EPISODE 1039

[INTRODUCTION]

[00:00:00] JM: Serverless tools have come a long way since the release of AWS Lambda in 2014. Serverless apps were originally architected around Lambda, with the functions as a service tool being used to glue together many larger pieces of functionality and API services. Today, many of the common AWS services such as API Gateway and DynamoDB have functionality built-in to be able to respond to events. These services can use Amazon Event Bridge to connect to each other. In many cases, a developer does not need AWS Lambda to glue together services to build an event-driven serverless application.

Jeremy Daily is the host of the Serverless Chats Podcast, a show about patterns and strategies in serverless architecture. Jeremy joins the show to talk about modern serverless development and the new tools available in the AWS ecosystem.

[SPONSOR MESSAGE]

[00:01:02] JM: Today's episode is sponsored by Datadog, a modern, full-stack monitoring platform for cloud infrastructure, applications, logs and metrics all in one place. From their recent report on serverless adaption and trends, Datadog found half of their customer base using EC2 have now adapted AWS Lambda. They've examined real-world serverless usage by thousands of companies running millions of distinct serverless functions and found half of Lambda invocations run for less than 800 milliseconds.

You can easily monitor all your serverless functions in one place and generate serverless metrics straight from Datadog. Check it out yourself by signing up for a free 14-day trial and get a free t-shirt at softwareengineeringdaily.com/datadogtshirt. That's softwarewareengineeringdaily.com/datadogtshirt for your free T-shirt and 14-day trial from Datadog. Go to softwareengineeringdaily.com/datadogtshirt.

Thank you Datadog.

[INTERVIEW]

[00:02:15] JM: Jeremy Daily, welcome to Software Engineering Daily.

[00:02:17] JD: Thanks for having me.

[00:02:19] JM: So you host a podcast called Serverless Chats, and I'd like to know when did you start working with serverless tolls?

[00:02:28] JD: It's been quite some time now. Actually, it was shortly after AWS Lambda went GA back in early 2015 is when the company I was at, I was at a small startup, and we started looking at it because we had had just before that an incident where our system did not scale when we got mentioned on the Today's Show. That was one of those things where we just started looking at how do we scale up? How do we scale up our application without having to set up millions of servers and have that all running in the background and paying all that money. At the time, Lambda functions seem like they were a potentially good fit for that. So that's when I first started to sort of take a look at them. Didn't put anything into production with serverless though probably until about middle of 2016, maybe a bit later than that once API gateway was available and we're able to actually start doing some web frontends.

[00:03:26] JM: In the early days of Lambda, it was often used as this tool for glue code. When did Lambda mature to something that was more integral to application development?

[00:03:38] JD: It depends. I think that what's interesting about the ecosystem that AWS has put into place is this idea of serverless being more than just Lambda functions, and I think a lot of people kind of get hung up on this idea of fast or functions as a service being what is serverless, and it couldn't be further from the truth.

Things like S3, and SQS, and DynamoDB, and SNS, and now Event Bridge, and all of these other tools that are serverless in and of themselves, the tools that you can use without having to worry about capacity planning and without having to worry about managing those service in the background. What Lambda does is it just gives you, as you said, that glue to kind of tie some of those things together, but we're even getting to a point now where we're sort of maturing past the need for Lambda in many cases, right?

So some of these tools like API Gateway, now the new HTTP APIs that just came out, they're providing ways in which you can do direct integrations into other systems. So you can have data come in from an API Gateway and write that into an SQS queue or into a DynamoDB table and have that, even use VTL templates if you want to to manipulate that.

Then if you have some business logic, something you need to do with that on the backend, that's something you can do asynchronously and use a Lambda function to do something like that. I mean, I would say Lambda functions started off as, like you said, sort of this glue code where you could kind of run some business logic here and there and maybe tie together some services. But the ecosystem itself and especially AWS is growing to a point where some of those connections are sort of built in to the tools themselves and Lambda isn't even necessary in some cases.

[00:05:17] JM: Could you explain that in more detail? Maybe give a concrete example about how you might have needed to use Lambda functions to glue together different AWS services before. But today, it's easier to just have them integrate seamlessly.

[00:05:35] JD: Sure. Let's think about maybe a webhook scenarios. You could set up an API gateway right now that had a service integration into Kinesis Data Firehose, and essentially what that would do is it would allow you to take the data in from whatever is emitting those events or whatever is hitting your webhook. Take that data, use VTL, translate that into some format that goes into a Kinesis record. Then with Kinesis Data Firehose, you can actually process that data and spit it into S3 buckets and automatically prefix it so you can say, "These were all the events from 2020-03-19 or whatever," and it does it by date and by hour, splits those all up. Then you can write Athena queries. You can use Athena, which is another serverless service that's similar to like Google BigQuery where you can just point that to those S3 buckets and you can actually query that data as if it was in sort of a MySQL table or something like that.

So that entire process of doing something just to get an analytics or to process some data off of webhooks, you can do that now without using Lambda at all. Lambda comes in really handy for a bunch of things that you can do within that process if you wanted to. Let's say that you wanted to maybe transform the data that was coming in from the webhooks. Maybe do some verification things like that? You connect it through Kinesis Data Firehose and then you can actually have a Lambda function that does that transform step for you, that after the data has been captured, it will transform that and then put that into S3 for you.

Then once you have S3 and the data sort of the way you want in S3, you query that with Athena, and now Athena even has a way that you can run queries, and when those queries are finished, those can actually trigger Lambda functions for you to do something else with. It's really about thinking about how you can use all the different tools that are available to you as supposed to just sort of focusing on what code can I write myself?

The big point here is something that, some of the DAs at AWS talk about quite often, is the biggest weakness with any serverless application is the code that you write. I mean, for the most part, you can trust the cloud to do the retries and to do things like throttling and to do some of these other things so that you don't lose events. But as soon as you write code that starts touching those events and starts trying to manipulate data and call on their systems, that's when you can start introducing bugs. The more you can avoid writing code and use the built-in systems or the built-in tools that are there for you, that's I think the ultimate goal of where we want to go with serverless.

[00:08:10] JM: If we compare the typical greenfield application path that people might have taken 10 years ago, standing up an EC2 instance with your first monolith and standing up a database server somewhere else, or maybe – I'm not sure if there are managed database services 10 years ago. But how does that compare to today? If I have my monolithic Node.js application that I'm standing up as a greenfield application today, is there a standard serverless stack or deployment medium? Should you still use EC2? Should you use Fargate? Should you really just try to use Lambda functions together with other various services? What's the greenfield application stack?

[00:09:00] JD: Yeah, that's a really good question, because I think that is something that definitely trips people up when they start thinking about serverless, and that also goes back to this idea of thinking that Lambda functions or FaaS is what serverless is because, again, like I said, it's the ecosystem around it.

Traditionally, when you are setting up applications, you would have to go through that process of saying, "Okay, we're going to build it on node, or we're going to build it on PHP, or we're going to write Java, or whatever we're going to do." Then someone would have to set those servers up for you. Now, if you were a small team, that was one of the biggest pains, is that you would spend a lot of time just trying to figure out what is my infrastructure look like? What do I need for a database? What do I need for an application server and things like that? What do I need from maybe a messaging server if I am going to build some things distributed? What other services might I use?

The problem with that was it always really easy set up that initial version, but then as soon as it starts to scale, that becomes incredibly difficult. Now you're starting to load balance things and you have to start sharding databases, and it gets really, really complex. If you're building a greenfield in 2020, you need to think about, one, building it in the cloud because especially if you're a startup or even you're a good sized business, just the idea of trying to use VM's and configure that and have all that maintenance that you need to do when you take that old sort of traditional approach. That's just a huge headache. That's something that is completely not worth your time and it's just going to waste your developers' time.

What you want to do is you want to think about how you build sort of what we are calling modern applications or cloud native applications. In some cases containers can fit in there. But really, the first thing that I always do at least for me, this is the advice that I give, is to say, "Can you build a serverless?" If say, "No, I can't." Then rethink it and ask yourself again and rethink the way you would build the application. Then if you really can't, then start thinking about maybe going down the Fargate route or something like that.

But in almost all cases with the tools available today, you could absolutely avoid using EC2. I have tons of applications that are running right now. They're running 100% serverless except for using like Elasticsearch on the backend for a few things, because there really is no good serverless search utility right now.

But to go back to your question about sort of the standard stack. I mean, there are a bunch of really great tools. Serverless framework is a framework that allows you to very easily sort of configure your Lambda functions, specify the resources you need, do all your mappings so that you can say, “Okay, when this endpoint is hit, I want it to trigger this Lambda function.” You can set up all of your resources all within a simple serverless.yaml file. But then there is cloud formation and there is a subset of formation called SAM, which the Serverless Application Model that is produced by the teams at AWS, and that's very similar to the serverless framework. You can configure all your resources in there. Configure your Lambda functions, things like that, and then it’s a simple deploy, SLS deploy, or serverless deploy, or SAM deploy, and it just pushes all that stuff up into the cloud. It's all infrastructure as code. It's all repeatable. You can create new environments.

Again, because the runtimes, if you really kind of hung up on the fast aspect of this, the runtimes available on AWS right now, there is Node, there's Python, there's Go, there is Rust, there's Java. I mean, basically any language you want to write it in, and even if you can't, even if there's not an existing language support for it, you can do custom runtimes. Really, there is little reason unless you have to build the stateful application that you wouldn't just go serverless out-of-the-box. At least that's how I feel about it.

[00:12:26] JM: And can you speak a little bit more about how serverless has changed the product direction of AWS? You're speaking to a little bit by just talking at how these services have become more integrated in a way that doesn't even necessitate glue code, but maybe could speak to what has come out in recent years and what it might say about the future product direction of AWS.

[00:12:50] JD: That's a really interesting question, because two years ago when I was at Rreinvent 2018, it was all about serverless. It was all the massive investments in serverless. This is where everything was going. Then at Reinvent 2019, it was very much so, “Here's how are going to support more with containers, and here's how we’re going to allow you to run

Kubernetes or Kubernetes pods on top of Fargate and some of these other things," which is sort of a serverless container system that they have.

If you asked me two years ago, I would say, "Yeah, I think AWS is moving 100% to the serverless sort of wave." But I think that the market has sort of adapted containers as sort of the main thing. You hear about all the time, Kubernetes, Kubernetes. Everybody is doing Kubernetes. All these big companies are building Kubernetes clusters.

The idea of where I think AWS is going right now from an enterprise standpoint. Again, they make their money off of enterprise customers. So they are going down that direction I think of trying to support sort of this more general idea or more accepted idea of cloud native, which has sort of become synonymous with containers. But they are still building tools that serve this sort of serverless piece of it.

One of things to think about between sort of what FaaS is versus what everything else is, is FaaS is very much so just the runtime, right? Just the ability to run code, whatever that is, business logic, glue code, whatever you want to call it. But it's the database. It's the messaging system. It's the queuing system. It's the streaming tools. It's all of those other tools in that ecosystem that add on to that business logic that makes it really, really powerful.

If you look at something like Kubernetes or containers, what people are building on containers unless they're building out very custom sort of implementations of these things, they're not building out their databases on containers. They're not building out some of these other things on containers. They're mostly focusing on the business logic, right? Again, there're some great use cases for it when it comes to the machine learning and some of these other things. Even just serving up HTML or serving up webpages or APIs. But all of the stuff around that, which really makes the application rich, so the DynamoDBs, the NoSQL databases, or even your large RDS clusters for relational databases and things like that, or Event Bridge to do this really scalable – This message bus that Event Bridge does for you. Those are the tools that are all peripheral to that.

Even people who are building on AWS and using containers and going down that route, they are still taking advantage of these other services, these other peripheral what I would call serverless services that are in that ecosystem. I think AWS has changed their sort of – I don't want to say they changed their direction, but they certainly have changed their marketing a bit to be very much so focused on saying, "All right, you want to do Kubernetes? Great. We can do Kubernetes. We can do Kubernetes."

But behind-the-scenes, they are working very hard on building all these other tools that essentially is going to make something like Kubernetes completely – I don't want to say obsolete, but something you don't have to worry about. I think for the majority of developers who are doing that or building things, they don't want to be tied down by having to manage some infrastructure. If you've got a big company and somebody else is managing that for you, that's one thing. But especially for small startups or even midsized companies that want to move fast and want to build things quickly, certainly avoiding that container trap. I don't know if you would call it. Maybe I would call it that, but avoiding having to manage that piece of it and being able to focus on these small components that scale almost infinitely and take advantage of that whole ecosystem. I think that's where AWS is kind of heading in the background while still from a frontend standpoint accepting the fact that Kubernetes is sort of a talk of the town. So they'll continue to do that while building out these backend components.

But between the things that have sort of just been launched over the last couple of years, Event Bridge is the big one. I mean, for me, that changes the way you do interservice communication. It is a massive messaging bus that allows you to do routing in all kind – Multiple subscribers, the pub/sub type system similar to like a Kafka or something like that, but fully-managed and allows you to sort of glue all kinds of things together with filters and messages and everything is decoupled. There's been a significant amount of investment in things like the API gateways. So now you've got the new HTTP APIs that are less than a third of the cost of what your original API gateway service was or the Rest APIs also dramatically reduced the latency of those things. You've got web sockets. You've got the event schema registry. I mean, there're just so many cool things that have come out that are completely managed for you. Once they are set up and once they're configured, you don't have to go in and install updates. You don't have to worry about something breaking and then having to go and replace the drive somewhere or reboot a server or set up scaling groups or any of that stuff. It's just taken care for you.

I think that's where AWS is quietly moving in the background even if they're more publicly talking about supporting things like containers. But I think they're in full agreement that eventually they want that to go away and they want serverless to be the way.

[SPONSOR MESSAGE]

[00:18:05] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

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

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

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

[INTERVIEW CONTINUED]

[00:19:54] JM: It has become an amazing competition of narratives here where you have the one narrative of we should be avoiding lock-in, and Kubernetes is our remedy to this. That is to some extent true. To some extent a hangover of the Microsoft days, and to some extent

Google’s kind of virtuous blathering about something that maybe nonissue compared to the convenience that you would get from the lock-in of AWS with the potential threat that AWS somehow turns into Oracle in the future and blocks you in and starts raising prices. It's anyone's guess as to which of these narratives should be more believed.

So you see these interesting compromises in how people are navigating the landscape. Some people are saying, “Yeah, I’m going to have a Kubernetes cluster in each cloud provider, or I’m going to try to replicate my infrastructure in each cloud provider.” Actually, I don't think anybody is doing that. But it is this weighing of pros and cons of convenience versus – The other thing is like even if you set up a Kubernetes cluster. You used like a Google Kubernetes service, you're still locked in, I think. Even just to the extent that you happen to have set up infrastructure on a cloud provider and it takes a lot of effort to migrate off of a cloud provider. You’re kind of locked in there.

On the other hand, going whole hog into basically running on the AWS operating system of services is a very scary proposition for some people, or it's a new exciting paradigm shift. It just seems hard to weigh these two futures, and I think what I'm seeing at least when I talk to people who are running large enterprises is basically we’re going to hedge our bets for as long as we can, as practically as we can, which often results in a feeling that they're doing something wrong. But today, there's really nobody who can say what the “best practices” around how aggressively you should adapt this stuff.

[00:22:15] JD: Yeah. I mean, I think the problem with vendor lock-in or that vendor lock-in argument, especially when it comes to something like Kubernetes. If you set up a Kubernetes cluster on AWS and then you set one up in Google Cloud, and then you build an application that runs on Knative or whatever and you have your containers all set up and you can say, “Well, I can just deploy these containers to AWS, and then I can also deploy the same containers to Google Cloud.”

I think the multi-cloud parity, sort of like having parity between two applications on multiple clouds, I don't know how much people are doing that. But let's say that that was the ultimate goal. So now I can run an application on Google Cloud and I can run it on AWS, and maybe it's portable in the sense that I don't have to change too much other than maybe few configurations.

But what is that application doing? Where is it getting its data from? What is it using for its message queues?

If you're building in the cloud today, it is a terrible, terrible, terrible idea in my opinion to choose the lowest common denominator. By that, I mean, if you say, “Well, we want to use MySQL, and that's what we want to back our database. We want MySQL to be our backing database.” If you're in AWS, are you going to spin up a bunch of EC2 instances and install MySQL yourself and try to manage that cluster with some sort of load balancer or something like that, or are you just going to use RDS, which is their relational database service? You're going to use RDS.

Now, if you go over to Google Cloud or you installed MySQL on a bunch of VM's there and managed it yourself, or are you going to use cloud SQL? You're going to already lock yourself into one of the products on one of those providers, because it would just be crazy not to. If you're going to use something like MySQL, why not take some of your application and put some of it in DynamoDB so that you have the scale and the operational scalability so you don't have to worry about managing all those databases? Why not use Firestore or Firebase on the Google side? Because it'd just be a smarter tool to use that would make it easier to build your application more scalable, less management on the backend. As soon as you start making those decisions, you're already locked in, right?

If you want the convenience of being able to move your application code, that to me is the easiest thing to do. What does that take? I mean, even huge application. Let’s say you had to do some refactoring to move it from Lambda functions to Azure functions. Maybe that takes a couple months to do that. How much time do you lose trying to build something out that is agnostic to these different clouds? What do you lose in capabilities? What do you lose in scalability? How much do you lose in terms of the number of people you have to set aside to be able to manage these clusters because you're not taking advantage of these cloud native services?

I think that the second you choose anything outside of running just application code, anytime you pick any service on the provider that does and that, and that might be EKS, by the way. EKS is the Kubernetes service that AWS has. You choose that, and all of a sudden now setting up another Kubernetes cluster in another cloud is going to be different. Now that’s something else you have to learn, something different you have to do. It's not just grab this code and upload it to a different server. It's a lot more complicated than that.

I get your point, and I can see why enterprises say, “Well, I don’t want to go all-in on AWS, because what happens if they raise the prices or do something like that?” I mean, that very well could be true, but I think if you're going to spend the time trying to be agnostic. What you really should be spending the time is using the tools in each cloud provider. Build an application that runs on that application or on that cloud provider using the best tools that they have available and do the same thing on other ones. Maybe they're not 100% portable, but a lot of the business logic would be. That's not going to change. Some of the interfaces into it and some of the ways that you deploy these things might be a little bit different, but again, you're going to use step functions in AWS. You're going to use logic apps in Azure, and that's the kind of decisions that I think you're going to have to make if you really want to be multi-cloud.

But this idea to me of trying to just be agnostic just seems like a huge investment and not only is it probably never going to pay off for you, but it's going to slow you down, and whether you have a team of a hundred or a thousand, there could be a team of five people that could build a product and get it up and running in a few months on AWS while you're still spending months and months trying to figure out how to make this work across all these different major providers.

[00:27:02] JM: What are you hearing when you talk to large enterprises? How open are they to the idea of starting to expand into more and more serverless services?

[00:27:16] JD: I mean, I think it depends on who you talk to, right? Enterprises everywhere are starting to adapt serverless at least as a peripheral sort of component. One of the really, really interesting use cases for serverless, which is something that I guess it only applies to companies who have very large infrastructure or very large cloud deployments as it is now, is that you can use serverless to React or Lambda functions to react to events from your EC2 instances.

Let's see you get an alarm, a CloudWatch alarm that goes off that says a particular server has spiked to 90% CPU or something like that. You can have a Lambda function that responds to that and either run some diagnostic scripts or reboots that EC2 instance or spins up another instance and then reboots that instance. I mean, there're all kinds of things. There's the use cases of development machines that there's a schedule, there’s a CloudWatch schedule that runs or a CloudWatch – Yeah, CloudWatch schedule, or a rule that triggers every night at 7 PM or whatever it is that it shuts down a whole fleet of development servers, and then at 7 o'clock in the morning it spins them all back up so that when the developers are working, that they have this environment, but they're not going to pay for it when it's not running.

There're all kinds of things like that. CloudTrail in AWS is essentially the stream of information of what's happening in your infrastructure. It doesn't matter what services you are using. They don't have to be serverless ones, but as different things happen in the infrastructure, security things, people login, people change permissions, those kinds of things. All that information gets sent to CloudTrail and you can read off of that stream and say, “I want to do something with it.”

The idea of integrating serverless into this sort of DevOps type role is a very, very popular thing that I see a lot of enterprises doing because it doesn't touch their main application. This is peripheral to it and they don’t have to worry. So they might be running Kubernetes or they might be running EC2 or they’re doing something like that, but they can run these jobs and these scripts and these other things that they might be running before on something like a Jenkins or whatever. They can now do all this stuff on the outside without having to worry about needing to run those servers and having the benefit of it being very event-driven, which is something that serverless is one of the key components of what we consider serverless.

[00:29:38] JM: You talked about Event Bridge in some detail earlier. Can you explain in more detail what Event Bridge is and how it changes serverless architecture?

[00:29:48] JD: Yeah. One of the things that you need to think about when you're building serverless applications as we start splitting our applications up into very, very small chunks. If you go and you think about microservice architecture, usually what you would do is you’d build a bunch of different services that would be separated, that’d distributed, and you might have a billing service and you might have an order service and maybe you have an inventory service and a mailer or an alerting service, something like that. Traditionally, the way that you build those is you would build those as maybe separate and you might have a cluster of EC2 instances or a set of containers that are running just that particular application. They have their own data store.

The typical way that you would communicate with those is you would send messages back and forth. Sometimes there would be API calls, synchronous API calls, but other times – And usually you would use some sort of a messaging bus. You essentially would say, “Okay, and order was just placed. So I’m going to send out a message on to this message bus that says, “Hey, and order was placed. Here's some information about the order.”

Then any service in your architecture that was interested in that particular event could listen for that type of event and then consume it. Now maybe your alerting service says, “Oh, a new order was placed. Great. Now I want to go ahead and send an alert to sales or I want to send a message to the customer and let them know that their order was received.” That's typically how you would communicate between microservices.

When you move to serverless, now you start breaking these microservices down even smaller. Now you're using individual functions. So you might have a send email, send text message, process alert. Those might be three separate functions that live in your alerting microservice. It's not just composing those functions together like you would in the past where they all ran on the same stack or on the same machine. These are all technically running as separate individual containers that do that particular job.

The ability for you to coordinate that stuff and say, “Okay. So a new alert came in and now I need to send an email.” You can wire all of that stuff together with messaging buses or a message bus and the tool that AWS has come out with is an extension of their CloudWatch events, but is now called Event Bridge. It's become a lot more powerful. It allows you to do a lot more integrations with third-parties, and also you can sort of get rid of webhooks with certain companies.

But let me go back and just sort of explain how that works. So let’s say I get an order that comes in. Someone places an order on the site. It hits the order service and a new order gets created. What you do is you basically can admit an event or send an event to Event Bridge and say, “Okay, this order,” and you choose your bus. Maybe have a main bus or you might have other buses. But you send a message to that bus and you say, “This order has been placed and here are some of those details.”

Then Event Bridge allows you to connect other Lambda functions. It allows you to send things to step functions. It allows you to send them to Kinesis Data Firehose. It allows you to trigger all kinds of other downstream services and multiple ones, right? It's pub/sub. You can have multiple subscribers to this.

Every time a new order is placed and that event is put on the bus, you may have 10 different services that are interested in that. You might have a service that writes something to Salesforce or to Marketo or some other sales system. You might have a tool that sends out the email. You might have a tool that goes and orders more inventory if it dips below a certain amount.

You can build all those things and they're all completely independent and decoupled. By decoupled, I mean, that none of those services have to know what any other service is doing. So as long as one of those services admits an event that says, “This has happened.” Those other services that are interested in that type of event can listen for it, but you can go ahead and add 30 more services that are listening to that event and that original service that was producing that event doesn't need to know that those exist. All it needs to now is that it needs to be able to do its job.

The ability to scale with Event Bridge is pretty crazy, because it's like you can send just – I don’t even know what the throughput is on it, but it's like billions of messages a day this thing is handling. I mean, it's crazy. That's everything from the messages that you send to do interservice communication. It's almost CloudTrail events that we talked about, that EC2 server or an alarm went off or something like that. All of that is built-in.

The other thing that EventBridge adds, which is really cool, is integrations with third-party services. We talked earlier about webhooks, and that webhook use case. Well, setting up a webhook is kind of a pain. Even if you go through that whole process, you still have to configure it. You still have to worry about scaling and some of these other things. Make sure that it can handle all that data.

What EventBridge does is for few partners. I think there're maybe 20 partners now, something like PagerDuty and a couple of other ones. What they allow you to do is communicate through events. When something happens in PagerDuty on your account, trying to hit an endpoint for you to do that webhook, instead of doing that, it actually will just send the message to an event bus in your AWS account.

So now you can subscribe Lambda functions or step functions or any of the other services. I think it integrates with 20 services now, any of those other services, so that now let's an alarm goes off in PagerDuty or there is a change in schedule or whatever some of those events might be. Now that gets posted to EventBridge, and you don't have to worry about maintaining that webhook endpoint. You don't have to worry about trying to make sure that data gets captured or if it fails trying to do retries. All of that is handled by EventBridge for you and then you just build in – You just add your subscriptions or your rules that route that to those different services.

It’s a really, really powerful system that makes it – What I'm hoping, and I think this is what AWS is hoping too, is that as more people start using this partner channel or these partner events, then you get to the point where all of that communication happens through EventBridge. Anytime I want to say or I want to process a credit card with Stripe, for example, maybe that's something I can submit an event to EventBridge and then Stripe will pick that up and then send a message back to me letting me know that the card was processed or something like that.

That I could get really, really exciting because that eliminates a lot of extra infrastructure and a lot of headache for many teams that are dealing with all these different services when you no longer have to set up those webhooks and be responsible for those. It is something where it just makes it a lot easier to do the communication. It standardizes it. It allows you to send it across different accounts. If you have multiple accounts, you can send messages between different accounts, and it’s just a complete sort of change the way you that I think we were building serverless applications before where we were doing some pub/sub, but we were often using things like SNS, which is fine, but you can't do step functions and some of these other things you can't kick off. I probably rambled on for quite a bit there, but you probably tell I'm excited about it, because I do. I have started using it now and I don't know how I did things without it, basically.

[00:36:59] JM: The difference in how EventBridge functions versus SNS, or SQS, or Kinesis, the other queuing services on AWS. Can you explain what the difference is in more detail?

[00:37:17] JD: Yeah. Kinesis is a streaming service. What's really great about Kinesis is when you add messages to Kinesis or to a Kinesis stream, it actually saves those messages for a period of time. I think 24 hours might be the default, but I think it can go up to seven days. But essentially what it does is it stores those messages and it stores them in order. I can have multiple consumers that can read off of that Kinesis stream and I can go back in time up to however far back I've stored. I can go back in time and reread events that have already been read maybe by another consumer. That’s really powerful. That’s super helpful use case for something like that, is when you need to have massive throughput and you need to be able to maintain ordering. Something like maybe clickstream data. But if you wanted to use it as a message bus, the problem is that you can't filter messages from EventBridge. Essentially, every single message that goes on to that stream has to get read by the consumer and the consumer would then have to decide whether or not they want to keep the message or not.

For pub/sub type things where you would want to publish a message and you want multiple consumers to consume that, something like Kinesis would not be a great choice for that especially if every consumer wasn't interested in pretty much every single message. But lots of great use cases for that specially when you want to maintain strict ordering and you have very, very high throughput.

SQS is the simple queue service. So simple queue service essentially is just a regular queue. Once a message is removed from the queue, it's gone. You can have really only one consumer per SQS queue, and what that does is that it checks, it downloads the messages, does something with them and then tells the queue to remove them. But you can only use the message essentially once. But very, very lightweight, tons of throughput. It’s like the failure rate on it is very, very, very, very low. So it's something where if you do just want to make sure you capture an event and have that message durability, SQS is a really great choice because it is so lightweight. But you are a little bit limited on how – Like I said, you can only have one consumer that can process it.

There is something called FIFO queues with SQS, which is first-in, first-out, and those allow you to maintain ordering as well. The problem with that is similar to what you have with Kinesis. Kinesis, if you have really high throughput, you have to break it up into what are shards. You basically have to shard it, and then you have multiple consumers that can read off those different shards. Same thing with FIFO queues for SQS, is you would have to set up different message group IDs that allow you to have different consumers for those different groups. They’re kind of like shards. But it's certainly something where you can do it and it works really well. It’s just, again, you get limited on throughput if you don't shard the data enough.

Again, SQS is great if you have a single consumer. Where EventBridge and SNS come in, and I'll talk about SNS for a second. SNS is the simple notification service. That was their original pub/sub service. Also a very, very cool service. You can essentially publish a message, and that will have distributed to however many subscriptions it has. SNS is really great for – We use it before to do things like to trigger Lambda functions, for example. But SNS is really, really, really powerful for very, very high throughput type pub/sub job.

Your subscribers can be SMS phone numbers. They can be email addresses. They can be mobile, like an APN for like Apple push notification, things like that. We’re talking about very, very, like a ton of subscribers with very, very high throughput. Can you use it to trigger a Lambda function? Yeah, you can. Can you use it to trigger an HTTP endpoint? Yes, you can. But the main thing that you would use SNS for now that EventBridge exists is for those very, very high throughput use cases where you need to send –I don’t know, hundred thousand push notifications or something like that. But you can use it. You could use it as that sort of traditional pub/sub. One message goes out, multiple consumers can consume it.

But where EventBridge comes in, is EventBridge really is built to be that sort of enterprise messaging bus, right? There's a bunch of retries built into the system. If you send a message to EventBridge, as soon as it gets that message, then you're pretty much guaranteed almost to have that delivered to whatever service you've got wired to it or whatever rule you have to route it to the downstream service or the subscribers. If you're sending it to like Lambda functions, for example, the Lambda function service only has to accept that job and then all of the retries are built in there. But EventBridge will actually retry for up to 24 hours for all these different services and not lose your message.

Really, I guess to summarize it all, Kinesis is great for high throughput streaming where you need to maintain order, but you still have to deal with shards, but it's not great if you have multiple consumers that only need certain messages. SQS is for sort of a single consumer, very high throughput and very lightweight. So great for just message durability and things like that. SNS, great for high throughput pub/ sub type jobs. Again, the emails or the push notifications. Then EventBridge is really for that solid enterprise event bus/pub subtype stuff.

[SPONSOR MESSAGE]

[00:42:35] JM: If you can avoid it, you don’t want to manage a database, and that’s why MongoDB made MongoDB Atlas, a global cloud database service that runs on AWS, GCP and Azure. You can deploy a fully-managed MongoDB database in minutes with just a few clicks or API calls and MongoDB Atlas automates deployment and updates and scaling and more so that you can focus on your application instead of taking care of your database. You can get started for free at mongodb.com/atlas, and if you’re already managing a MongoDB deployment, Atlas has a live migration service so that you can migrate your database easily and with minimal downtime, then get back to what matters. Stop managing your database and start using MongoDB Atlas. Go to mongodb.com/atlas.

[INTERVIEW CONTINUED]

[00:43:34] JM: Now that we've covered EventBridge in some detail, another newer application in the serverless family of tools is AWS Step Functions, which is a tool for building these serverless applications with a visual workflow. Can you explain how step functions are used?

[00:43:53] JD: Yeah. Step functions have actually been around for quite some time. What they released recently last year or two years ago – Well, 2018 I believe, were what they call Express Workflows. So Step Function Express Workforce.

Step Functions or state machine. If you're familiar with a state machine, essentially what it is, is it’s just a tool that does or allows you to do orchestration so that you can have multiple components. Do some sort of job, and then based on the results of those jobs, control that workflow. The ordering workflow is the perfect thing where you say, “All right, an order comes in, a new order comes in. The first thing we need to do is make sure that we have the inventory to process the order, or the first thing we have to do is submit the order itself and capture the data. Then once we do that, we have to make sure that the inventory is available and we have to subtract the number from the inventory.

Then we have to go and make sure we put a shipping request in for it and then we have to send an alert to the user and then we have to – Once it goes out, we have to send a tracking number to the user, something like that. If any one of those steps breaks down in the order, that’s sort of a bad experience. We don't want the user to say, “Hey, we got your order, but then the inventory or the shipping stuff never actually happened.”

What state machines do, and this is something you would with Step Functions, is essentially you can say, All right, I want to go ahead and process the order.” So you process the order. You run that as a task in your state machine and then that comes back and says, “Okay, that's done.” Once that's done, then you moved to the next stage and you then run another task and you say, “Okay, now I’m going to deduct my inventory or I’m going to check the inventory and make sure that that's there.” That comes back and says that's good to go.

Then you go and you try to do the shipping. Now let’s say the shipping comes back and says, “Oh, there was a problem with the shipping. we couldn't do it right now.” The Step Function can build in all kinds of retries and it can retry that same job or that same task multiple times. Let's say that we say, “Okay, we want to wait five minutes between every try to process the shipping. If it fails 10 times or 3 times or whatever, then we basically have to roll back the entire application.”

You can handle all of that logic and all of that coordination of those different steps. You can handle that with step functions, which makes it a very, very smart choice if you have a workflow that is complex and all of the steps have to complete. You can do things like parallel executions. You can use it for doing things like fan-outs and these other complex patterns. But really the main thing to understand is the ability for you to have a lot of control over how all of these different steps process and what the retries and all the guarantees and things like that.

The reason why that is super important is because if you just try to use more of a choreography approach, and I'm a big fan of approaches to distributed systems or to microservices for very loosely coupled things where it’s like if for some reason the marketing update service didn't trigger right away and we had to manage retries in there, that's not as big of a deal as if we weren’t unable to process somebody’s credit card and then we still ended up shipping them something. That would be bad.

If you have those very strict workflows using, something like Step Functions just gives you all that extra control and it just – It’s just sort of a smarter way to do it. The step functions, the original step functions, are long-running. You can run them for a year. You can do all kinds of cool things where you can even have a system that when it's eventually done its job, it can ping the step function and say, “Okay, I'm done in this step,” and that can kick-off and do something else. Again, that can be a year. It can run for quite some time.

The new ones that they launched a year or so ago were the express workflows, and those are the ones that need to run within five minutes. They’re for very quick sort of things. The order process could be something that would fit into that because, again, most likely that whole order process would run very quickly. But those are for, again, very high throughput state transitions because regular Step Functions are little bit pricey when you have to – It’s something like 2.5 cents per 1000 state transitions or something like that, whereas the express workflows are very, very inexpensive and they can run very, very quickly.

But overall, they’re state machines and they allow you to build these complex workflows and there is the visual piece of it. It’s not a visual builder. It's a visual sort of renderer, I guess. It shows you – You build it all out in JSON and it will show you how it works. It gives you this visual representation, which is very nice to see. So it's very, very helpful.

Typically, the workflow or I should say the way that Step Functions work is that you would build these things out and the task would generally execute using Lambda functions, because that's where you can run that business logic to handle whatever that services you needed to do.

[00:48:44] JM: There was this announcement. I think this was a few years ago or maybe less than two years ago, but about Firecracker, which is an underlying runtime for serverless applications. I think it's for functions as a service. Do you have any knowledge of how Firecracker has changed the underlying runtime I guess at least for functions as a service?

[00:49:08] JD: Yeah. Firecracker isn't actually a runtime. It's actually the container. It's the virtualization layer itself. It's the way that you can run these very, very lightweight containers in order to serve up functions as a service. If you're familiar with like Hypervisor and those sort of things where with VM's it's sort of a heavy process to startup a new VM on a large machine. It takes a couple of minutes to spin something like that up.

What Firecracker did is it basically just gets those containers closer and closer and closer to the bare metal of the machines that they run on and allow you to spin up these containers that are very, very secure, they're lightweight, but they can spin up these containers very, very, very fast, and that's what they've moved to running their Lambda functions in.

Essentially, Lambda functions themselves are containers. I guess it's transparent to you that that's what's actually happening, but they do spin up a container that is running a Linux server and they run that on top of Firecracker, and that's that virtualization layer. That can be done very, very quickly, and what they're also doing is moving things like Fargate to run on those as well. So just the ability for you to run these services, run them very quickly and get them loaded very quickly, which is the big thing, but still have all that isolation and still have all that security. That's what Firecracker is about.

That was sort of one of those big announcement that they made where people were like, "Well, yeah. Yeah, okay. What does it matter to me?" because it really doesn't matter to most people. It just means that you're able to run Lambda functions much faster behind-the-scenes. But what it does is it actually spins up the container, and then within that container is whether or not is runtime that is installed, whether it's Java, or Node, or Python or something like that.

[00:50:59] JM: I would be remiss if we didn't talk a little bit about podcasting.

[00:51:02] JD: Sure.

[00:51:03] JM: Since you run Serverless Chats, I'd just love to know about your experience in the medium and what you like and whether you're optimistic about the future of software podcasting.

[00:51:15] JD: Yeah. So I had never done podcasting before. I have a newsletter that I do on serverless called Off By None, that I've been doing. I think I'm on issue 82 or something like that of it now. It comes out every single week and it's essentially just a wrap up of everything that has happened during the week in serverless. I focus a lot on AWS, but I try to pick some stories out from other clouds as well. But every week, I put 40 or 50 links with descriptions. It's a long process to do this.

I was doing that and I kept on seeing all of these really interesting stories that people were writing. Stories of them implementing serverless and stories of them – Or problems they were having with serverless or confusion around what are best practices, or what are leading practices? The constant containers versus serverless arguments and some of these other things, and people coming up with a million different definitions of what serverless was.

I was going to conferences and I was having chats with people that I wasn't recording and they were just really interesting to me. I thought to myself a year ago when I was having these conversations I was like, "Oh! I would've loved to have heard this a year ago," like somebody else have this conversation. People who were further along in their journey.

I said, "You know what? I'm going to try a podcast." A ton of work to get set up. More work than I thought I was going to be. But once I started doing that and having these conversations with people, I mean, it's just been amazing. I mean, I love chatting, as you can probably tell. I love talking about serverless. So finding other people that are as excited as I am and bring new perspectives and different ideas is just super fascinating. I always learn something new every time I have a new guest. I always make a new friend. Then I bump into them at a conference and we can have another conversation.

I think I've heard a lot of feedback of people who say that sharing these conversations with them not only are entertaining, and I try to make my podcast a little bit entertaining at least, but also very educational and these things where people can go back to over and over and over again. I mean, I know you actually had an interview with Rick Houlihan from AWS. I recorded an interview with Rick Houlihan I think shortly after you did. It ended up being two episodes long, and I go back and listen to it myself just trying to catch all of the nuances and all the little details that are in that episode. I mean, it's great, and I think that this is a medium where people can consume it when they're in the car, when they're sitting on the train, when they're out for a walk or for a jog or something like that.

[00:53:42] JM: Where they're in quarantine.

[00:53:43] JD: When they're in quarantine. Exactly, right? But I think it's a really cool medium because I can't sit and watch videos. Not on software stuff, like not for more than a few minutes. Like more than five minutes or so, if you don't get to the point, like I kind of lose it. But I love listening to shows like yours and some of the other shows that are out there where it's just you can listen to somebody. Get a bunch of knowledge. Just kind of download it into your head and you can be doing something else while that's happening.

I mean, when I mow my lawn or doing yard work or going for a run or any of those things, I most always have a podcast on. I know for me it's super interesting. I don't think I'm in the minority here. I think there are a lot of people who just love podcasts and love the idea of being able to get information like that that they don't have to stare at documentation that's probably poorly written and try to figure it out themselves or read 800 blog posts or have to watch videos for it. This is just this.

I always liked the scene from The Matrix where Neo wakes up and says, "I know kung fu." Just downloading information into your brain that's not a super active, not as active having to read or watch a video or something like that.

Anyways, I think there's a future for it, and I mean I know there's more podcast coming out. I hope more people come out with podcast. I would love to listen to more things. I mean, I'm going strong now. I just finished episode number 41. I've been doing this for almost a year now and I hope to keep going. I don't know how you do one every day or whatever it is that you do, but I'm trying to do my part and spread the good word of serverless as well as hopefully entertain people and give them another way to learn.

[00:55:21] JM: One thing I would say is it’s been remarkable to me how little my life has changed under quarantine, because I just spend probably too much time producing podcasts. That's like been my big realization of this whole event, is like this is actually not probably a healthy way to live. The fact that I have been – I am under quarantine. My life is under quarantine, because I'm podcasting too much. I think there's going to be a dramatic shift in my own life after this, if this thing ever ends. Do you think are podcast listener habits changing with the virus? I mean, this thing is like consuming all of people’s spare brain bandwidth. I'm not sure people even want to hear about software topics.

[00:56:03] JD: Yeah. Well, I can tell you, I am getting tired of hearing about virus topics. I mean, I checked in on the – Anything that can distract you from some of these things. I mean, obviously, this is a really weird time we’re living in right now. Like you said, I work from home anyways, but now I've got my kids home because they're out of school for however long. I don't know. My wife is a teacher. So she's home. So they’re upstairs watching a movie right now while I'm trying to get work done here and have this conversation with you. So things have certainly changed.

Yeah, I don't know. I mean, I think it's going to be interesting to see. I think there's this weird transition period I think probably for the next week where people are going to try to figure out how to relive their lives. Not much has changed for me either because I’ve been working from home. I really haven't stopped much other than to check the news every once in a while. But I don't know. I mean, again, once everything kicks back up, maybe they’ll just be a backlog of episodes if people are going to need listen to on their way to work.

[00:56:57] JM: Let's hope so. Well, Jeremy, thanks for coming on. It’s been great talking to, and I'm a fan of your work. It's great stuff.

[00:57:03] JD: I appreciate it. Thank you so much.

[END OF INTERVIEW]

[00:57:13] 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 stock from L2 to L7 and 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]