EPISODE 1517

[00:00:01] AD: All right. John Ceccarelli, welcome to Software Engineering Daily.

[00:00:02] JC: All right. Thanks for having me.

[00:00:04] AD: Absolutely. John, you're the Senior Director for Product Management at Azul. And the best way I can describe Azul for people that haven't heard of you, but just like a focused Java optimization shop, right? Full service. You have services and support. You have product, including your own JVM. You have SaaS tools, and just things like that. But maybe give us some background on what you do at Azul? What Azul does? And how you got there?

[00:00:24] JC: Sure. Sure. I'm the Senior Director of Product Management here for Azul. And I take care of our Azul Platform Prime, which is our hyper-optimized build of OpenJDK. I've been here at Azul a couple of years now. Azul it was actually a long and storied company. We actually started out building appliances. We started out building servers, specialized servers, for financial institutions who wanted really low latency. And we would actually ship those servers and install them with our own version of Java on them.

And over the years, we have moved from doing hardware to just doing software. And then recently, from just doing software. Just doing one JVM, which used to be called Zing. Most people know us from before. Remember it as Zing and was rebranded to Azul Platform Prime.

We moved from just having that one to also doing our Zulu JDK, which is just a straight supported build of OpenJDK. And then from that, to actually providing SaaS services, which we're bringing to market today. So yeah, we've been doing Java a long time. We are the company with the largest team of developers who work solely and completely on Java.

[00:01:36] AD: Yep. I love that. And I love just like the benefits you can get from just deep specialization, right? Because most Java developers, it's not worth it for them to go super deep. But then if you just have this like core people that are going deep and can debug this stuff super quickly, and just tap into that expertise, it's so useful and awesome. One thing I want to ask you is, are a lot of people making their own JVMs? Why did y'all build your own JVM?

[00:01:57] JC: Right. Right. So, yeah, there used to be more JVM. There was a time when there were like lots of JVMs and then it kind of all standardized on the OpenJDK. And now we're seeing, again, more flavors of Java branching out. I mean, back in the day, you had JRockit from Oracle, and you had HotSpot from Sun. When those two merged, they kind of merged the two teams, merged the two technologies and brought things in. And then Sun had already open sourced Java as OpenJDK. And so, all of that technology went into OpenJDK.

Zing had been around for a long time. Throughout that, had been around as a secondary JDK. And IBM had a JDK called OpenJ9 and so forth. Things seem to coalesce pretty clear, really around OpenJDK. And now what you have today is you have a lot of different distributions of OpenJDK where the code doesn't change at all, right? All it is, is open JDK doesn't really provide builds. And there's a source code. There you go. So, how do you actually get those builds, right? So yeah, various companies have started providing their own builds of OpenJDK. If you're in a cloud, you'll get Corretto, which is Amazon's build OpenJDK. Or Microsoft has just started doing their own build of OpenJDK, and so forth. And we offer one of those as well, which is Azul Platform Core and the Zulu builds of OpenJDK that we have in there.

But those are really just repackaging and redistributing the base JDK that isn't OpenJDK. So why would you want to make a significantly different JDK? And there's only one reason to do it, because it's a lot of work, right? So, you don't do it just because you think it's a fun science project. You do it because you think you can do it better, right?

And today, what you have is OpenJ9 is still out there used mostly internally for IBM products. You have GraalVM, which is not based on OpenJDK at all. But it is a TCK-compliant Java runtime. And you have Prime, which has really been pushing the limits of what a JVM can deliver in terms of performance, and in terms of reliability, and in terms of total cost of ownership.

[00:04:15] AD: Yeah, absolutely. Can you tell me a little bit about how it's different than some of these other JVM's that are out there?

[00:04:19] JC: The important thing is that Azul Platform Prime is not like a cleanroom reimplementation of Java. It is a built of OpenJDK. It is standard Java. What we do is we take OpenJDK, and then we take a couple of the components that are in OpenJDK, and we replace them with our optimized versions of those components that we feel delivered better performance and better user experience.

[00:04:43] AD: What are some of those components? Any particular ones there?

[00:04:46] JC: Right. Yeah, it's mostly focused around the JIT compiler and around the garbage collector. And then we have some add-ons that we put on top of it that give you additional power. But as far as replacing components that were already in OpenJDK, it's the JIT compiler, and it's the garbage collector.

Let's take the JIT compiler first. The JIT compiler, this isn't Java C, which compiles source code down to bytecode. This is the just-in-time compiler that compiles your bytecode down to specific optimized methods for the exact machine that you're running on. And this is what makes Java really, really fast. Java doesn't do ahead of time compilation. Java does compilation as you're going, so it doesn't have to compile everything. And it compiles it. It optimizes these methods based on the usage patterns that you're actually seeing. It makes it really, really fast for the way your program is actually being used, rather than trying to compile something that's going to be fast for every possible usage of your program.

And so, ours is called the Falcon JIT compiler. That's the component that's in Prime. And we based this on a different technology than HotSpot. We based this on the LLVM technology that's inside of Linux. And Falcon just produces faster code, right? We do very intense optimizations. We've been doing this a long time. And we produce, on average, code that runs a lot faster. Individual transactions get executed more quickly. That means you can handle more load on a single instance. That means a fleet of what used to be a hundred VMs to handle your peak load can be handled with 70 VMs or 75 years, right? That's what the JIT compiler delivers.

[00:06:33] AD: Yeah. And is that just a drop-in replacement? If I'm using a different JVM, and I want to switch to Platform Prime, it just changes the binary I have on my machine and good to go?

[00:06:42] JC: Yeah, you download, you install, you set Java home, right? And then the thing that then needs to happen is tuning some of the JVM flags that you have on there. A lot of this is actually a lot easier. For example, garbage collection, right? A lot of what goes into tuning JVM today is tuning the garbage collection activity. And the garbage collector is another component that we swap out. We swap out – there are several garbage collectors inside of OpenJDK. What we offer is the C4 pause-less garbage collector.

And what this does is it allows you to basically perform garbage collection activities without stopping program execution. With most of the normal Java collectors pre-JDK17, that were in open JDK, these were all what we call stop the world garbage collectors, right? You execute until you reach a certain safe point where it's safe to stop the execution. And then you stop all execution, and you clean up all the memory, and you compact it, and you move it, and you update all your references, and then you start again.

Well, what happens during that stop time is that you're not responding to any requests. If you have an SLA on response times that says, "I must always, always, always respond to an SLA within 100 milliseconds." And for some of our customers, we're not measuring this in milliseconds. We're measuring this in nanoseconds, right? You think about algorithmic trading companies, right? Every millisecond is money lost, right?

Yeah. If you have that SLA that says, "Hey, I need to respond to everything within 100 milliseconds, right?" Well, then those thoughts are all garbage collections can just blow your SLA out of the water, right?

And the problem is linear to how large your Java heap is. If you have very large heaps, means lots of garbage to collect, means longer, stop the world, garbage collection pauses, right? And this has an effect on how people run their Java, right? People scale horizontally, right? They want to have – instead of having one VM instance with a very large heap, they will have lots of small VMs with very small heaps. So that at any one given time, no one of those VMs is engaged in a really long garbage collection cycle. Whereas we support like multiple terabyte heaps. We have customers running with 10 terabyte heaps in production, right?

[00:09:05] AD: That's amazing.

[00:09:05] JC: It’s amazing. It’s crazy.

[00:09:05] AD: Yup, yup. That's pretty cool. And it's amazing to think about that stuff. And just all this sort of infrastructure that's been built on Java over the years, and especially like core database type infrastructure, Elasticsearch, Cassandra, things like that. And still having that GC is like some of the issues I hear come up with that. But having one that doesn't stop the world and have those GC pauses I'm sure is a huge win for using those.

[00:09:27] JC: Yeah, we were the first in the market to do it. And we've been offering it for a long time. It's battle0tested. Its run in some of your most household names. And since then, success begets imitation. And we're happy to see that pauses garbage collection has gone mainstream. There are two new garbage collectors in OpenJDK. There's ZGC, which is zero garbage collection, I think. ZGC comes out of OpenJDK, out of Oracle. And Shenandoah, which came from Red Hat. And both take two different approaches towards implementing it. But basically, what they're trying to achieve is the same thing. Those things are coming along nicely. And we talked with the teams that are doing those and participate in the discussion about non-pausing garbage collection. Those are available really from 17 as the first place where you would want to start using them. Ours are, of course, Prime. You can get this functionality on JDK 8 and JDK 11. So, without having to undertake an upgrade to a newer JVM.

And yeah, and there's still aspects that we have already done that they haven't done quite yet. But yeah, pause-less garbage collecting is becoming a thing in a lot of use cases, especially any latency sensitive use cases. Cases that stress the heap. You got high allocation rates and so forth. It's very much a concern there. And it's an accepted way to tackle that problem.

[00:10:50] AD: Yeah. Cool. Well, speaking of use cases, I want to talk a little bit more about that. And first, just add a little higher level, like, what are the general types of applications that are using Platform Prime? And when I think of Java, I mentioned huge infrastructure components like Elasticsearch Cassandra, Kafka, Spark. Are people using with that? Or is it web apps with Tomcat, JBoss? Is it another custom build stuff? Where are you seeing a lot of adoption for Platform Prime?

[00:11:15] JC: Yeah, like I said, we started off as really targeting the financial sector and doing algorithmic trading, and anything that has like super sensitivity to latency. That was really our wheelhouse. You had a lot of first party code that was – code that was written by these people. Not these kind of Java infrastructure pieces that you pull off the shelf, but the code that was actually written there, and was very sensitive to latencies, and needed consistent execution throughout the whole runtime of the trading day.

But since then, we have really moved on to working with a lot of companies on a lot of different use cases. We still see a lot of the classic latency first party app use cases. A place we're seeing a huge shift isn't exactly those pieces of Java infrastructure that you were mentioning; Cassandra, Kafka, Solar, Elastic, and so forth. We are seeing great uptake in those.

I was just actually out at Current last week at the Confluent Kafka Conference, and we were mobbed at the booth with people being very excited about the gains that we deliver there. And really, it's been our work on the JIT compiler, I think, that has broken us out of the low latency niche that years ago we occupied and has really made us a widely applicable JVM that can deliver really great benefits for lots of people.

For example, Kafka. Kafka is a perfect example of this. Without Falcon JIT compiler, we wouldn't have had a lot of value to show there, because Kafka is not memory-bound at all. Kafka is IO-bound. It's really bound by network traffic and less by how much the heap is being stressed.

But with our Falcon JIT compilations, we can achieve on our extreme pedal to the metal, just throughput saturation. Try and jam as much traffic through this incidence and see where it knocks over. We got that up to 45% higher on Prime than you get on OpenJDK on Kafka. And those numbers were just blowing people away at Current.

So, yeah, Cassandra as well. With Cassandra, it's very interesting, because that's where you really get this SLA consideration comes in, right? Because if I take a look at running on OpenJDK, if I perform that pedal to the metal test, right? And I do that using all of the different garbage collectors, I can see that if I actually want to get the highest throughput that I possibly can, right? Because if I do that pedal to the metal raw throughput tests on Cassandra using the different garbage collectors that were in there, we see that while Prime delivers the most absolute throughput that I can get through a Cassandra instance, the next one is actually G1 and CMS, right? As far as how much throughput can I get through?

My maximum throughput using a Shenandoah and ZGC is actually lower than my maximum throughput if I use CMS and G1. But then you bring SLAs into it, and you say, “Okay, I don't want to know just how much I can run through this one. I totally saturate it. Because nobody's going to run a completely saturated Cassandra node in production, right?

What I want to know is how much traffic can I get through this instance while still maintaining my 100 millisecond SLA? We have this test suite that we've open sourced and developed an open source called TUSSLE, which automatically does just that. Basically, it takes the highest level that you attained, right? And then it goes down and it tries at different levels of that. Let's take 40% of that. Then let's take 50%. And then let's take 60%. And it measures at what level was I able to just get that throughput but make sure that my maximum time for like a 99th percentile, or my three nines, is not over 100 milliseconds?

And when you look at that, CMS and G1 are actually the worst, right? They can't guarantee 100 millisecond response time at any level, right? It's always bouncing up above that, right? ZGC and Shenandoah are much better. But if you take the combination, that one-two punch of raw throughput, plus responsiveness and SLA, we get the nearest competitor to us is Shenandoah. And we get over 2X the amount of throughput through an instance while still meeting that SLA.

[00:15:42] AD: Yep, absolutely. Man, it just shows some of the difficulty of even understanding what you're doing there. Because someone might come in and naively try something and just say, “Pedal to the metal, how much can I jam through there?” Without thinking of that flipside of also the latency and the SLA at high percentiles. And yeah, you really got to balance that.

[00:15:59] JC: Yeah, it's one of our biggest challenges. One of the things that we run into the most is just performance testing in general, and performance optimization in general, right? It's so influenced by what you're testing and how you're testing. And we do this all the time. These results that we publish, these are just the results that we can get on an artificial benchmark in an artificial synthetic environment. We've seen much better numbers than these at our customers, right? In production.

We see this also – You also see this when you go out and you just get into a new account. And they're like, “Okay, well, let me test it in my test environment first. And then if I see the results I like, I'll go out and I'll do it in production.” But the thing is their test environment, the problem that we are solving just doesn't exist in a test environment. Because the test environments been testing under artificial load. And there's just one thing happening. And there's not a bunch of network interference. And there's not a bunch of like scaling going up and down. I set up one VM, and then I ran it and off I go, right?

But then when you get into production, that's where you really see the blips, the freezes, the pauses, the things that you're trying to solve, right? And that's where you really see the benefits. So yeah, what you're testing and how you're testing it is a big deal. And I know in the industry, lots of people spend lots and lots of time moving artificial and incorrectly-formed test metrics that end up not having the desired effect on the actual performance of their app in production.

[00:17:38] AD: Yeah, truly. And it's hard to even trust benchmarks that get put out there. Because it's like, “What are you testing? Are you an honest broker? And some of that stuff. I guess, when someone's looking to adopt Platform Prime, does that often involve some support from your team to help do some realistic testing and tuning and that sort of stuff? Or what does that engagement sort of look like?

[00:17:58] JC: We have, I think, one of the world's finest teams of security – Security as well. Because that's a place we've been working in a lot. But Java performance engineers. And Java performance tuning, it's hard. Java performance testing is hard. You can obviously come to our website. You can download the builds for free to test and using to testing and development. And we have lots of guides, tuning guides, and so forth set up. But we really recommend to just get in touch with us, because you can work with our folks to really get those flags right. Make sure you're measuring the right thing. And make sure that you're getting the results that you want to see.

[00:18:37] AD: Yep. Have you had any difficulties as just different workloads change? We have latency sensitive stuff, like Elasticsearch and Cassandra. But then also just enormous data processing with Spark or different things. Have you had any particular workloads change over the years that are particularly challenging or anything like that?

[00:18:53] JC: Yeah. I mean, the types of workloads people are interested in have definitely changed. The types of things that people are doing with data in motion, and streaming pipelines, and on the fly analytics of data, and so forth. These are things that we just weren't doing 10 years ago, 15 years ago, or we were doing in very different ways. And also, the technology itself changes, right? Teams or ISVs that would have spent a lot of time in the past tuning their applications to get good latency or good throughput. The existing open JDK and Java tools can suddenly find if the tools themselves, if the JDK itself, is handling that latency better that they can kind of go off and spend time elsewhere.

[00:19:43] AD: Working on those things. Yep, absolutely. What about just the shift more and more to the cloud over the last 10 years? I guess, what is your customer mix look like? Is it a lot of on-prem? I know I'm sure the financial and sort of algorithmic trading a lot of on-prem type stuff. Do you have a lot of cloud stuff as well? Or what's that look like?

[00:20:00] JC: A ton. A ton. I think everybody's going through the transformation. Everybody's moving out to the cloud. It's less lift and shift from what we've seen. It's less of just like let me just take this thing that I had running on-prem and put it on the cloud. And it's more of rearchitecting for scalability and to take advantage of a lot of the things that are in the cloud.

One of the interesting things that we've put out there is a cloud native compiler, which basically that JIT compilation service, rather than running it on the JDK itself, now – or if you're on the cloud, and you've got expandable capacity and so forth, we can actually off source that out to a dedicated Kubernetes cluster that does the compilations for you.

Because think about it, to me, JVM's are kind of like that guy in the movie Memento, right? He wakes up and he's like, "What am I doing? I don't know. I guess I'm doing this thing. Okay. Da-da-da-da." And a couple of minutes passes on, "What am I doing? I guess I'm doing this thing, right?" And it has to do that work. JVMs are like that, right? They wake up and they go, "Okay, great. What am I doing? Oh, what's this? Kafka? I guess I'm doing this thing called Kafka. I've never heard of it. Right? Okay. Well, let me start seeing how you use it. Okay, this is the way you use it. And how you – Okay, I'm going to compile these optimizations. And I think these are going to make you run really fast." And then the machine goes down, and all that information is lost, right?

And now imagine you've got a thousand Kafka nodes, right? Each of them running the exact same program over and over and over again. And each one of them waking up, like, "What am I doing now? Oh, I guess I'm doing this thing." Da-da-da-da. Like, wouldn't it be nice if, since you already know that you've run that thing a thousand times, if you had some sort of record somewhere of what were the optimizations I needed to perform? And gee, wouldn't it be nice if instead of a thousand JVMs having to reserve capacity and eat up their CPU to perform those optimizations, if there was some other cluster somewhere that could say, "Hey, you know what? Right now, I'm seeing a lot of compilations. I'm going to scale up to a lot of capacity. And I'm going to do all these compilations for all these guys. And then I'm going to scale that capacity back down," which means I don't need to reserve that capacity for the entire life of the runtime, right? I can actually scale down my machines to just handle the application logic, and I'm outsourcing my stuff elsewhere.

And wouldn't it be even nicer if that component had a cache and actually kept the compilations that it did? So, if you request the same compilation again, I don't have to do it again. I can just serve it up. And that's what we've built. It's the cloud native compiler. You don't necessarily need to run it on a cloud. We've got people who are testing it on-prem as well. But it just means that you're scaling out this thing that every JVM needed to do all the time and reserve resources for all the time. Now, those JVMs can just offload that. Only have the resources they need for running our application logic. And they can reuse compilations that have happened in the past.

[00:22:58] AD: Yeah, that's fascinating. And I love just seeing all the changes that happen as a result of these other changes. The elasticity of the cloud and that sort of thing. And what we can use that for in different ways. Cool. Any other use cases, examples, you can tell us of people that sort of picked up Platform Prime? Some issues they were having? And how Platform Prime was able to help them there?

[00:23:18] JC: Yeah, sure. I talked earlier about multi-terabyte heaps, right? And that is one of the things that a company Taboola does with us, right? Taboola is a recommendation engine, right? They do about 30 billion recommendations daily, right? They were having real problems with latency. They ran a lot of Cassandra on us. They run a lot of first party code, and also these ISVs, like Cassandra, and so forth on us. And they were having like up to 15 full minutes of pauses where the servers would just stop responding, right?

And threw in Prime and turned it on. And what they were really able to do was that vertical scaling I was talking about, right? You can't do this without a pause-less garbage collector, and especially it pause-less garbage collector that's multi-generational and has really been tested and been through the trenches.

And they moved from having lots and lots of nodes with small heaps, to running monolithic nodes with 10 terabyte heaps on them for Apache Cassandra, for NameNode on Hadoop and so forth. And you just can't get that using plain Vanilla Java. That's been very exciting with them.

We also are working with retail companies and so forth, where we're really pushing how much can we bring down your total cost of ownership just by being able to run faster code by having a higher carrying capacity under your response SLAs and so forth. And really working – and that's been on everything from the ISVA appliances, like Kafka, Cassandra and so forth, to first party code. We work heavily in the financial industry, obviously. We've done fraud detection. MasterCard does all of their fraud detection on top of us with Platform Prime, recommendation engines, SaaS providers, and so forth. We've just been seeing great adoption across the industry.

[00:25:20] AD: Awesome. Yeah. One thing, how much of that – Like, I know, there's just so many knobs I can tune in a JVM, whether that's yours or a different one. How much of those improvements are tweaking some of that tuning? And how much is the core changes in the JVM and the garbage collector and things like that itself?

[00:25:35] JC: Yeah. The core changes in the JVM are what enable it all the time, right? All the time. Without the Falcon JIT compiler, you wouldn't be able to get the faster curve. Without the changes that we make to open JDK, you literally would not be able to achieve the types of numbers that we're seeing and the types of infrastructure savings that we're seeing.

Having said that, tuning is tuning. And you do have to tune, right? And there are certain situations where it depends what your primary concern is in that situation, right? There're people for whom the primary concern is not so much how small of a VM can I fit this app on? But their primary concern is, “Hey, from the moment I bring a VM online, I need this thing to run fast.” We've got some major game makers who do massive multiplayer games, right? And when they see peak times, they were seeing lags when they brought in new JVMs, right? Because in the first 5, 10 minutes, the JVM was still slow, because it was still like warming up and so forth.

And we have this other technology called Ready Now, in which we basically record that profile. Like I said earlier, we record that profile of what are all the optimizations you need to do. And then we front load those optimizations. By the time your JVM is starting to accept traffic, it's already warmed up, right?

When you're tuning, it depends on what you're tuning for, right? At the same time, for some people, what they really need is just the latency solution, right? They just need to bring down latencies. And they're not so concerned about how much throughput they can actually get through it. And they're more concerned about the latencies. And they're concerned about how much CPU they're burning or so forth. At which case, you might like turn down some of the optimizations so they don't burn as much CPU. And you might have different configurations of it. The tuning is still there, I mean. But you just have a lot more that you can do with Prime as far as the tunings in order to get whatever it is you're interested in; response latency, total throughput, smallest possible VM size, time to first transaction, whatever your main concern is, right? We've got levers in there that you can tweak to get the best outcome on that.

[00:27:48] AD: Yep. Cool. All right. So, just to close out here, I want to look a little bit in the future and what you see going forward. I guess, like, what excites you? Anything new you're seeing either in the Java world or at Azul that you're sort of excited about?

[00:27:58] JC: Yeah, sure. Well, I think the ARM 64 chips that are coming out now are a huge game changer, right? Amazon Web Services, already coming out with its Graviton 1, 2, 3 chips, which we support now. And we've seen great performance on those.

I think the interesting thing about Graviton is that people are seeing, obviously – absolutely seeing the value for machine learning algorithms and so forth. And those architectures have been tuned very well for that. The interesting thing on something like Java is that we've been optimizing Java for x86 chips for a long time now. And we've been optimizing Java for ARM 64 chips on a couple months now, right?

The interesting thing is, as you would expect, and this is for all JDK versions, you actually get lower throughput on Java when you run on ARM 64 than when you run on x86. When you're doing your calculations, you have to make sure, when you're looking at it, you're going to – Most people look at it and they say, "Oh, Graviton 3 instances are 30% less than the old x86 instances. That means I'm going to have 30% less CPU cost." Well, not quite, because you got to build in another padding for the fact that you're going to have to have a couple more instances to serve the same traffic. This is a completely temporary situation, right? And both the Open JDK team, and the HotSpot team, and the Prime teams were all working very significantly on getting it to hum as fast as it can on Graviton ships. Having said that, of course, our Prime JDK runs faster on Graviton than OpenJDK for sure.

I really like what's happening there. And having gone to current and just really seeing the whole paradigm shift on how we deal with data. And going from data at rest to data in motion. And how that unlocks so many use cases and so many ways to do things at the massive scale that we're doing things that now is very exciting as well.

[00:29:57] AD: Yeah.

[00:29:57] JC: Those innovations also inside of JDK, and the language, and the API's that are available and so forth, which are very exciting as well. And we love implementing those jets and getting them out to our customers and making sure that everybody can play with all the new goodies in the JDK.

[00:30:13] AD: Yep, I totally agree. I see a lot of energy around ARM. I see, like you're saying, with the changes in data and Java underlying so much that I think there's a lot of interesting stuff happening here.

Hey, this has been a great conversation, John. I really appreciate it. And thank you for walking me through some of the stuff about JVMs and optimizations you're doing at Azul. If people want to find out more about you or Azul, where can they find out more about you?

[00:30:35] JC: Well, just come to azul.com and check us out. You can download Azul Platform Prime builds there. And check out our blog. We blog quite a bit about various things of interest to the Java community, whether it's obscure algorithm and how you do garbage collection, to very mainstream things, like benchmarks and new features in Java. It's a great resource.

Another great resource that I'd love to plug is Foojay, which is Friends of OpenJDK, which is a real community. I mean, we participate in it. We sponsor the hardware, basically, to keep the site running. But for the most part, very community-driven effort to really talk about what's happening in the JDK? What's coming up? And so forth. That's a great resource as well.

[00:31:17] AD: Nice. Thanks. I'll check those both out, especially that Azul blog. I love it when experts really go deep on some of these optimizations and teach stuff. Thanks for sharing those. John Ceccarelli from Azul, thanks for joining Software Engineering Daily today.

[00:31:29] JC: All right. Thank you, Alex.

[END]