EPISODE 1096

[INTRODUCTION]

[0:00:00.3] JM: AWS has over a 150 different services, databases, log management, edge computing and lots of other services. Instead of being overwhelmed by all of these products, an engineering team can simplify their workflow by focusing on a small subset of AWS services, the defaults. Daniel Vassallo is the author of *The Good Parts of AWS*.

An excerpt from the book is as follows: “The cost of acquiring new information is high and the consequence of deviating from a default choice is low, so sticking with the default will likely be the optimal choice. A default choice is any option that gives you very high confidence that it will work. Having confidence in your workflow, even if it is a simple workflow has advantages. S3, EC2, elastic load balancers, these are tools for simple web applications and that's all you really need to build many businesses.”

Daniel worked at AWS for more than eight years before leaving to become an entrepreneur and an author, so he is authoritative on Amazon Web Services and he joins the show to talk about what the good parts of AWS are and his strategy for building applications with that subset of services.

If you are interested in sponsoring Software Engineering Daily, send me an e-mail, jeff@softwareengineeringdaily.com. We're always looking for sponsors. You can also support the show yourself by becoming a subscriber. You can go to softwaredaily.com and click subscribe.

[SPONSOR MESSAGE]

[0:01:34.3] JM: Scaling a SQL cluster has historically been a difficult task. CockroachDB makes scaling your relational database much easier. CockroachDB is a distributed SQL database that makes it simple to build resilient, scalable applications quickly. CockroachDB is PostgreSQL compatible, giving the same familiar SQL interface that database developers have used for years.

Unlike older databases, scaling with CockroachDB is handled within the database itself, so you don’t need to manage shards from your client application. Because the data is distributed, you won’t lose data if a machine or data center goes down. CockroachDB is resilient and adaptable to any environment. You can host it on-prem, you can run it in a hybrid cloud and you can even deploy it across multiple clouds.

Some of the world’s largest banks and massive online retailers and popular gaming platforms and developers from companies of all sizes trust CockroachDB with their most critical data. Sign up for a free 30-day trial and get a free t-shirt at cockroachlabs.com/sedaily.

Thanks to Cockrach Labs for being a sponsor and nice work with CockroachDB.

[INTERVIEW]

[0:02:57.6] JM: Daniel Vassallo. Welcome to the show.

[0:02:59.1] DV: Thank you, Jeff, for having me.

[0:03:00.5] JM: You worked for AWS for more than eight years. You helped build some parts of AWS. Eventually, you left the company and you have now written a couple books, you have started your own company. Let's talk a little bit about AWS. Tell me about your experience working at the most prominent cloud provider.

[0:03:18.1] DV: Yeah. I joined AWS in 2010. Before then, I had worked a couple of years in a small startup. I was what nowadays you'd call a full stack developer. I built web applications, dealt with databases and things like that. I always felt like I knew how to use technology, I knew how to use a database, but I had no clue on for example, how to build a database from scratch and sort of things like that. That's what led me to start looking at working at a place like Amazon.

I was using AWS before then, so I was a fan already. When I joined, I learned a lot. Obviously, I got to see how the sausage gets made from close by. Yeah, and it's obviously, it's technology at large scale, something that I was never exposed to. It would have been highly unlikely that I

would have been exposed to these kinds of things if I weren't working at a company like Amazon.
I spent all my career at Amazon working in CloudWatch. It's the monitoring product. It's actually quite high-level in terms of if you look at it in terms of high-level stack. I wasn't working on the operating systems, or close to the hardware, or things like that. Still in the field that I had experience and still basically building web apps with databases and other things, but I got to see a different perspective of dealing with technology and building technology that especially in the first few years, I got to absorb a lot.

Then I mean, I spent eight years there. Then obviously, the lessons and information start to diminish a little bit like in everywhere else. A long story short, maybe we can talk more about it. I think I realized that probably if I called it as a full-time employee wasn't the ideal one for me. I spent quite a few years thinking about it and that's why last year, I took the plunge to leave. I was living off my savings for a little bit until I figured out something else to do. I think instead of working as a self-employed developer, I think it's a better fit for my preferences.

[0:05:44.0] JM: Since leaving the company, how has your perspective on building software changed? Now that you've started to write your own software, rather than having management around you, how is the process of writing software and thinking through these problems and setting directives for yourself been altered?

[0:06:00.3] DV: Yeah. I think once you have a bit more skin in the game, I think you start to make decisions a bit differently, obviously even things like that were sacrosanct at Amazon with regards to sometimes even testing and test coverage and things like that, their work policies and implicit rules about how to do things, I think you start to be a bit more even more pragmatic, I would say, that you start to evaluate things differently.

Nevertheless, I think something to Amazon's credit, something that I really enjoyed is that when it comes to programming and building technology, lots of decisions get made very locally within the team. It wasn't like the company was giving us directives on you should use this programming language, you should use this type of hardware. Most of the time, it was basically a small team, five to 10 people. I think my teams were never bigger than five people. We used to decide within ourselves what programming languages to choose, what technology to depend on, what servers to use, what testing do we need.

I think there wasn't that much of a huge gap compared to say, a more legit environment where you get told exactly what you need to do. I was already accustomed to having to make decisions on my own, or with a small number of people. I think, yeah, it becomes a bit more extreme. Nowadays, I'm paying for development costs myself. I have a full-time employee who has been helping me for the last 10 months, which I'm paying out of pocket from my savings. It's a different perspective, what you expect out of people and out of things when you're paying literally a 100% out of pocket, versus when a huge trillion dollar companies funding your development.

[0:07:56.4] JM: AWS has many, many services. The book that you wrote is *AWS the Good Parts*, which is suggestive that there is perhaps some extraneous stuff. Maybe it's useful for some developers, but most developers don't really need to know about these other features. All of this different infrastructure that we have available on AWS can lead to a feeling of analysis paralysis. You get into this state of mind where you have this optimization fallacy that you described, this belief that you can get better and better infrastructure by optimizing what you're doing. Can you describe this fallacy of optimization?

[0:08:32.2] DV: I think what happened, to look at it a bit from a more macro level. When I started using AWS more than 10 years ago, before I joined Amazon, I remember going to the AWS website, the attended products spoke to me. I was fascinated that I could download data, but there wasn't even a console before when I started using it, download the TCLI types on instances or whatever it was. I got like, or you could SSH with server within 45 seconds. The documentation spoke to me as a developer.

I think what's changed recently mostly in the last five years is that AWS started targeting a different type of customer, started going after speaking to CTOs of Fortune 500 companies and governments and things like that, which are very different from me. I think what's resulted is that it resulted in these 150 or 170 different products all with their own dozens of options, lots of different things.

I think it's very easy to be daunted by all these seemingly infinite number of choices. If you go look into the advanced documentation, there's no opinionated perspective. I mean, you get presented with this matrix of if you use S3, this matrix of storage classes and they don't know how many they are. There are six with all their different quirks and features. It is hard to decide. It's hard to, because they're presented, like if you want nine nines of durability, you should choose this if you want five nines, you should choose this, but what does it really mean to me?

Again, I think the reality is that these emerged just because there was likely some huge customer that wanted this particular feature to save a penny per gigabyte stored right in them, otherwise was built it. Then that made it available to everyone and the documentation explains it as if it's a viable option for everyone.

At the risk of not sounding very modest, I think I tried to describe some of the very good products that AWS has in a way that I think describes them better than Amazon does, without all the ignoring probably 90% of the options and details that very likely, you wouldn't need almost certainly in the beginning. I think if you end up needing some of these things, you will likely realize on your own right and then you research them and realize that maybe I'm spending a lot of money on storage, maybe I should switch storage class.

Basically, what I try to do in the book is ignore everything that you might have learned about AWS, look at these services as I said, to describe most services as primitive data structures, as an easier way how to reason about them. They have these basic concepts and primitives and just probably start with these default options and I explain why. I think it's a better way of going to your options this way by restricting the number of things you have to choose from. Again, like then maybe. If you want to specialize on something or optimize something specifically, you could always do that later.

[0:11:52.9] JM: Right. This is something you call your default heuristic, the idea that you should stick with what you know. Maybe you don't assume it's exactly the best choice, but it is a choice. In many cases, it's going to do what you need it to do. Your book describes these default services that you can pick for websites and web services. These are the “good parts.” We can get into these good parts.

Let's talk about first, DynamoDB, which obviously we're going to need a database if we're building an application. You're generally by the way, talking about web services, websites and web services. You're not really talking about really building machine learning or data engineering pipelines. Let's start with just DynamoDB, because every application needs some transactional database. Describe the use cases for DynamoDB and the areas that it's not well suited for.

[0:12:41.8] DV: Yeah. I think this is a bit of a controversial opinion. I think Amazon is doing a disservice. By the way, I believe then would be a great product. I really like it; probably my second favorite service up to this today. I think Amazon is doing a disservice by categorizing it as a database and I know this sounds like, what? I think, I see DynamoDB as a very primitive B3 in the cloud. I think if you expect things that you typically get from a DBMS out of DynamoDB, you're going to be highly disappointed.

I believe this is one of the reasons why it's not that popular. People tend to expect even nowadays, even from non-relational databases, Mongo, Cassandra and others, people tend to expect that they can show a database, a query at a black box and get a result back. Maybe it's relational, maybe it's a NoSQL query, maybe there's some restrictions, but that's typically what they imagine. Then the DBMS comes with features, like backups and indices and other things.

I think DynamoDB is more similar to Redis than it is to MySQL. You literally just have this primitive data structure, highly durable, highly available. You don't have to think about running out of disk space, or availability, or things like that, then it does a great job to resolve those issues. Then you just have two very primitive operations to query data out of DynamoDB. A key value get input, or basically a list, which is they called – it's called the query API, which is basically just a list of sorted keys.

In fact, I would say that DynamoDB and S3 are significantly close in terms of technologies. They both actually support a put and get operation, S3 also has a list API, which sends you a sorted list of keys which you can basically mimic the DynamoDB query API. There are some differences on the edges. Obviously, DynamoDB has for example, secondary indices which S3 doesn't have. S3 allows you to put an object as a public, to public access, so then if it doesn't, then it really has point-in-time backups, S3 has versioning, which are somewhat related.

Fundamentally, they're both in my opinion just basic data structures with very primitive operations. They're both highly durable, highly available and highly scalable. I think developers in particular are better off if they think about these technologies like that, rather than this is a MySQL or PostgreSQL replacement.
That's why I think, I believe that if DynamoDB was put into the storage category under S3 with just some – because, I think mostly what the way they defer is pretty much in terms of cost and performance. Behind the scenes, S3 is running on spinning disk, which results into very low cost per gigabytes, DynamoDB on SSD, so very high, very fast, but more expensive. That's what ends up being the two characteristics that differentiates them most.

Apart from some of the features on the edges as we talked about before, like secondary indices are nice and some other things. Yeah, I think that's probably my most controversial point of view about DynamoDB. That is probably best seen as something radically more simpler. I think if you think about it that way, then Amazon's mostly doing a market. It has a marketing problem. It's transitioning DynamoDB as a database. Whenever I see DynamoDB discussed on Hacker News, on Reddit, it's always people complaining that they expect this, but doesn't have this. It's a pity, because it's really an incredible service if you look at it as a much simpler proposition.

[0:16:38.2] JM: Those things that the Hacker News people complain about, it’s mostly around just the ability to query it like a relational database and get some big, specific select statement, right??

[0:16:51.6] DV: The general idea that with nearly every piece of technology that calls itself a database, you tell a query, regardless of how complex it is and you wait some time and this black box that tells you back the answer. DynamoDB doesn't work that way. If you want to do a simple aggregation, I have an order stable and they only want to sum the number of orders by date. DynamoDB doesn't know how to do that. You can get the list of orders, you can pull it down in your application, but then you have to aggregate it yourself. That's one of the most fundamental differences that I think puts people off when they realize, “Oh, I have to do all this work on my side,” which by the way, has performance implications as well. I talk about it in the book.

When you're making most of the queries application side, or some of the queries, you have to consider you might be pulling tons of data, I mean, over the network, closes them locally on your side. There's a performance aspect, not just a convenience aspect. I think that's one of the most challenging issues if you try to put it alongside Mongo and Cassandra and obviously, the relational databases, where they're just this big black box, this big complex thing and just throw something, just works its magic and it turns back the results.
[SPONSOR MESSAGE]

[0:18:21.1] 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 that you can get end-to-end visibility quickly. It integrates seamlessly with AWS, so you can start monitoring EC2, RDS, ECS and all your other AWS services in minutes.

Visualize key metrics, set alerts to identify anomalies and collaborate with your team to troubleshoot and fix issues fast. Try it yourself by starting a free 14-day trial today. Listeners of this podcast will also receive a free Datadog t-shirt. Go to softwareengineeringdaily.com/datadog to get that fuzzy, comfortable t-shirt. That's softwareengineeringdaily.com/datadog.

[INTERVIEW CONTINUED]

[0:19:24.8] JM: Can you just go a little bit deeper on why, architecturally, DynamoDB is not well equipped to fulfill the same semantics of a SQL database?

[0:19:33.9] DV: Oh, it was designed to build this way. I don't know if you know that, but do you remember SimpleDB? It used to be the predecessor of DynamoDB back in, I think it was launched probably in 2009. It was significantly more ambitious than DynamoDB. It was meant to be [inaudible 0:19:53.2] in a query. It was non-relational. It was more like Mongo document based. You even said documents and then you saw a query and it sends you the answer.

This is actually, it is probably one of the few. I can't think of any other service, one of the few duplicated services from AWS. It's technically still supported if you're still running it, if you're still using SimpleDB, the APIs still work, but basically, Amazon just hid it under the carpets that you don't find anywhere. You won't find it in the console. It's not in new regions and things like that.

The problem was that Amazon found it super hard to make this type of database scale and to have predictable performance guarantees. One of the biggest problems that was happening that with SimpleDB, you might throw in some complex query, you might not have an index about it and the query would take two minutes, it would time out randomly, it was completely unpredictable. Some queries take 200 milliseconds, some will take minutes. It was very hard on the service side to reason about queries, like how to allocate the sources, how to reason about capacity.

DynamoDB was the answer to that. It was a radically different perspective. Completely predictable, versus completely unpredictable. There's two operations, get and put. They are literally going to be to a B3 index behind the scenes and updating a single item; very predictable. They all take just a few single-digits, milliseconds to read them scientifically. There's this query API, which again just goes to the starting point of a B3 reads is a sequence of records up to a megabyte. Again, it's like the upper bound predicts how much expensive the query can be as it tends to results and that's it.

If you want to continue to paginate, you go with the pass the next token and you need another 1 megabyte. It became very easy for the service provider to reason about how expensive a query can be, how fast it can be, how to allocate resources. It became hugely successful within Amazon itself, because I remember we used to run services on top of the relational database and we used to have the same problem. Sometimes, the relational database, it's a complex machine. It might start choosing, it might start to use a sub-optimal query and suddenly, your query that used to take a second is now taking 20 seconds and suddenly using all the memory.

When we started thinking in terms of much more primitive technology like DynamoDB, things become easier to reason about it, as long as you managed to model your queries and what you needed to do to its limitations. Then the days where you're fighting your database, because it's suddenly spiking to a 100% CPU and everything is slowing down, disappear, so that element of predictability is highly valuable. Yeah, there were different. It was designed to be this way. This wasn't designed to be throw in a query of arbitrary complexity and will give you the answer.

[0:23:04.4] JM: What do people do when they have built their infrastructure around DynamoDB and it's not fulfilling the requirements that they have?

[0:23:11.8] DV: Yeah. I think you will struggle if the limitations end up surprising you. I think this problem ends up, being discovered very early in development. For example, if you're expecting to be doing lots of aggregations on huge amounts of data, during development you're going to realize that you're going to be downloading everything out of Dynamo and doing it locally.

Hopefully, you'll realize early that this is worth considering that whether you should use another type of database, or the relational database, or something else. I don't have any first-hand experience for example, where the limitations ended up surprising you later, which is I think is a good thing. Again, the fact that it's significantly restrictive helped you – it's very hard to abuse it, or expect more out of it. You realize immediately that these are the limits, which again, I think these tend to be sometimes a trap in more sophisticated databases, because during development your query just returns in 100 milliseconds. Then once you have lots of data or things are running hotter, they start to become more unpredictable. Dynamo just eliminates that issue. Just there's no unpredictability. It's actually incredibly predictable at the cost of the constraints that it comes with.

[0:24:38.1] JM: You write in some detail about S3 and S3 I think of for obvious use cases as slow file system, it’s blob storage, it’s static website hosting, it’s a data lake. Tell me about the other applications of S3.

[0:24:52.8] DV: Yes, yes. I think one of the least appreciated values of S3 is that you can think of S3 as having infinite bandwidth for all practical purposes. If you have a terabyte of data, you could basically in S3, you could download it as fast as you want. You can basically throw as many threads as you want at it, as many servers as you want at it, you can chunk it up in pieces and just download the terabyte in a second.

For example, one of my biggest projects at Amazon was launching and working on CloudWatch logs insight, which is basically a monitoring tool that allows you to run arbitrary complex queries against your log data. This was pretty much entirely built on top of S3 and this is – it surprises

people, because this, unlike DynamoDB, we actually chose to support give me an arbitrary query of any complexity, including regular expressions and things that are super costly to evaluate and will return you the results. We built it literally on top of S3 and in a very cost-effective way, because we relied on the assumption that for example, log data tends to be very big in generalized, especially in our days, like applications tend to generate gigabytes and terabytes of logs. You want to store them somewhere where it's cheap and S3 is a perfect place for that.
You tend to query them infrequently. When there's a problem, or you want to analyze something about your application. I think one of the ideas that works really well with S3 is this technique where you separate compute from the data. Basically, once there's no query running, there's no compute running. Basically, you can just have the data sitting in S3 there. You're just paying the 3 cents a gigabyte per month and there's no other cost. If you open up the console and [inaudible 0:28:33.6] insights and you do a query, will spin up some EC2 instances. Well, I mean, behind the scenes with some pool of warm instances lying around.

Fundamentally, you can think about it as spin up some ephemeral instances and we enough, such that we can download data from S3 as fast as we really want it to. Then you can just churn over the data very quickly. Just because S3 can literally saturate your network if you're able to keep up with it right on every instance that you allocate.

Amazon internally is doing this a lot, this idea of separating compute and storage, even data, even technologies like [inaudible 0:29:17.1], which is Amazon's relational database product. I don't see it too much outside of Amazon. S3 really enables this. Just because you can have just data sitting around with basically a huge pipe, an infinite pipe, I think for you can think of it like that, where whenever you want, you can just start pulling out of it and doing intensive workloads for a short period of time, which even though they might be expensive for the short period of time, the fact that you run them for just a few seconds, or a few minutes, end up being very cost effective in terms of churning to data in particular.

[0:28:13.0] JM: There are some different storage classes in S3. Tell me about how a developer should consider choosing between these different storage classes?

[0:28:21.9] DV: Yeah. Personally, I think I would just go with the default one is the easiest one to reason about. I think all the others are just optimizations for very special cases. I think as we talked about earlier, you're very likely going to, if you realize that you have a huge S3 build, that it's a significant part of your spend, then it might be a good time to start looking at all the other options. Again, these are not – these are the versatile decisions. I think, it's easier to just go with the tried and tested classes, the ones that have passed the test of time, the ones that are easier to reason about their implications, rather than try to prematurely optimize for something.

Again, unless it's immediately obvious, for example that you're going to be storing data and never reading it, just because you're keeping it for regularity reasons, then maybe some of the niche storage classes like glacier and these super low-cost ones might make sense to use immediately. If you're expecting to be doing the [inaudible 0:31:15.5] with some unpredictability, you don't really know exactly how you're going to be reading, or how much, or how frequently, I just default to the default storage class appropriately named.

[0:29:42.3] JM: You also write about EC2 and everybody knows EC2, it's the classic server option. There are compute options other than EC2. You could spin up a Fargate container, or you could try to build your application around lambda functions. Is EC2 still the same default of server infrastructure?

[0:30:01.2] DV: I think for most workloads, yes. My problem with Fargate and ECS and the related services is that I still see them as somewhat in match your offerings. This started as Amazon mostly acting due to the changes in the industry, where Docker and containers that become popular. In my opinion, just do something goes very quickly to decide to have a solution and the story for that demand.

I have to admit first of all, I've never used personally first-hand experience with Fargate and ECS, but from my secondhand experience talking with people, others that use it, I think you run into the risk of running into too many limitations and restrictions for basic things, like how we do deployments and how we do things, just because it's something very new, hasn't really passed the test of time. Unlike EC2, which was built very differently, Amazon built it for itself. I mean, internally it had a similar solution. The more recently, Amazon has been starting using EC2 internally very aggressively. It's used by millions of customers or service company, huge customers. It definitely last the test of maturity, in my opinion.

I think what I like about EC2 in particular is that yes, just its freedom is just, I can do whatever I want with it without being constrained by some other concepts and restrictions that the platform has imposed on it. Lambda is different, I think. I think it's even more concerning to default to building your application around lambda in my opinion, but I still believe that lambda is great for if you have a piece of code and you want to run it in the cloud, I see lambda is just a code running in the cloud. It's not a computer in the cloud.

I'm very optimistic about a future where the operating system and the server get abstracted away. I don't think lambda is there yet for general purpose applications. Again, I think if you try to build something on top of lambda, you're going to have to grow to lots of gymnastics to break up your application and to functions. Then I think you will very likely end up being surprised.

Unlike what we talked about DynamoDB earlier, very likely end up being surprised later in the development process. Whether you realize you might want to install a log agent on your host, but you constantly do it, or it's super hard to do it, or you want to use WebSockets and even though lambda the supports them, they are not stateful, so you're going to have to touch DynamoDB and it's going to cost you an insane amount of money.

There's lots of traps and pitfalls and problems, I think in my opinion, if you try to do something very sophisticated on top of lambda. Lots of people disagree with this position and I'll admit. That's why I like to think in terms of just defaults like this. Just if I want to host something in the cloud, I would just default to use EC2. If I have Adobe on this application running on my laptop, I can get it running on EC2 very easily. It's pretty much the same environment.

If you try to get it to learn on lambda, it's almost impossible. You just have to break it up into an insane way. Fargate and ECS have been different obviously. If you have something already running on Docker, or Kubernetes, or something else, I think they're worth a look. I would still advice caution with regards to limitations and hidden issues that might not have been exposed, just because I believe they're not that widely used, even I'm sure that our customers obviously, but it's not in a different category than EC2, where almost any issue has been discovered and fixed and addressed.

[0:34:20.9] JM: As far as security, what do I need to know about securing EC2?

[0:34:27.4] DV: There are two concepts, which I think are really important to understand the concept of security group and the concept of VPC rules, which I think you can think about them as the security group is like a firewall for your instance, little tied to what gets in and out on the machine. The VPC is around the network. You have a group of instances. You have another firewall on the network boundary.

I think from a network security perspective, at least you can start with just everything logged down and then you just go to the usual process of just opening up what you really want, maybe just a port to the load balancer and then the load balancer faces the Internet and things like that. With regards to keeping the instance and the operating system patched and secure, there's typically one of the concerns people tend to have. I think it's also, it's not as problematic as many people think. I mean, it's definitely a burden and something under your responsibility, but it's literally just a matter of and something you could automate, a method of just doing young installs, security or whatever it is on your operating system and rebooting your instance periodically, which I don't know if it's a secret, but this is pretty much what Amazon does behind the scenes for all the other services. I just know nothing magical.

I think there's a new feature. I haven't played with it yet. I think it's a new feature in auto-scaling, I believe, or maybe it's native built-in in EC2, that allows you nowadays to just reboot your instance on a periodic basis. I think if you're running a fleet of instances this might be just a very reasonable way of just keeping your operating systems patched by just doing a rolling operating system update and restart on a weekly basis and with some basic monitoring to make sure that everything comes up back as you expect. I think it much cover you in terms of security.

[0:36:45.0] JM: With EC2, there are all of these varieties of tiers. You have CPU and memory and storage and network options at different tiers, pricing tiers. My guess is that it’s going to be more of a discussion than the S3 storage classes. There’s more to discuss around this subject. Can you tell me about suggestions for getting started with picking EC2 tiers?

[0:37:05.9] DV: To be honest, I think again, it's probably even easier to move to a different instance type than to move from an S3 storage class side, because I mean, if you have a petabyte of data stored in S3 in one class and you want to move the class, you might have to pull it all down and upload it. Again, it will take time and probably be expensive.

With EC2, it's just a method of – most of the time, running your cloud formation templates in a couple of minutes. You might change your instance type. I think it's an even more easily reversible decision. I think again and even when I worked with Amazon, typically we didn't use to overthink the instance type a lot, obviously. If you're doing something very specific like relying on instant storage, or the disks attached to the instances. There are obviously some instance types that have those available, or some that have bigger this and things like that.
I think for most use cases where you're just learning web applications, where the storage is either necessary or in a database, I would just default to the M and C categories, which is the compute optimize the memory optimize, which are probably the most general-purpose ones. They have slightly different ratios of CPU and memory. They're both roughly the same cost.

Yeah, I just wouldn't over-think it. I would err on if I'm doing it, if I have a distributed application, I would err on – or I would be inclined to choose multiple smaller instances than the bigger instances. I think you just get somewhat better availability and you just – it's easier to deal with instances that way. Yeah, I think instance types are very easy to change. Unless it's immediately obvious what you should be choosing in the beginning, just choose what seems good enough and start with that.

[SPONSOR MESSAGE]

[0:39:07.8] JM: You probably do not enjoy searching for a job. Engineers don't like sacrificing their time to do phone screens and we don't like doing whiteboard problems and working on tedious take-home projects. Everyone knows the software hiring process is not perfect, but what's the alternative? Triplebyte is the alternative. Triplebyte is a platform for finding a great software job faster.

Triplebyte works with 400-plus tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves the hiring process by saving you time and fast-tracking you to final interviews. At triplebyte.com/sedaily, you can start your process by taking a quiz. After the quiz, you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple on-site interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte, because you used the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more, since those multiple on-site interviews would put you in a great position to potentially get multiple offers. Then you could figure out what your salary actually should be.

Triplebyte does not look at candidates' backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. I'm a huge fan of that aspect of their model. This means that they work with lots of people from non-traditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There's very little risk and you might find yourself in a great position getting multiple on-site interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out. Thank you to Triplebyte.

[INTERVIEW CONTINUED]

[0:41:24.9] JM: Revisiting lambda. You touched on this a little bit earlier, but I'd like to know more about what your perspective is on how to use lambda and maybe what more broadly, what a fully serverless application would look like today. I've talked to some people who are pretty excited about building fully serverless applications. I talked to a company called Courier recently. It's a fairly new application for doing optimization of sending text messages and private messages to customers, e-mail messages to customers. It's a very infrastructure-heavy product, but they built it entirely out of AWS lambda functions and triggers and they're all in on AWS, to steal the AWS marketing jargon. They're really happy with AWS. I'd like you to tell me more about the fully serverless application stack and how that contrasts with the default heuristic, that simple default heuristic you've been advocating.

[0:42:16.9] DV: In my experience, what ends up happening is that the benefits that you thought you were getting by having the operating system abstracted away and the security updates and the servers, the limitations and the other issues end up out buying those benefits, to put it in those terms.

What I've seen happening and I think it depends on how much and probably the main reason why people say they love serverless is probably they find dealing with services operating systems super daunting. It's like, they're nearly – can't imagine themselves having to configure security group, or configure the VPC, or update the operating system and things like that. I've seen discussions and conversations where sometimes people don't even consider that an option, just because I'm not a DevOps person, or I'm not a systems engineer. I can't even think about dealing with it.
Then they tend to gravitate towards lambda, because it promises to eliminate those concerns. I think what you end up fighting with end up being things like limitations, like the fact that the function can only land for 15 minutes if you're doing anything stateful, it's a challenge. If you want to install something, you want monitoring software, you want to install Datadog or something. It's just how do I do it?

I mean, there are ways of doing it, but it's complicated. You have to install a Kinesis stream and create cloud watch logs and pipe it to Kinesis and then pull out and send it to somewhere else. That's just from ND infrastructure side. You end up with lots of little pieces. Not to mention the software side as well, which I think leads to this extreme microservices, micro-function level breakdown, which in my opinion hurts our software architecture significantly.

I'm still a bit old-school in terms of preferring the Ruby on Rails-like web applications, the monolith, the thing that has a single binary, or single entry point. I can land on my laptop with a single command and a 100% of the functionality runs from a single place. I love this concept. I love that I can open the debugger and walk to a hold request without having to jump to different processes, or that I have to start – that I don't have to start five different [inaudible 0:44:50.9] for my application to be doing everything.

I think if you try to build something on top of lambda, you're going to be forced to break things up into something that is even more extreme than the typical microservices, where you might just be breaking it up by functionality, or by the team that owns it or something like that. You end up breaking it up by every different request type, or every different tiny bit of activity, which from the examples that I've seen, they've all – in my opinion, I never really looked at the code architecture, or the infrastructure and I envied it that I said I wish my software was like that, or my infrastructure was like that.

Definitely there's a lot of preference here, but I just prefer the traditional way and the way it's been done for a long time, where my application that's when a single binary, or if I'm a single place, if I want to distribute it, I just run multiple versions on it and put a load balancer in front of it. It's super easy to reason about it, even deployments.

One of the other problems with breaking things up is it sometimes complicates deployments as well, unless you create some very strict protocols, you might have a function running version 5 and another function running version 4 and they're touching states that is in common, which there are frameworks that facilitate all these things to make sure you're doing the right things. I think you end up still fighting all the time. I think one funny thing I was just thinking about it recently, if you go to Reddit or AWS, I think probably 50% of the questions are how do I do this basic thing on lambda?

I think it's an indicator of how confusing it is for people to just learn very basic web applications and websites, things on a platform like lambda. Again, like talking how do I communicate with them with IDS? How do I keep the connection post open and not run out of connections? Basic things. How do I keep a cache, or how do I send things to a third-party service without having to send them on FD and lambda invocation?

Then apart from that, there are also limits of the platform, which would likely get fixed in the future, but the fact that you are cause bundle conducts it, I don't remember what the size is, but conducts it some certain number of megabyte or gigabytes, which are less problematic. Even those things end up being another headache that you end up having to deal with.

For example, I think a use case where lambda works well and we try to – when I was at Amazon, we tried to use it this way was for example, to learn integration tests during the deployment pipeline and just before in the CD pipeline. We were quite happy with it. They start on demand. They run a bunch of tests against a piece of software that we had.

Yeah. Then suddenly, our test because it was written in Java and Java brings in the whole, I don't know, gigabytes and gigabytes of dependencies. One fine day, we just couldn't deploy the tests anymore, just because we exceeded this arbitrary small, I think it was 250 megabytes or something like that, limit of the bundle size. Suddenly, we're stuck for a few weeks until we figured out a way how to remove some dependencies and reduce the bundle size. It's just a significant cost. When you think about that, we could have been running this on EC2 and not have to worry about things like that.

[0:48:40.2] JM: ELBs are the load balancing infrastructure on AWS, elastic load balancers. Tell me about aspects of load balancing on AWS that I might not be aware of.
[0:48:50.8] DV: Yeah. I think probably one of the things that many people don't know, I mean, there are two – there are two modern ELB types. There's a classic one, which even Amazon doesn't recommend using anymore. There's the application load balance and that's where log balancer. I think the way that they're documented on the Amazon products pages makes it look like the network load balancer is something very niche and you only use it, only needed if you want, extreme high performance or something large scale and things like that.

Actually, I think I would likely default to the network load balancer in general. Probably, one of the reasons is that the way they work is that an application load balancer, you can think of it as behind the scenes, there's an instance somewhere that's doing load balancing work for you. There's a monitor that's tracking your number of the requests coming to, and as your requests fluctuate and increase, AWS will put more load balancer instances, would basically also scale up or down. This sometimes tends to create problems at scale, or when there's a big change in demand just because the auto-scaling might not adjust quickly enough.

In fact, almost all the commands that if you know that you're going to have a big spike of traffic to inform them beforehand, cut a ticket and they will warm it up for you. That's the term that they use. Basically, is that they set a minimum number of instances. The problem is that these limits are not documented. Nobody knows what whether it's a 100 requests a second or it's a 1,000 or if it's a million. There's always this thing in your head like, is my thing going to get popular when I'm launching? Suddenly, everything is going to start turning a 502 error, just because load balancer hasn't scaled.

This is only a problem with the application load balancers. The network load balancers don't work this way. They're basically just a big multi-tenant router. I think that's the way that I like to think about it like that. Pretty much FDA, the best customer is using the same thing and there's no single capacity dedicated to you. The only way you could exhaust the NLB, if you like, use all of Amazon's capacity, which would be a hard thing to do.

Despite the NLBs have fewer features, like NLBs work on a lower level at the network packet level, so they don't really have fancy features, like request level loading on the URL part and things like that. Or for some along the NLBs, you can even attach lambda functions to them and serve some parts from a lambda function and some very interesting features. NLBs are super primitive. They just route requests and load balance them from the Internet to your instances. They support TLS termination on their site, which is super fascinating that they managed to do it at the level 4 level. It's quite a interesting piece of magic, but it works really well.

If all you need is just to route HTTPS requests from the Internet to your instances, I would recommend defaulting to NLB, unless you really need some of the unique features of the application load balancers, just because it eliminates that burden of scaling and capacity from your responsibility.

[0:52:17.8] JM: Let's talk a little bit about infrastructure as code. Infrastructure as code in AWS world is generally talking about cloud formation. I'd love to get your perspective on cloud formation relative to other tools, such as terraform. What is the use case for cloud formation? How does it contrast with terraform?

[0:52:35.0] DV: Yeah. In my opinion, they're both relatively equivalent. I think I would choose again, which one you would be more comfortable with. I tend to default on cloud formation mostly, because I've used confirmation a lot. It's the level that I know. I know its quirks and its issues. I used terraform a little bit. I know it less. If I were to choose something, if I were to do something new, I would like to choose cloud formation, just because of the familiarity.

From my perspective, the differences are not that significant that warrants a fixed recommendation for one or the other. I would definitely encourage using either one of the other and wouldn't recommend in production, unless you're doing a demo or a toy project or whatever, to do anything without infrastructureless code. Yeah, I don't really have strong opinions on the two different pieces of software.

[0:53:33.5] JM: Okay. Well, just getting a little more context on what you're doing now, so you've left AWS. You wrote *AWS the Good Parts* and you also run a company, which is called Userbase. Explain what your company, Userbase, does.

[0:53:47.1] DV: Yeah. Userbase is a very – it's a service that allows web developers to add user logins and data persistence without having to deal with any back-end code, any lambda functions, any compute, whatsoever. Basically just a very basic JavaScript SDK, can it be run in the browser that has basically create user, log in, log out, insert item, delete item in a database and some basic database crud operations. It's similar to firebase from Google and amplify a little bit from AWS.

One of the key differences that Userbase has is all user data is end-to-end encrypted. Basically, when the end-user creates an account, a key gets created. That's never gets sent to the service. Well actually, the key gets encrypted with user's password and gets backed up on the service, like with the combination of the password and the key never gets – the server never receives that.

If this opens up, some interesting use cases were either for privacy-focused applications, where the website owner doesn't even want to see the user data. The user data becomes a liability and does general security benefits as well. That's significantly harder to misuse, or misplace, or leak out user data if it's all encrypted with keys that never leave the service. It's focused, yeah, it's focused on simplicity and ease of use with this unique feature that none of the other vendors have.

It's still very new and that I mean, it was a – I started working on it approximately almost a year ago, like 11 months ago and launched the first version about three months ago. We're still building features on it, so we still don't have file storage, for example, just database storage or data sharing between users is something that we're about to release.

We just released an interesting feature, where you can connect your Userbase account with Stripe if you want to build a page web app. We tied in the payment feature with the authentication part, so you could configure your app, such that it allows your users to log into your app, for example for seven days for free. After seven days, they will be prompted to pay for access. If they paid money, the subscription money goes to your Stripe account and Userbase facilitates all of that again, without writing any webhook handlers and encode on the back-end. You literally just connect your accounts with Stripe, just a couple of clicks from the console. You define your payment plan and that's it.

I like the space. It's something that I like building products for other developers. It's something that I really enjoy doing at Amazon and I still enjoy it. It's a space that I like. If I chose this, I choose to focus on this product first, just because I think it matches my skills, like as we said, it matches that I like building tools for developers. I also have some ideas myself on building web apps eventually that might benefit from Userbase itself. It's part of the strategy that I could be my own customer as well. Yeah, that’s one part.

I’ve also been very active – pretty much when I left Amazon, I was pretty much unknown to the world. I had never, which was quite shocking when I realized that if I did something and I released as public and nobody would hear about it. I intentionally had started trying to build an audience. I settled on Twitter. I’ve mostly been documenting my journey all these decisions, like word. For example, user base pretty much I’ve been stored it in public on Twitter. I tweeted about it. It evolved a little bit from feedback from people that's responded.

As you mentioned in the meantime, mostly as an experiment, I did a couple of information products, we call them as – one of them is the AWS book, which I released about four and a half months ago at the end of 2019. Again, it's something that I had never done before. Either thought it might be interesting to see if it was viable to create something, like basically to sell a

PDF of some knowledge that I had. It's how I saw it, like a brain dump of some things that I knew. They might be interested.

I had no clue whether this was viable from a financial perspective, whether this would sell $100 or a $1,000, or more. It turned out to be – exceeded my expectations. Database books sold, I think it exceeded $90,000 in four and a half months, which basically I've marketed pretty much exclusively all on Twitter organically. Just recently, I started doing some paid ads on Reddit, but only about 3% or 4% of my sales.

Yeah. I mean, I've taken the self-employment parts. I didn't go with my company Silicon Valley style, where I had this grand idea and I decided to make it work at all costs. It was started pretty much the other way around, where I sat down here, I tried to think what could I – what could I do, what could I sell, what could I maintain? I tried to work backwards from that. What do I enjoy doing, because obviously, it's not some business that I intend to just build and then exit out of. It's something that I hope to keep doing.

The information products like the book and I released recently another course on building a Twitter audience, like focus for on developers and people like me. These were just experiments to just supplement my income. I mean, Userbase is still very early. I mean, we have 68 paid customers. It's just not significant enough to pay my bills. It's doing a few thousand dollars in revenue.

I'm willing to give it time. I mean, I saved enough money from my work at Amazon to afford to give it five years to become this level business and that's my target that I could live off it eventually. I understand that it will likely take time, because it's just the timing of the business. People need to build a web app, need to be ready, just needs to fit the expectations.

These info products are just supplementing my income. I'm super happy that they're working out and I enjoy doing them. I've done two so far. I might do something else later this year. I was also doing a bit of freelancing a little bit on the side with a friend of mine, so I was mixing those three things, as business info products and saving time for money. I stopped freelancing now now that the info products are doing well. That's been a little bit more journey. I'm just trying to figure out how to make a living on my own.

[1:01:08.2] JM: Awesome. Well Daniel, thank you for coming on the show. It's been really great talking to you and I enjoyed reading your book.

[1:01:14.5] DV: Thank you very much, Jeff. Thank you.

[END OF INTERVIEW]

[1:01:25.6] JM: Today's show is sponsored by strongDM. Managing your remote team as they work from home can be difficult. You might be managing a gazillion SSH keys and database passwords and Kubernetes certs. Meet strongDM. Manage and audit access to servers, databases and Kubernetes clusters no matter where your employees are.

With strongDM, you can easily extend your identity provider to manager infrastructure access. Automate onboarding, off-boarding and moving people within roles; these are annoying problems. You can temporary access that automatically expires to your on-call teams. Admins get full audit ability into anything anyone does when they connect, what queries they run, what commands are typed. It's full visibility into everything; for SSH and RDP and Kubernetes. That means video replays.

For databases, it's a single, unified query log across all database management systems. strongDM is used by companies like Hearst, Peloton, Betterment, Greenhouse and SoFi to manage access. It's more control and less hassle. strongDM allows you to manage and audit remote access to infrastructure.

Start your free 14-day trial today at strongdm.com/sedailly. That's strongdm.com/sedaily to start your free 14-day trial.

[END]