EPISODE 475

[INTRODUCTION]

[0:00:00.3] JM: Developers can build network applications today without having to deploy their code to a server. These serverless applications are constructed from managed services and functions as a service.

Managed services are cloud offerings like databases as service, queuing as a service, or search as a service. These manage services are easy to use. They take care of operational burdens like scalability and outages. But managed services typically solve a narrow use case. You can't build an application entirely out of managed services.

Managed services are scalable and narrow. Functions as a service are scalable and flexible. With managed services, you can make remote calls to a service with a well-defined API. With functions as a service, you can deploy your own code.

Functions as a service execute against transient unreliable compute resources. They aren't a good fit for low latency computation, and the code that you run on them should be stateless. Managed services and functions as a service are the perfect compliment. Managed services provide you with a well-defined server abstraction that every application will need, like a database or a search index or a queue. We all need those for building our backend applications.

Functions as a service offer flexible glue code that you can use to create custom interactions between the managed services. The term serverless is used to describe the applications that are built entirely with managed services and functions as a service.

Serverless applications are dramatically simpler to build and easier to operate than cloud applications of the past. The cost of managed services can get expensive, but the cost of functions as a service can cost one-tenth of what it might take to run a server that is handling your requests.

Whether the size of your bill will increase or decrease as your company becomes serverless is less of an issue than the fact that your company will have more productive employees. Serverless applications have less operational burden, so developers spend more time architecting and implementing software.

It's been five years since the Netflix infrastructure team was talking about the aspirational goal of a NoOps software culture. This NoOps desire was about your software being so well-defined that you would not have to have regular intervention of your operational staff to reboot your servers and reconfigure your load balancers.

Serverless is a newer way of moving operational expense into capital expense. It moves us closer to this idea of NoOps. Today's guest, Randall Hunt is a senior technical evangelist with Amazon Web Services. He travels around the world meeting developers and speaking at conferences about AWS Lambda, the function as a service platform from Amazon.

Randall has given some excellent talks about how to architect and build serverless applications, which are in the show notes. Today, we're going to explore those application patterns further.

[SPONSOR MESSAGE]

[0:03:33.6] JM: Amazon Redshift powers the analytics of your business. Intermix.io powers the analytics of your Redshift. Your dashboards are loading slowly, your queries are getting stuck, your business intelligence tools are choking on data. The problem could be with how you are managing your Redshift cluster.

Intermix.io gives you the tools that you need to analyze your Amazon Redshift performance and improve the tool chain of everyone downstream from your data warehouse. The team at Intermix has seen so many redshift clusters, they are confident that they can solve whatever performance issues you are having.

Go to Intermix.io/sedaily to get a 30-day free trial of Intermix. Intermix.io gives you performance analytics for Amazon Redshift. Intermix collects all your redshift logs and makes it easy to figure out what's wrong, so that you can take action, all in a nice intuitive dashboard.

The alternative is doing that yourself; running a bunch of scripts to get your diagnostic data and then figuring out how to visualize and manage it. What a nightmare and a waste of time. Intermix is used by Postmates, Typeform, Udemy and other data teams who need insights to their redshift cluster.

Go to Intermix.io/sedaily to try out your free 30-day trial of Intermix and get your redshift cluster under better analytics. Thanks to Intermix for being a new sponsor of Software Engineering Daily.

[INTERVIEW]

[0:05:18.9] JM: Randall Hunt is a senior technical evangelist at Amazon Web Services. Randall, welcome to Software Engineering Daily.

[0:05:25.2] RH: Thanks for having me.

[0:05:26.7] JM: I want to start by discussing a use case of AWS Lambda, because we have been talking about serverless a lot on the podcast recently. One show we had was with a guy named [inaudible 0:05:37.7]. He was on the show to discuss the refactoring of infrastructure at a company that he was working at called Yubble.

The situation was that the social network that he was working at had extremely bursty traffic workloads. It was a social media site, and for most of the data was very little traffic, but occasionally there would be a social media start that would log on to the site and then a ton of users would come online to hang out with this social media star. The bustiness created really big problems, because it was in a situation where most of the day, okay they only need one or two servers. But occasionally, they need this massive amount of server workload quantity.

This turns out to be quite a common problem. You're an evangelist. You talked to lots of different clients. What are the kinds of bursty workloads that you see customers dealing with?

[0:06:34.0] RH: The situation you're describing is basically the hot shard problem, but applied in a traffic sense rather than a data sense, where Kim Kardashian or Justin Bieber's tweets are way more popular and need to go to way more users than someone else's.

I see that pretty often, but there are a bunch of different ways you can design around it sort of architecturally. But with regard to AWS Lambda and serverless applications, I think the best way to design around that is figure out what capacity that you need to be at, and you can set canaries and cloud walk or otherwise to keep those containers warm and running.

Then the other advantage of Lambda is, let's say 90% of the time that capacity isn't needed, so the promises of cloud, all that nonsense. Sorry, it's not really nonsense. It's just me being funny. But the promise of the cloud is this elasticity thing, where you can go in and you can say, "I want a million servers for five minutes and I don't want to be charged anything else afterwards."

You really can go out and ask for hundreds of thousands of invitation per second, and then go immediately back down to a thousand invitations per second.

[0:07:58.6]JM: The reason I'm starting the conversation with this discussion of the bursty workload is I think it exemplifies some of the virtues of the serverless architecture as defined by an architecture that uses a function as a service on-demand compute. Regardless of that, people have been dealing with bursty traffic events for a long period of time.

Just to give an idea for how the Lambda-type architecture makes things a little better, what are the other ways that we could handle bursty traffic, like using auto-scaling groups, or load balancers, the traditional way that we've been handling bursty workloads for the last 10 years, what are the ways that that works and what are the disadvantages of those approaches?

[0:08:52.2] RH: Bursty workloads for the past 10 years, I think 10 years ago it was a very different story. Even five years ago it was a very different story. Auto-scaling groups were interviewed probably six or seven years ago now. Essentially yeah, you would set up your auto-scaling group and it would be measuring some kind of check and typically that was an instance level check.

Eventually it was able to be a business level metric or an application level check that you were doing, that would tell you whether or not you needed to be adding more instances to your pull of things that are answering this.

Then you had a load balancer, and in this case ELB. An ELB would – Elastic Load Balancer would also be performing various checks in making sure that everything was running as it needed to be.

From time to time, you'd still run into issues. You might have a poorly – you may only be requesting instances of a certain type, in which case there is not enough capacity for that type. Then we introduce Spot Fleet. Spot Fleet allows you to say, "I want anything that would give me this amount of capacity at an instance level." But then you can go even further. You can go to ECS. ECS says, "I have this task that I need to accomplish and I need this amount of compute and this amount of memory and this amount of network. I don't care where it runs, or how it runs. Just give me that much stuff." Then you can go even further.

The other advantage that we recently added, I guess would be permanent billing. Previously, if you were bringing instances up and down, you would be charged a full hour for each of those instances that was in a spot and sends – where the spot market by the way is just this – it's like a price that's determined and fluctuates based on the availability instances. You can get 90% off what the on-demand instance cost would be from the spot market.

By going back to my point of previously you would be charged for a full hour every time one of these instances came up and every time you went down within that hour you'd be charged a full hour. But now, we charge per second. You can bring an instance up for two minutes and five seconds and you'll be charged only for those two minutes and five seconds and you can scale back down.

But then with ECS, you can granularize the stuff that you're getting. It's those continuum. As you move forward on that continuum from EC2 to ECS and then to Lambda, you get into this very small deployable unit, which is just, "Hey, here is my code. I want to run my code without having to deal with any networking or compute or anything else." The only dial that I have to figure out

is my RAM, my storage, my memory. Then I just turn that up to 11 and get my 1.5 gigs of RAM and run my code, and everything else is managed for me.

The deployable unit gets much smaller and the level of stuff that I have to worry about in order to scale also gets much smaller.

[0:12:02.6] JM: I think a lot of people have gone from a monolithic type of application that's deployed on a VM to breaking their model of the application into services that they deploy onto containers, and then deploying those containers onto Kubernetes or Amazon Elastic Container Service, that's ECS that you described, or mesosphere or whatever management platform of their services they want to use.

What does that re-architecture typically look like when people want to go from a containerized architecture to a place where they can move some of those – some of that container functionality to functions as a service?

[0:12:51.0] RH: This is the methodology that I use. This is what we used at SpaceX, and this is the method that works for me. I'm not saying it's the perfect method for all situations, but what I do is I like the idea of the Lambda monolith. You basically take whatever working application you have and you move it into API Gateway and Lambda.

This is for web applications. Other applications might benefit from a different approach. But for web applications, you take your flask app, or your Jingo app, or whatever, and you move it into Lambda and API Gateway. You can deploy these things with something like Chalice, or Zappa, or a bunch of other – the serverless framework there. There are a bunch of different community run and based frameworks that are really good for deploying all the stuff, or even SAM's, the Service Application Model, which is just a cloud formation transform.

I realized, I'm throwing out a lot of vocab here, but I can recap later. The way that I do it is I take this biblical honka code and I throw it into one Lambda function. Then inevitably something goes wrong. When that something goes wrong, that's my first foray into micro-services. I start decoupling components of my architecture over time, because in most of the situations that I'm in, it is more important to get it working than to get it perfect.

Now if you have the luxury of time, I would strongly recommend actually taking a step back, looking at your architecture, looking at the components that are available to you and restructuring the entire application to take advantage of Lambda, because what I'm describing this Lambda monolith is not going to take full advantage of all of the different components of Lambda and API Gateway. It's in fact pretty limiting, but it is a way to get started very quickly and it's something that I see a lot of customers doing.

[SPONSOR MESSAGE]

[0:14:58.8] JM: Thank you to our sponsor, Datadog; a Saas cloud monitoring platform for cloud infrastructure and applications. Datadog Datadog integrates seamlessly with more than 200 technologies including AWS Lambda.

With powerful visualizations, sophisticated alerting, distributed tracing and APM, Datadog helps you get a handle on the performance of your serverless applications, as well as their underlying infrastructure.

Start a free trial today and Datadog will send you a free t-shirt. Visit softwareengineeringdaily.com/datadog to get started and get a free t-shirt. Thanks to Datadog for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:15:47.7] JM: Can you talk more about that example of migrating when you were at SpaceX? I think that would be a pretty interesting case study, if you can run through a little bit more of what exactly you're refactoring.

[0:16:01.4] RH: Yup. I can't talk about it.

[0:16:03.8] JM: Okay. All right. It's fine.

[0:16:06.5] RH: Sorry.

[0:16:06.9] JM: That's all right. As to the mystery of SpaceX, I suppose. Okay, let's contrast those two architectural approaches we could take, where we've got – we're up against time and we want to migrate to serverless as quickly as possible and we just throw our entire monolith onto a functions as a service platform, and then we see what breaks. Then whatever breaks, we decide to refactor that breakage into more functions as a service. That's certainly one strategy.

The more prudent strategy is take a step back, draw out your UML perhaps and figure out which of these different components we should break into which services and which of these components, perhaps maybe we should move into a managed service. Maybe some of these parts of our application can be moved into some managed service that would be easier for us to work with.

What kinds of applications have you seen broken in that more prudent measured fashion? Is there a case study that you can talk about that would help to illustrate that more measured approach?

[0:17:19.5] RH: I'll highlight a couple different used cases here. You have real-time fall processing, which is what he Seattle Times uses. Let's say a photograph is taken, it's uploaded to an S3 Bucket and Lambda events triggered, that Lambda runs and it resizes all the images and optimizes them for various platforms. The New York Times also does a lot of similar components beyond just photographs, they have this idea of event streams that they are processing. That's really cool.

That's built on top of a Kafka instead of Kinesis. But somebody that uses Kinesis and Lambda is localytics. Localytics, they process billions and billions of data points in real-time, and they use Lambda to process both their historical data and their live data. So the stuff coming in over a Kinesis stream, which is like live event stream, variable to go through and generate metrics from this data to indexing, social media analysis and cleaning up the data and all that fun stuff.

Then you have ETL customers too, so Zillow they use Lambda and Kinesis to track a bunch of different mobile metrics in real-time, like clickstream type of data. Then with Kinesis and Lambda, they can go back and put things into Redshift, which is a data warehousing solution, or

even perform their analytics right there and immediately serve into the people asking the questions.

The interesting part of that particular use case is that it only took them about two weeks to deploy it. To go from, “Hey, we want to build this to deploy it and in production,” only took them about two weeks, which I thought was really cool. Then when it comes to back-ends, like the web application backend, I mean they’re probably – I’m going to say millions, but there are thousands upon thousands of different customers and use cases for the web backend situation.

The one that really stands out to me is A Cloud Guru. So they’re a AWS training website. But not just AWS, there are cloud training website where they serve commercials and videos and I think little quizzes and stuff as well. They’re entirely serverless. So they talk about what it cost for them to run their entire application serverlessly.

They built it serverless from scratch, which I thought was that’s different from the typical migration scenario that you hear about. So they’re truly born serverless. They talk about their architecture from time to time in various places, and you could probably look up one of their talks on YouTube. It’s worth checking out.

[0:20:04.0] JM: I am beginning to see the contours of what you might call a typical architecture for serverless applications. I talked to your colleague Danilo Pocha a while ago and he outlaid something similar. Basically, the commonality that I see is you’ve got some kind of queue of events that is your queue of changes that are going on across your infrastructure, and you have event handlers that are reading from that event queue and making changes to your stateful infrastructure.

Maybe you have a DynamoDB instance that does one thing. You’ve Redis doing another thing. You’ve got Elastic search doing another thing. You just have these little services that publish events to your Kinesis stream, or your Kafka stream of events, and then you have other event handlers, other function as a service, event handlers that are reading from the Kinesis stream and are communicating with your data stores and are updating the data stores.

This event handling is a really useful thing to put into Lambda functions, because going back to the burstiness, if you're just thinking of the event handlers as the changes that are going on across your architecture, well that's going to be a bursty kind of workload. You're going to have different times of day, you're going to have different rates of change appearing in your application and being able to think about the rate of change as the thing that is scaling variably with the functions and the service that seems to make sense.

Are most of the people that you're talking to, are they doing something where they have this event stream, where they're putting it on Kafka, or Kinesis or whatnot?

[0:22:09.2] RH: No. I would actually say, that's a huge portion, but I wouldn't say it's a quarter or – I mean, it's really all over the place. There are two different APIs for invoking Lambda. There is Invoke and then there is InvokeAsync. InvokeAsync is this idea of I don't care about the answer. I just want you to perform an action on this payload that I'm sending you. Invoke is, "Hey, I want to – it's the request response model. I have a request and I need this function to fire and give me a response preferably quickly."

Two APIs alone enable pretty vast swath of different applications. People always ask me for different use cases and my mind goes wild, because there are so many different use cases. I mean, one of the use cases that I built was just for communicating with my parents. I travel a lot for work and I check in on four square, so I basically pull four square API every five minutes. Pull them in my location and throw that into Dynamo.

Then whenever somebody text me, calls me, anything, I invoke a Lambda function that responds with my current location. That single Lambda function is responsible for reducing the amount of communication that I have to do, primarily with family members tremendously.

[0:23:32.1] JM: It's interesting. I guess, maybe I get over-indexed on the – when I talk these big companies that have had to scale. I guess, once you hit a certain scale, you want to do something where you have this event sourcing model. You're describing a world where there is obviously a lot smaller scale applications that don't need an event sourcing model. You just want your application. You have a small set of functionality like calling your parents in response to a certain event, and you don't really need a event buffer to handle that kind of thing.

[0:24:14.1] RH: Right. I mean, Lambda has its own built-in event buffer, so your request can get throttled and stuff like that. Even Alexa skill, so Alexa you can ask a question, "Hey, Alexa. What's the weather?" Alexa with theoretically fire off a Lambda function that goes and looks up the weather and returns back the response. You don't need a lot of other complex architecture for that.

Maybe the backend that Lambda is hitting to get that weather information is drastically complex. There are a couple of different layers. There is the invocation model, which is interesting to me. The design of it and the interaction with it is pretty cool. Then there is the backend model, which is – even beyond the backend model is kind of the data processing model. Lambda plays a point at all different parts of that architecture. It's the glue that holds all of your services together.

[0:25:10.6] JM: Are there any applications where people have come to you and they have said, "Hey, here is what our current application model is. How do we move to a serverless architecture, where you really had to stretch your imagination to figure out how to refactor it?"

[0:25:26.0] RH: Well, there are some things that don't easily lend themselves to a serverless model. It's not like everything has to be serverless. Maybe one day that will be the case, but I think right now – I'm not going to try and – if somebody has an application that doesn't lend itself to a serverless model, I'm not going to try and sell them on it.

Yes, I've had people come to me with stuff that was pretty data heavy. I don't want to sell people on this myth that Lambda can instantly scale from zero to a billion indications. It does take a little bit of time to go from zero to a billion. Not a lot. There are limits that are in place, primarily to protect the user. I think the limit right now is 1,000 simultaneous indications. That's a soft limiting if you open a request, you can get that limit raised.

It's a lot closer to this idea of instantaneous scaling. But there's still edges in places, and sometimes those edges can be a deal breaker for certain situations. FINRA uses Lambda pretty expensively to look at billions and billions of trades, but they're not doing it in real-time. Their response isn't necessary in real-time in order to make a financial decision.

Whereas, let's say you're a high frequency trader and you wanted to invoke Lambda and get a response and say Lambda was going to run this proposed trade against a model that you had stored in S3 and you wanted to load it from S3 and continuously update that model with each trade, all these other stuff. I think you'd be much better served by ECS in that case just because of the latency and also the intensity of the work.

GPUs and things where you need a significant amount of compute that will last beyond the five-minute limit of Lambda, those are places where you run into those edges of Lambda. The subset of customers that I deal with that have that is actually pretty small. I think the gross majority of the people that I talk to on a day-to-day basis can benefit from a serverless architecture. But that doesn't mean it's going to be the right thing perfectly.

[SPONSOR MESSAGE]

[0:27:50.5] JM: Dice helps you accelerate your tech career. Whether you're actively looking for a job or you need insights to grow in your current role, Dice has the resources that you need. Dice's mobile app is the fastest and easiest way to get ahead. Search thousands of tech jobs, from software engineering, to UI, to UX, to product management.

Discover your worth with Dice's salary predictor based on your unique skill set. Uncover new opportunities with Dice's new career pathing tool, which can give you insights about the best types of roles to transition to and the skills that you'll need to get there.

Manage your tech career and download the Dice Careers app on Android or iOS today. You can check out dice.com/sedaily and support Software Engineering Daily. That way you can find out more about Dice and their products and their services by going to dice.com/sedaily.

Thanks to Dice for being a continued sponsor. Let's get back to this episode.

[INTERVIEW CONTINUED]

[0:29:07.8] JM: You're talking about applications where state management over a long period of time is relevant. That or you might need a specific type of hardware that maybe a GPU is going

to handle your job a lot better than some random CPU. I think with Lambda, you can't specify what kind of processor it's running on, is that right?

[0:29:37.9] RH: You cannot. Basically, the more RAM that you provision, the more CPU you'll get as well. I think we make some guarantees about maybe not literally what instruction set, but at least the generation of processor that like I think we make some guarantee about the minimum generation of processor, but I'm not sure on that. Actually, my colleague Chrismon would be the right person to talk to about that one.

[0:30:00.4] JM: Okay. Well, maybe that's another conversation I can have some time. Why don't you help clarify for people why is statement management – given the constraints of building a function as a service platform, why is state management difficult and maybe what are some of the problems that exist today that will be solved in the future, or hopefully will be solved in the future.

[0:30:24.9] RH: A lot of state management can be solved currently with step functions, which is another kind of Lambda-focused service. You can basically have a JSON document that defines state transitions and inputs and outputs from various functions and you can have weight conditions and all that good stuff. It's like a workflow, or a state machine for Lambda.

I think that makes a lot of the state management easier. But if you were trying to maintain some sort of external state that requires either an intense compute job, or a very long-running process that's maybe training a model or something, that's definitely not the right use case for Lambda, just because there is limits put on how long functions can execute and also how many resources they can use.

Whereas, it is advantageous when take the model training instance for example. It's advantageous for you to get that model trained as quickly as possible. You don't care if you use millions of resources, because they're only going to be around for as long as it takes to train the model. Maybe there is a cost function for you where you can go in and you can say, "I'm willing to spend this much money to have this model ready in this amount of time."

With Lambda, there is no way to coordinate that gigantic piece of state, that gigantic set of matrixes. You can invoke a Lambda, go out and test three and say, “Hey, download this file because I think it’s 500 megs of temp space in Lambda.” But even then, downloading 500 megs, modifying it, chipping 500 megs off again, that’s not the right model. You want shared memory and stuff like that in order to be able to accomplish the goals faster.

[0:32:12.6] JM: When people are trying to build state into their serverless applications, are they typically using these step function handlers, or are they using something like Redis? What are the different ways to manage state in a serverless application?

[0:32:31.9] RH: I think a lot of people just talk right back out to Redis. DynamoDB is the really easy, straightforward way of managing various states. That’s what I think almost everybody uses. If you’re just talking about state like in API limiter, a re-limit or something like that, most of that can be handled with an API gateway itself.

If it’s another kind of limit, like you’re only allowed to invoke this endpoint this many times before I start to cut you off, then yes I’ve seen Redis is used a lot for that. You can run that in your VPC and you can run your Lambda functions on your VPC, so everything sits tightly together and can talk very quickly and stuff.

Then the other side of things is just if you’re talking about crud type apps, like create, update and destroy and delete, you can very easily go right out back to a relational database. There is nothing preventing you from making those sorts of updates. Even long-running transactions, not huge along running transactions, but transactions that lasts a minute, you can still complete those and run those within Lambda. You’re not limited in that way. That’s where I see most people managing state.

[0:33:48.6] JM: When people are deploying and managing their serverless applications, that they have figured out how to architect and they figured out how to refactor them, how does the day-to-day usage of their serverless experience – how does that go, like monitoring and testing and deployment, how does the overall experience of operating a serverless application work?

[0:34:17.7] RH: Well, I'll start with the dev cycle. What I primarily use in my dev cycle is SAM Local. SAM Local basically just spends up a – like docker containers that have an API gateway local Lambda and all this other stuff. It's basically like running Lambda, but locally. That's where I start.

Then it comes to deploy. I think you have this continuous integration model, which is you create a code pipeline, or a code deploy scenario where you're saying, "Hey, I want you to upload this Lambda function, or this artifact into Lambda and I want you to deploy this API." You can do all that through cloud formation, you can do it through terraform, whatever floats your boat, whatever you have the most expertise with already. If you don't have any expertise already built up, I would strongly suggest just focusing on cloud formation and SAM.

Then if you don't feel like setting up all that infrastructure and everything to start with, then you can just go over to CodeStar and it will spin it all up for you. CodeStar, you go in and you say, "I want a ruby on rails after. I want a fast cap deployed on Lambda," it will take care of it all for you, and it will spin up the git endpoints, so every git push you do it will run a set of test. If those test pass, it will deploy all that good stuff.

Then when it comes to the monitoring section, one of the things that I really like to do for my biggest applications is using AWS X-Ray. X-Ray is a type of instrumentation, and what it allows you to do is get a service map of all of the different kinds of things that are – all the different connections in your system.

Serverless applications – serverless microservices in particular differ from these monolithic apps in that you may think that your service graph is relatively simple, but when you look at it you realize you have data coming from tons of different places, tons of different clients and then going out to tons of different places as well.

If you have a trace ID coming in with these requests, what you can do is you can actually propagate that trace ID through all the other places where this data is flowing and you can generate this very pretty service map, then you can visually alert on that service map when something goes wrong, or a request getting throttled by DynamoDB.

Maybe I needed to turn on auto-scaling, or maybe I need to increase my read capacity, or my request to recognition, which is this computer vision API that we have. Or my request to recognition getting throttled, “Oh, maybe I need to open this four ticket to get more limit, or maybe I need to figure out why we’re resubmitting the same thing every time. Maybe hashing is broken.”

There is this company called Sky Scanner that was using X-Ray to alert – not really alert, but at least to instrument and monitor their system and they realized they had this one call that was taking a lot longer than they were expecting. When they looked and dove deep on what it was doing, it turns out they were reading from the cache every time and that was – it was always a cache miss.

Then when they looked at their code they realized, “We forgot to set the cache after we went to the database to grab the info.” That was something that they actually found in production with X-Ray and it only took them two or three minutes of looking through their metrics and they have that instrumentation available to them.

They will break it down by individual call. You can see like each call you’re making, this is what’s happening, this is the latency for it. Then beyond that code and instrumentation, you have the classic cloud watch model. You have your logs and your logs are going out to S3 and you can setup various alerts on those for 500s, or whatever your traditional log or learning mechanism is.

Then you also have the ability to create these custom dashboards. What I tell customers to do with the custom dashboards are rather than focusing on technical components, focus on some like business deliverables. To focus on some technically or programmatically confirmable business deliverable.

If that goes outside of a well-defined range, then alert and try to find some action that you can. The business metrics, you alert a human on. Then all of the other metrics, something breaking, something – a service not being available. Those are metrics that you alert on, but the alert should go to something that can take a programmatic action to resolve it.

You want to alert humans for business problems, but if it's a problem that can be technically solved, you want to try and solve it with, I'd say a Lambda function, but you want to try and solve it through some other means. That's my philosophy and it works well a lot of the time, but there are also a lot of complications when you're building that kind of – that level of alerting. It takes a little bit of time and investment, and it takes some familiarity with your system that not everybody has.

[0:39:32.5] JM: I know we're up against time, but you talked a little bit about machine learning and places where building a model, or working with a model may not exactly makes sense in the context of serverless architecture. Are there any use cases that you've seen where people are successful using functions as a service in conjunction with a complicated machine learning pipeline?

[0:39:59.5] RH: Sure. I don't' know if I can talk about the customer, but I will explain in bigger terms our use case. Running a model on Lambda is totally doable. Plenty of people do that. In fact, I think we have our own fork of MXNet.

[0:40:14.7] JM: It's like a model that you have trained and deployed. You deploy it to a function as a service, so people are just making stateless calls to that model.

[0:40:23.1] RH: Right, exactly. I mean, that's pretty common. I see it very often. But this other company wanted to taste each input that they had and use it to reinforce their existing model. They had an API gateway that would invoke a Lambda, which would immediately invoke another Lambda that had their call out to push the input into their model. Then they would invoke that asynchronously, then they would have their original Lambda just return whatever the existing model had.

Then they had a Lambda that would run every few hours and it would say, "Hey, start retraining this model," and it would trigger some instance that would launch, download all the new data, start retraining the model and then push that model wide again. Then they would – all the Lambdas would start pulling down that new model.

Then in order to refresh the Lambdas, they would update function configuration and then change environment variables or something, just to make sure that they were getting enough fresh container.

[0:41:40.8] JM: Okay. Final topic I just wanted to ask you. This is not very related to AWS Lambda, but you used to work at SpaceX, you've also worked at NASA. Bezos, Jeff Bezos the CEO of Amazon, he talks about his goal with Blue Origin, his space company to build an AWS for space travel. Like basically a platform where random hackers could build their space companies without too much upfront investment, like if I want to build a space transport company, I can spin it up using Blue Origins, whatever back in infrastructure.

It's a bright future. It's pretty exciting. Do you have any perspective of how far of that is, or what are trajectory to getting to that bright future will look like?

[0:42:36.9] RH: I think that particular future is pretty far away. One of the downsides that I found when I was working in the space industry is that it moves pretty slowly. That sometimes an advantage – so Jeff Bezos has the same where slow is fast. I like that to some degree, but as a super ADD technologist, it's also really hard to be on the same project for four years and then it still hasn't launched.

I worked on the Orion Project way back in the day and nothing that I worked on has made it into space yet. It's many years later. Then at SpaceX, what I liked about it was stuff that I was working on one day might be on a rocket 10 minutes later.

I think Blue Origin and SpaceX both have this idea of making that cycle of iteration much faster. When you're dealing with chemical propellants, there is a lot of danger. Stuff can go wrong, especially if you're caring people and people getting hurt. It's important to make sure that cycle of iteration is as safe as possible. I think those components balancing out and moving forward, getting all that backend infrastructure that you talk about is stable enough that it's as easy as taking a commercial air flight. I think that will take a while.

The commercial air industry took a while to get to the point that it's at today. It still is cumbersome and frustrating. I think there will be a even longer kind of spin up for the space industry. Maybe that wasn't very hopeful. Sorry.

[0:44:23.6] JM: No, no. I prefer realism to baseless idealism. Thank you. It's a good note to end on. Well, Randall I appreciate you making the time despite the fact that you're just on the tail-end of an overseas flight. You're about to go give a talk at a conference. I really appreciate you taking the time and I hope we'll talk again soon.

[0:44:45.0] RH: Yeah. Thanks a lot for having me. I appreciate it. I hope your other guests are awesome too.

[END OF INTERVIEW]

[0:44:53.6] JM: Are you a Java developer, a full stack engineer, a product manager or a data analyst? If so, maybe you'd be a good fit at TransferWise. TransferWise makes it cheaper and easier to send money to other countries. It's a simple mission, but since it's about saving people their hard-earned money, it's important.

TransferWise is looking for engineers to join their team. Check out transferwise.com/jobs to see their openings. We've reported on TransferWise in past episodes and I love the company, because they make international payments more efficient.

Last year, TransferWise's VP of Engineering Harsh Sinha came on Software Engineering Daily to discuss how TransferWise works. It was a fascinating discussion. Every month, customers send about 1 billion dollars in 45 currencies to 64 countries on TransferWise. Along the way, there are many engineering challenges. So there's plenty of opportunities for engineers to make their mark.

TransferWise is built by self-sufficient autonomous teams. Each team picks the problems that they want to solve. There's no micromanagement, no one telling you what to do. You can find an autonomous, challenging, rewarding job by going to transferwise.com/jobs.

TransferWise has several open roles in engineering and has offices in London, New York, Tampa, Tallinn, Cherkasy, Budapest and Singapore among other places. Find out more at transferwise.com/jobs.

Thanks to TransferWise for being a new sponsor of Software Engineering Daily. You can check it out by going to transferwise.com/jobs.

[END]