EPISODE 1070

[INTRODUCTION]

[00:00:00] JM: Amazon's virtual server instances have come a long way since the early days of EC2. There are now a wide variety of available configuration options for spinning up and EC2 instance, and these can be chosen from based on the workload that will be scheduled on to a virtual machine. You might want a certain virtual machine for lots of memory or for machine learning workloads, and there're also Fargate containers and AWS Lambda functions. There're many, many options for somebody who wants to deploy virtualized infrastructure.

The high-demand for virtual machines has led to Amazon moving down the stack. Designing custom hardware such as the Nitro Security Chip and low-level software such as the Firecracker virtual machine monitor. AWS also has built Outposts which allow for on-prem usage of AWS infrastructure .

Anthony Liguori is an engineer at AWS who's worked on a range of virtualization infrastructure, software platforms, hypervisors and hardware. Anthony joins the show to talk about virtualization at all levels of the stack. This was an excellent episode and a wonderful window into how AWS works.

[SPONSOR MESSAGE]

[00:01:19] JM: If you listen to this show, you are probably a software engineer or a data scientist. If you want to develop skills to build machine learning models, check out Springboard. Springboard is an online education program that gives you hands-on experience with creating and deploying machine learning models into production, and every student who goes through Springboard is paired with a mentor, a machine learning expert who gives that student one-on-one mentorship support over video.

The Springboard program offers a job guarantee in its career tracks, meaning that you do not have to pay until you secure a job in machine learning. If you're curious about transitioning into machine learning, go to softwareengineeringdaily.com/springboard. Listeners can get $500 in

scholarship if they use the code AI Springboard. This scholarship is for 20 students who enroll by going to softwareengineeringdaily.com/springboard and enter the code AI springboard. It takes about 10 minutes to apply. It's free and it's awarded on a first-come first-served basis. If you're interested in transitioning into machine learning, go to softwareengineeringdaily.com/springboard.

Anyone who is interested and likes the idea of building and deploying machine learning models, deep learning models, you might like Springboard. Go to softwareengineeringdaily.com/springboard, and thank you to Springboard for being a sponsor.

[INTERVIEW]

[00:02:57] JM: Anthony Liguori, welcome to Software Engineering Daily.

[00:03:00] AL: Thanks, Jeff. Really happy to be here.

[00:03:02] JM: When I spin up in EC2 instance on AWS, that is a virtual machine on top of a physical host, and that's managed by a hypervisor. What are the responsibilities of a hypervisor?

[00:03:15] AL: Hypervisors are super interesting. It's the space that I love the most. It's something I've worked almost my entire career on, and effectively you can think about hypervisor like an operating system. It really plays the same role as an operating system, but instead of allowing you to run multiple applications at the same time, a hypervisor allows you to run multiple operating systems at the same time. The hypervisor running on the bare metal server is ultimately what carves up the resources and allows multiple virtual machines to exist in a safe and secure manner.

[00:03:48] JM: EC2 has been around for more than a decade. What have you learned about hypervisors over that period of time?

[00:03:55] AL: There's an awful lot that we've learned about virtualization throughout the history of EC2. When we first started, we used the Zen Hypervisor, which was really a state-of-the-art

at the time, and virtualization has really evolved in the last decade or so. When EC2 first started, there really wasn't a lot of hardware support for virtualization. Sort of the big theme in the last decade has been a lot of specialty hardware to enable virtualization to deliver indistinguishable performance from bare-metal.

At the starting point with Zen, it did a really good job of working with kind of what was available at the time. But then what we sort of learned after a number of years is that we really needed to invest in building hardware to make the virtualization experience as close to native as possible. That really got us started with the Nitro Project. The Nitro Project began in about 2012, and what we really asked ourselves was if we were going to design hardware not to run general-purpose operating systems in general purpose software, but if we're really going to design a hardware platform specifically for virtualization, for only virtualization and only EC2, could we make dramatic simplifications that would improve performance for customers? Improve reliability of the system, and ultimately give us a better product? That's really what resulted in the birth of Nitro.

[00:05:18] JM: We'll explore Nitro in more detail a little bit later. EC2 has a very broad range of use cases. There are users that have very different compute, and storage, and memory, and networking requirements that fit various use cases. How has AWS historically handled the fact that users may want a very specific configuration of hardware on demand?

[00:05:47] AL: Our goal with EC2 is to make available as many options for customers as possible. Through the years, when we first started EC2, there was only one type of virtual machine you could get. In fact, the earliest EC2 API, which is still the API of today, there's just more options to it. You didn't even specify an instance type, which is what we call the different hardware configurations. You just called AWS EC2 run instances and you got a virtual machine of a fixed configuration. Overtime, based on feedback from customers and different workload medians, we started introducing instance types as a concept. Instance types give you different shapes of memory, but also gives you access to different types of hardware platforms.

Besides being able to get a different ratio of virtual CPUs to physical memory, today we have instance types that provide AMD versus Intel processor types, ARM instance types through the Graviton processor family, but then also gives you access to specialty types of hardware. We

find that there is a lot of workloads that need really dense storage as an example. So we have a number of different types of instances that offer either hard drive based storage or large volumes of NVMe flash storage.

But then also pretty recently, we launched a series of instance types, which we generally call the N family. So these are things like C5N that provide really high-network bandwidth, so up to 100 gigabytes of network bandwidth for workloads that really need that level of data transfer. We've been introducing a wide variety of different instance types to be able to offer customers all the different kind of configurations that they need for a given workload.

[00:07:29] JM: I'd like to talk about another facet of virtualization, which is Lambda. AWS Lambda came out in 2014. How has the execution of AWS Lambda functions evolved in those six years since the first instantiation?

[00:07:50] AL: Lambda is a super interesting product, because it really changes the model. With EC2, when you run an instance, you're getting a full-blown operating system. The idea behind Lambda was sometimes all you really need is a function, right? You just need a single JavaScript function to do some glue code or you just need a little bit of Python here. The idea behind Lambda was what we now like to call function virtualization. Instead of virtualizing the full operating system, let's just provide individual function virtualization.

When we first launched that, the most important thing that we consider at AWS is security, and security within the context of virtualization is a really, really hard problem and it’s something we spend a lot of time on, and it's very challenging to get right. Our initial implementation of Lambda, reuse the security primitive that we felt very comfortable with, which is EC2 instances. Every Lambda function was an EC2 instance.

A few years ago, we launched a new virtualization technology called Firecracker, and Firecracker allows for sort of a purpose built function virtualization hypervisor if you will, and this allows much faster startup time along with some hyper plain technology. It largely eliminated the cold start problem which was caused a delay in the execution of a Lambda function based on whether the instance was already spun up. Generally, it has made a big improvement in the overall experience with Lambda.

[00:09:16] JM: Can you give me an overview of the virtualization stack that runs under a Lambda function today?

[00:09:24] AL: Ultimately, Nitro is – We'll talk about this in a little bit, but Nitro is our fundamental primitive for everything. At the base of the stack, you've got the Nitro hardware platform and the Nitro software, and that's running a bare-metal instance type. One of the cool things that we're able to launch with the introduction of Nitro is the ability to do bare metal instances. Within that bare metal instance, we are then running Firecracker, which is available actually as an open source project on GitHub today, and then Firecracker is the thing that actually runs the functions. Within the firecracker virtual machine, there's a small bit of code that sort of allows the function to get bootstrapped and execute, but otherwise integrates pretty deeply with the rest of the system.

[00:10:15] JM: Fargate came out in 2017, and that allowed for longer lived serverless programs. What were the engineering problems that Fargate presented that were novel at the time?

[00:10:30] AL: Containers is a super interesting space and it's very related to virtualization in the sense that what you're trying to do is have multiple workloads run side-by-side. A lot of the same concerns apply with respect to isolation and packaging and things like that, but the cool thing about containers is I think they have done a really good job of making containers a really great sort of code packaging environment, right? Even though I'm a virtualization guy at heart, like it's so much easier to create a Docker container than it is to create a virtual machine image. Lots of customers are super excited about Docker and love using containers for software packaging.

The real challenge with containers though is that they're not based on the same sort of fundamental technology as a hypervisor. They just use Linux as an abstraction boundary. The container is a lot more like an application than it is a virtual machine. What we really wanted to do with Fargate is allow customers to really just focus on running containers without having to worry about running the operating system that host containers, right?

That's really fundamentally what Fargate is, is it's allowing you just execute containers and not worry about the underlying operating system that's running those containers so that you can really just focus on your business and not kind of worry about maintaining the infrastructure underneath of it.

[00:11:54] JM: In all of these containerization that you're doing with Fargate and Lambda, there's a multi-tenancy model that gives you economies of scale, but presents some security issues. Can you describe the tradeoffs in the economies of scale versus security?

[00:12:15] AL: We never really ever when a tradeoff security. For all of our products, security is something that we don't compromise on. The way we solve this with these technologies is that we always use virtualization as the primitive. Virtualization is always the mechanism that we use to isolate multitenant workloads. Fundamentally, the difference between virtualization and sort of running an application is that there are special hardware capabilities and modern architectures.

On Intel, this is called virtualization technology or VT. On AMD, it's called SVM, but there's a special mode of the processor that you can use, and that special mode of the processor gives you a really strong isolation boundary. There's a whole lot else that you have to do, but it's a really strong building block. To The way we've solved this multi-tenancy problem with whether it's Fargate or Lambda or any other container-based technology is that we've built a specialized hypervisor that it still uses this fundamental building block of virtualization technology, but is really optimized for container use cases.

A really concrete example of this, if I run a Windows VM in EC2 today, there is a whole bunch of legacy hardware that's being emulated as part of that virtual machine. For instance, there's a VGA graphics card that literally implements the VGA spec from like from 30 years ago of CGA graphics and a whole bunch of stuff that you probably never heard of before unless you play old DOS games and stuff. In the world of containers, that's just not important. That that's all just wasted overhead. It slows down the creation process. It slows down launch time and all that kind of stuff.

With Firecracker, this is our hypervisor that's really oriented towards containers and function virtualization, we've kind of stripped out all that legacy, all of the stuff necessary for running

operating systems that are trying to support hardware from 30 years ago and really focused on just that isolation technology to be able to provide the secure multi-tenancy.

[00:14:25] JM: There was this effort several years ago. I don't know, maybe there's still ongoing efforts around something called unikernels. My understanding of unikernels is that it's like a Linux version that does kind of what you're describing where you strip away things like the USB port or the USB drive. You don't need USB drivers if you're spinning up virtual machines, but you're talking about Firecracker, which is a virtual machine monitor and then there's also the term hypervisor. I get confused when it comes to the terminology because it sounds kind of like what you're describing is something like a unikernel, what I heard about the unikernel, or it sounds like a hypervisor, but it's described as a virtual machine monitor. Can you disambiguate some of the vocabulary here for me?

[00:15:16] AL: Absolutely. I think one of the general challenges in the spaces that people talk about things and use terms interchangeably. I'll tell you the way I try to disambiguate these things, and other people might have different opinions here. But when I think of a hypervisor, I think of the full solution for being able to run some kind of virtualized platform, right?

As an example, Nitro I would call a hypervisor even though it entails a bunch of hardware and specialty software. I still would refer to the entire system as a hypervisor. VMware ESX I would refer to a hypervisor. They're kind of full solutions. Within a hypervisor, there is generally a lot of different components.

For instance, there's going to be a scheduler. There's going to be something that schedules tasks. There's can be a memory manager, and then there's a really special component within a hypervisor, which I like the call it VMM, a virtual machine monitor. The VMM is really the thing that implements all of the I/O emulation for the operating system.

For instance, if I'm running a Windows virtual machine on top of EC2 and it's trying to write to disk to the local storage within the OS, ultimately there's can be a bit of code running somewhere within the hypervisor that's going to trap all of the attempts that Windows makes to try to access that virtual disk hardware and then it's actually going to implement that virtual disk in software, and it's that kind of logic that I like to call the VMM because it helps to sort of

distinguish that from everything else. That's generally how I think about hypervisor versus a VMM.

A unikernel is actually super interesting area and I think it's one of those things that hasn't had its day in the sun yet. I actually think it's going to be something that down the road that becomes a lot more popular than it is. A unikernel really is all stuff that runs with in a virtual machine. Unlike a hypervisor or a VMM, which is running directly on the bare metal, the unikernel is really based on the observation that, “Man! I'm running all of these applications in the cloud or on top of virtual machines and they all do like one specific thing.” Maybe I've got a Ruby on Rails application or maybe I've got a Python application,” and the question that a unikernel asks is, “Do I really need a full Linux operating system? Do I really need to have new system and journald and all of the kind of baggage that comes from general purpose operating systems and could we really optimize things if we try to move a language runtime to basically run as an operating system?

There's been a lot of interesting unikernels throughout the years. There is a really popular one called Mirage, which actually based on the OCaml language. There's long been unikernels based on Java in the past. It's a super interesting space. It's not super popular today, but the sort of operating system nerd in me still is holding out that one day it will become a more common thing.

[00:18:21] JM: The VMM distinction versus a hypervisor. So did you say that – You think of the hypervisor as this full platform for spinning up distinct virtual machines on top of and you think of a VMM as a monitor of one of those specific VMs that you're spinning up? Do I understand that correctly?

[00:18:44] AL: Yup. That's absolutely correct. A really good concrete example of this would be if you looked at a system like KVM, sort of common usage of KVM sort of in the industry, the scheduler and the memory manager, you're really just using Linux, right? So KVM is kind of a small bit of software that's added to Linux to turn Linux into a hypervisor. Generally, I would refer to KVM and then QMU and user space as the VMM, and that addition of the VMM to the general purpose Linux operating system turns Linux into a hypervisor.

[00:19:20] JM: Okay. Firecracker is a VMM. Firecracker is a virtual machine monitor. It uses Linux KVM infrastructure to create minimal virtual machines. Can you revisit why you built Firecracker? The ultimate goal was to try to get to a point where we could launch really small workloads, namely containers and functions, in less than 150 milliseconds. That was actually how the entire project started. It was sort of a thought exercise of, “Okay. What would it actually take to launch in 150 milliseconds and with really minimal memory overhead?” The latter thing is a little subtle, so it's worth talking a little bit more about.

When you're dealing with function virtualization, it's actually really common for a function to just take a small amount of memory, like 128 megabytes is pretty common for functions. What that means is that you're going to have way more functions on a physical server than you would normally have virtual machines, because normally virtual machines are taking tens, even hundreds of gigabytes of memory, right? It's just different orders of magnitude in terms of resource usage.

When you're running this many kind of virtual machines on a single server, you really need to minimize that memory overhead. We started Firecracker with the idea that we wanted to create a virtualization platform where we could launch a VM in 150 milliseconds and it would consume less than a few megabytes of memory, and that's ultimately what we built with Firecracker, and it's been really exciting to see kind of all of the reaction in the community to it and there's been a lot of work that's been done around this. We’re now starting to see that there's a lot of other sort of open source projects that are following the same path with these, what now is often called micro-VM runtimes.

[SPONSOR MESSAGE]

[00:21:21] JM: Today's episode of Software Engineering Daily is sponsored by Datadog, a monitoring platform for cloud scale infrastructure and applications. Datadog provides dashboarding, alerting, application performance monitoring and log management in one tightly integrated platform so you can get end-to-end visibility quickly, and it integrates seamlessly with AWS so you can start monitoring EC2, RDS, ECS and all of your other AWS services in minutes.

[INTERVIEW CONTINUED]

[00:22:25] JM: What is the lifecycle of a Lambda that gets spun up with Firecracker?

[00:22:32] AL: When the request comes in, the first thing that the Lambda control plane will do is try to figure out if there's already a Firecracker instance available from that same customer running that same function, because ultimately the best way to get the best performance is just not to do any work. If you already happen to have something sitting around, then you want to use that and you want to try to use that.

If there isn't something available though, then what you would have to do is you have to spin up a Firecracker instance and that can happen very, very quickly. Then the next step is there's a bit of code that runs very early in the operating system boot. Generally, this is Linux for what it's worth. There's a small Linux image that's running within Firecracker, and that will kind of rendezvous with the Lambda control plane in order to get the actual payload that you're trying to run as your function virtualization.

Then as part of the initialization there, anything that's kind of a constructor within your language runtime – So this is kind of a subtle feature of Lambda where like if it’s C++,it’s literally static variable. But if it's NodeJS or something like that, if you've got code outside of your main function that executes kind of unconditionally, that will get executed to start with. Then it'll sit there and wait for the actual function implications, and then that's ultimately what the Lambda function returns.

One of the things that's kind of tricky in this model is some language runtimes, notably Java. JVM has a pretty long start-up cost. This is just the nature of the way Java works. It does a lot of translation ahead of time and it prepares the heap and things of that nature. So that work also

helps to get amortized by reusing existing virtual machines, which is one of the reasons that that sort of constructor model is so important.

[00:24:23] JM: Most of the shows, we focus specifically on software, but so much of your work is on EC2 and optimizing EC2 that you have gone into the hardware, and this is what Nitro includes. Why did you need to create specific hardware to improve the EC2 platform?

[00:24:46] AL: I actually kind of think there is a bit of a revolution happening right now. The way that I think the industry has historically thought about hardware is that hardware is Silicon that takes 5 or 10 years to build and it's this very waterfall type process, right? I think one of the things that's happening these days, and Nitro is a really good example of this, is that we're discovering that we can actually really shorten the hardware development process and that combining hardware with very, very purposeful system firmware, you can get really tremendous benefits at more of a software pace.

Ultimately, the reason we've decided that we had to invest in hardware technologies with Nitro was that if we go back to the earlier conversation about the virtual machine monitor and the role that it plays, one of the trends that you see happening in the industry, if we go back to the start of EC23, our network was built with 1 gigabyte networking technology. The same stuff that you probably have in your home, the cat5 cable that's kind of been ubiquitous for a really long time now since token ring, basically. But what's been happening in the last kind of 5 to 10 years is that like networking technology has just exploded. We've gone from 1 gig, to 10 gig, to 25 gig, to 100 gig, and the ethernet consortium just announced 800 gig. The innovation that's happening in terms of broad network performance is just incredible right now. We see the same thing happening with storage. We were stuck on spinning drives, and because of limits in physics, a spinning drive can't spin faster than a certain amount without exploding.

The performance of a spinning drive just really hasn't changed that much over the years, but then we had SSDs get introduced. This is flash-based storage, and that was a huge improvement, and that's just been within the last 10 years or so. Now we see things like an NVMe, which is an order of magnitude faster than SSDs, and on the horizon we're saying things like persistent storage, which is even faster. You have this incredible innovation happening in hardware, and if you're trying to emulate an NVMe drive, a modern NVMe drive by taking

software exceptions and handling that all within general-purpose software on your x86 processor, two things are going to happen. One is you're just never going to get great performance, because you're just not going to be able to keep up no matter how good your software is. But then the second problem is that you're going to have to consume a lot of those resources. You're going to have to take a lot of the resources of your x86 processor in order to be able to keep up with this performance. That's both bad for customers because it's wasteful, but it also means that there's just certain workloads that you're not going to be able run because so much of your processor has been taken away in order to do this I/O virtualization.

What we try to achieve with Nitro and the hardware that we built really enabled us was we try to focus on taking those really critical performance paths and creating specialty hardware for it so we could offload that to specialty hardware so that as much of the underlying x86 processor would be available to customers as humanly possible.

[00:28:00] JM: Right. In Nitro, you have the specialized hardware devices. You're moving some of the operations for storage, networking, security. You move these things into specifically created hardware and that allows you to improve the CPU and memory utilization of the virtualization platform. Can you explain a little bit more like if there's a tradeoff in the overall architecture? You're adding some complexity there with these additional hardware modules. Is there some cost to doing this?

[00:28:41] AL: There absolutely is, right? Nothing's ever for free, and the cost and the complexity of doing this hardware offload is that, one, it takes a long time. There's a level of development and effort that needs to go into it that is just longer than what a lot of people expect, because it involves actually building hardware. We began Nitro in 2012, but we didn't actually start talking about Nitro publicly until 2017. Actually, it took us five years of iterating before we got to a point where we really felt like, "Hey, we had something really great to share with people about how the overall architecture worked."

There was a lot of learning in that too. The way we introduced Nitro is very incremental. We started with networking, ten we moved to block storage, then we moved to local instance storage, and then finally we moved the control plane and we learned a ton during all of that. There was a lot of iterative development there.

The other thing that is just kind of an overall challenge is we set really high bars for our self with Nitro. Among other things, we set really high security bars. So we wanted to build a system that we really felt strongly would give the best possible protection for customers data. So there're a lot of things that we did within the Nitro system that created challenges for us, but ultimately it's better for customers.

A good example of this is that there is no kind of interactivity to a Nitro system. To give you kind of more concrete example of this, if I was to go and I was to create a KVM-based virtual machine using kind of off-the-shelf Ubuntu or some other operating system, I'm still going to have a full Linux install. If I have root on that server, if I SSH into it and I have root permissions, I have full access to the system.

Within Nitro, there is no concept like that. There is no concept of SSH or user accounts or any kind of way for anybody to kind of login in a privileged way. The whole system is designed strictly for creating virtual machines and operating at the level that EC2 operates on. Yeah, there's a lot of complexity for us, but ultimately that results in wins for customer, which is what we're here for.

[00:30:52] JM: When you make a dedicated hardware card for storage or for VPC, what's the process for designing the hardware spec?

[00:31:05] AL: The way we do it is different, and sort of the traditional approach to this is a hardware team goes off and they build a really detailed specification. They review it with the software team, iterate on it, they are simulation. It's this kind of very, very long and drawn-out process. One of the things it's been really exciting working on this project at AWS is that the software teams and the hardware teams work in lockstep, right? It's a very collaborative process for us. We don't just kind of trade specifications and kind of have dividing walls between teams. It's a highly iterative process and a lot of this just comes down to the way that our culture works.

Generally speaking, it starts like all things start at Amazon. We have a customer problem that we're trying to solve. We'll generally do working backwards narratives to figure out like what's the best way to solve the problem. Then the engineering team start working together to come up

with a solution and we try to get it done as quickly as we possibly can to get it in the hands of customers.

[00:32:08] JM: Hypervisors traditionally have this broad set of responsibilities, and the hardware that you design has the goal of breaking these responsibilities into specific hardware pieces, like you got the Nitro security chip, for example. Tell me more about the virtualization advantages that are gained from defining your own hardware.

[00:32:35] AL: One of the things I always look out for is simplifying assumptions. One of the things I really just strongly believe in is that the best way to make something better is to simplify it. One of the problems that the industry has been trying to solve for many, many years is how do we make sure that system firmware, your bias, things like that, the management engine or BMCs, how do we make sure that that software is protected? Because one of the necessities of this type of software is that the operating system has to be able to update it, because all software, no matter what it is, is going to have bugs. If you don't have a strategy for being able to update your software, then you're going to live with those bugs forever, and that's not a good place to be in.

You have this kind of fundamental problem that the earliest bit of software in the system has to prove that the system is secure before they can let other things run, but that software can be modified later. How do you make sure that you're kind of trusting your route of trust? There's really complicated mechanisms in the industry for dealing with this through mechanisms like UEFI secure boot, all sorts of like crazy, complex, cryptographic challenges, and signing schemes, and every time this has been implemented in the industry, you're going to deal with keys getting leaked and things like that. It's really complicated, and honestly it hasn't worked very well in the industry to-date.

What we tried to do with Nitro is something that is just much easier to think about and rationalize. Number one, software on the x86 processor never needs to update the bias, because those are virtual machines. That's the hypervisor, and we have these Nitro cards, so we have a side channel. We can rely on this side channel for doing the update.

The second thing is it's a lot easier to think about introducing simple hardware mechanisms. Every kind of flash chip has some kind of like write-protect pin, some physical pin on it or some mechanism that's relatively simple from a protocol point of view to say stop allowing writes. Just don't allow modifications to the flash anymore.

It turns out you can put a relatively straightforward ship in front of all of your nonvolatile storage and you can kind of mediate access and you can get this really strong guarantee that says, “You know what? I know that from now on, this media is not being modified. It’s just a really nice and clean, simple way. Then the other benefit of this is that you can do all this verification and monitoring before you run your very first instruction on the x86.

We generally call this first instruction security, and by being able to modify the hardware, we’re able to implement a much stronger security mechanism that's also a lot simpler and easier to understand. To me, like that fundamentally makes it better being a stronger mechanism that’s also simpler. That's the big winner right there.

[00:35:35] JM: We’ve talked through the brief timeline of virtualization technologies on AWS EC2, Lambda and Lambda getting improved through Firecracker. Then we talked about hardware, which is an improvement for the EC2 platform itself. Is there a connection between the Lambda improvements with firecracker and this conversation around hardware?

[00:36:03] AL: Absolutely. As we look at improving the overall platform, one of the things that we want to be very aware of is making sure there we’re making improvements for next-generation workloads too. It's great if we come up with the world's best hypervisor for running full-blown operating systems, but a lot of customers today are doing serverless. So we have to make sure that we’re thinking about how do we also innovate for serverless.

Fundamentally, the thing that enables Lambda or Fargate is this idea of having bare-metal instances, and you can find places that offer things that they call bare-metal today. But generally speaking, that's not really an on-demand model. Oftentimes, you have to order it and it takes like four or six weeks to arrive or they have kind of not a great kind of security story and you get limited in a lot of ways.

What we built with the Nitro security chip was a mechanism the provided security that was so strong that we are able to say, “You know what? As long as you're running a single tenant workload on top of a server, we don't actually even need hypervisor anymore. All of the I/O offload is happening in this Nitro card so that you can just talk directly to the Nitro card and that all just work fine.” We feel really comfortable about it, that we’re able protect all of the nonvolatile aspects of the system. Fundamentally, the ability to do bare-metal instances with Nitro is what enables us to innovate and be amends with things like Firecracker. It's all kind of build on top of each other.

[00:37:38] JM: The idea is that I can run a Lambda on a bare-metal instance that is equipped with the Nitro hardware because all of the I/O from the Lambda instance is going through this hardware?

[00:37:55] AL: Absolutely, yeah. All the I/O goes through the Nitro cards. Absolutely.

[00:38:01] JM: Sorry if this is naïve, but is I/O like frequently been the vector that's going to lead to security risks?

[00:38:09] AL: It's one of many, right? Ultimately, it comes down to what the surface area is. If you look at a modern hypervisor and what the surface area of a hypervisor is, there're really two classes of surface area. One class is kind of the CPU virtualization. The things that trick the operating system into thinking that it's running in a privileged mode, and then the other area is I/O virtualization. From a code perspective, I/O virtualization tends to be at least an order of magnitude, if not more, just more code because there's a greater diversity of hardware that you're trying to emulate.

[SPONSOR MESSAGE]

[00:38:55] JM: JFrog Container Registry is a comprehensive registry that supports Docker containers and Helm chart repositories for your Kubernetes deployments. It supports not only local storage for your artifacts, but also proxying remote registries and repositories and virtual repositories to simplify configuration.

Use any number of Docker registries over the same backend providing quality gates and promotion between those different environments. Use JFrog Container Registry to search the custom metadata of the repositories. You can find out more about JFrog Container Registry by visiting softwareengineeringdaily.com/jfrog. That's softwareengineering.com/jfrog.

[INTERVIEW CONTINUED]

[00:39:50] JM: Can you tell me a little bit more about what the interaction is between a Lambda that's running in addition to Nitro? Like is it just talking directly – Or is the Lambda getting spun up on a bare-metal instance and there's no EC2 in this case, or is there still an EC2 that's running under the Lambda instances?

[00:40:12] AL: Oh no. It's still what we consider to be EC2. Today, you can go into the AWS console. You can go and do the EC2 console and you can launch and i3.metal instance and you still use [inaudible 00:40:24] as your root volume. You still can attach EBS volumes. You can still attach VPC, EMIs to the instance. It still has instance metadata. In every way, shape or form, looks like an EC2 instance. The only difference is there's no hypervisor. It is exactly the same as any other EC2 instance in that regard, and that's all enabled by Nitro, because Nitro, it's more than just kind of that hypervisor software layer on the x86. It's the full hardware platform that enables it.

[00:40:56] JM: Changing the topic, a much larger physical piece of hardware that you have done work on is Outpost. Outpost is a full server product that users can buy and set up in their data center or their on-prem environment. When you were designing this spec for the first version of Outpost, you had to know that it wouldn't be possible to have every AWS service run on-prem. A lot of people are excited about Outpost because the potential is that I can have AWS on-prem, but of course implementing that in practice with just a single rack that you're giving people, that's not really feasible. How did you decide which services to make available through Outpost?

[00:41:43] AL: We've had customers asking about running AWS on-premises basically since the start of EC2. It's just been this common theme that people have been asking for and it's exactly this problem that you mention, is the thing that has kind of kept us from doing it even

though there's a lot of folks that have tried to do stuff like this. Because ultimately, if you try to replicate AWS in a single rack, you're going to have to make way too many compromises. You're not going to be able to have the same monitoring .You're not going to be able to use the same kind of microservices architecture. You're not going to be able to have the same scalability. It's not going to be the same, and I think the most common theme that we hear from customers is that when they say they want AWS on-premises, they want their applications to work exactly the same way. If I had an application that I was running in the Oregon region, I want to just be able to lift and shift that application and run it locally.

What we found in talking to customers is that there is very specific reasons why they need to run applications locally, and it's not because they need all of AWS. It's usually that either they have some kind of legacy on premises infrastructure, so maybe a legacy database, and are trying to convert their applications over to a more modern architecture. But in order to do that, they need to have equipment that's close to that legacy database, right? That latency really becomes important.

We also see a number of customers. Unusually, this is in the industrial areas, manufacturing, things like that, where they have to keep the manufacturing lines running no matter what. Oftentimes, these factories are in rural areas that have kind of sketchy WAN connections. So what they care a lot about is making sure that like they let up using the cloud. They're really excited about using AWS services, but the bits of software that ultimately are controlling the robots in the factory, that stuff has to stay on-premises, but they're really happy to use CloudWatch to store logs and make use of things like GuardDuty, where if the WAN fails for six hours, that's totally fine. They're okay with that.

Based on all of the stuff we built with Nitro and kind of the change security posture, one of the things we realized was that we could actually take a rack of Nitro hardware, we could put it into a customer's on-premises data center and we can manage it as if it was any other capacity within the region. We could use the same control planes. We could have access to the same services and we could make VPC span on-premises and in a region. Ultimately, that's kind of how we figured out how to sort of solve this problem and really make true kind of AWS available for on-premises data centers.

In terms of services availability, what we've really been focusing on is services that really need to have that low-latency interaction. As an example, some of the early services that we've made available in Outpost are ECS and EKS, and that's because these are services that are going to host you the software that ultimately is going to be talking to these robots and things like that.

We've also been looking at making databases available. So RDS is something that we've announced a preview for that will be coming shortly for GA, and that allows you to start bringing your database workloads and kind of doing that incremental conversion. It's still pretty early in the lifetime of Outpost, but like all things in AWS, the way we're thinking about bringing services to it is based on customer conversations and feedback.

[00:45:23] JM: If I want to have one of these outposts, that means that I'm going to be deploying a custom piece of hardware into my pre-existing infrastructure. What is the process for deploying an entire foreign server and integrating it with my pre-existing hardware?

[00:45:42] AL: One of the things that's been super fun for me in building Outposts is dealing with getting racks of equipment into customers' data centers. If you've ever been to a data center before, they're super fascinating. I actually try to go on and install as much as I can in order to just see the experience and see the diversity within data centers. But the overall process starts with submitting an order and then there's a survey that we have customers do so that we can work through things like can the floors between the loading dock and the actual final location of the rack support the weight of the outpost?

A like a typical outpost is going away thousands of pounds. It is quite significant to move around, and so we have to make sure that the path is clear and things like that. Then a stereo that always tends to be an interesting challenge is the way that we build the cloud, every single position in our data centers, every single place where a rack goes, it's exactly the same. It has the same power. It has the same network capabilities.

When we land racks in our data centers, we can just wheel the rack [inaudible 00:46:47], but what we find in customer data centers is that depending on how the data center got built, there can be diversity in the position. Sometimes one position has fiber-optic cables. Some just has copper cables. Sometimes it's 10 gigs. Sometimes it's 25 gigs. We have to work with customers

to figure out exactly what the position supports, and then based on that, we'll make sure that the right rack arrives and then we will make sure that the rack gets installed, that the power is correct and that the networking works out of the box. Then that is ultimately what will give you a fully functioning outpost.

[00:47:28] JM: We already talked about Nitro and the hardware that you custom-built for improving the EC2 experience. What is the importance of Nitro when it comes to designing the Outpost server infrastructure?

[00:47:44] AL: One of the really big challenges to solve in trying to have an AWS region span into a customer's data center is that we can assume that customers' data center is trusted. We have to assume that something bad can happen. Even though customers take a lot of care centers, we have to assume the worst.

A really common way that folks build their infrastructure for large-scale virtualization is things like they have shared [inaudible 00:48:15] or other mechanisms where they assume that because the data center is secure and controlled, you can kind of simplify some of the security problems.

In order to make Outpost possible, we really needed to have the Nitro model where every server is kind of fully hardened independent and kind of, crucially, the entire infrastructure is designed to very much be an appliance. Something that just kind of responds to commands and things of that nature, and so because of this overall design and architecture, we're able to take this rack of servers. Put it in an untrusted location and yet still be able to manage it in a secure and kind of faithful way.

[00:48:59] JM: I have a variety of other questions that are kind of taking a step back. If we'd come back to the question of functions as a service, one of the engineering problems that I've talked to several people about with functions as a service, like with Lambda, is the question of state management. If I am running a Lambda function, today the way that I am managing memory, for example, is I use Redis in many cases. I use Redis and I just store my state in Redis and different functions can access that Redis this instance. Is there a view towards getting to stateful functions as a service?

[00:49:49] AL: It's really interesting, because a lot of it comes down to what customers are looking to do and kind of the feedback that we've been getting. There's a lot of work these days on the container side to provide kind of similar functionality as function virtualizations. This is often called scale to zero, and a lot of people are doing scale to zero workloads on top of things like Fargate. It's not entirely clear to me if the answer is to figure out how to build stateless functions or if the answer is to bring kind of better scale to zero support to container runtimes. I think that's probably where the industry will end up going, but it's an interesting space. I think it's still really early days. From an AWS point of view, we're taking the lead based on the feedback that we get from customers in the space.

[00:50:39] JM: When you look at the current model of serverless computing, like you have backend as a service tools, you have these systems with hooks in them so they can essentially talk to each other, or you have functions as a service to glue them together where necessary. How close are we to the ideal of what people want out of a serverless platform?

[00:51:06] AL: I think people want change all the time, right? There's never a final version of anything. I think, today, like if you look at the way that we build things internally, we love Lambda. We build a lot of our services with Lambda today in its current form, but it's also easy to imagine lots of improvements down the road and lots of features that can be added. I think that it's all just part of kind of the evolving way that people are building applications. I think it's a super exciting space to see how it develops.

[00:51:36] JM: How do you envision the outpost platform? The on-prem platform getting closer and closer to what the ideal of an AWS experience would be like for on-prem?

[00:51:51] AL: What's really interesting about an outpost is that it really is the same hardware that we use in our data centers and it's running the same software that we're running in our data centers, and this is one of the things that makes it fundamentally different from anything else that's out there. If you order an outpost rack and you get it delivered into your data center, actually, there's search Nitro cards in it, there's the Nitro security chip in it. It's running the Nitro hypervisor or it can run bare metal if that's what you'd like to do. In terms of being core EC2

capacity, an outpost is already there. That already is what it and it's already able to do that and it has very, very high fidelity.

I think the thing that's going to be interesting and it's going to be one of the areas where there's a lot of evolution, is thinking about how customers build applications that really do span on-premises and in a public region. If you think about it, if we take a factory as an example, where there are some amount of software that you need to run to keep the conveyor belts going. There're a lot of things that you don't actually need to run locally, and a lot of these factories are really limited on physical space. they really want to introduce more computer to do things like machine learning, to do quality control and things like that, but they just have the space to do it locally.

There's these really exciting models that I think we're going to come to over the coming years of services that can kind of span, do a bunch of workloads in the public region, but then kind of do the really critical stuff locally, and that makes it really easy for customers to build these highly resilient applications while still getting all of the benefits of the elasticity of the public cloud. I think that's a super interesting space that we'll see a lot of innovation on.

The other thing that's really cool and is coming soon is the world is kind of in the process of building out a 5G infrastructure, and what this does is this brings a lot of additional bandwidth and much lower latencies pervasively just everywhere. There's been a lot of interesting discussion and thought about what happens if we can inject AWS compute in all of these different locations? We announced a project called Wavelength at last year's Reinvent where we're working with telco providers to bring compute capacity throughout the 5G wireless network.

I think there's a whole new set of things, new types of services, new ways to think about how as a developer you can engage with N-customers that takes advantage of this much more pervasive computing paradigm, and that all really comes from being able to have real AWS in all of these facilities that aren't just kind of the big regions that clouds typically have today. I think that's kind of the really exciting future to me, is the really kind of pervasiveness of AWS globally wherever you need it to be.

[00:54:58] JM: That is really exciting. We've talked about granular virtualization. We've talked about entire racks of hardware that you're building. You move up and down the stack. You investigate everything. Can you give me an area of the AWS infrastructure stack that you're starting to think about but it's too far in the future to start executing on?

[00:55:25] AL: I think that one of the things that is really interesting to me and still is kind of mind-boggling from me personally at least is quantum computing. We had a really exciting set of announcements around quantum computing at this past Reinvent, and we've got a really great team of folks working on it. Just as an engineer, it's one of these places where I feel comfortable with what I know about traditional computing at this point in my career and I feel like it's starting over with quantum. It's a completely different world that we just are starting to understand and just starting to figure out. That's one of the things that I'm really excited about, but we're just at the very, very, very beginning and it's really exciting to me that kind of the way that we're approaching it from AWS's perspective is that we're just making it accessible to people, because we're in this highly experimental phase right now and who knows what the right solution is going to be? Letting people start figuring out how it works and play around with it and try to figure out what the right solution is, I think it's exactly the right thing to do. But man! It is early days like. It has the potential to completely change the industry, which is so exciting.

[00:56:37] JM: Anthony Liguori, thanks for coming on the show.

[00:56:39] AL: Thanks for having me. I had a lot of fun.

[END OF INTERVIEW]

[00:56:50] JM: Apache Cassandra is an open source distributed database that was first created to meet the scalability and availability needs of Facebook, Amazon and Google. In previous episodes of Software Engineering Daily we have covered Cassandra's architecture and its benefits, and we're happy to have DataStax, the largest contributed to the Cassandra project since day one as a sponsor of Software Engineering Daily.

DataStax provides DataStax Enterprise, a powerful distribution of Cassandra created by the team that has contributed the most to Cassandra. DataStax Enterprise enables teams to

develop faster, scale further, achieve operational simplicity, ensure enterprise security and run mixed workloads that work with the latest graph, search and analytics technology all running across hybrid and multi-cloud infrastructure.

More than 400 companies including Cisco, Capital One, and eBay run DataStax to modernize their database infrastructure, improve scalability and security, and deliver on projects such as customer analytics, IoT and e-commerce. To learn more about Apache Cassandra and DataStax Enterprise, go to datastax.com/sedaily. That's DataStax with an X, D-A-T-A-S-T-A-X, @datastax.com/sedaily.

Thank you to DataStax for being a sponsor of Software Engineering Daily. It’s a great honor to have DataStax as a sponsor, and you can go to datastax.com/sedaily to learn more.

[END]