on this new role at New Relic and started going around talking to people.

[00:03:00] KP: Well, I definitely want to get into the book. But one more sidetrack before we get there. I'm curious, at the time you joined Amazon, at that point, where were they at in their cloud journey?

[00:03:11] LA: Oh, that's a good question. So the time I joined Amazon, this was in 2005. So there was no AWS. And it was just Amazon Retail. And Amazon Retail was a monolithic application. It was called Obidos. And my first job was I joined the team that was responsible for driving the migration of their website from a monolith to a service-oriented architecture. So that's like jumping in headlong into a large project just getting started. There're about 100 engineers involved in that project. But they were a small company at the time. Their stock price was $30 instead of $3,000. It was a lot smaller organization.

But I did that, and I worked in the video game group for a while. I built the first software download store for them and created the group that built that that ended up becoming the App

Store eventually. And then this upstart group within Amazon, called AWS, which was just getting started at the time, they called to me and asked me to move over to that side of the organization. So I moved over there and started working on Elastic Beanstalk, which was their first foray away from infrastructure as a service, like EC2, and into platform as a service offerings, which was what Elastic Beanstalk ended up being. And so did that for a number of years, and seven years total at Amazon. So I was there until 2012, I believe, before I moved to New Relic.

[00:04:46] KP: And at that point, I mean, is that when scale become a prominent feature of your career? Or had that been there for much earlier times?

[00:04:53] LA: No, absolutely. That's when scale entered my vocabulary. Even when Amazon was smaller back in the 2005 days, everything you did was done at scale. It's an amazing mindset shift that occurs when you move to an organization like Amazon. That scale is such a central critical component of everything they do. And so that's absolutely when scale entered my mind set and when I started to think about scaling and scalability. And I didn't really ever consider myself an expert in that area until I moved to New Relic.

And what happened when I moved to New Relic is I moved from this large company with huge scale to a small startup that still had huge scaling issues. They were, at the time, like I said, they were about 100 people total in the company, small company. A relatively small company compared to where they are now. But they still had a big data intake pipeline. Big for them anyway. And so they had scaling problems. And they were just starting to go through the growing pains that I'd seen so many other times within various projects within Amazon.

And I was watching them struggle with some of these growing pains and I said, “Well, I know how to do this. I know how to help with this. I know what needs to happen here.” And I realized that the things that I took for granted, when I was at Amazon, the things I took for granted when I was at Amazon weren't things that everybody just knew. They were things that were very unique and very specialized skill sets.

And so I kind of led the effort within New Relic to help them improve their availability, which was based on scale issues and improve the scaling. And that's when I really focused a lot on the organizational changes, the strategies for building, scaling into the DNA of the company, if you will, and to help make that a mindset shift that occurs, which is really what you need in order to build a highly scaled company and a highly scaled application. And so I kind of built that mindset in. And then that's when I started realizing that this is really unique information that would be valuable for other companies as well. It's things I took for granted at Amazon, but really benefited New Relic. So that's when I wrote the book.

[00:07:19] KP: So the lessons that are found in the book, are they for an executive who wants a scalable company? A director of technology who has to get the team on the right track? Or who's really the right audience?

[00:07:31] LA: I think the main audience is the range from – Depending on the size of the organization, the range from manager, senior manager, all the way up to director, a hands-on CTO, in that sort of range. So that range would be the ideal range for this book. It's not designed for the true executive, the CEO, or the CTO of a large organization. Nor is it designed for the new engineer that's just joining a company and trying to figure out what to do. It's really for that middle area.

[00:08:06] KP: Well, scale is something that the cloud has been in some ways a solution for. Is it going to become to the point where maybe I can just consider scalability a problem I outsource? That it's Amazon's problem, and I just focus on my code? Are there critical lessons I'm going to need to learn?

[00:08:22] LA: There absolutely are. I think that's one of the biggest challenges that I faced when I talked to companies, and both when I was at New Relic as well as in my consulting business now, is the concept that scaling is a thing that you do. And that since it's something you can do, you can hire someone to do it for you, and you don't have to think about it. But scaling is really about a cultural change within the organization.

The scaling and availability go together. And those can't be solved with tasks They're solved with the mental change occurs within an organization, to change the mindset of the organization, to be thinking about a scalable company, or a scalable organization, or a scalable application and a highly available application.

Now, once you do that, the natural progression is the cloud becomes a tool that helps you accomplish your goals. But you can't just say, “Oh, I'm using the cloud. So therefore, I solved my scaling problem.” It's the chicken comes before the egg sort of sort of thing. You need to be thinking about scalability. And that leads you to the cloud. But the cloud doesn't solve all your problems. It just enables you to solve your problems.

[00:09:40] KP: What are some of the things that initially people tend to bump into is their early growing pains?

[00:09:47] LA: Most early scalability problems show up as availability outages. Brownouts, blackouts, things like that. And usually they're caused from not thinking through ramifications of process decisions that they're doing. For instance, a smaller company will – They'll have a deployment process. It's less formalized. How they process that someone can make a change to a production system and not log the changes in the right systems that make sure other people know what thing isn't going on? They're a lot less formalized in how they do things. And that's okay to a point. And certainly, we can talk about Agile on how Agile fits into all of this. But the problem that happens is, by not thinking through the problems, you start making stupid decisions, or stupid mistakes. Stupid mistakes end up being outages that you're not prepared for, and you don't have the tools for, you don't have the processes to solve quickly. You don't think about them as problems in the same way.

I’ll tell you. I can't tell you how many companies I've talked to that think about downtime as a kind of a throwaway concept. We're going to launch a new version of the app. And we'll just bring the app down for an hour in order to do that. But they just naturally think that that's an okay thing to do. But those are the sorts of decisions. That's the sort of mindset that leads you into this trap, that leads you down the path where, as you scale, you start having more and more problems, and more and more availability problems that turn into more and more scalability problems.

[00:11:31] KP: Well, I'm thinking of typical startup are the ones I've been interacted with, I don't know that any of them have a good handle on what their availability is. Are there best practices for establishing that or coming up with a true measurement of how many nines I can award myself?

[00:11:47] LA: Lots of different things that that come to mind there. One of the first things you want to do, though, is measure everything, right? And, of course, I came from the background of a measurement company. But even before that, I realized the reason why I joined New Relic is because they did these sorts of things that companies really need to do in order to keep their applications going.

But you don't need something like New Relic. You just need a way to monitor your application so that you know what the performance is. You can't solve problems. And you don't even know where your problems are unless you know what is currently available. So measure, measure, measure, measure, measure first. And once you measure, then you start knowing where you're doing well, where you're not doing well. And you know the places you need to invest. And you know what to do to make that better.

The next step beyond measurement is establishing baselines, establishing SLAs, and understanding what you – Baselines are, so you can see where you're improving and how you're improving. And SLA is to establish what the minimum expectations are from one component in your system to another, which ultimately amounts to your customer expectations.

[00:13:06] KP: I imagine this could vary a lot by organization. But are there any common metrics or measurements that most people are typically looking at?

[00:13:14] LA: At the highest level, most measurements having to do with latency, with first byte availability. Those sorts of measurements are really the basic things that most people think about. When I think about measurements, I think a lot about time to resolution, MTTR, MTTD, mean time to resolution, mean time to detection. How long does it take you to notice that there's a problem on your site? And then once you notice there’s a problem on your site, how long does it take to resolve that issue and fix it and bring your application back up to a normal standard? Those are two measurements I think a lot of companies don't think too much about, especially in the early days, but really are critical ones to building a highly available, highly scalable website.

[00:14:05] KP: And what's the share of responsibilities between traditional software engineer and someone in more of a DevOps role?

[00:14:12] LA: Yeah. So I don't like separating the two out. I'm a big believer in the development organization is the one ultimately responsible for all aspects of the service. You've divided your application into services. You divided you're – Each of the services is owned by a particular development team. That team is responsible for all aspects of the service. That's including – That's building the service, designing the service, architecting the service and operating the service up to an including on-call for running and keeping the service going.

Now, whether or not in order to accomplish that, that team is all engineers, all DevOps people, whatever the combination is. That's less critical. And in fact, I'm not a huge proponent of role division by I'm a developer versus I'm a DevOps person. I really like organizations that play those roles a lot more fluid. Every developer should understand how to do the things that a DevOps engineer does. And a DevOps engineer really should understand the concerns and problems that the developer is running into. So I like running organizations where those are much more fluid concepts.

[00:15:29] KP: Yeah. I've seen a lot of success with the do one thing, do it well concept. I'm not so much for the professions, as you said, wear many hats, but in terms of the microservice. So having a small team that owns that end to end makes a lot of sense. But then people will say, "Well, now you have a problem of multiple microservices and the coordination of what happens when something fails. Are there extra challenges there as we scale?"

[00:15:52] LA: Yeah, certainly, one of the problems I see some companies running into is deciding where to put service boundaries and how many services is the right number of services for this application. And sometimes, companies, when they move from a monolith to service architecture, they go too far, they make too many services that are too small. And what happens when you move to a service-based architecture, is the individual services themselves, the smaller you make the services, the easier they are to support, and the less cognitive load it takes in order to understand how the service works. But the more complex the interactions are between the services, and the more cognitive load it takes to understand the overall system architecture and how things work together.

So what tends to happen when you move your services smaller and smaller, is you need less expertise, and smaller teams to actually build and operate the services. But you put a greater load on your architectural team and the team that owns the interactions between the services, and you create problems at that layer. So that's kind of the sweet spot where your services are big enough that you don't create the cognitive load of the service interactions that occur, but small enough that each individual service is really just a mini monolith.

And finding that sweet spot is kind of a – There's some trial and error to it. But there's some ways you can go over that. And I talk about them in the book a fair amount, about how do you size your services and where do you draw the service boundaries in order to make the load between what it takes to manage the services and what it takes to manage the interactions between the services be the right balance for your organization?

[00:17:48] KP: There's been a growing interest in serverless functionality, most notably, I guess, Lambda functions. One of the things that's appealing to me about them is it's an opportunity to maybe focus a little bit more on my code, worry a little bit less about how it's going to be run and scaled up. How do you feel about that? Am I giving up too much in a typical Lambda environment? Or are things still manageable there in a responsible way?

[00:18:12] LA: I think anyone who's read some of the things that I write will find that I'm not a huge fan of serverless. I'm not a big advocate of serverless, in general, and primarily because it forces and drives decisions, in my opinion, for the wrong reasons. A serverless architecture really dictates a style and an architecture and a sizing of your services that I think is, in general, often too small for the services to be. You end up with this problem that I'm talking about where your individual services are a lot less complex and easier to manage. And so the advantages of serverless work there. But you end up with substantially more services and a more complex interaction between them. And so you end up with taking that complexity and moving it out of the service and moving into an area that's a lot harder to deal with.

And that tends to happen more with serverless architectures than they do with servers. You're going to have to deal with the issues around how your infrastructure supports your application, whether you’re doing serverless or whether you're running on top of servers directly. Dealing with it, and deciding to deal with it, and planning for it, and building it into your processes and your decisions upfront gives you the flexibility where you can then make your code decisions based on where you want your code to be, not where the infrastructure requires it to be. So that's one of the big things I have against serverless, is that forcing a specific style, and structure, and size of services that isn't necessarily consistent with your application.

Another problem with serverless is the inconsistency and how they operate in performance that goes along with it. Serverless functions, really, you have to deal with an unpredictable performance curve. And a lot of applications have a hard time dealing with that. And a lot of people don't think about that. And a lot of people don't deal with that problem. And that ends up showing up in performance issues later on and other aspects.

And finally, I think people don't understand that serverless, even though it's less infrastructure, presumably, that doesn't mean it's cheaper infrastructure. And I've seen many companies just go so overboard with serverless in areas where they really weren't designed to be used, so that their cloud bill really shows the cost of that value in ways that weren't consistent with how they use that value. You see what I'm saying?

[00:21:02] KP: Yeah. Well, in contrast, could I get your thoughts on Kubernetes?

[00:21:07] LA: Yes. I am a huge Kubernetes fan. I've always been a big fan of containers as a way of delivering service technology into an operational environment. And Kubernetes has been the Godsend of how do you run containers in an operational environment in a positive way? That's my view. Now, there certainly were other technologies before Kubernetes, kind of became the golden standard. And certainly, there's nothing wrong with things like Amazon container service. But Kubernetes is the gold standard, if you will, in container management. And it does a great job with it. It's got a learning curve that's higher than you would want it to be. But it's well worth it. And you end up with a with a solid infrastructure that's easy to deploy new capabilities into.

[00:22:05] KP: Well, if I were personally responsible for scaling up and maintaining high-availability on some system, I would definitely be concerned with how good my source code is, that I'm deploying the right services. Doing everything in-house correctly. I also, it seems, might have to worry about my cloud provider. We've seen some big outages in that regard. What's a good way to split my time between worrying about my house and my provider?

[00:22:30] LA: That's a good question. Let's look at the cloud providers and the outages that have occurred recently. And certainly, I think one of the problems with the cloud provider outages that occurs is that they get a high level of visibility. And certainly, let's take out this last round of outages that have occurred recently. I think it's fair to say that the outages tend to be outweighed by publicity than by real impact. Now, the last rounds were pretty bad. And they may change the scales a little bit here. But I think an application running on a cloud provider versus an application running on on-premise environment, and how often the infrastructure causes problems that cause the application have a problem in those two environments, I think you would find that cloud providers are very, very, very stable and very high-quality and very, very performant, etc. And that the vast majority of outages – I mean, from a percentage standpoint, you're going to get a lot more outages when you're running on your own data structure, or your own data center in your own environments.

And I think even the question is what this last set of outages, does that change that ratio? I don't think it does. But I haven't thought far enough along yet to know whether that's really changed things or not. Bottom line is when a cloud provider goes down, it affects a larger group of people. And so you hear about it in the news. But it's not necessarily for an individual company and an individual application any worse of an outage than they're currently used to if they're not using a cloud provider and running in their own data centers. So I don't really see the outages that have been occurring as problematic. Just more of a reality of running a highly scaled application. I still trust AWS to run a high-scale website.

[00:24:34] KP: Well, if you're someone who has the opportunity to work on something that could have unpredictable spikes and traffic and need to be available and grow, there's just a certain degree of uncertainty. And the only way I know to manage that is to have smart planning and risk assessment. So no surprise that risk management is a major section of the book. It's not something that appears in a lot of computer science curricula, however. So I think it's something a lot of times software engineers can pay lip service to, but don't really know where to begin from a quantitative sort of perspective. Can you share some ideas about how we frame and measure risk?

[00:25:10] LA: Yeah, yeah. So I strongly recommend that each individual service team builds and develops their own risk matrix for the operation of their service. And what a risk matrix is, is it's a list of known vulnerabilities, known problems, known things that you think might be a problem or you know are a problem and a grade them by using two criteria. One is the likelihood that this risk will happen, and the severity of this risk if it does happen.

So the likelihood is what's the percent chance of this thing happening in some sort of scale? And then the critical masses, if this does happen, how major of a problem does this occur for you? What you'll find is, is a lot of risks that have high likelihood, and they're very likely to occur, but the seriousness of the risk is not very high. And there's other ones where the likelihood – Or very, very serious that this problem occurs, but the likelihood of them occurring is very low. But then you'll find there's other risks that are not very likely to happen and aren't very serious that they do happen. Those are the ones you can simply ignore. But then there's the ones that are likely to happen and are very severe when they do happen. And those are the ones you end up paying the most attention to.

So anyway, you brainstorm this list. You keep the list up to date. As new vulnerabilities become known to you, you add them to your list. This becomes your gold standard for the risk that's built into your system. And you associate these two variables, likelihood and severity, to each item and at risk, and the you track everything using that. You focus on the ones that are the high highs, the high severity and the high likelihoods. You ignore the lower ones. And you use this as a tool to help prioritize the work that you do to improve the availability and quality of your service.

Then individual service owners should share these matrices with their dependencies as well as up the management chain so they can combine them and create application level or component levels or company levels risk matrices that contain all the information about everything that's going on within the organization. So the risk matrix becomes a way of communicating technical debt up and down the management chain, as well as with your dependencies. So they become a communication tool that allow you to share where you are in your organization, and what's important.

When you come to trying to schedule a project now and you run into issues where management wants it done this day, and you say it's going to take and all sorts of problems, you can use the risk matrix as a tool to try and show you, “Look, we have these risk items that we need to resolve before we can fix this or before we can do this, or that sort of thing.” Becomes a tool that can help you with your communications to understand and let management make decisions like, “Well, it's okay for us to have these risks. We understand what these risks are.” So we are willing to trade these risks off for these other things that we want to have done. So it becomes a tool that allows you to talk technical language and management language or business language and have everyone understand what they mean by these risk items that were talking about.

[00:28:48] KP: Well, earlier on, we've been following the thread of your career, and I think we left it off while you're at New Relic. The book has come out. You going around touring a bit about it. Then decided to go off on your own. That's a risk to be taking some risk to be managed. What made the timing right for that to be going on a new venture at that point?

[00:29:06] LA: Well, a number of personal things were going on, as well as some changes within the company that were going on, with the New Relic that were going on. And ultimately what finally decided is – So the timing was between some personal items, things ever going on. Some changes that the company that were going on. And then the start of the pandemic, which basically meant that it was very hard for me to go and talk to customers now and do a lot of that. Made me realize there's a good time to be thinking about doing something different.

And so New Relic and I came to an agreement. I left New Relic and started my own consulting company. And that was now – Let's see. It was May of 2020 – 2019. I'm sorry.

[00:29:55] KP: Pre-pandemic?

[00:29:57] LA: Well, 2020. So May of 2020. Sorry. So it's going on two years now. It's been about a year and a half. And I've been going like gangbusters ever since and really enjoyed. It is a risk. And it's a risk that you wonder if you can ever do that risk anytime in your life. And it was probably a time in my life when I could afford to take a little bit more risk. The kids were out of school. They're living on their own. It was just my wife and I at home. So we didn't have the family issues to worry about. So it's a good timing from that standpoint.

So I decided just to go off and make the change. And it's been a great positive for me. I've been very happy with what's been going on. And I've got some good clients right at the very beginning, which gave me the motivation and been growing and going like gangbusters. I think, recently, I've started to make a change, where I've been focusing less on individual clients, where someone will bring me in and use my expertise to help them. And I've been doing more educational outreach. I did the second edition to the book. I'm working on another book. I did a book for Redis Labs on caching and working on another book for them. I did a series of courses for LinkedIn Learning, and a bunch of other content output like that. I've been focusing a lot more in doing that and selling that content. So I'm not sure in the future how much of my time will be traditional consulting versus push education. But some ratio between all or nothing on one of the others of those. I'll be doing some combination of both of those, I'm sure.

[00:31:45] KP: Well, the books now in its second edition. Are there any notable changes from edition one to two?

[00:31:52] LA: Yeah, it was actually a fair amount of change I went on there. So what happened was the first edition of the book came out in 2016. And that's when I started my new role at New Relic where I went out and it'd be the promotional engine going and talking to customers and doing tradeshows and things like that. So I ended up talking to an awful lot of customers in an awful lot of situations and trying to help with the problems they were running into. And so I've learned a lot after the book came out about how other customers were going through the same problems that I saw at New Relic and Amazon and other places. And so I use that material that had developed over the course of now several years of doing that, from 2016 to 2019. And that's when I decided to do a second edition of the book.

So the second edition of the book came out in March of 2020. And basically, it's a complete reorganization. It also has about 30% to 40% more information, as far as by page count, and that sort of thing. And so there's a lot more information in it. And it's radically reorganized into topics that I think will make it easier to read and easier to understand by individuals.

So basically, it's divided into five tenants that I talk about. I talk about availability. We talk about service-oriented architectures. I talk about organizations, people, management. Building organizations that are designed for scalability. I talk about risk management. And then I talk about the cloud. Actually, the cloud was a brand-new section that was added to the book as well and a bigger focus on how technology in the last several years.

So a lot of changes that occurred. And response has been very, very positive. I've gotten lots of people who have bought the first edition then bought the second edition and said it's like night and day comparison between them. So I think it was a good upgrade as it has a lot of good material for people who really are needing to solve these problems.

[00:34:09] KP: Well, Lee, before we wrap up and send listeners on some links or whatever we do there, are there any particular trends or technologies you're following and excited to see going on here in 2022?

[00:34:20] LA: Yeah, I'm very excited about citizen developers, and about the trends that are going on in that area. I'm also rather intrigued, but not quite sure where it's going in the idea of AI-assisted development and some of the technologies there. Of course, AI, in general, is a big area. IFT is a huge interest of mine. But I think the general areas of citizen development and AI-assisted development are two areas that I'm in particularly intrigued about.

[00:34:55] KP: Very cool. And we touched on a number of educational solutions you're coming up with. Where’s the best place for listeners to learn more?

[00:35:03] LA: Best way is to go to leeatchison.com. You'll see my books there and links to where to get them, as well as the courses I've done. Right now, there's four LinkedIn courses, plus a couple other courses. And then my O'Reilly book, my Redis book is there, and a couple other – I've been contributors on a couple other O'Reilly publications as well, and they're all listed there as well.

[00:35:31] KP: Well, Lee, thanks so much for coming on Software Engineering Daily.

[00:35:36] LA: Thank you very much for having me.

[END]