EPISODE 525

[INTRODUCTION]

[0:00:00.3] JM: On Software Engineering Daily, we have been covering the serverless movement in detail. For people who don't use serverless functions, it seems like a niche. Serverless functions are stateless, auto-scaling, event-driven blobs of code. You might say, "Serverless sounds cool, but why don't I just use a server? It's a paradigm I'm used to."

Serverless is exciting not necessarily because of what it adds, but because of what it subtracts. The potential of serverless technology is to someday not have to worry about scalability at all. Today, we take for granted that if you start a new company, you're building it on cloud infrastructure.

The problem of maintaining server hardware disappeared for 99% of startups and this unlocked a wealth of innovation. The cloud also simplified scalability for most startups, but there are still plenty of companies that struggle to scale. Significant mental energy is spent on the following questions, "How many database replicas do I need? How do I configure my load balancer? How many nodes should I put in my Kafka cluster?"

Serverless functions are important, because they are an auto-scaling component that sits at a low-level. This makes it easy to build auto-scaling systems on top of serverless functions. Auto-scaling databases, queueing systems, machine learning tools and user-facing applications. Since the problem is being solved at such a low level, the pricing competitions will also take place at the low level.

This means that systems built on serverless functions will probably see steep declines in pricing and their cost in the coming years. Serverless compute could eventually become free, or nearly free with the major cloud providers using it as a lost leader to onboard developers to higher level services. All of these makes for an exciting topic of discussion that we will be repeatedly covering.

Today's show is with Eduardo Laureano, the Principal Program Manager of Azure Functions at Microsoft. It was a fantastic conversation and we covered applications of serverless, improvements to the code-start problem and how the Azure functions platform is built and operated.

Full disclosure, Microsoft is a sponsor of Software Engineering Daily. We're planning meetups for Software Engineering Daily. If you are interested, you can go to softwareengineeringdaily.com/meetup and sign up for either an upcoming meetup, or just to be notified about future meetups.

Right now, we are planning to visit New York. We're going to be going to the Datadog offices. We're going to HubSpot in Boston. Those are both in March. In April, I'll be at TeleSign in LA. We're going to try to do more meetups this year. I hope to see you at one of them.

Also, we're announcing summer internship applications to Software Engineering Daily. This is a remote opportunity. If you're interested in working with us on the Software Engineering Daily open source project fulltime this summer, send an application to internships@softwareengineeringdaily.com. We would love to hear from you.

If you don't know about the Software Engineering Daily open source project, you can check it out at github.com/softwareengineeringdaily. You could find our apps in the app store. You can go to softwaredaily.com and see what we're building. I hope you enjoy it.

[SPONSOR MESSAGE]

[0:03:43.5] JM: LiveRamp is one of the fastest-growing companies in data connectivity in the Bay area. They're looking for senior level talent to join their team. LiveRamp helps the world's largest brands activate their data to improve customer interactions on any channel or device.

The infrastructure is at a tremendous scale. A 500 billion node identity graph generated from over a 1,000 data sources, running a 85 petabyte Hadoop cluster and application servers that process over 20 billion HTTP requests per day. The LiveRamp team thrives on mind-bending

technical challenges. LiveRamp members value entrepreneurship, humility and constant personal growth.

If this sounds like a fit for you, check out softwareengineeringdaily.com/liveramp. That's softwareengineeringdaily.com/liveramp. Thanks to LiveRamp for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:04:50.1] JM: Eduardo Laureano, you're the Principal Program Manager at Azure Functions. Welcome to Software Engineering Daily.

[0:04:56.2] EL: Hey, thanks for having me here. I'm super excited.

[0:04:58.5] JM: Yes, yes. It's great to have you. We've done so many different shows about serverless and the related technology set. I think the definition of serverless that I have arrived at is that it mostly breaks down into two things, which is that you have these serverless functions, the functions as a service, which are these small, scalable functions that you can deploy and then you've got managed services.

Those are two separate definitions of serverless that managed services would be like this big abstract services, where you just are not addressing specific servers, but with that preamble, maybe you could start us off by – correct me if you have a different definition of serverless and maybe we could start off with a discussion of serverless functions, which is what you are working on.

[0:05:50.5] EL: This is always an interesting debate on what serverles really is. The way we think about it is really from the developer point of view. If an application developer, where you're developing any code in the cloud, if you don't have to think about the service underneath you, it's serverless.

That goes into a lot of dimensions. You shouldn't have to think about your CPU requirements, your memory requirements, the operating system, where the server is located. The last you have to think about there, then the more frivolous you are.

That's pretty much the definition we go with – I know it becomes a little bit of a broad definition, so there is almost like a range of serverless, but that's how we see it. In our particular case in Azure functions, we have a serverless offering, which is called in upper like the consumption plan.

We also have functions a service dedicated and we can have functions running on let's say, IoT devices and the function runs on let's say a thermostat or something like that. That's not really serverless, because where are you going to scale to, because you can only scale within that device.

The ability of scaling and scaling up and down and even literally is something that you can have it in the cloud, but some other devices you can still do the event-based programming and that can be functions as a service, but not solely serverless. That's how I see it.

[0:07:18.0] JM: Okay. Well, let's drill into serverless functions, because that's what you're focused on. When are serverless functions useful?

[0:07:27.2] EL: There are several cases that come to mind, but I think where customers today are converging to and we do this a lot that we go out and talk to companies on how serverless is applicable to them is really to extend or innovate your existing application, that I think is when particularly useful. I'm going to give an example.

Let's say you have a working code, whatever application you have and typical special enterprise applications. They're complex, they consist of multiple pieces. You need to add something to that code. Typically, you have to go through the process of sometimes refactoring, sometimes extending something, sometimes you have to define a new API.

I think that's a really sweet spot for serverless, where you can think of your application – typically application already emits some type of events, and you can see which events your

function can hope to and then you can extend that with functions. Like more concretely with some that we see is tons of applications build things that log, that have some sort of logged information.

Those logs are not always structured. Everybody wants now to add more and more intelligence to their applications. How can they harvest that data and make it structured, process into it, process to some sort of machine learning model, and get the outcome of it. That's where functions come really. Actually, it's a collection of functions really. One that can, let's say listen to logs and when new entries come, you're calling to – you structure that data, you cleanup, you augment it, you normalize it, you put it elsewhere some sort of structure data set.

That one could do some aggregation and then you could send it through a machine learning model to come up with some outcome out of those logs. The meta-point is extend the existing applications and innovating on top of what you have I think is really right now the sweet spot for serverless.

We don't typically go around recommending rewrite everything as serverless, because in some cases it's really a lot of work for you to rewrite anything and having to structure and thinking the different way where things are – each little piece is stateless and runs for a short limit of time. Extending applications is really working out well for tons of customers.

[0:09:42.0] JM: A lot of the excitement around the serverless functions is the fact that they're auto-scaling. We've had auto-scaling infrastructure for a while, but I think what's new about the notion of serverless, and please tell me if you believe differently, but the serverless stuff scales both up and down quite effectively. This scaling up and down quickly and responsively to your application workload, this would've been desirable five years ago, but we only got serverless technology more recently.

I wonder sometimes why that happen. Was there some enabling set of technologies that allowed these serverless auto-scaling up and down platforms to really propagate and become more widespread recently?

[0:10:37.8] EL: The first part of what you talked about; one, what's different about it? Scaling serverless versus just regular platform as a service. By the way, I worked on the platforms of service, app service Azure for four years before jumping into functions. We had like you said, the auto-scale working. That was based on machine, was based on CPU and memory.

What's really different about serverless scaling is that it's scaled based on your code, is based on your codeless and so some type of event. Previous scaling was in some ways like really clueless about what you're coding. You just put code there, but the code doesn't matter in terms of the scaling decisions.
I think that's one concept that was introduced that was different. Therefore, all the cloud providers have to adjust to scale based on events and based on what your code cares. The second part of what you're asking is what allowed for us to do that now, right?

Part of it is just growth of the cloud in general and having the capacity for structure to provide that type of promise, because all the cloud providers are serverless, we give you the ability of – if you have a burst of events that come out of a sudden, your application will scale accordingly. For that, you need to have that spare capacity there available for you to meet that need.

Not the majority of the cloud providers inside of the global presence it's key to that. Otherwise, you wouldn't be able to fulfill that promise. That's one thing. The other thing is for us as providers of such services, how do you make that economically viable really? Because as you know, it's very financially and cost-attractive to our customer serverless, bcause you really only pay for when your code is running.

For us, in reality you can't just provision a VM and say, "Hey, I don't want it anymore," in terms of a cloud provider. We still have the VM to be used for some other customer. We have to get the technology mature enough to be able to say, "Okay, this VM is now used for one customer. When his workload is done, let's use that same VM for a different customer."

The platform have to mature to be able to make such tradeoffs, to be able to really use our available capacity in different ways and look at different customers to it. The other part to why is it only now, I think we also needed to have more maturity on event in technology pieces.

In Azure, we have now event hubs, service bus and queues for a while and now we have event grid. We needed that part to be mature for you to have mini-fault triggers for functions. We actually had this thing in app service called web jobs, which not sure most listeners mean about it, which is the basis for our Azure functions product.

That's when we start venturing into event-based programming and how to start looking into scaling based on event load. Getting that maturity of, "Oh, how do you do a custom trigger? How do you trigger based on something beyond a timer, or HTTP? How can you trigger based on some other event, like a item added to a queue?" There was the maturity of the event and technologies that was needed to for you to truly provide serverless.

[0:14:04.3] JM: You needed a lot more higher level functionality in all kinds of ancillary services in order to make this work is what you're saying, because you want things like triggers, you want like – you wanted to have a database where you can trigger a serverless function elsewhere and you want to have a queueing system, where you can trigger a serverless function elsewhere. Because that's how a lot of people use these serverless functions is event-driven tools.

These things don't just work magically together. You have to link them together somehow and you have to have very smart infrastructure that's well tested, that integrates with everything else, where one event can propagate to triggering an event on another piece of the infrastructure and it has to work homogenously consistently across your infrastructure. That's the stuff that had to be built. It wasn't like there had to be some fundamental invention. It was just a lot of engineering.

[0:15:07.2] EL: Exactly. I think the for Azure cloud provider, the engineering is where some of these things really – it's really where the complexity was. For it to have an idea before we had such mature event infrastructure, people would write web jobs that were timer-based that would wake up, let's say every 5 seconds and would check, is there a new item on the queue for me before there was a queue trigger?

You can see how that one is inefficient. Two, as a cloud provider we didn't look into the depth of that queue to know how much can we scale this guy. Once you had a queue trigger, now you

say, “Okay, my queue length is such that I will need hundreds of machines in parallel to process this properly.”

Whereas before, when you had very simple type of triggers, like really, like scheduler type of triggers, you would have to deal with it yourself in your code and to be able to deal with that complexity. Through those tons of platform work on the eventing side, like I said and the other one is how to allocate capacity in a very smart way, so it still as a cloud provider we can fulfill a promise, but it’s also always have the capacity available for you as a customer.

[0:16:20.0] JM: The comparison between the scaling with your serverless functions versus previous scaling infrastructure, so what you said was that the previous auto-scaling models were more about the resource consumption and the newer auto-scaling model with the serverless functions is more about an awareness of the code that you’re writing, the volume of code, the volume of execution.

Could you please drill down a little bit deeper into the difference between those two? How are serverless functions aware of execution in a way that’s different than just resource consumption?

[0:17:01.1] EL: For sure. I’ll give an example. Let’s say you decide to go to function that listens to a queue and items come in the queue and you want to process your function, now what do we do in the platform; one, as soon as you write code and you say, “My trigger is a queue type of trigger and you are having your config, you’re going to tell us where the queue lives.”

As a platform, we’ll be able to now and we have a component up on code scale controller, which really is the one that decides how to scale things, that to be able to look into that queue and see what’s the incoming rate really of items being written to that queue, to make decisions in advance for you, because a lot of times what happens, the big difference there between machine-level scaling and code, if you will, or even-level scaling is the time it takes for you to scale.

A lot of customers that scale just machine-based sometimes their CPU gets to a 100%, or the top amount of memory they have in that hardware when it's a little bit too late. Requests are going to start failing, because they hit that limit, or they are just spiking way too quick.

Now with looking at the key with this scale controller type of technology that we have in advance and see what it looks like, also knowing information about the current executions and how long it's taking, how many resources you're using, you can make really smart decisions. You can make decisions on not only how much – how many VMs you will need in parallel to process those requests at an acceptable rate that really meets the goal of the incoming rate in which events are written to the queue, but where to really put it.

Do I need to pack multiple requests into the same machine, because it's a request that doesn't take that much memory, or CPU? Or do I need to really process this massively parallel? That's the big difference the platform had to build some of these components that skew based on your code.
We truly inspect your code, see what the trigger type is like and do some smarts there. One that's pretty different will be HTP, because it's not predictable and some of the other workloads that you can really seen advanced what's coming, so that is one that we deal with that pretty differently.

[SPONSOR MESSAGE]

[0:19:27.7] JM: Your enterprise produces lots of data, but you aren't capturing as much as you would like. You aren't storing it in the right place and you don't have the proper tools to run complex queries against your data.

MapR is a converged data platform that runs across any cloud. MapR provides storage, analytics and machine learning engines. Use the MapR operational database and event streams to capture your data. Use the MapR analytics and machine learning engines to analyze your data in batch, or interactively across any cloud, on premise, or at the edge.

MapR's technology is trusted by major industries like Audi, which uses MapR to accelerate deep learning in autonomous driving applications. MapR also powers Aadhaar, the world's largest biometric database, which adds 20 million biometrics per day.

To learn more about how MapR can solve problems for your enterprise, go to softwareengineeringdaily.com/mapr to find white papers, videos and e-books. MapR can leverage the high volumes of data produced within your company. Whether you're an oil company like Anadarko, or a major FinTech provider like Kabbage, who uses MapR to automate loan risk, and has done 3 billion dollars of automated loans to date.

Go to softwareengineeringdaily.com/mapr to find out how MapR can help your business take full advantage of its data. Thanks to MapR for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:21:09.8] JM: I do not spend as much time actually building systems as I would like. I often like to just ask people how they're building stuff and what programming stack they're using these days to try to get the temperature of how people are building stuff. I know that people are using serverless functions in a variety of different places. Do you see them in widespread use within Microsoft?

[0:21:37.1] EL: We see them used in tons of scenarios, like internal and some external ones. It still compare to platform as a service for instance. It's still not just not as mature and hasn't been out for that long. Although we do get widespread use, I think the usage is doing crazy. I think we're still seeing the growth curve of serverless.

In some ways, I feel like although our own product has been generally available for over one year, it's still somewhat early days, if you will, especially compared platform as a service. What's interesting there is we see the number of applications is really increasing. The applicable scenarios are increasing. In the past people thought, "Oh, we can't do orchestration type of workloads with functions. Or we couldn't do map reduce type of scenarios."

Now we're seeing those really coming together; one, because the technology grew and because truly customers just tried these things for different scenarios and they succeed. We're seeing a really good pick-up internally and externally really.

[0:22:43.1] JM: Is there any hurdle to evangelizing them in terms of explaining the event-driven nature of it? Because this is a responsive event-driven paradigm, it took me a while to really understand and maybe I still don't quite understand, but to really understand the nature of why there is this connection between event-driven and these serverless functions. Is that a problem with the adoption internally? Is it just evangelism of the event-driven paradigm?

[0:23:16.1] EL: I think if you are internal or external, I think that applies, which is if you truly haven't been doing anything event-driven and you've been doing applications your own way, monolithic, controlling state and you have your own model of developing things, there is always the usual problem of introducing a new technology, a new way of doing things.

There are a few concept aside from event-driven, there is also the fact that now you have – typically you want things to process asynchronous too, which is another route that some people are just initially not – that's not what they do. Making things stateless, make them run like short amount of times. Like functions, they run 5 minutes, or if you change setting up to 10 minutes, but still relatively short. It's not something that runs forever.
There are all these new concepts really. Although it could hurt the adoption, what we see though is when – we truly observer, we go to customers and see them trying serverless for the first time and look what their reactions are going to be like. They start grasping, like the concept of events like if you did like Y programming, there is events from way back when you've done this. When did like C++, there would be events if you done a UI. The comps event itself and responding to events is not new necessarily, but when you put this altogether in new technology, there is always – it's a new way of doing things.

We see some people take a while to adopt, but as they try and that's the thing I really like seeing, when they try you see them eyes light up and get excited about how easy it is to get started. I think it's probably the easiest way for you to just put a quick application out there today.

[0:25:03.3] JM: You mentioned something pretty cool, which is that people are using these things – if I heard you correctly, the people are using these things to do map reduce jobs, is that right?

[0:25:11.8] EL: Yeah. It's somewhat new and I don't know if you had on the show before, but in all cloud providers, we all are serverless mature, everybody invested a little bit on how to orchestrate process, because of this limitation of functions being short-lived and in reality you have workloads that require state were to be long-lived. All of us cloud providers I should say on behalf of the other ones is provide some sort of orchestration mechanism.

In Azure, we had for a while Azure logic caps, which is these visual designer way of building application, integrates with tons of services, but it's not really code. For those that live in like really coding their behavior and coding the orchestrator, we launched in functions the concept of durable functions, which has been extremely successful, because of this type of scenarios that –

Let me tell you one scenario of what people are doing. I can tell a map reduce one too. There is a company I talk to and they are doing payment collection, and their payment collection consists of if you're late on your payment, they would text you and say, "Hey, you're late for your payment. It's time for you to pay." If one day later you haven't paid this and another one and they keep doing this for a few days.
The they go to e-mail after those days. Then they start an escalation process too, trying to reach other parties and they can send you – print something to send you a mail. It's a framework that can run over a month. Every time it runs, it really only runs for seconds, for like say an SMS is a real quick action. In the past, they had the infrastructure that was allocated to run this thing. It runs once a day really for a few seconds, but you paid for the infrastructure just be sitting there. Serverless did well.

The other one is you want this to orchestrate, so you want to know if you already sent a text message, because the text of your following text message is a little different. It is, "Hey, we already reminded you were coming back." The e-mail only comes after the text messages. You need to carry that state around. You need to know where you are in the workflow to know which task, or which function to run.

Things orchestrators, like durable functions being one of them allows you to implement such workloads. This particular example has cost me and we were super excited about it and really cut his cost, he was able to develop faster and all that good serverless stuff. That's one example.

Map reduce is when we see now, again with durable functions is when people can do fan out and fan in, if you will, patterns now with serverless. It's something that in our case, we have that now in preview it's on there. Super excited to move it forward and make it generally available, because enable so many more scenarios now.

[0:27:56.1] JM: One related topic we could talk about is the streaming system. We've done some recent shows about streaming, and applications will often write these large volumes of data to a system like Kafka, or you have a event hubs managed queueing, distributed system queueing service.

When people write these large volumes of data, they're called streams and this term can be really confusing to people, so I always like to redefine it. The way that I think about stream is just a big array that is append-only, and you can take chunks of that array that might be in Kafka, or in event hubs or whatever, you could take a chunk of any it, any portion of this really long array that's sitting in Kafka or event hubs that are partitioned by this – or that are separated into these different topics.
You can do performance – you can perform operations over those large volumes of data. This is useful, because you have this one centralized place for all your data. All these patterns are emerging around Kafka and any kind of the stream storage system. Streaming plus storage, plus durability, these things are hard to explain on the fly, but anybody that's curious about this can probably look it up.

There's a abstraction of the stream where you have this append-only log of events and you can perform operations across it. This seems like a good place for using these serverless functions, because you have all kinds of operations that you want to perform across these different streams of data.

For example, you could get a stream of GPS coordinates coming in from IoT devices that people are wearing. This is the example I like to give is like, you're walking around with a Fitbit on and your Fitbit is constantly broadcasting your – or sorry, messaging your GPS coordinates to the queue, the distributed event queue stream, and you've just got GPS coordinates and then you could have some kind of function that is running and picking up those GPS coordinates and enriching them with other data, like the location that you're nearest to. It could ping the Yelp API and find a location that you're closest to and then write that back to the event stream on a different topic, so that now you have a enriched data.

It really gets interesting when you start to join streams together, or you start to do operations over lots of events across stream. You could do this – the map reduce for example, the fan out map reduce example that you gave would be perfect. If you've got a billion events across a certain topic, a billion GPS coordinates, you don't want to process those just sequentially. You would like to parallelized that processing.

Maybe you could talk a little bit about what you're seeing in the integration between serverless functions and the streaming systems.

[0:30:59.3] EL: It's definitely a pattern, I can tell you that much. One thing we've done is there not a product and we have a measure stream analytics, which listeners are over, but there is also event hub. What we see is two things; one, there is stream that's really fast-pace, like tons of events coming at the same time.

They might or they might not need processing on every one of them. You can through event hub, or event with event grid today, you can trigger functions to process let's say batches of them. You have a way to store them and put them on a batch and then have a function to process it and create these different batches that get all processing parallel. That's one pattern we see, and then you can have with durable function some type of orchestrator, you can have that step that's the first step that you process tons of things in parallel.

You can have another function that knows when that's done and aggregate all that information back. That's one pattern we see. We also see what you're seeing sort of anomaly detection, like

you process all the events, they go through some type of function that will find out which parts of the stream could be off according to the pattern that you're trying to observe.

I think the core concept here that's interesting is how do you go from something that streams, which I guess data is always coming to a function that's short-lived? That will process one entry at a time.

I think it's interesting to internalize, you could either have each one of those events trigger an individual function that will process that data, but in most cases what we see is that that's not optimal, because depending on how much volume you have in each one of these units, you could compact them to process much bigger workloads into each function. That's one thing we see. You can process tons of things in parallel if you do the right batching.

The other thing that's interesting is sometimes order really matters in those applications, if you think of video stream, media stream. You can't reprocess things out of order. Otherwise, you are messing up the incoming data source.

It could do those controls of making sure things are batched in such a way that when you're put back together, they put back together in the same order. We actually someone on our team just posted about how to process events and events in a orderly fashion.

[0:33:26.5] JM: Wow. Okay. That sounds like a cool post to check out. Because I certainly heard – I mean, on the show we've covered this a couple times about how hard it is to guarantee that you have exactly want processing. I'm about to check that out.

Since you're working on the serverless functions, we should focus on that a little bit more. I like to talk about the engineering behind building these serverless function platforms. We've discussed this a little bit around other serverless function systems. What happens when I deploy a serverless function, but have not called it yet? Then what happens when I actually call that serverless function?

[0:34:11.4] EL: Yeah, that's a great question. Let me go a little bit through the architecture and I'm going to – my explanation is based on the architecture we have in Azure functions. I would imagine it maps well with some of the other cloud providers.

The first time you create a function, in our world all the functions are part of – or an application. It can have multiple functions for an application. We'll give it an internal DNS, so you have a function, so in case it's HTP. In our case HTP by nature and anything else, but – you put that DNS entry, you allocate all those resources for your function.

When you're creating the function, you're going to tell us the platform two things, you're going to tell us what's a trigger type and what language you're going to program on. That's allows for us to know which dependencies you have, what do we need to bring and your code itself will also indicate some dependencies. Let's say you have no Javascript. You're going to say which NPM packages you want.

What do we do is the first time you create that function, we create all these resources that we know how to, send it to a new virtual machine that obviously there's a serverless customer doesn't have access to it, but as the infrastructure providers, we do.

We have tons of machines in a pool and we have this pool of machines we call skate units. We spread across all of our data center, 20 plus data centers that we have and there's a pool machine, they're sitting there waiting for it to be used, for it to be allocated to customer. We pick one of these machines and first, we initiate our platform code into that VM. That VM, it's a managed VM, that we know how to process it.

Then we put the functions specific type of coding there, so it's a function's runtime and functions host. Then ultimately, we put your customer code and your dependencies on top of all of that. Those three things happen and then your VM is ready to go, if you will, and ready to respond to request.

Now when you first request, this is always going to happen actually when your first request comes. The first time your function is being called, the frontends of our platform, which is a set of VMs recognize, "Okay, you need to hit this particular endpoint." This endpoint is not yet alive.

We have that skew controller component that I alluded to you earlier, that who know, there is no VMs allocated to it. It will allocate the first VM with that code.

Now from that point on we can look at the incoming rate of request, let's say if it's a keyword event hub and then allocate more events to it. We have to put these three pieces there. The VM management part, the functions host and your own code into each VM.

[0:37:05.2] JM: This leaves us to the code start problem, which we've discussed in the serverless episodes where the first time that you call a serverless function, your code has to be loaded into a VM and executed. There is some latency associated with that. The subsequent calls will be lower latency. How has your approach to the code start problem matured over time?

[0:37:34.3] EL: That's a great problem for us to have. I think all cloud providers that do serverless have that. What's happening like I was explaining, there is tons of step that happens until your code runs. Those steps are costly and it gets specially costly because of the amount of different languages we support and amount of different things you can do in a function.

There are a few things we've done to optimize that. We've done one thing which is we're trying to now keep tons of VMs already having the functions runtime code running and have already the management piece running. The only part that's not there, it's your own customer code. That was one optimization we've done on that space.

The other one is we try to be smarter about when to add more instances to respond to your request, because one interesting part about code start is people think it's only the very first request as a code start, but the reality is every time you're adding more VMs, the first request to in additional VM could potentially hit code start again.

As you're scaling very rapidly, you could experience multiple instances of code start. Now we're also be smart about when to put those additional instances around and starting them up as much ahead of time as we can, so it can predict what you need.

The last part we do is today, customers put the code server running a VM and they process some events and eventually don't have any more events to process. Normally what the platform

would do is after some minutes, I think around in our case was around 20 minutes, we would de-provision that, so you would go back to a cold state, if you will.

We tried to now be smarter about that to only the allocate you based in some heuristics. Now there will be higher chances that you are not the allocated as quickly, so you would not experience cold start as often. Some of the things I'm saying some where we implemented, some were in experimentation phase, but these are some of the bets we're taking as a platform to deal with that problem.

[SPONSOR MESSAGE]

[0:39:58.0] JM: This episode of Software Engineering Daily is sponsored by Datadog. Datadog integrates seamlessly with container technologies like Docker and Kubernetes, so you can monitor your entire container cluster in real-time.

See across all of your servers, containers, apps and services in one place with powerful visualizations, sophisticated alerting, distributed tracing and APM. Now, Datadog has application performance monitoring for Java.

Start monitoring your microservices today with a free trial. As a bonus, Datadog will send you a free t-shirt. You can get both of those things by going to softwareengineeringdaily.com/datadog. That's softwareengineeringdaily.com/datadog. Thank you, Datadog.

[INTERVIEW CONTINUED]

[0:40:52.1] JM: It's exciting to think about the economies of scale that you could eventually get to, because – so today maybe it's hard to predict how many instances of Java people are – how many serverless functions in Java people would want to run, so you have no idea how many JVM instances you would want to spin up and actually, I don't even know if you'd be able to do that, because maybe you need the code in order to spin up a JVM, but I'm not actually sure about that.

Can you maybe can spin up the JVM and then have code separately? Anyway, the same thing would be with NodeJS. I mean, you want the right numbers of NodeJS environments to be spun up on VMs so that you can just have – when serverless functions gets called, all that it matters is you take – grab the NodeJS code that somebody has you loaded onto a VM that already has NodeJS preloaded on that VM and you just execute it, rather than having to spin up a VM with that environment.

It's a problem that is going to get solved, which is what I find interesting about. Another one of these problems were it's like you look at it and you're like, "Oh, here's another problem where you just have to engineer a lot of stuff and then didn't work." There's not like a magical solution to solving it.

The advantage that you get is the serverless functions, which sounds like a really appealing way to write code. You talked about these durable functions a little bit earlier. I did some shows about these container instances, which are I think related to the spectrum of execution environments, that we've been talking about container instances where you have a – your deployable unit is a specific instance of a container and you don't have to think about a Kubernetes, for example. You don't have to think about a container orchestration platform. These things are new.

How do you see the usage of the different deployment units from serverless, to container instances, to Kubernetes instances, to maybe VM still? How do you see these different use cases evolving over time? Have you seen people using these in different ways?

[0:43:06.7] EL: There are compelling to use each one of them. One trend, to give one, one trend we see now with a lot of applications processing, machine learning and intelligence being the pattern and massive growth on Python and Python adoption. We see a lot of applications, they have not only code, but terms of dependencies with it, like machine learning, some model that you have to have closed your code, and sometimes on configuration that you have to have.

Containers is great for that, because you can just package all your dependencies, run it in different environments and they will behave the same way. That's great. Some people are really used to that environment.

Now serverless, you see it growing for the reasons you have been talking about. Folks want the rapid scaling on having to think about a lot of the underlying details of machine, how it scales and –

What we're starting to see is that two things marrying together, right? Like how do you get this ability to customize all I want, like have that high-level of customization, bring what I need, develop in a containerized fashion because some people like having that type of environment. At the same time, abstract from the underlying infrastructure.

You can deploy those containers somewhere, but the scaling is still event-based, because your scaling is not going to be based on the amount of dependencies. It's going to be how much can you process in parallel and what's your income rate of requests.

We're trying to see some of these patterns emerging together with containers and serverless coming together and Kubernetes too. You see them all having their own space and very successful. I think all of those trends are still growing We see them starting to play with each other. We have Azure functions running on containers today. It's not serverlelss yet, but we see that being a natural reaction as just merging both efforts.

[0:45:13.2] JM: I'm sorry. Wait, I didn't quite understand what you just said. The last bit. Azure functions running containers versus what?

[0:45:18.9] EL: Azure functions today has an offering that you can bring your own container. A normal Azure function, the serverless Azure function you bring your own code, no dependencies, it just runs for you, on infrastructure normally, today runs on Windows, right?

Although for developer, it doesn't really matter if you use our latest runtime, it runs on dotnet core, so it doesn't matter to you the underlying OS. Another option is if you have your own container and that container it's kept with running functions, you could extend that container and bring it in to Azure and say, "I want my functions to run on this particular container with my own customizations."

That's one thing we already – we have it in preview, because we're trying to match the concepts of function as a service and containers. Then moving forward, also bring in serverless into the mix.

[0:46:12.0] JM: Okay. I see. Talking more about some scheduling questions, because we're talking about all these different ways of deploying serverless, or deploying containers, I imagine as a cloud provider this leads to a lot of interesting conversations around scheduling, because for example the conversation that – well, the phrase that I just uttered in before my last question a couple questions ago was just the idea that you're going to have to schedule and provision these different containers with runtimes according to the runtimes that users are going to be wanting to execute functions on, like you want a NodeJS environment. You want enough NodeJS environments for the NodeJS serverless functions that are going to run.

These scheduling questions, they play out in all of the different products that you're going to want to build. Have you had any insights about scheduling and across these different tools that are being built, are people able to reuse the same scheduling paradigm, or does everybody have to write their own scheduler?

[0:47:19.6] EL: Your question is regarding Oz as the cloud provider how we –

[0:47:23.2] JM: As the cloud provider, yeah.

[0:47:25.3] EL: One thing you mentioned earlier and I meant to comment is we do have that problem you said, which is instantiating NodeJS or Python.exe. We can't do that every time that you want to run some Python code. We need to predict that, which makes it super interesting, because let's say if you support I think we want to support multiple languages, would support like six or more languages, and in reality in our data centers, we have limited capacity, so you can't instantiate every single one of those languages in case people want to use it. How do you predict that? How do you make sure things are ready and waiting for you? That's how I understand your question. A couple things –

[0:48:10.1] JM: Sorry. I'm sorry. I didn't articulate it very well. What I'm really curious about is the question of how you schedule containers. If you want to schedule a Kubernetes cluster, if

you want to schedule a container instance, if you want to schedule a serverless function, all of these seem like different request types where different types of resources have to be provisioned, because I'm wondering how much of that stack gets to be reused and how much lower level work that I would – I'm trying to ask a question that would be uniquely – you would be uniquely qualified to answer by virtue of working at one these cloud providers.

[0:48:49.4] EL: I don't think our scheduling bit that allows for us to provision the capacity or the functions to run when you need it, I don't think is a beast that we leverage across products really. I think each product have a very, very unique needs. We truly have to function as – it's only unique in the sense that we have all these different event types are coming, all these different languages that need support that we had to really write their own thing in terms of when to make things available for our customers.

There is, obviously there is products out there that specialize in scheduling services. It's a generic enough of a product that wouldn't fit our needs for Azure functions and probably for some of the other serverless architectures as well. This is a I would say a custom part and I would imagine cloud providers do it in different ways, because a lot of optimizations can be done at that layer.

[0:49:47.6] JM: Definitely. Let's move up the stack a little bit and talk about some higher level applications. You discussed machine learning a little bit earlier. How are you seeing people use serverless functions for machine learning models?

[0:50:02.5] EL: Two main ways; one is really in its infancy if you will, and one is a little bit more mature. There are different machine learning kits, I should say out there that allow you to get started and generate to your models of machine learning, like you would provide training data out, some data set and would spit out the model. Obviously, I'm abstracting a lot of the machine learning parts there.

Assuming that you have the model, a lot of it is how to get this model to run. How do you get that model to be wrapped into a web service and run that model when it's needed? I think that's where – I think is a great integration of serverless, because running such models could be called – could be event-based.

You never know when you want to call those models. Can be base in HP request, can be based on some other event. You could have a function that calls into – either calls into some sort of API, which is the machine learning API that process something and returns the results. We see a lot of text, or sentiment analysis where you sent blurb of text over an API and returns back some sort of sentiment, whether let's say a tweet if it was paused to or not.

We see that type of stuff, but now instead of calling API, we're seeing a little bit deeper integration, where the machine learning data scientist or whomever is working on the model can automatically updates its model and that gets sent to a function and be loaded as a dependency, a local dependency and not being like an external API that you'll have to call.

The function itself has what it needs to process that machine learning model and spit out the result, or upload that result to some other data source. That's one integration. It's becoming either the web service, or the event layer on top of the machine learning runtime. That's one. I think that one is more concrete. I mean, on our side we have integration with cognitive services and some other services out there that you could add intelligence, if you will, to your functions.

We were exploring the route of deeper integration with Azure machine learning. The steps in our mission is possible. Another route that's a little – I would say newer, I think we haven't explored as much is the training of the model itself, depending on the model is something that traditionally it could have been done with map reduce with tons of data requirements that you have to process, break the data down into small pieces, process them all, all these little pieces separately and bring them altogether into a single model.

Model training has been either resolved map reduce, or really high-powered hardware that can process that stuff. I think that's one route I think where serverless can help as well for you to do that distribution of that workload. I think model training is something that it's not too explored, but I think there's an opportunity there.

[0:53:06.3] JM: Yeah. I completely agree. That what you just described resonates with what I've heard from other people on other cloud providers as well. What about IoT? What are the opportunities for serverless functions in IoT-related applications?

[0:53:20.1] EL: IoT is super interesting, because by nature tons of these devices are noisy, if you will, they send tons of events. They are always saying, “Here is my temperature all the time, or here is my position all the time.” That nature, they’re event-based, they’re trying events all the time.

There are two opportunities here. There is one where instead of each one of those devices is sending ton of events, you could potentially have functions running inside the advice itself. There is some processing for each event before they get sent out. Some of these devices are getting more capable in terms of what they can do in terms of processing, which allows you to run some functions inside of them. That’s one pattern.

The other pattern is if this IoT device can be connected to some sort of gateway, or hub, or something that can send events to you, you can put the functions on those gateways to respond to those events. As there is burst of flow that scale serverlessly, those gateways can be on the cloud, so you can take advantage of the scalability of the cloud, which we’re talking about earlier.

You see a little bit of both, like in terms of functions as a service and running on the device, we had our product running on IoT edge today and we see companies that run manufacturing. Those companies are adopting that type of solution, where they can have coding process on the device itself.

A lot of folks on the team like doing demos on raspberry pies and things like that, that functions running on such devices. Your code is the same. You’re unaware of where it’s running. Like as a developer, you code the same way as you code for your function to run on the cloud.

[0:55:01.0] JM: Definitely. I guess, to wind down, I mean what are the other futuristic opportunities for serverless-like technology? For example, I think this notion that you have resources that scale up and down based on the volume of code that you have versus the resource management, this fundamental difference that you outlined, this difference in resource allocation, this is actually really important.

This is a fundamental, I mean cost emerges from your resource expenditure. If you can lower the resource expenditure, you're lowering the cost and you could potentially do that across every aspect of infrastructure. What are the areas of infrastructure that you find the most appealing to apply that change in the economics to?

[0:55:53.4] EL: There are few directions I think a particularly interesting. Let me tackle them one by one. One for instance, I find super appealing and we're not quite there so we're talking future direction, which is a lot of these super simple event processing technologies be running on the cloud. I think a natural evolution is to have more and more of that to be processed on the edge, right? To have those functions running.

You as a customer, you just imagine that now you can say, "Hey, deploy my function to this data center, or deploy it around the world." It's really being deployed to your cloud provider data center. Imagine if you could say, but you know I want this to be super low latency, because the scenario requires very fast response. I want that same function to run on the edge. I want them to run as close to my customers as possible.

I don't want whatever many hops it takes, like over 10 hops probably for you to get the data center. I want to be one or two hops and then I want answer. Having functions at the edge of the network, I think that's – it's just – I think is a matter of time type of thing that it's super appealing to customers. That's one direction we see it as going.

The other one is have tons of customers that are using the cloud, but they're just not using it the most effective way. There is tons of I would imagine hardware sitting idle for when you need it, right? I think that's where one is a little bit of folks getting used to serverless, implementing applications that way, and some is just workloads on the cloud provider side, being super smart about de-provision, re-provision things based on what's that workload used for.

I think we can make more and more things who are traditionally just provisioned for you and they stay there forever. Typically a VM, people don't bring it down often. What about making some of those applications, like more serverless? Things that only spun up when you need them. The critical piece like you said is compromise would be super smart about when to bring resources up and down. That's another direction we see.

The last direction related in mail that we're talking about is on one side you're seeing with all these talk about intelligence and IoT. If you'd imagine like drones and rocket ships that process tons of images and you want to do some image processing on them, which is really rich machine learning models. Now you're talking about very powerful hardware, or functions would need to be run on to process that type of scenario.

This image-intensive machine learning scenario for instance, a lot of times you need machine with GPU, or FVGA, some sort of acceleration on those machines. Wouldn't it be great that if you're as a developer you code your function and your function data depends on such model, and when you deployed the cloud provider, we could make the decision on where your function needs to run based on your processing needs, without you having to tell us.

Would be able by being smart, inspecting your code and dependencies that you would need that type of hardware and automatically allocate you there. I think that will be a really expanding the concept of serverless to also be adjust my processing and my memory needs according to what my code needs. That's another direction I think we'd get there. I don't know how long, but I think it will be natural to go there.

[0:59:23.9] JM: Well, it's definitely a bright future. That's really exciting to hear about all that stuff. I mean, those are a lot of the same ideas that I'm hearing from other people, but certainly some unique ones as well.

Thank you, Eduardo. This has been a really good conversation. I appreciate you coming on the show.

[0:59:38.2] EL: Yeah. Thanks so much for having me here.

[END OF INTERVIEW]

[0:59:43.0] JM: If you enjoy Software Engineering Daily, consider becoming a paid subscriber. Subscribers get access to premium episodes, as well as ad free content. The premium episodes

will be released roughly once a month and they'll be similar to our recent episode, "The Gravity of Kubernetes." If you like that format, subscribe to hear more in the future.

We've also taken all 650 plus of our episodes. We've copied them and removed the ads, so paid subscribers will not hear advertisements if they listen to the podcast using the Software Engineering Daily app.

To support the show through your subscription, go to softwaredaily.com and click subscribe. Softwaredaily.com is our new platform that the community has been building in the open. We'd love for you to check it out, even if you're not subscribing. If you are subscribing, you can also listen to premium content and the ad free episodes on the website if you're a paid subscriber. You can use softwaredaily.com for that.

Whether you pay to subscribe or not, just by listening to the show you are supporting us. You can also tweet about us, or write about us on Facebook or something. That's also additional ways to support us. Again, the minimal amount of supporting us by just listening is just fine with us. Thank you.

[END]

