EPISODE 908

[INTRODUCTION]

[00:00:00] JM: Amazon Web Services first came out in 2006. It took several years before the software industry realized that cloud computing was a transformative piece of technology. Initially, the common perspective around cloud computing was that it was a useful tool for startups, but would not be a smart option for large, established businesses. Cloud computing was not considered economical, nor secure.

Today, that has changed. Every company that writes software is figuring out how to utilize the cloud. Software companies with on-prem servers are migrating old applications to the cloud and most companies that have started in the last decade do not even have physical servers. Applications that are started on the cloud are referred to as cloud native. The architecture of cloud native applications is a newer topic of discussion, and some software patterns that became established in the pre-cloud era might make less sense today.

Cornelia Davis is VP of technology at Pivotal and the author of *Cloud Native Patterns,* a book about developing applications in the distributed virtual world of the cloud. Cornelia was previously on the show to discuss Cloud Foundry. In today's episode, our conversation centers on her book and her perspective on the emerging patterns of cloud native software.

[SPONSOR MESSAGE]

[00:01:31] JM: Hired simplifies the job search for engineers with a data-driven personalized matching process. Head to hired.com/sedaily and create a profile today. By creating one profile, you'll be matched with over 10,000 companies looking for engineers like you. Hired uses intelligent matching technology, data science with years of experience matching engineers with jobs. And hired also has a human in the loop. Hired gives personalized career coaching to match you with opportunities based on your skills, industry, interests and desired salary. Create a profile today at hired.com/sedaily. Find a job that you truly love that is personalized to your background and your preferences. And if you aren't an engineer, Hired also helps designers, engineering managers, product managers and other tech workers find their dream jobs. Just go to hired.com/sedaily and check it out.

[INTERVIEW]

[00:02:44] JM: Cornelia Davis, welcome back to Software Engineering Daily.

[00:02:47] CD: Thank you. It is so great to be back. It's been quite some time.

[00:02:50] JM: Sure has been. I think we last spoke at the DevOps Enterprise Summit. Is that right?

[00:02:56] CD: Yeah, quite a number of years ago. When I continued to be involved in the DevOps Enterprise Summit, I think we might have met the first year that I was involved, but I've been part of the programming committee for some time, and dare I say so delighted to have established to really start working relationship with Gene Kim who runs the summit, and I would dare even call him a friend. He wrote the forward to my book.

[00:03:19] JM: That's awesome. I mean, he has become kind of a – He's like a cult figure, and DevOps Enterprise Summit has become something of an enterprise in and of itself.

[00:03:28] CD: It has, and you're very right. He's definitely really seen as a kind of hero in the space, and he is just so tremendous. The Enterprise Summit, I always tell people whether it's like colleagues or my customers that it is literally my favorite conference of the year and we do it twice a year. Once in North America, once in Europe, and the community that he's built and being part of the programming committee, I can see the care that he puts into curating a super great conference and really has a good team around him to do it. It's quite impressive.

[00:04:04] JM: Yeah. What they're doing is really important, and I become more intrigued by it every year. I mean, this is obviously a kind of – We're originally talking about it, but could tie in. The strategy around "digital transformation" is really an evolving thing. Basically, the way that a gigantic ancestral organization, if it's 50-years-old or 80-years-old, it's not going anywhere, and they need to adapt software whilst new software comes out every year.

So it’s this moving target, and it’s really interesting seeing every year the talks that come out of that place. In some cases, there are these total themes that develop, like, “Okay. You finally started your digital transformation. Here’re the things you have to do.” Then here’re the things that just started happening this year, because Slack started becoming a platform or something like that.

[00:05:03] CD: Yup, and even one of the really interesting areas that Gene and I have been working on together is that we, about a year ago, started talking about functional programming, and he personally made the journey from being an imperative programmer, programming traditional languages like Java or something like that, and now his favorite language is Closure.

We’ve been doing a lot of riffing on how that’s impacting even the software that we build and the architectures around event-driven systems. So I like to say if automation is the way that we use to do the imperative way of doing even systems, deployments of allocations, deployments of systems, what is now our functional alternative to that and it tends to be more event-driven, and that’s where Kubernetes has become really interesting, because they’re playing it that systems spaces as supposed to the application programming interface. So we’ve been doing some work around that, and that’s been starting to show up increasingly in the program.

[00:06:07] JM: All right. Well, we’re here to talk about your book, which is about cloud native applications, and let’s start with some history. AWS first came out in 2006. It took some time for AWS to become really popular. And obviously in the early days, it was much more slimmed down. It was just server in the cloud, a small storage service in the cloud. It’s really changed since then. When did you start to realize that the cloud was going to change software engineering?

[00:06:43] CD: It definitely wasn’t in 2006, and you just put your finger on why it wasn’t in 2006, and that was because it was really just running the same fundamental components of compute servers up in somebody else’s data center. Although I will say that even though that wasn’t cloud native at that time, it was cloud, which I make the distinction that cloud is much more about where, and cloud native is about how. In those early days of 2006, nobody saw that. It was really seen as much more of a, “Hey, I’m just going to run my stuff in somebody else’s data center. But nothing really changed around that.

When the cloud started to change software architectures was when we started to understand a little bit more about the fact that it wasn’t just a how, but that some of the characteristics of those clouds, AWS, and now some of the other cloud providers, were changing. Characteristics like, “You know? We didn’t promise you that that server would stay around forever.” The fact that you now have to worry about a server not staying around forever, whereas before the cloud, we really built our data centers to have stable infrastructure. And if the infrastructure went belly up, we would throw our hands up in the air and say, “Well, our software can’t run, because the infrastructure went belly up.” But that’s no longer an excuse. The infrastructure is having all sorts of changes happening to it, and our software still has to run.

So 2006, were we there? Absolutely not. I would say that probably by – Well, when I’m thinking back, we started to really get serious about that, and I think I saw the first couple of really interesting talks from AWS people. People like Werner Vogels and other folks. I would say it was closer to 2010.

[00:08:47] JM: Why did you decide to write a book about cloud native applications?

[00:08:51] CD: The reason I wrote a book is because I actually have been delivering the messages that ultimately came into the book in one-on-one settings with our customers. I work for Pivotal, and Pivotal, our enterprise customers want to get better at building software. To a large extent, the way that that’s coming from the top, from their boards, for example, is the boards are saying, “Hey, we want you to go over the cloud,” and they’re basing that on ample evidence of these startups that have been successful on the cloud and they’re starting to see those startups see the way of their business. So they’re saying, “Hey, we need to be that good at software.”

So that is what I do in my day job, is I’m out there talking to customers about what does that mean. It’s not just taking your same practices and your same software that’s running on-prem and running in somebody else’s data centers. The characteristics of that infrastructure in the other data centers are different. Therefore, you have to write your apps in a different way.

The subtitle of my book is building or designing change tolerant software. So making that the first class thing that you're thinking about, those are the lessons that I'm working day-in and day-out with these large enterprises that have been sometimes building, but definitely running software for decades. I wanted to sail that. I wanted to be able to – I'm a teacher at heart, and so I didn't want to just teach the few hundred customers that we have as Pivotal customers really wanted the opportunity to share what I have learned so far, and I emphasize so far, with a broader population.

Now, I'll also tell you that I have been a long-time teacher when I was in graduate school, was when I learned that when you teach something, you learn it at a level that you just won't until you teach it. So it's not all altruistic. It's also that I have learned a tremendous amount. I've become much, much deeper in the process of teaching these patterns to my readers.

[00:11:04] JM: As you said, that subtitle is designing change tolerant software. In what ways does our software sometimes fail to tolerate change?

[00:11:15] CD: Ah! So there are quite a number of examples. The simplest one is this. I am willing to bet that some of our listeners are very familiar with the concept of the stick session. So their applications have multiple instances deployed. They know that there are some good redundancy – Redundancy servers a good purpose there. But they have written the applications so that they store – The way that applications are written today is very request response, and you string together a bunch of requests to be able to achieve an outcome for your consumer.

In the past, we stored a lot of the state that took us from one request to the next so that we wouldn't have to retransmit everything that we had transmitted so far, that we would store that state in-memory, or we would store it on local disk. Well, change tolerance means that – So sticky sessions where a way that you could say, "Hey, a multitude is sequence of events coming from one consumer will keep routing that to the same instance so that we can take advantage of the memory that's stored in that instance of the application.

Well, in an environment where change happens all the time, and that change might come from, "Hey, I have a memory leak and my application crashes." But it needn't be something that is catastrophic. It needn't be an error. It could be that, "Hey, my infrastructure team or my platform team is in the process of patching the latest operating system from some vulnerability." Being able to do that without being in lot step with the application team is one of the huge benefits.

If you're using stick sessions and they actually role the instance and your user goes to a different instance and loses all of the context of their shopping cart, that's a pretty bad customers experience, and you're losing revenue.

So, it's that type of thing. Getting rid of the use of sticky sessions, but instead handling state from request to request in a fundamentally different way. That's the simplest example I can come up with, and there're tons and tons more examples of that in the book.

[SPONSOR MESSAGE]

[00:13:46] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW CONTINUED]

[00:15:34] JM: Earlier, we talked a bit about functional programming, and one significant development that happened just a few years into cloud was the infrastructure as code movement. And that has gradually evolved from things like Puppet and Chef, which is I've never programmed in Puppet and Chef. But as I understand, they're mostly imperative languages. And that's evolved into Kubernetes, which is more of a stateless-like description language, which is more in the direction of – Well, functional programming people tend to talk about statelessness a lot more than the imperative programming people. What's your retrospective on how we went from EC2 servers to infrastructure as code, to immutable infrastructure, and what's the tie in with these movements towards functional programming?

[00:16:31] CD: Great question. So immutability is absolutely essential for change tolerance, because immutability says that I – When you have immutability, what you can effectively do is you can recreate from scratch, and that's what infrastructure as code is. So, for example, we're just talking about this notion of sticky sessions and I said, "Well, okay. You don't want to use sticky sessions. So let's assume we're not using sticky sessions." But I've got my application running. By the way, it's increasingly running in containers. That's where Kubernetes comes in and other technologies. We can come back to that. But we've got our application running in those containers, and now I need to cycle the host that that container is running in, because there's an operating system vulnerability that I need to patch as soon as possible, because come tomorrow, the embargo is going to get lifted on the vulnerability and the CVE and every hacker out there is going to know about it and they're going to start pointing their boss at everything out there. So we have to patch the operating system today.

Well, in order to facilitate that to allow that to happen in a few hours as supposed to a few months, which is literally how long it used to take to patch certain vulnerabilities, and things like Equifax show us that it sometimes wasn't even done in months. Sometimes it would take years to patch all of the operating systems in an infrastructure.

The way that we can achieve that is by automating the deployment of all of those things, and that's where infrastructure as code comes in. An immutability is what says, "Once I've established whatever that entity is, whether it is code running on a virtual machine or code running in a container, I'm not going to allow somebody else to come in and mutate it in a way that can't be reproduced. And that's going back to the question of functional programming.

It's extraordinarily interesting when you start to study this, because if you were to look at an application where everything is happening with side effects. So by that I mean, you declare a variable and then you run through some logic and you're changing the values of that variable as you run the program, as you run the imperative program. In the end, all you have left over is a state. But exactly how that state was achieved is completely lost.

Now, if you take a functional programming approach, everything is immutable. So you can simply replay the – In fact, you could start with an expression of the problem that you're solving and the functional program will deterministically go through the same execution and yield the results at the end. So immutability, which functional programming, sometimes people have this misnomer that functional programming is, "Oh! As soon as I can pass functions, I have a functional programming language." It's actually far more important – The immutability portion of functional programming is far more important than passing around functions.

[00:19:47] JM: What are the most common failure domains that a cloud native architecture needs to protect against?

[00:19:53] CD: So I think that there are a number of them. So I actually start the book, the first four words in the book are it's not Amazon's fault. And I start the book with a story of an Amazon outage that happened in 2015, and there were quite a number of companies that of course are depending on Amazon to run their businesses. And the outage for Amazon lasted somewhere between 5 and 8 hours depending on how you count, because lots of systems were affected.

So the outages for companies like IMBD and Nest lasted for at least that long, if not longer, by the time their systems came back up after Amazon systems were back up. Another company that was running on that same infrastructure was Netflix, and there was a great article, a blog post that they posted not long after the outage where they said, and I quote, "Yeah, we had a brief availability blip." And I just think that phrase is so fantastic, because it's this like surprise. Everybody was blaming the infrastructure for their outage, and Netflix said, "Oh! We had a brief availability blip. No biggie."

Of course we know Netflix is kind of the poster child for cloud native and microservices, and it wasn't that 8 hours or 10 hours is brief for them. Clearly, it's not. But what they had done was that they had protected themselves. They had understood very clearly some of the failure domains. What happened in this instance is that Amazon had a full regional outage. So a whole bunch of the services in a particular region were affected. What Netflix had done was they, first of all, recognized where that boundary, that failure domain was. By the way, that failure domain is very clearly defined by Amazon, which is why I say it's not Amazon's fault.

Amazon never promised you that a region would never go down. Sure, there are some SLAs where if it went down 30% of the time, you'd be crazy to use it, because you need to have some level of stability. But they never promised it was zero downtime for a region. Therefore, they gave you the access to multiple regions, but it's up to you to use them.

So things like data centers, which are largely in Amazon parlance described as regions, the abstraction that they use is regions, availability zones which at least on-prem. Now, I've never worked at Amazon, so I don't know exactly what the physical analog is, but I'm betting it's very close to what we see on-prem in our customer infrastructures, which is that availability zones are usually racks.

So there're some abstractions that generally map to physical resources. Now, there are some others as well which are network outages. So that's another failure domain that is very, very common. I like to use the analogy of when you're browsing the web and if you click on a link and your browser spins and never renders the page, that really very lightly is just a network outage somewhere. And it might only be a few seconds, but it just lost a packet somewhere. So it can't complete the request response protocol that it was in.

So network boundaries are there. Of course, there's storage as well. So there are some failure domains both in all three, compute, storage and network. Now as we move to more container-based systems, then we have to understand where those failure domains are as well even in kind of the container space. And then when you do things like in Kubernetes, like you have name spaces. Then of course there's a relationship of the containers to the network and so on.

[00:24:02] JM: Does a cloud native architecture necessarily mean a "microservices architecture"?

[00:24:11] CD: I mean, that's the simple response. It's really funny that I sometimes fall under the trap of trying to be a little – I'm a little OCD. So I try to be very, very crisp with terms. So I probably would not have used the term microservice. But then, again, you have to be pragmatic and you have to use the terms that are adaptive in the industry.

But the term microservice is a little bit of a misnomer. It's actually not the size, because we talk about micro and we think, "Oh! It has to be this small thing." And it actually turns out that, in fact, we work a lot with customers where they have applications that predate the microservices era. So they might have a three-tier application that has the database tier. It has some services and it has a web tier, and those could be pretty macro. They could be pretty substantially sized services.

But if you follow some of the practices, like statelessness, or you're employing other practices. Like earlier I used the analogy of when you're browsing web and you click on a link and the webpage doesn't render. What do you do? You hit the refresh button. Well, the analog of that in writing software is you do retries.

So if you're employing, it's more around the patterns than the size of the services. So you could have relatively substantially sized things. Now, there are some disadvantages. So if you have a very large Java application that takes 5 minutes for everything to load, for the class path to load, and you've got large class paths and things like that. Then some of the benefits that you'd get from containers spinning up quickly, yeah, the container may spin up in fractions of a second. But if it takes 5 minutes for the Java virtual machine and everything that you're loading into it to be ready to serve traffic, then you kind of start to lose some of the benefits that could otherwise gain. So don't get me wrong. There are absolute benefits in going micro, but you can be, at the very least, cloud friendly and maybe even claim cloud nativeness if you are not so micro.

[00:26:37] JM: After several years down this microservices hype cycle, do you ever reflect like, "Why is the monolith so bad? Is it possible that we're living through a microservices industrial complex and maybe we should just be thinking about how to build our monoliths more intelligently?"

[00:26:58] CD: No. I think we're on the right path, and I think that we're going to stay on this path. I'll say a little bit more about it. I'm just grinning from ear to ear. What a fun question. What a really great questions to ask. But I do believe that this idea of microservices in monoliths are a thing of the past. What it all comes down to is it comes down to agility and it comes down to tight coupling.

When you are doing a monolith – In fact, we see this. I always forget what the term is, but sometimes people used a term that essentially says, "Well, I've broken everything up into microservices, but then I deliver. The only way that I deliver things is when I actually pull all of the microservices together into a distribution bundle, which is monolithic."

Inside that microservices, distribution is monolithic. That's really where you might – If you do that, you might still have some of the kind of runtime benefits of microservices and of cloud native, and that if you've built the systems so that each one of the components can kind of regenerate itself, just self-healing and those types of things, you might gain that benefit. But one of the things that's very, very clear is that organizations have recognized the value of being able to release software very, very frequently.

I just did a meet up last week with one of our customers, and it's Albertsons, and they talked in this public meet up, which is why I can quote it. They talked about it used to take them three months to do a release of a piece of software, but they're now releasing three times a day. What that does is it allows you, for example, to experiment more. It allows you to fix bugs more effectively and all of those types of things. There're organizations out there like DORA, Nicole Forsgren and Jez Humble, who DORA in the meantime has been purchased by Google, that have done studies and have the metrics to show the increased business value from being able to release software frequently.

I remember a new of years ago with the DevOps Enterprise Summit when Nicole and Jez were up on stage talking about the latest state of the dev ops report, that Jez said flat out, architecture matters. What he was talking about there is a loosely coupled architecture actually has direct impact on these metrics that show higher performing organizations versus lower performing organizations.

[SPONSOR MESSAGE]

[00:29:51] JM: GitLab Commit is GitLab's inaugural community event. GitLab is changing how people think about tools and engineering best practices, and GitLab commit in Brooklyn is a place for people to learn about the newest practices in dev ops and how tools and processes come together to improve the software development lifecycle.

GitLab Commit is the official conference for GitLab. It's coming to Brooklyn, New York, September 17th, 2019. If you can make it to Brooklyn on September 17th, mark your calendar for GitLab Commit and go to softwareengineeringdaily.com/commit. You can sign up with code COMMITSED that's, C-O-M-M-I-T-S-E-D and save 30% on conference passes.

If you're working in dev ops and you can make it to New York, it's a great opportunity to take a day away from the office. Your company will probably pay for it, and you get 30% off if you sign up with code COMMITSED.

There are great speakers from Delta Air Lines, Goldman Sachs, Northwestern Mutual, T-Mobile and more. Check it out at softwareengineeringdaily.com/commit and use code COMMITSED.

Thank you to GitLab for being a sponsor.

[INTERVIEW CONTINUED]

[00:31:20] JM: There are some people I talk to who think that the only thing separating us from using monoliths in a more proliferate sense is that the tooling around monoliths is not great. So if you look at Facebook, for example, Facebook has a very monolithic backend and it has a ton of tooling built to support it. And you can't find a lot of that tooling in the external world. So I don't know. I just find it an interesting thought exercise. I mean, I'm with you. I don't think this microservices thing is slowing down, and I see plenty of reasons to go in that direction. But it is an interesting thought exercise.

[00:32:04] CD: Yeah. Again, I think even just what I was talking about a moment ago, which is you have all these business benefits that are tied to releases and those types of things. Those monoliths – I mean, Amazon is well-known for releasing software into their environment, on average, every second of every day. They absolutely are not doing that with the monoliths at the backend.

Now, there're a couple of other interesting exemplars. So Netflix, it's not usual to start with a monolith. In fact, it makes a lot of sense. Netflix was a monolith. Took seven years to re-architect the thing to get microservices. Again, they're kind of the poster children. Most of the organizations I work with all have monoliths that are very, very difficult for them to not only to do new releases on, but also to retire, because there's so much business value that comes from those monoliths. But in fact is an important part of microservices, not microservices, but I would call them cloud native architectures, which is why my book is not titled Microservices, by the way. That's why it's titled cloud native, is because it's not about – It's what we talked about a moment ago. It's not about microservices. It's about cloud native.

But there are some important patterns that allow you to have a certain level of agility while you are still depending on a slow to evolve monolithic backend, and there are things live event-driven architectures where you're starting to have your services, your microservices, have their own local storage and using techniques like events, and eventing systems, and event sourcing, and there's a difference between messaging and event sourcing. Using something like events sourcing to keep the data that those microservices are using up-to-date, where the system of record might be the backend store or the event source. That's one of the interesting things.

But then those microservices can actually continue to evolve very rapidly. If weren't just on audio, I'd be sitting here turning one hand very rapidly up in the air and the other one is going very slowly. It's a little like tap the top of your head and rub your tummy type of a thing. There's this impedance mismatch, but there are actual cloud native patterns that allow you to kind of mediate and moderate the impedance mismatch between these layers that are very quick moving, quick evolving and this backend that is a little bit slower.

[00:34:48] JM: Yeah. Well, I mean, the whole monolith versus services thing is it's also – These are two extremes, where in reality you have a gradient. If you have an older company, there're probably been mergers and acquisitions and there're different software stacks in different areas of the organization. And some of them are "monoliths", and some of them are services, and some of them are like some kind of service shim over the monolith. You have like 18 services that talked to this monolith, and nobody has any idea how the monolith actually works. But, luckily, somebody had the wherewithal five years ago to architect a system of services that speak some decipherable language to the indecipherable monolith. So it's not like you ever retire the thing, you just kind of paper over it in a service-based fashion.

[00:35:42] CD: Yeah. But there's a big difference between putting a layer of services in front of a monolith, and the way that we used to when we did SOAP. Remember SOAP? Websplat or SOAP-based web services and all that.

[00:35:56] JM: Sure. I definitely heard these terms. This was like when I was getting out of college and I like heard these terms. When I was interning at eBay, I heard these terms, and then they kind of like gradually faded out of my nomenclature. So I never really had to like really learn the subtleties. I remember, there's something to do with XML and I was like, "I don't want anything to do with any of these."

[00:36:22] CD: The XML actually, I was a huge XML fan, because there was a language called XSLT that was functional in nature, and I'm a functional programmer at heart. So I loved XML from that. The syntax to me that – I don't care about syntax. It was all about the actual execution model and the interpreter. But the XML was a nuisance, but not the real point here. But we were doing these services layers on top of these monoliths. But here's the thing, we weren't – What we weren't doing that we're doing now as we move into this cloud native world was we were creating these services, but we were essentially trying – All of these services were tied back to this monolithic backend.

So if you think in that monolith backend as a big old monolithic database. Now, if one of those services needs something different from the monolith, and let's say a different service wants to just stay on the track that they're on now, but there's overlapping. There's conflicting needs from those two different services that were in front of this monolith, then you have to get all of the services in lockstep against the monolith, because it's this shared system.

What we sometimes think is as soon as we have services, they're loosely coupled and autonomous. But if they are too tightly coupled to some backend, because it a database or some monolithic system, then we only have the illusion of lose coupling, because the coupling is actually at the backend tier.

So some of these patterns around eventing and event-driven systems and every service having it's own materialized view of some source of data, those are so essential to actually achieve the lose coupling that microservices promises. We've gotten really good at doing it at the compute layer, but we haven't carried that all the way through into the layers of networking and storage. We're getting there now.

In my book, in the very first chapter, I built a mental model where I talk about cloud native, is you have to think about cloud native compute, cloud native data and cloud native interactions. Those are the three, services, data and interactions, and you have to think about patterns in all of those.

[00:38:48] JM: Well, I was talking to a guy yesterday who works on Istio stuff, and Istio – Well, and Envoy, are interesting because they are the cloud native manifestations of the load balancing and routing layer, and there's kind of the changing of the guard from the hardware-based load balancer days. The ngnx-based load balancer days, and it seems to be shifting towards an Envoy up and down the stack from your load balancers to your microservices instrumentation, which seems like a more friendly way to be doing this routing and doing this instrumentation and so on, this load balancing. That's just kind of one of these – How you just see these – The propagation through the whole architecture. Because like an architecture is a very – Software architecture, or I mean an application architecture, because you can include hardware in some context in that. It's such a wide-scoping blueprint of different things that are fitting together. So as you said, we are seeing overtime the cloud creep into and affect all the different areas of this application stack.

[00:40:08] CD: Yeah, I think that's brilliant, Jeff. That's another example of what I was just talking about with a monolithic backend being the database or some monolithic system. You've just put your finger on another one, which is to say we used to have these centralized things, like a centralized API gateway. Sure, the API gateway probably was deployed as multiple nodes. But from a logical perspective, it was a single thing, or the load balancer from a logical perspective was a centralized thing.

What we're doing is we're breaking apart not just the applications. Not just the services, but all of the things that used to wire together or topology of an application. So I see the whole Envoy, really the sidecars, and then Istio is the control plane for that as the distributed API gateway. A distributed logging system, a distributed load balancing system. I think that's brilliant.

When I first seeing some of that coming out in the industry I was like, "I work on a platform where we actually have some of these things that are architected in a more centralized way." I was like, "Oh my Gosh! It's like a distributed PaaS. So, there's a certain element of that, and it's an important part of the evolution of platforms moving forward. Yeah, super cool stuff.

[00:41:31] JM: Side note. This doesn't need to be a conversational point, because we're just talking before about how as we're speaking today, the announcement of Pivotal's acquisition by VMWare is formally announced. But that's not the only point of consolidation that's going on. I think it was yesterday that the Splunk acquisition of SignalFX was announced. That's like, to your point about the centralized logging stuff, I mean, we might see much like we saw with business intelligence like a month ago, the total consolidation within like a week of the entire business or the vast majority of the business intelligence sector. It looks like we're about to see the same thing with logging, perhaps monitoring. I don't know who the candidates are for consolidation, but it's an interesting time. It's a time of creative destruction, certainly. Thanks to this cloud thing rolling through all the markets. So, yeah, you don't need to comment on that, because I know it's the business level sensitivity. But I think it's an interesting downstream propagation.

I want to talk about kind of late stage cloud. So, obviously, we started with the EC2 and S3, and then we got some database services. We're pretty straightforward. But then we've gotten these more interesting things. We've gotten Redshift and Lambda eventually. We have SageMaker, BigQuery. These things that are born in the cloud and there's not really analogs to on-prem services. I mean, there are rough analogs, but these are really abstractions that are given to us by the cloud providers with the scalability and the constant updates and so on. All the things you get out of a platform as a service or infrastructure as a service. Well, I should say platform as a service probably more accurately, just like backed in. So you get these small abstractions with the platform baked into it and it really changes what you can do. So how have our software architectures changed as a result of these really beautiful, useful abstractions?

[00:43:43] CD: Yeah. I mean, those abstractions certainly serve the goals of cloud native software. I think without – It's always a chicken and an egg thing. Some of those things were born out of a need that was arising from these new architectures, and then the new architectures were enabled through the availability of some of those new services.

I also think it's super interesting that these things that are coming from these cloud native companies, not just cloud native software, but cloud native companies, like AWS and Google, and interestingly enough, not a cloud native company, but really kind of reinvented itself in a way that I think – Or is extraordinary, is Microsoft. Definitely got that. Is that they understood these patterns that people have to unwire these things that they have to unwire in their brains and unlearn right from the get go.

Some of these things that you talk about here, like BigQuery, and Lambda, and Redshift and even database as a service, although database as a service is an interesting one, because in many cases, the implementation of the backend were traditional databases, that they figured out how to project in a cloud native way. But that later example that I just talked about is where a lot of enterprises are today. They're so used to having their operational practices and their software depend on these systems that are not cloud native. So they have to work around that.

Quite frankly, a lot of the difficulty in many cases is that there's just this mindset around – We talked about it earlier. Stability. That was the assumption, is that infrastructure has to be stable. When I, a couple of weeks ago, was with a client and was chatting with a group of individuals and saying provocative things like, “You should intentionally cycle your containers every few days, because if there’s malware and then you automatically get rid of the malware, they were like, “What? We intentionally kill containers? What are you talking about? That’s crazy.” There’s that whole interesting thing. I think, again, it’s a cycle. It’s a chicken and egg. It’s enabling other things, and that’s the way I think technology is great. It feeds on itself and it builds on itself.

[00:46:13] JM: Taking your experience in a predictive direction. I want to get your take on where we’re going in terms of API economy. And I don’t know if you have a take on this, but low-code tools. So let’s imagine we’re thinking about the DevOps Enterprise Summit in 5 or 10 years. I think they’re going to be talking about things like the low-code tool suite, like the Airtables of the world. And some of these low-code tools are sponsors of the show. So I don’t want to seem like I’m like doing on-air sponsorships. But like Webflow, for example. That’s not a sponsor of the show. Then the API economy. Obviously, that’s expanding. You’ve got Twilios and Stripes and these kinds of things. I think these will only become more proliferate, and there will have to be some kind of standardization around how the large enterprises are adapting these things. How does the expansion of the newer suite of software like API economy stuff and low-code tools, how does that affect your thinking around software architecture?

[00:47:22] CD: So the first thing, even before I get to software architecture, is that particularly when you mentioned low-code, but also related to the API economy. I have to tell you that it drives me absolutely crazy that being able to write software is still something that is only available to the chosen few, and I’m like waving my hands up in the air. It is still too darn hard to build solutions. Imagine what will happen when anybody comes up with an idea of something that they want to be able to solve can actually implement a solution. I think that these things that you’re talking about here are the early stages of doing that.

Now as somebody who has a programming languages person, I love languages. I still feel like, and I don’t have the answer. I don’t know what it is, but I feel like we need to have a fundamental shift in the way that we view programming. The idealist to me says functional programming is way better, because you can do things like you can declare your intent. Then the interpreter of that declaration is where we do the hard, heavy lifting.

And I do believe that there's something to be said for that. I believe that one of the things that functional programming and declarative systems do, and Kubernetes is a great example, where Kubernetes has built in that hard question of where do I place my workloads. That's something that we used to have to do to, and to large extent are still doing. But now, all I have to do is tell Kubernetes, "I need five instances of this." So it takes the burden. It takes that cognitive load off of the human and says, "Let's let systems do that."

So now that I'm riffing on this a little bit, I think that you've actually put your finger on a couple of things that by having consumable resources that, by the way, make a lot of sense. I think in the past, as engineers, we have had consumable resources, but I would call them the engineers interface. So we thought about the way that we would code this or we looked at the way that we would code it and we started exposing just the class member variables.

That makes coding against those things still very hard. When you think about things like Twilio, it's starting to actually project capabilities in kind of more of a business tone and in a business language. So what are the business values of that? So if we can start to stitch those things together and make them consumable by a larger – People who don't have a degree in computer science who haven't gotten through a boot camp, then I think we're headed in that direction.

I will admit that I haven't looked at the low-code tools in sometime. The early, at least quite a number of years ago when I looked at them. It was still the same fundamental programming model, but just with a little GUI put on it. I think we need to change the programming model. I don't know where we are with that. I haven't looked lately.

[00:50:46] JM: Fair enough. So as we begin to wind down, I'd love to know what's the hardest part about writing a book?

[00:50:54] CD: So I think that the hardest part about writing a book for was that despite the fact that I teach these patterns in my day job, the actual writing of the book, not my day job. So I wrote this book at nights, weekends and on vacations. I published the book a few months ago, and this summer had the first vacation that I've had in three years where I wasn't writing a book.

I wrote when I was in Thailand. I wrote when I was in Greece. I mean, no matter where I went, I was always writing a book. So that was definitely hard, was trying to write a book as a side job. So that was hard. But I think that in terms of aside from the time, the thing that was hardest for me was that that really more of a transition that I went through was when I first started writing the book, I honestly was trying too hard. I was trying too hard to “write a book” as supposed to just express myself.

So the very first chapter I wrote was like 70 pages long and it was like uber formal and it just – It was painful, quite frankly. I didn’t enjoy the process at all. I’m a novice. I mean, I’ve written blog posts and those types of things, but I’m a novice book writer. This was my first book that I’ve ever written. So I really didn’t – I haven’t found my groove. It took me about a year. Once I found that groove and said, “First of all, relax. I’m going to let my own voice come out.” I reached a point where I told my development editor, Christina Taylor, who was just awesome, “Look, I’m just going to start writing in my own voice, because it’s more fun that way.” So that was a transition that I needed to go through, was kind of learn to use my own voice instead of trying so hard to “write a book”.

Then another element of trying too hard was that, quite frankly, I just tried to cram everything in there. So that first chapter that was 70 pages long, it was like, “Wait a minute. Do I really need to do all of these here and now, or is it going to be the best models are the simplest models?”

So just relaxing a little bit and saying, “Look, I don’t have to prove quantum mechanics or something here. I’m writing a book that I want to be practical and I want to be enjoyable to read.” So that was definitely a transition for me. Dare I say, at the end, I was having a blast. But in the beginning, no so much fun.

[00:53:41] JM: Corneila Davis, thanks for coming back on the show. It’s been great talking.

[00:53:44] CD: Jeff, thank you so much for having me. It’s been a delight.

[END OF INTERVIEW]

[00:53:55] JM: Software Engineering Daily reaches 30,000 engineers every week day, and 250,000 engineers every month. If you'd like to sponsor Software Engineering Daily, send us an email, sponsor@softwareengineeringdaily.com. Reaching developers and technical audiences is not easy, and we've spent the last four years developing a trusted relationship with our audience. We don't accept every advertiser, because we work closely with our advertisers and we make sure that the product is something that can be useful to our listeners. Developers are always looking to save time and money, and developers are happy to purchase products that fulfill this goal.

You can send us an email at sponsor@softwareengineering.com even if you're just curious about sponsorships. You can feel free to send us an email with a variety of sponsorship packages and options.

Thanks for listening.

[END]