EPISODE 1018

[INTRODUCTION]

[0:00:00.3] JM: Full stack JavaScript applications have been possible since the creation of Node.js in 2009. Since then, the best practices for building and deploying these full stack JavaScript applications have steadily evolved as the technology has evolved. React.js created a consolidation around the Vue layer. Then the emergence of AWS lambda created a new paradigm for back-end execution. Serverless tools such as DynamoDB offer auto-scaling abstractions in the database layer. CDN such as Cloud, Flare and Fastly can now do processing at the edge. All around the ecosystem, there are changes and these changes have affected how people want to deploy their applications, particularly full stack JavaScript applications.

Brian LeRoux is the Founder of begin.com, a hosting and deployment company built on serverless tools. He's also the primary committer to Architect, a framework for defining applications to be deployed to service infrastructure. Brian joins the show to talk about his work in the JavaScript ecosystem and his vision for begin.com.

Brian's also speaking at Reactathon, which is a San Francisco JavaScript conference taking place March 30th and 31st in San Francisco. This week is all interviews with speakers from Reactathon. If you're interested in JavaScript and the React ecosystem, then stay tuned. If you hear something that you like, you can hear more at the conference in person.

You can also hear more podcast episodes about React by listening to the Reactathon Podcast, which is available at reactathon.com/podcast.

[SPONSOR MESSAGE]

[0:01:44.3] JM: Today's show is sponsored by Datadog, a scalable full stack monitoring platform. Datadog synthetic API tests help you detect and debug user-facing issues in critical endpoints and applications. Build and deploy self-maintaining browser tests to simulate user journeys from global locations. If a test fails, get more context by inspecting a waterfall visualization, or pivoting to related sources of data for troubleshooting.

Plus, Datadog's browser tests automatically update to reflect changes in your UI, so you can spend less time fixing tests and more time building features. You can proactively monitor user experiences today with a free 14-day trial of Datadog and you will get a free t-shirt. Go to softwareengineeringdaily.com/datadog to get that free t-shirt and try out Datadog's monitoring solutions today.

[INTERVIEW]

[0:02:53.2] JM: Brian LeRoux, welcome to Software Engineering Daily.

[0:02:54.9] BL: That's awesome. I'm excited. Thanks.

[0:02:58.0] JM: Yeah. It's great to have you. You were part of the JavaScript community from the early days. How did you become involved in JavaScript?

[0:03:04.3] BL: I was a web developer really early. That Internet thing just happened right when I got out of high school, around '96. You couldn't really avoid JavaScript if you became a web developer. Those early days, it was not cool to be a web developer. That solely focused on front-end. I don't think people actually really would have said, "I was a JavaScript developer in 2003 or whatever."

It was not until around Google Maps and maybe Gmail that it became a thing. Yeah, I just loved building software. I mean, I don't really self-identify as one particular tribe. JavaScript is obviously the runaway train that we're all on now. I was lucky to be a part of the web early on. It wasn't really a conscious choice or a thing, so much as I'm old.

[0:03:52.8] JM: When Node came out and developers started building server-side JavaScript, were you skeptical at first? Were a true believer from the beginning?

[0:04:00.5] BL: Oh, I was all in. There was a bunch of us looking for this thing quite early. The concept of Node was predicted well in advance of its actual existence. There was a number of people that realized that there were these huge companies, largely incentivized to work on the

VM. You can't have a tab open and then pause and wait for the JVM to start up. When you open a new tab in a browser, it's got to be instantaneous.
That cold start was a really big key problem for early cloud and early orchestrated scaling. A lot of the big vendors that were out there that were building browser, or JavaScript runtimes into the browser realized this and we knew that eventually, there was going to be a server-side runtime to complement it. Early on, a lot of people actually thought it was going to be this thing by a guy named James Duncan. A camera was called as Server JS or something like that.

James had a good approach. I used it a little bit and I liked it. There was nothing called Jaxer at that time. Ryan Dahl's approach to Node was just immediately the right thing. Everyone as soon as they saw it was like, “Oh, that’s what it's going to look like.” A lot of previous attempts we're trying to model after what was popular at the time, which were these big runtimes, like the JVM and the .NET runtime, whereas Ryan's approach was more Linuxy, or low-level primitives that you could compose together. Yeah, it hit, but it hit with a thunderclap and that was it. Everybody was just like, “Oh, that's what's going to look like. That'll be the one.” Then, yeah.

[0:05:25.8] JM: Do you have a perspective on why React became the most popular front-end framework?

[0:05:31.1] BL: That's a great question. I suppose it addressed a really major core problem. I think what was really interesting about when React came out was it wasn't popular. A lot of people immediately eschewed it for its uniqueness. People saw it and they thought, “Well, that looks like something we've seen before, which was E4X and that was a disaster in the browser, so there's no way this will be a thing.”

Facebook was very persistent in keeping the message on about what the benefits were. They kept showing it to conferences and talking about immediate render. At that time, the JavaScript community somehow suffers from repeated collective amnesia. At one point at that time, people were talking about two-way data-binding, like it was a good idea. Two-way data binding was a bad idea in the 80s and it was a bad idea in the 90s. We didn't really remember that, and so we've tried to re-implement it. I think things like Angular and Knockout and Ember were trying to popularize this idea of two-way data-binding.

React really pitted itself against that with this mediate render concept. When you treat your DOM like a game loop, declarative programming and immediate render is not news, that's something we've been doing in video game programming for a very long time. It just eliminates a huge class of bugs due to state. I think it became popular for that reason. It was a little bit different and it solved one really hard problem really well.

[0:06:53.7] JM: Has the popularity of React had downstream effects on middleware and back-end?

[0:07:00.1] BL: What a great question. It absolutely has. We see people now transpiling their back-ends, which is fascinating. People are building whole architectures and deploying this stuff to the cloud in a dialect that is not JavaScript in a third-party dialect; worse, a third-party proprietary dialect controlled by an advertising company that sells ads to hostile foreign governments. Aside from that, it's great.

That third-party dialect has nothing to do with your back-end. But because your front-end requires it and you want to have interop between your templates and your back-end, they're transpiling the whole lack. I think it's having the predictable result. We're seeing back-ends get a little complicated with this extra weight. Really, it's all in service of just being able to render a template, which is pretty funny.

[0:07:48.0] JM: You spent a few years as part of the Apache Cordova community, which enabled people to write mobile applications with JavaScript and HTML.

[0:07:57.4] BL: Yeah.

[0:07:58.1] JM: This was back in 2012 and 2014, that span of time. What kinds of applications could you build with Cordova and where did the framework hit its limitations?

[0:08:10.8] BL: Yeah, that's a great question. Cordova came from PhoneGap. What happened was we had built PhoneGap at around – I think we started working on that in 2008 and we were acquired by Adobe in 2011. The broader open source community was really scared that Adobe was going to try and kill it, and so were we.

We made a deal with Adobe that we wouldn't kill it, that we would donate that source code to the Apache Software Foundation, so it would remain open and free. This is a super well-known story, but I think it's been long enough that I can tell it. At that time, we were in litigation with the gap. The original PhoneGap logo was Helvetica. At that time, it was trendy to bold half your word, if you put two words together. We left it Helvetica Phone and bolded Gap. Guess what bold Gap Helvetica is?

[0:09:03.4] JM: It looks like the clothing company. Yeah.

[0:09:04.4] BL: It was literally the Gap's logo. Apache was very gracious and wanted to accept our source code, but did not want to accept our litigations. We had to pick a name really fast. I lived on Cordova Street in Vancouver, so that became the name of Apache Cordova.

[0:09:21.2] JM: Could you just change the logo?

[0:09:23.5] BL: Yeah. It was more complicated than that. They wanted to take it to a full set of lawyers talking to lawyers. We weren't in a position to move fast enough to make all that go away, so we were just like, "Fuck it. Whatever. Just find and replace the name and give it to Apache under a different name." The downstream distribution can be PhoneGap. We ended up selling with them and it was all fine. I think that the deal we made was that we couldn't make any apparel. We almost vindictively named it Apache Khaki just to be snarky, but we didn't do that.

Anyway, so what did it do, it let you take HTML, CSS and JavaScript and build native apps for the phone. We achieve this by embedding onboard web browser and then just full screen it and letting you build your app.

By the time I left in 2014, about 20% of the app store was using this technique or technology. It definitely had a big impact. I think a lot of people are mad about it still, as popular things get in software. Some people like it, some people don't. You can build a bad website, that's Sturgeon's law. We know 90% of the Internet it's not great. Yeah. Of that 20% of the app store, I'm sure

90% of it was also bad. Cordova and PhoneGap garnered this reputation as not being the greatest platform for building for native devices, which isn't true.

You can build something bad with any technology. You can build something great in any technology. Early versions of Instagram were embedded WebViews and it did okay. It's possible to build a great thing with these things.

[0:10:57.7] JM: I just wanted to get a little bit perspective on your experience with Cordova, because I understand that it works to a certain extent, but my sense is that React Native, Flutter, these technologies have made cross-platform mobile development a whole lot more of a reality.

[0:11:21.3] BL: Oh, probably. I imagine they have. I mean, it was a reality before too. I think all of these things in aggregate might add up to a small percentage of websites. Good for them. What I see now though is that the web is coming back into its own and that would be your default distribution channel. You wouldn't build a iOS app, because you'd be limiting your audience just so much. There are cases for that. It makes sense to do that. I'm not saying don't do that, but I am saying that you would probably want to start with the largest distribution channel possible for your software and that would most likely be the web.

[0:11:54.4] JM: Do you have a sense of what – if there were any technical bottlenecks back in 2012-2014 timeframe that have been eased up on today? If you're talking about cross-platform mobile development.

[0:12:07.9] BL: Well, I'm trying to think of examples. Apple used to have this 300-millisecond tap click delay on links. There are just lots of small things that have changed over the years and an aggregate have added up to a much better experience. Skia was this thing that Android slowly rolled out for their rendering layer, which has made a huge difference for animations on the web.

I think actually WebKit is now a little bit behind as far as the mobile web goes, which is pretty sad. Apple was a real leader there in the beginning. Across the board now, I just don't know why you would default to choosing building a native mobile app personally. I don't know what the use case is either. If your use case is edge machine learning, then obviously, yes. That would be a good way to do it. If your use cases display words on page, unless you're a restaurant, I don't

know why you wouldn't use the web to do that. If you're a restaurant, you would publish a PDF through a flash website obviously. Yeah, that's my view of it.

In tech, we have this thing where we think that there's this winner-take-all and that there's this mutual exclusion. As soon as Flutter came around that everything else was going to go away. That's actually not true. There's no real case of winner-take-all. There's winners take most, that's for sure. There's a power law.
Tech is not exclusive and mutually exclusive. Old tech doesn't go away. It's all additive. Cordova's existence didn't negate the existence of previous tech, any more than Flutter's existence negated the existence of React Native, or whatever else came along. These are just options.

[0:13:40.4] JM: Around 2015, you became interested in lambda. The original use cases for lambda were mostly around glue code. Did you believe even back then that lambda could be used as a main back-end server?

[0:13:53.4] BL: Oh, fully. Yeah, it blew my mind. In 2012, the predicate for our acquisition to Adobe wasn't that PhoneGap was awesome, even though it super was. The predicate for our acquisition was that we had a lot of people using a service that we had built called PhoneGap Build. PhoneGap Build is a hosted cloud service. You upload a zip file of HTML, CSS and JavaScript and then you can download an IPA file or an APK file. We did all the compiling ourselves. Magic.

That magic turns out to be a rails app that I built in 2010, that we just slowly bolted stuff on and duct tape and Band-Aids and the prayers and hopes. It got bigger and bigger and bigger as it became a part of the Creative Cloud, it started to pump some serious major, major traffic, like I'm talking millions and millions of builds. I learned a very valuable lesson. I learned that I will never do that again.

As soon as I saw API gateway get announced I was like, "Oh, thank God. I'll never have to load balance another monolith again." I was immediately there. I think this is a common story. The people they get this whole surplus thing have suffered by the old ways of doing things. The people of that haven't suffered by the old ways of doing, things just think this is obvious. Yeah,

of course that's how you build it. It's going to be easy. Just put this thing up on the Internet and it'll scale forever.

It's never been that easy. Lambda was my first glimpse into what a stateless, isolated runtime architecture would look like. As soon as they said they'd take care of the HTTP lair, I knew that was it.

[0:15:24.2] JM: You make a good point. We don't really deal with scalability issues anymore.

[0:15:28.4] BL: Well, I think we do. Now there are people issues. We've realized that the technical side was actually the easy part. I think building out a very large team that can collaborate on these small interlocking pieces is still very much an unsolved problem and something that we're figuring out. We're getting there and we're creating technical solutions, like service discovery. These are old problems and we're getting there, but it's a coordination issue now more than anything else, I think. It's not really how. It's who and what.

[0:15:59.3] JM: How did lambda change your perspective on how to build back-end architecture?

[0:16:05.2] BL: It was a bit of a cold shower. I did the thing that everybody else does. I just did it a little earlier than most people. I took lambda and I've made a greedy proxy at the root and I put a web server in it and it works. You want to put express inside of a lambda function, you can do it. If you want to make that lambda function accept all incoming traffic, you can do it and it will scale. It'll scale great for a hello world.

Then as soon as you start adding code to it, it's going to get progressively worse, because cold start is directly correlated to payload size of your execution environment. If your execution environment is small, cold start is going to be pretty fast. If it's less than 5 megs, we've measured it's usually sub-second. I don't know of any sub-5 megabyte monolith. I'm sure they exist. I don't think they're going to have a lot of functionality.

This is the trap. A lot of popular frameworks popularized this trap. Our stuff doesn't do this, because we learned this lesson early. If you put everything in one function as that function grows, it's going to get slower, you're going to have worse cold starts and that'll fail eventually.

You can't fix a quilt start with pinging it. This is another myth that's been propagated. If the container gets started, then you just keep it alive and that'll work, but that's not actually thinking about how this thing works. These things scale horizontally.

You're just keeping one container alive. You're still going to get cold starts for the other 4,999 possible provision capacity lambdas that you have. Pinging it just changes the probability of you getting that cold start, but it doesn't fix a cold start. The solution is the same solution for all complex problems; you break it down into smaller pieces and you eliminate that cold start.

[0:17:51.2] JM: Cold start is still an issue today?

[0:17:53.5] BL: It's an issue for people that build lambda functions bigger than 5 megabytes. It's not an issue for anybody else. It's an issue with the other cloud platforms that aren't Firecracker-based, I should say. Things that are K-Native based or whatever are going to be slower, because they are.

[0:18:09.2] JM: Let's get into that in a sec. Actually, what you said about the fact that you can't just keep a single lambda around and keep paying it. Then as soon as you need to scale, scale up then. Why doesn't that strategy work?

[0:18:27.4] BL: Because lambda scales for you, right? You can control – it's called your concurrency. How many instances of the container that they boot up. It's actually a micro VM, but whatever.

[0:18:38.0] JM: As in scale to one, or scale to two, scale to three.

[0:18:40.6] BL: Right. Your default is a 1,000 I think. Generally, the first time you call support they'll bump it to 5,000.

[0:18:46.8] JM: You max is a 1,000.

[0:18:48.7] BL: Mm-hmm. Mm-hmm. That's for your whole account too, which is worth noticing. If you have a whole bunch of lambdas, one lambda can DDoS the rest of them.

[0:18:55.7] JM: This is also the scale up, right? This is not necessarily the scale – now they have a scale to a minimum, right? You can keep three lambdas running in the home times if you want to for a single –

[0:19:06.9] BL: That's the keep alive, warm ping start thing. That's great for legacy architectures that are running jar files or whatever and how bad cold starts. That's a good idea. It defeats the purpose in my view. The idea of this whole serverless thing would be to scale to zero. The default should be that if I'm not using it, I shouldn't be paying for it. I shouldn't have provision capacity sitting there.

This is the problem with a lambda warmer, the pinging thing, is you're just keeping a container alive. You didn't change the – the cold start is still going to happen. You're still going to get it and somebody's going to get it. You won't see it as often.

[0:19:43.3] JM: Right. You won't necessarily get the cold start on the first end-users. You will get them on every time you need to spin up a new container, whoever is hitting that new container, whoever's getting routed to that new container is going to hit the cold start.

[0:19:57.5] BL: Exactly. This has been the I think the biggest stumbling block for a lot of people, because we're used to building these monolithic architectures where we have these large balls of code that we deploy all at once. As soon as you embrace breaking the code up, you can deploy it in parallel, which is obviously a lot faster and then the cold start is also faster, so there's less payload to start up. We just haven't really internalized building this way yet.

[0:20:24.4] JM: Firecracker, this is the open source virtual machine system that AWS released sometime last year?

[0:20:35.2] BL: Yeah. They released it at re:Invent 2018. It was a big surprise. I wasn't expecting them to do this. I don't think anybody was really saw that one coming. It's the first major in for a core piece that I know of that Amazon's really gotten out there with. Yeah, it would be competitive with things like Kubernetes. It's an orchestration engine that's –

[0:20:56.3] JM: It's with Knative, right?

[0:20:58.0] BL: I don't even know, because I don't care, honestly. It allows you to start micro-VMs. We could totally get into the weeds about how that's different than a container. It's got a better cold start and isolation property.

[0:21:10.5] JM: Let's do it. Let's go deep. Give me as deep as you know.

[0:21:14.8] BL: Well, I would put it on the listener to go check out the re:Invent talks about Firecracker and come to their own conclusions, because I think this will raised hackles, especially with people that are big on Kubernetes. I'm not saying this is Kubernetes going away and I'm not saying it's bad and I'm definitely not saying you're bad if you like Kubernetes. It's fine. It's totally okay. There another thing out there with better characteristics for isolation and startup and that would be Firecracker.

[0:21:41.1] JM: What makes it better?

[0:21:43.0] BL: I think it's smaller. I think there's just less stuff. It's written with the sole purpose of being stateless. It's written in Rust. That's all I got for you. I don't really know what to say. The benchmarks are out there. The codes are out there. It's all free and open. It's used in production with this small company called Amazon running this small service called AWS lambda, which apparently makes up 60% of the traffic on EC2. Probably worth looking into.

[0:22:10.3] JM: Well, sure. But the API for accessing it, in the developer experience. The fact that Kubernetes is used largely for long-running services, which are not lambdas –

[0:22:27.8] BL: Totally.

[0:22:28.2] JM: I mean, it seems a very different application than Kubernetes.

[0:22:31.8] BL: Well, it's load balance orchestrated operating systems, so you tell me. I don't know. Maybe they are very different in that regard.

[0:22:39.0] BL: I mean, if it's closely designed with lambda in mind, lambda are supposed to be these flaky spin up functions that are –

[0:22:50.7] BL: Fargate also runs on it too, their other thing. Yeah, I agree. I don't know that we want stateful long-lived workloads. I mean, I think we've come to the conclusion that's a bad idea. We lose parallelism, resiliency. Data is harder to guarantee that you're going to keep it around. Lots of problems. Keeping a stateless operating system or runtime architecture seems like a pretty good idea.

[SPONSOR MESSAGE]

[0:23:24.4] 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.

If you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW CONTINUED]

[0:25:12.7] JM: Do you know anything else about what they did to speed up the lambda spin up? When they swapped out Firecracker under the surface for whatever there was before, what specific advantages were –

[0:25:24.3] BL: I think it's just less stuff.

[0:25:25.6] JM: Less stuff.

[0:25:26.4] BL: Yeah, it's just less stuff. It's a smaller, lighter weight thing that sits right on top of – Well, they say it runs on top of the metal, but I believe it runs on top of EC2 virtual machines and then itself runs VMs. They are very specific. These are micro-VMs and not containers. I'm not really sure what the exact distinction between those realities would be personally. If you go and look at all their literature about this stuff, everybody uses the same terms for these things roughly.

I don't know. What I do know is it's faster and it's also open. I also know that I'm never going to run it. Running Firecracker would be equivalent to me like running Kubernetes. That's work I'm just totally uninterested in doing. I'm not saying that people don't have reasons to do that. I'm not saying that it's bad. I'm not saying that you're bad if you do it. I totally get it, but –

[0:26:15.5] JM: I've seen a lot of advertisements that tell me that I need to run Kubernetes. I'm pretty sure I got to run Kubernetes. Some of those advertisements have been on this show.

[0:26:23.9] BL: I don't know how I've managed to avoid this. Yeah, I haven't. I'm speaking from a place of ignorance. Perhaps I'm missing out. But yeah. I don't run it and I don't know anything about it. I'm not using it. I'm really happy not running Firecracker too. I'm a very happy customer of lambda. It's totally fine for me.

[0:26:42.0] JM: What aspects of a fully serverless back-end should be fulfilled by lambda and what should be fulfilled by rich services, like API gateway?

[0:26:53.6] BL: Well, you want to offload as much as you can. I mean, that would be just the general predicate, I think, when we're thinking about a software project. It's that old Larry Wall saying that a good programmer is a lazy programmer. We want to outsource as much of this stuff as we can. Now this can also be bad if we're building our applications with as many third-party solutions as possible. That's also very risky. I think there's a balance to be struck. I don't know. If I'm a regular line of business out there, like I'm an auto body shop conglomerate of some kind, running data centers is not my core competency. We can generally agree on that now. Most people would agree if you're fixing mufflers, your job isn't fixing DCs.

This is just the logical conclusion of that whole thing. If we keep taking that further and further, why am I running pods, or instances, or VMs, I shouldn't be doing that either. I should just be focused on my runtime logic. Hopefully when I deploy it, all the rest is magic and it just scales. That's a wonderful vision for the future. I don't know if that future truly exists, because there are times when you got to drop down and go low-level. There are circumstances where you can't outsource all of the undifferentiated heavy-lifting.

Some of those circumstances might not even be technical. It might be as simple as that Amazon is a competitor, or whatever. My general vibe is lambda is your last knife in the toolbox and you have a whole bunch of other tools available to you. If you can do the job without lambda, please do, because all that runtime logic that you're writing is liability. Just like configuration is liability. Runtime logic is the worst liability. There's a service you can use that does the thing better. Outsource it absolutely.

[0:28:42.5] JM: Tell me more about the stack of AWS services that every AWS user should know. What is the prototypical AWS stack, or the must know AWS stack?

[0:28:54.6] BL: Well, I mean, that's a great question, because I think a lot of people come to the table with these cloud providers and they think, “Oh, my God. Way too complicated. There's no way I can deal with this. There's no way I can grok this. I don't blame them. When I open up the

AWS console, I have a minor panic attack if anything's changed, because I'm like, “Oh, shit. I might not be able to find anything again.” It’s just too vast and too big.

How do we deal with a big complicated problem? We break it down into smaller constituent parts. Amazon's really no different. You can treat it like a big complicated problem. It turns out they have a lot of solutions and you probably don't need them all. If you're building a web application, you're not going to need SageMaker is their machine learning tool, so you can just not worry about that whole suite of tools.

For my problem space, the things I'm interested and I typically am doing web app development, I think there's eight services that you need to build fully serverless web apps. I think you need cloud formation. As you want to build this stuff with infrastructure as code, you don't want to outsource that to a third-party. I think you need CloudFront, which is their CDN. It's the oldest CDN. It's got over 220 points of presence, so it's pretty good. I think it's the most points of presence.

You need application server of some kind. I'd say API gateway is a good candidate. API gateway gives you lambda. Lambda gives you any runtime you want. You can even mix and match them on a per route basis. You need background tasks, some queue, SQS maybe. You need asynchronous unordered background task, that would be SNS, which also gives you push notifications. I've got database, got CDN, got an app server, scheduled function, so you need to run backups every hour. That would be EventBridge with their scheduled functions. Sometimes people call those cron lambdas. Yeah, those are the only primitives I can think of off the top my head.

AWS has over 300 services, but I think I only just named maybe eight. You can build an app that can scale forever and is only bound by your credit card and will cost you nothing if no one's using it. Those are pretty powerful concepts.

[0:31:05.0] JM: As far as database, you're a big fan of DynamoDB.

[0:31:09.0] BL: I am.

[0:31:09.7] JM: Why is Dynamo such a strong choice of transactional database?

[0:31:13.6] BL: Well for me, Dynamo was the database I've been waiting for. Again, I'm a little older and I've had a few databases in my time in the late 90s, early 2000s. Believe it or not, we very often would use access, which is a database technically from Microsoft. We would connect to it with a thing called ODBC and it would always fall apart. We'd upgrade to Microsoft SQL Server.

At some point, open source database started get popular, because these database licensing fees were just so high. We started to use things like MySQL and PostgresSQL. I love those databases for what it's worth, right up into the minute I had to shard one. Now that I've been through that experience, I want to manage database all the time. A managed database that I still have to shard is not managed enough.

When Dynamo came out, initially I was pretty skeptical. I didn't like the idea of a proprietary database. I didn't like the idea of a database I couldn't run locally. I didn't like that there was this weird proprietary, like a query language I had to learn. I hated all those things. What I like though is single-digit millisecond latency, no matter how many rows I have. I like automatic backups. I like that that database only pays for what you use.

I found the query language fine after a while. It's built a rough spot. They have a local development environment now. The tooling is much better and it meets all of my personal requirements. I think you get 25 gigs free in their free tier. Yeah, you're locking into a proprietary thing. Data is where all the gravity is at these cloud providers. I would rather outsource that problem until I had it later. Yes, Dynamo is expensive. I'm sure people right now are saying, “Oh, Dynamo is really expensive.”

All of the people that think that, I guarantee you have never hired a DBA, because that's expensive and that's a problem you will not have if you're dealing with Dynamo. I like it. I'm willing to outsource that part of the differentiated heavy-lifting. It's been fine for me.

The other aspect that makes Dynamo a little bit weird is you have to figure out your queries upfront, which makes model-end a little harder. It's also extremely flexible though, so it's pretty

cheap to spray this data all over the place. If you need to change things, it's actually not that challenging. Whereas I find with SQL databases, changing things can be very painful.

[0:33:36.7] JM: Do you have any issues with the idea of going all-in on AWS?

[0:33:43.3] BL: I do not. I think a lot of people do. I think that's normal. I'm trained to have the exact same reaction to things. I think it's more risky to go with one of Amazon's competitors, just based on historic precedents. I don't think it's going to work out well for them. I don't think you can disintermediate Amazon. I think they've become a de facto standard.

We can just look at the adoption of their lambda functions; Atlassian, Twilio, Netlify, Zeit, us. So many people are just running on their functions platform, which is their signature. How many S3 clones are out there? There's at least three, every major cloud player. They've already established themselves as a de facto standard, whether or not we're willing to admit that as an industry is a different topic. I think if you're worried about lock-in and you should be, I'd be very worried if I chose not Amazon.

[0:34:34.0] JM: Are their services and other cloud providers that you find particularly useful?

[0:34:38.8] BL: Well, I have issues with Amazon's onboarding experience and their development experience overall, but I'm dealing with that with our own company. I think there's good reasons to go adopt a layer2 cloud provider, or a different cloud provider if it's comfortable for you. I don't think Google and Microsoft are going to go out of business. I think the layer2 providers are actually providing a very helpful service for onboarding newer users, or younger users, or people without mission critical workloads.

Yeah. I think it's fine if those are your reasons. I'm sure if you're a Kubernetes expert that you're probably interested in whatever Google's got going on. It seems logical to me. That's supposed to be the home, or maybe Red Hat is. I don't know. Yeah, go for it. If you're a Microsoft shop and all you know is .NET, then obviously it makes sense to be looking at Azure.

I think there's going to be more than one player. I don't know that there'll be 10. Most likely, there's going to be two or three. We're pretty sure who we know who number two and one are anyways right now.

[0:35:42.4] JM: Are there situations where a developer cannot be completely all-in on functions as a service and things like API gateway? When does the developer have no choice but to use VMs, or Kubernetes, or container as a service?

[0:36:00.8] BL: Well, long-live stateful workloads would be one. I'd say machine learning, especially stuff that you're doing in-house. If you're drawing outside the lines with Tensorflow, you're going to have a real hard time doing that with a managed service right now, which I think most people that are doing hard core ML are running their own workloads. That's yet to become managed. If you're just building a database-backed web app, you have no excuse in my view. You're living in the past, wasting time.

By the way, I'm not saying this and don't feel bad. Take this as an attack on your developer world if you're doing that. I'm just saying you would be well-served to take a look at these managed solutions that take care of a lot of this stuff for you now.

[0:36:44.3] JM: If have a fully serverless application, there are a variety of ways I can define it and describe it. I can use cloud formation. I can use the serverless framework. I can use a framework that you've worked on called Architect. Describe these different methods of application description.

[0:37:02.1] BL: Yeah. Inference code is a really important concept. I'm not sure this concept is actually permeated our industry yet. Inference code is downstream of an earlier idea called configuration management. This came about in the Rails years. Rails got really popular in around 2005, 2006. Everybody jumped on board, started setting up their own Rails servers and realizing how hard it was to get Linux into a place that was friendly and possibly normal for everyone.

This was probably the predicate for Heroku existing. It was just making that part easier, because it was just so hard. Which Linux and which package manager on Linux and which packages. We

started creating these pretty Byzantine deployment processes, where we would embed bash scripts inside of YAML files. It calmed down eventually. We have things like Puppet, Chef, Ansible SaltStack to make a lot of that easier.

In more recent years, a second generation of these tools has come along. I'd say Terraform was probably among the vanguard of these tools. The ideas that were entirely declarative. We're not embedding scripts anymore. We're going to have a deterministic single artifact on the other side of this and it's going to be exactly the same, no matter who fucking runs it anywhere in the world.

That's a big deal, because that reproducibility is key for us to get the bug resolution and to keep velocity. We want these deterministic file formats that are checked in with the code that it depends on. The way I like to describe them for as code to people is that it's like a lock file for your cloud resources. We're used to having lock file for our code resources that we depend on that are third-party. One, I have a lock file for the cloud resources we depend on too, so that when we deploy, we get the thing we expected.
This all seems obvious. If you didn't come from the old world, this would be like, “Yeah, that's what you want.” The old world, that is not what we had. We he had Linux servers all over the place. We didn't know what version they were running. We had SSH into these things and patched them ourselves. We're moving there slowly.

Cloud formation is Amazon's attempt at configuration management, which has morphed into in for as code. Cloud formation is ancient. The cloud formation documents have a date stamp on the top for their version. That date stamp is from 2010. Cloud formation is 10-years-old, which is unbelievable to me, but they've done the Amazon thing. It didn't get worse. It just got more and more stuff and got better with time.

Cloud formation is also extremely verbose and it's hard to write. I think if you are a newer developer, it is extremely steep curve to get up. There are community ecosystem projects out there that compile down to cloud formation. One of them is serverless framework; great.com, excellent name. It takes YAML, turns it into more YAML. Serverless frameworks predicate is to go wide, not deep, so they support all the major clouds. I think they even have a Knative

adapter now. If you want to deploy functions to your own Kubernetes cluster, you can do that in a way that would be familiar, I suppose, to those developers.

Then Architect is like a serverless framework that it compiles to cloud formation, but it's different and it's focused for web apps. We only support out-of-the-box those eight services that I was talking about earlier. If your goal is to try and get to a whole bunch of clouds, ostensibly you would think that that would be with the same workload, but it's not cross-platform in that way. It's just it supports the different clouds, same as terraform; then this would be a good tool for you, Terraform or serverless framework. If you're just trying to build a database-backed web app and get it on Amazon and you don't really care about all this stuff, then Architect might be a good choice for you.

[0:40:49.9] JM: Tell me more about what you're trying to do with Architect. This is the application description file format compilation system deployment method that you're working on.

[0:41:03.5] BL: Yeah. so arc.codes we started working on in 2015. We did that, because at that time the only other serverless framework at that time was a thing called Jaws, which became serverless.com. I really admire Austin and serverless, so I don't really believe we're competing. I think we're competing with people that don't care about serverless in general. We're not competing with each other. No slide on them.

We could have easily adopted serverless had it been a few more months along. We built architect, because there was nothing else at the time that was web-focused. We built it initially, because we needed it and we realized very quickly having in-house proprietary framework for building serverlessly was a huge liability for our startup. We donated to the Open JS Foundation. Would not be coupled to our startup and it would just be its own entity that could live separate from us.

After my experience with Adobe, I was not really into having that IP as a part of my company just for a variety of reasons. Not the least of which I just wanted to see this thing continue on and have outside contribution. That was around 2017 that we donated it to Open JS Foundation. Yeah, it's been steadily growing ever since and it's not the most popular framework

and I'm totally okay with that. I think a lot of people will be like, “Oh, well what's the most popular?” It's like asking, Nickelback's popular. Who cares what's popular? Is it good for your use case is the question, right? For web app developers, I know it's very good.

[0:42:38.7] JM: This is a application description format that assumes you are on top of AWS, right?

[0:42:48.0] BL: It does, but it doesn't – this is something that is different from the other ones and that it's totally agnostic. We don't bring in Amazon service names into your code. Your code can be just your code and focus on your stuff. One of the big problems of Amazon's complexity is that they use a lot of branded names for their business. They have bad names for stuff, like API gateway, HTTP APIs. That's a thing. Try googling that.

[0:43:14.0] JM: You'll never get anything useful.

[0:43:15.9] BL: There's other examples, where they just blanket. Your code looks like a NASCAR. It's covered in Amazon's logos. You want your code to focus on your business problem, and so we abstracted at a very high level and then we just give things generic names. Instead of calling an SQS queue, or an SNS topic, you publish to a queue, or you published an event. We just use these high-level terms and totally just try and walk it back from the Amazoness.

[0:43:44.0] JM: If I'm a developer, why would I want to use a framework on top of AWS? Why not just go directly to the AWS ecosystem? What do I gain from using this layer on top of AWS that you’ve built?

[0:43:59.3] BL: I think that answer is a little more tricky now than it was a few years ago. Amazon is aware of this problem and they're working on it. I think you still want a clean entry into this world, an entry that's going to be a little bit more developer-focused. This is what's creating the opportunity for the layer2 cloud providers. Amazon is somewhat intractable to get started with. These frameworks give people a head-start. It's like you get the last four years of our expertise the minute you hit NPM install. That's a huge shortcut to figuring out all of this stuff yourself.

The other thing we can do that Amazon cannot do for you is tell you what not to use. Amazon hates it when I say this, but it's true. Here's some things that are true about Amazon. Amazon's not getting smaller, right? They're not going to ship less features next year. They're not going to tomorrow say, "We'll only use these five services." They're never going to get slower and they're never going to stop growing this complexity.

What we can do is say here are the eight services that we can just put you on Rails with and get you to the other side. We're not going to block you from using all this other stuff, but we're going to make it so fast that you can deploy to Amazon within 10 seconds. That sounds crazy, but you actually can deploy to Amazon in 10 seconds right now if you want to and you could do it on your phone.

[0:45:19.8] JM: Okay. Talking more – I don't think I wanted to deploy to my phone. Do I?

[0:45:25.3] BL: I think if you have a developer experience that's mobile-friendly, you'll start to like it. I think the reason that you think that's weird is because you're not used to it yet.

[0:45:34.3] JM: Conversation for another day.

[0:45:36.4] BL: Yeah. Yeah, yeah. Yeah, yeah.

[0:45:37.0] JM: Your system, Architect, you have a file format called arc. There's a process of parsing that format and turning it into a deployment. What does that process look like?

[0:45:47.8] BL: We have a lecture and a parser and all that good stuff, but the format's actually not that interesting. We also support JSON, YAML and TAML if you want to write in those formats. We felt that those formats have – they're more data serialization formats and they are for configuration. YAML is definitely a configuration format and TAML and E-file is definitely a configuration format, but they're brittle configuration formats.

If you get one character wrong in a YAML file, it's still valid YAML, but it's probably not a valid deployment artifact. TAML has similar problems, where it's got significant ordering of items. We

felt there was a justification to create our own format, but we got a great deal of pushback about that. We ceded to the mobs demands and we made it open enough that you can write in JSON if you want to. We don't think you want to.

We think that we've created a nicer format and it's just a traditional lexer parser that turns it into JSON at the end of the day. If you want to check that out, you can check it out. If you want to avoid it completely, you can avoid it completely. I've had people say you can't just go create a format to me and I'm like, “Well, somebody just went and created the other formats.” We've been using them now for over 10 years and they've shown their brittleness. There isn't two YAML parsers that treat YAML the same way and that's a major problem.

[0:47:05.6] JM: What kinds of applications are good fit for using the Architect format?

[0:47:11.1] BL: Oh, definitely web apps. That's what it's designed for. Yeah. Database-backed web apps, just boring, old, cruddy web apps, like what, 90% of us are working on right now.

[SPONSOR MESSAGE]

[0:47:30.7] JM: Looking for a job is painful. If you are in software and you have the skill set needed to get a job in technology, it can sometimes seem very strange that it takes so long to find a job that's a good fit for you.

Vettery is an online hiring marketplace that connects highly qualified workers with top companies. Vettery keeps the quality of workers and companies on the platform high, because Vettery vets both workers and companies. Access is exclusive and you can apply to find a job through Vettery by going to veterry.com/sedaily. That's V-E-T-T-E-R-Y.com/sedaily.

Once you're accepted to Vettery, you have access to a modern hiring process. You can set preferences for location, experience level, salary requirements and other parameters, so that you only get job opportunities that appeal to you. No more of those recruiters sending you blind messages that say they are looking for a java rock star with 35 years of experience, who's willing to relocate to Antarctica. We all know that there is a better way to find a job.

Check out veterry.com/sedaily and get a $300 signup bonus, if you accept a job through Vettery. Vettery is changing the way people get hired and the way that people hire. Check out veterry.com/sedaily and get a $300 signup bonus if you accept a job through Vettery. That's That's V-E-T-T-E-R-Y.com/sedaily. Thank you to Vettery for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:49:20.5] JM: Most of my web apps, I am using external APIs, like I might use Twilio, maybe I'm using Firebase as an external database. How do I define those kinds of abstractions if you're only definitions are these AWS services?

[0:49:41.3] BL: Yeah. You define your AWS primitives in .arc file or in one of the other manifest formats we support. Then we generate up your web app, which is usually an API gateway with an S3 bucket and DynamoDB. Now you want to integrate two third-party services and you do that the exact same way you do in your monolith today. You probably have some environment variables in your write code. That's about it.

[0:50:06.2] JM: Got it. Right.

[0:50:08.2] BL: I can make that more complicated. There is a Amazon service out there called EventBridge and I'm actually really excited about it, but it's early. Don't jump on this thing, because you heard it on a podcast. EventBridge is super we're checking out. It's Amazon's blessed way to deal with third-party ecosystem event payloads in a synchronous fashion. I have very strong feeling that this will be a common event bus for Amazon apps in the future.

[0:50:32.1] JM: That's one of the eight services that you're focused on?

[0:50:35.2] BL: Well, yeah. It actually is part of CloudWatch, weirdly. Amazon's got this thing called two pizza teams. You read about in the Internet where it's like, you can only feed a team at Amazon two pizzas. I always thought that was just something they said, but it's true. Everything at AWS is a two-pizza team behind it. Now when you think about how wildly different these services are and how weirdly integrated they are, it starts to make sense. Why isn't these

things so coupled is because they're literally two small teams maybe on different sides of the continent working on their own thing.

The thing that gets me excited about EventBridge is it's got built-in schema registry and schema validation built in. You can wire it up with whatever services that you want from a third-party like Twilio or whatever and you're still dealing with their event bus and their scalability and their retries and all the other stuff that Amazon brings to the table.

[0:51:34.9] JM: Got it. You run begin.com in addition to working on this arc, the open source Architect file format. Tell me what begin.com is.

[0:51:44.6] BL: Yeah. Along the way building Architect, it became really clear to us that the CICD side of this story is still pretty nascent and not very good. Where it does exist, it's relying on metaphors from the last generation of software. That's okay. This isn't a condemnation of how things are. I'm not saying you're bad if you run your own Jenkins. It's okay.

We think there's an opportunity to take a look at this stuff from the lens of serverless and deploy serverless applications and take advantage of those application's characteristics. One of the characteristics of these serverless apps is they're pretty fast to be deployed. You get the single deterministic artifact. We know exactly what our compute and our storage and our network requirements are. They're really well-documented separate from each other and we've gotten our deploys down to a sub-10 seconds because of this. Your lead time to production can be 10 seconds if you want. We think that's a powerful idea and that's the predicate for Begin.
[0:52:43.7] JM: You're focused on fast deployments, fast time to market. Tell me more about how you accelerate the developer experience on Begin.

[0:52:51.8] BL: Yeah. Everything that Begin deploys is just a cloud formation document. If you're on board with this world of building for Amazon and you've ceded to the fact that they're probably the incumbent that you want to target and if you've ceded to the fact that you're going to write cloud formation or use something to generate it, then this world is going to be pretty awesome for you. You work locally. You build just normal you would normally. Then when you commit your code, this cloud formation document will get generated. If it's the first time, it'll do a

cloud formation deploy two stacks. It will create a staging stack and a production stack for you instantly.

Subsequent deploys will only update the function code in the staging stack immediately directly, so you don't have to wait for a full cloud formation cycle. We'll just shoot the code straight into lambda functions. This seems scary and dangerous and weird and it is. That's why we only do it to staging stacks. The production stacks are totally deterministic cloud formation derived things. Yeah, that's how it works. It's just a normal developer experience. You work locally, you commit your code. When you commit it, it deploys. It just happens to deploy really fast.

[0:54:03.2] JM: What's the connection between the Architect project and Begin?

[0:54:08.5] BL: Begin is weirdly both written in Architect and deployed by it by itself. It's like a coin of its own making. We built Begin for ourselves. As people saw us building stuff with it, they were like, "Oh, that's amazing," and that became the predicate for the Begin product being that continuous integration thing. We deploy cloud formation documents. That's really it. I mean, it's built with itself too. It's constantly feeding into itself. I think as a startup, you want to use your own stuff.

Some choices in Architect look weird. People will look at it and they'll be like, "Oh, why did you do it that way?" Likely, we did it that way whatever that way is, because we knew that would be the best way based on the direct experience of the building Begin itself.

[0:54:56.0] JM: Do you expect the users that deploy their applications on Begin to define their applications in Architect?
[0:55:03.6] BL: Currently, that is a thing. Maybe by the time this is out, actually should be able to just give it any cloud formation and it will deploy that. That's our goal with it. Right now, it is our Architect-aware and Architect-special as it were. We want to move away from that though and be more cloud formation and more agnostic.

Either way, honestly as a developer, you shouldn't know or care. I feel a little bit of this is an implementation detail. We don't talk about how we're deploying our Terraform stacks or whatever, right? We're just deploying. At some point, I think that it's going to get like that for the

cloud formation serverless world, but we're still in the early moments of this thing. It's just the idea of inferred as code is still a bit controversial. I know the front-end developer community thinks it's a unnecessary ceremony right now.

They might be right, honestly. Maybe we can infer all this stuff. I don't know how to infer a database table name, or back one up that is inferred, but I would love it if it worked that way. I think that that's a worthy goal if we can make this be part of the substrate. I'm not saying it'll go away, but I could very much see this world of inferred as code become like an implementation detail, or perhaps a generated artifact of some kind.

With Architect, we've got something insane. It's like a 80 to one lines of code delta. For every line of code you write in Architect, you'll get 80 lines of cloud formation on the other side. This isn't something special or because we're really smart, this is just how things go. Stuff is always moving up the stack. We're always getting more abstracted. I think over time, this stuff will get easier, because it always does.

[0:56:44.9] JM: There are a few examples of hosting companies that are built around an open source project. Some past examples are like Zeit.

[0:56:55.5] BL: Meteor.

[0:56:56.4] JM: Meteor, the serverless company.

[0:56:58.8] BL: Sentia.

[0:57:00.2] JM: Sentia. I haven't heard of that one.
[0:57:01.4] BL: Appcelerator.

[0:57:03.1] JM: Right. Yeah. Okay. How does your strategy compared to those other companies?

[0:57:08.9] BL: I think that's complementary and similar in many ways. The idea is that you have an open core of some kind and then that's a funnel to your actual business. I think you can

dive by your open source framework too. When React becomes unpopular, which it will because these things do. If that is your core thing, then that'll suck, right? If your whole thing is based on React being popular and to say – and I'm sure people right now are saying, “There's no way that will happen.” I was a part of the jQuery Foundation and we totally agree, it'll never happen. You'll always be popular relative to where you were, but you will never be more popular than the new thing that supplants you in the future.

There's always a cycle of this. This is how tech works. This is why we have to keep innovating. This is why Apple had to release an iPod. You're going to have to add new stuff in the future. I think being open core based is smart, as long as you know that it's temporal and you have to stay with the times.

Architect, the original version of it was SDK-based. We didn't use any cloud formation. People always ask me like, “Why would you do it that way?” I'll tell you why, because cloud formation didn't support anything I wanted to do. Cloud formation does not come out with support typically right away for the newest services. That's changing a little bit at Amazon right now, but it wasn't always the case.

When we first built Architect, it was SDK-based. It was very clear to us that we have to change that, because the that shift towards inferred as code is very obvious. It's also obvious to us that we're going to need tools to make inferred as code a little more tenable. Yeah, I think being open core based is fine, as long as it's not your total identity and you're adaptable. I also think that open core can be smarmy. It really depends on your licensing and stuff.

If you're open, but the only people that can contribute have a Google badge, are you really open? That's why open governance is important to me. I really feel that if open means proprietary venture back startup, that's not very open actually. That's pretty close. There's a huge lock in risk with that, because eventually the venture capitalists want a return on their money. What will happen with that open source project? If it's the funnel, not good things. That's something to be aware of too.

[0:59:27.6] JM: Yeah. Talking more broadly, developers are increasingly using CDNs as a deployment medium for your app. What's the role of the CDN, or the edge server in modern deployment?

[0:59:42.1] BL: Yeah. We have to thank Netlify for making this a thing. CDN is used to be where I put my fonts and stuff. Now CDNs are by app. That's the edge of my application. It's absolutely the right way to do it. I think it's a far better user experience to have your time to interactive be as low as possible. I think it's interesting that it's become such a big deal too, because it's just something you set and forget. We broadly group cloud resources by network compute and storage and this would be a network thing. The lifecycle for network resources is usually pretty long. You don't change your domain very often. You don't change your CDN every day, so it's a set and forget thing. It's like, yeah, this is really important. You will definitely do it once in the life of your company.

I don't really have deeper, exciting thoughts. I'm definitely watching the edge compute thing. Lambda functions on the edge I think is a big future space, but we're not there yet. It's something to watch. I think the CDN is typically also been very static. It's where you put stuff and stuff stays there and it doesn't really change. As a result, invalidation is actually pretty bad for the most part and that's where these second layer providers are winning right now.

The reason Netlify inside are so popular is because they invalidate the CloudFlare. They invalidate their cache like that. There is CloudFront takes 20 minutes. Very confident that Amazon is both aware of this problem and can fix it. I think they can anyways. I don't know. Maybe they can be out-resourced buy a small venture back company. We'll see. Yeah. Cache invalidation would be the big delta between the major players right now. Once that's fixed, I don't really think it's something people will think a whole lot about. It'll just be how things work.

[1:01:32.7] JM: How will WebAssembly effect web development in the limit?

[1:01:37.2] BL: Well, right now it doesn't affect it at all, because you have to write pretty low-level code for it to be a game changer for you. I think there is an opportunity when the tooling gets there in the future that this could be a disintermediation of lambda, where we could see the edge become the lambda. We're deploying these blossom bundles to the edge. You only have

to play with Go and Rust for about a half hour before you realize how far we are from that reality though. It's a ways away.

The future is bright. It's up there on the shining hill. I can't wait for it to happen. I totally want to write a front-end in Python and deploy it to a CDN, but you're not doing that today, not without a lot of effort. It's not the first-party ecosystem that's the problem. You can write Rust or Go or even Python and get that into a wasm binary and put that at the edge. It's all the third party ecosystem interactions that are the problem. Your DynamoDB client is not currently going to compile into that wasm binary, and so how are you going to talk to it? Probably HTTP. Have you done Sig 3 signing with HTTP before? It's not fun. Now you got to do that. It's just real low-level still.

Everyone that's excited about this space knows it though and it's coming. It's one of those things, it's like fusion. It's right around the corner. We're almost there. This isn't to condemn people working on Fusion. Don't want you feel that. I know it's coming. It's probably a ways away.

[1:03:08.1] JM: It would come faster if they used WebAssembly.

[1:03:09.5] BL: Right?

[1:03:12.2] JM: How did you get begin.com?

[1:03:14.2] BL: Oh, I have an amazing co-founder and he's extremely meticulous. His name's Ryan Block. Ryan ran a process.

[1:03:22.1] JM: By begin.com, I mean, domain name.

[1:03:24.1] BL: Yeah, yeah, yeah, yeah. It turns out five-letter domain names can be bought and anyone can buy them. Usually, they're pretty coveted though. Once you get down to five letters, that's a big deal. There's the sub industry. I don't know if you know about domain brokers. Have you heard about these –

[1:03:40.8] JM: Sure. Yeah, yeah, yeah.

[1:03:42.1] BL: Yeah. This was new to me. We found a domain broker and we knew we wanted an actionable five-letter word that could be a verb. We started to go through the dictionary and we made a huge long list and until we got to B before we found one that we could afford. Yeah, that was it. That was the process. You can't be attached to the outcome of the domain name. You have to be prepared to get one that you might not be ready for, but that will suit your criteria and then you just got to work through a giant list and find something within your budget.

Our budget was big, but it wasn't actually so big that it was not impossible for a precede startup to do so. It took us I want to say seven months however, and we didn't know the name of our product or our company during that seven months. That was actually tough. We had a shitty placeholder domain that was deliberately bad. It was like a cheesy TLD. We did that on purpose, because we're like, “This isn't it.”

Then we've we found Begin and we found an awesome person that had had it since the 90s. They were excited about what we were doing, so we sold them on the startup and got them excited about it and we were able to come to a deal that was within our budget.

[1:04:55.7] JM: Nice. All right, last question.

[1:04:58.5] BL: All right.

[1:04:59.7] JM: How does AWS look in the long run, 10 years, or five years, however long you can project out what does the experience of using AWS look like?

[1:05:11.4] BL: Yeah, that's a tough question. That's great. I think it's interesting. There's a lot of people out there that think UX is a moat and that you can build a better UX and that Amazon wouldn't be able to copy you. I think that's very optimistic. Amazon has resources to do whatever they want. One of the things that they don't care about right now is your small startup, whatever that small startup is. Sure, yeah, your UX is better right now.
I’m very much Microsoft with operating systems, the UX at some point will be something they care about and then they'll work on. I think they are already working on it. They're just doing it in

two-pizza team fashion. There's the AWS amplify team, they are doing a really great job of creating an onboarding experience. It's something that normal developers would want to see. There's the AWS SAM team, which is coming out from the DevOps Eastside. Well, there you go. There's two really easy ways amongst the hundreds of bad ways to get started with Amazon. Do I think it gets easier to use? I think for certain verticals, it'll get easier to use. I think they're going to continue to accelerate and get bigger and bigger. I think they're going to take out competitors. I think they're going to buy some competitors. I don't see them stopping. I don't see them slowing down. I do see it getting easier over time as they enter more and more markets, which is pretty scary.

[1:06:28.7] JM: Brian, thanks for coming on the show. It's been great talking to you.

[1:06:30.7] BL: Yeah, likewise. Thanks, Jeff.

[END OF INTERVIEW]

[1:06:41.4] JM: As a company grows, the software infrastructure becomes a large, complex distributed system. Without standardized applications or security policies, it can become difficult to oversee all the vulnerabilities that might exist across all of your physical machines, virtual machines, containers and cloud services.

ExtraHop is a cloud-native security company that detects threats across your hybrid infrastructure. ExtraHop has vulnerability detection running up and down your networking stack from L2 to L7. It helps you spot, investigate and respond to anomalous behavior using more than 100 machine learning models.

At extrahop.com/cloud, you can learn about how ExtraHop delivers cloud-native network detection and response. ExtraHop will help you find misconfigurations and blind spots in your infrastructure and stay in compliance; understand your identity and access management payloads to look for credential harvesting and brute-force attacks and automate the security settings of your cloud provider integrations. Visit extrahop.com/cloud to find out how ExtraHop can help you secure your enterprise.

Thank you to ExtraHop for being a sponsor of Software Engineering Daily. If you want to check out ExtraHop and support the show, go to extrahop.com/cloud.

[END]