EPISODE 808

[INTRODUCTION]

[00:00:00] JM: A software application requires compute and storage. Both compute and storage have been abstracted into cloud tools that can be used by developers to build highly available distributed systems. In our previous episode, we explored the compute side. In today's episode, we discuss storage.

Application developers store data in a variety of abstractions. In-memory caches allow for fast lookups. Relational databases allow for efficient retrieval of well-structured tables. NoSQL databases allow for retrieval of documents that may have a less defined schema. File storage systems allow the access the pattern of nested file systems, like on your laptop. Distributed object storage systems allow for highly durable storage of any data type.

Amazon S3 is a distributed object storage system with a wide spectrum of use cases. S3 is used for media file storage, archiving of log files and data lake applications. S3 functionality has increased over the years developing different tiers of data retrieval latency and cost structure. AWS S3 Glacier allows for long-term storage of data at a large cost reduction in exchange for increased latency of data access.

Kevin Miller is the general manager of Amazon Glacier at Amazon Web Services. He joins the show to talk about the history of storage, the different options for storage in the cloud and the design of S3 Glacier.

This week is the last week for the FindCollabs hackathon. FindCollabs is the company that I'm building, and our hackathon has $5,000 in prizes that are being awarded. You can find out about it by going to findcollabs.com/hackathon. You can enter with any kind of project. FindCollabs is a place to find collaborators and build projects. I started it from my own need to find collaborators for software projects, music projects, art projects, all kinds of things. So if you're interested in building stuff, check out findcollabs.com.

[SPONSOR MESSAGE]

[00:02:26] JM: Deploying to the cloud should be simple. You shouldn't feel locked-in and your cloud provider should offer you customer support 24 hours a day, seven days a week, because might be up in the middle of the night trying to figure out why your application is having errors, and our cloud provider's support team should be there to help you.

Linode is a simple, efficient cloud provider with excellent customer support, and today you can get $20 in free credit by going to linode.com/sedaily and signing up with code SEDaily 2019. Linode has been offering hosting for 16 years, and the roots of the company are in its name. Linode gives you Linux nodes at an affordable price with security, high-availability and customer service.

You can get $20 in free credit by going to linode.com/sedaily, signing up with code SEDaily 2019, and get your application deployed to Linode. Linode makes it easy to deploy and scale those applications with high uptime. You've got features like backups and node balancers to give you additional tooling when you need it. Of course, you can get free credits of $20 by going to linode.com/sedaily and entering code SEDaily 2019.

Thanks for supporting Software Engineering Daily, and thanks to Linode.

[INTERVIEW CONTINUED]

[00:03:55] JM: Kevin Miller, welcome to Software Engineering Daily.

[00:03:57] KM: Thanks for having me.

[00:03:59] JM: In computer systems, we learn about the memory hierarchy, and this illustrates the storage tradeoffs of a single node operating system, or in the grander sense, the distributed system. Before we get to talking about Glacier and other storage solutions within AWS, I want to talk about just the abstract idea of the memory hierarchy. Give us a review of the memory hierarchy.

[00:04:28] KM: Well, sure. I mean, I think the general idea of the memory hierarchy is that you have the cost – In general, there's memory that is lower latency, faster to access tends to be more expensive, and thus generally provisioned in smaller quantities than memory that is slower or higher latency, slower to access, where the cost will generally be lower.

If you think about the memory that's sitting [inaudible 00:04:54] with your CPU, that's going to be faster than the memory that would be sitting, for example, in a router. Routers often have very fast memory very close to the forwarding components, because they're doing constant hits against that memory. You have general purpose memory on a computer, solid state kind of below that and then spinning disk, and below that you'd have tape. So sort of the – At each of those points, the costs per unit tend to go down and the access times go up.

[00:05:25] JM: Why does a software engineer need to care about any of these?

[00:05:27] KM: Well, I think particularly these days where software engineers are being asked to build very large distributed systems that are processing X-bytes of data, frankly, on a daily basis, it's incredibly important to understand the locality of your data. Where is your data currently and which data do you need to have to process and what can you do to essentially hide some of that latency? If you know that over the next 24 hours I want to process an X-byte of data, you don't want to just say, "Okay. Give me all that data right now." You won't be able to pay for an X-byte of main memory to store all that data. So by definition, you're going to be bringing some of that data into main memory. Other parts of that data will be staged coming in off of other storage medium.

[00:06:15] JM: Do you have any examples from your earlier days as a programmer maybe in college, or just getting started when you first dealt with the memory hierarchy intimately?

[00:06:27] KM: It's a good question. I don't know that I can think about a specific example. I can think of plenty of situations where I looked at a program and I said, "Why is this running so slowly?" and you realized that because of the way you're accessing memory, and it can be as simple as – Going back to college, one of my favorite classes in college actually was a class where we were asked to essentially reverse engineer the cache lines on a CPU and figure out exactly what it was doing with the memory.

You could clearly see there that if you accessed memory in one order, it would be very efficient. You'd have, say, a millisecond of latency total for your whole program, versus if you just changed the order of a couple operations, you would be sitting there for 10 milliseconds while the CPU constantly was turning while it was in its L1 or L2 cache.

[00:07:18] JM: So you've mentioned cost tradeoffs. I think there's also durability tradeoffs in different storage mediums. Can you talk about the different durability settings that different types of storage can have?

[00:07:31] KM: Well, sure. When you think about the fastest memories that we use today, they tend to be volatile memory. I mean, there's clearly a differentiation between volatile and non-volatile memory, right? Your solid state drives, hard drives, tapes, those are all non-volatile. Whereas, CPU memory, and memory, main memory on a computer is all considered volatile. So you take the power away, it goes away. Whatever you have there goes away.

So, obviously, customers and programmers, developers, have to think carefully about what data they need to be durable and not. Certainly, as we think about cloud computing, I think actually the options for durability have gone way up actually than what used to be available. It used to be that your durability was sort of governed by the fact that you would probably be operating in one data center and you may, for non-volatile, you may store one company on a hard drive or maybe like a RAID configuration where you have essentially one extra copy. If one hard drive fails, you can still restore the data if you can get another hard drive and install it quick enough to repair it. That would kind of a traditional way of thinking about non-volatile memory.

But enter cloud computing, where going back to even the earliest days of Amazon, as the Amazon was growing up in the late 90s and early 2000s, we recognized that for the durability we wanted, we really needed to have three data centers in a close proximity in a region within a few milliseconds of each other. So that's what built, and that was been a fundamental premise for AWS since S3 launched out just about 13 years ago actually as a Pi Day in 2006. We've built all of our regions on 3AZ availability zones, because we know that, among other things, the durability benefits of having 3AZs is pretty important in terms of knowing that the data is stored and you can read it back when you need to get it back.

[00:09:31] JM: So we've talked so far a little bit about the single node memory hierarchy that we might have. In fact, a little bit about the distributed memory hierarchy we might have in the cloud computing world. Go a little bit deeper on the options for cloud computing storage and how cloud computing can explore a wider set of dimensions of tradeoffs.

[00:09:56] KM: Yeah, it's a great question. So within AWS and obviously the cloud provider, I'm familiar with that. Within AWS, customers can provision storage in a number of ways. You can provision box storage, which looks just like a hard drive that you might attach to a computer.

So you have block storage, which is very similar to what you might attach a hard drive to a computer. You also have a number of file storage services, including our Elastic File store, and that allows for more file level access, kind of like mounting an NFS share or mounting a Windows share. Then we have our object storage service, which, again, object storage was actually one of the first services we launched in AWS, or actually the first one in production. We have always viewed S3, our object storage, as a fundamental building block to cloud computing. Because, again, going back to the earliest days of Amazon, we realized that a lot of what we needed out of storage was an ability to have a blob of data parked in an object and to be able to access it via APIs, read and write, and manage access control on it and all the rest.

We really wanted it to be something that would scale much more than you might expect from a block storage or a file storage in terms of we wanted to have many, many concurrent accesses to it. We didn't want to have to force a semantic where you would mount a file system. So that's what we built S3 for.

I think cloud computing certainly has expanded the options in storage, and each of those options comes with our stated durability expectations for it. So with S3, with our S3 standard and S3 infrequent access and S3 Glacier, and now S3 Glacier Deep Archive, all of those storage classes store data in a minimum of three availability zones. So we say that it's 11 9's of durability, 99.9999, lots of nine, because we store the data across three data centers. So we're really tolerant. Those services are incredibly tolerant to even issues that might impact an entire facility.

[00:12:01] JM: I'd like to throw out an example that can make the different storage classes and different storage products that we're going to explore a little bit more concrete. So let's say we're building a photo sharing app. So program who's building a photo sharing app, they might use a database, like a document database or relational database to store information about the users and perhaps the photos that are being posted by the user, and then they might use a different storage backing system to actually have the bites for the photos themselves. They might use a bucket storage system for the photos and then have a URL for the bucket address in that database.

Why does that make sense? Why do we need multiple storage types for our application?

[00:12:51] KM: Well, I think the simplest thing is it comes down to cost and scale, cost and scale really, which is to say that, "That programmer and that instance wants to build an application that can scale to store petabytes of images." We have many customers who store petabytes of images or movies. So they don't want to have to provision that storage and they also want to pay as low as they possibly can for that storage knowing that in a typical photo sharing application, let's say 5% of the content is really active and 95%, for some apps, that means it was posted 12 hours ago, and now it's hardly ever going to be accessed again.

So they're looking for not just that it's inexpensive to store when it's active, but also that they can save money overtime. They're going to optimize their cost overtime. That's really what S3 provides. In fact, in another storage class we launched just at Reinvent last year, is what we call Intelligent Tiering. Intelligent Tiering, the idea there is you just store data in the intelligent tiering class. We charge a little bit just to monitor every object that you put in. What we're doing is we're monitoring to see if it's accesses, and if it's not accessed for 30 days, we move it to a lower cost tier, what we call infrequent access, where you save money on the storage. So we charge you less for the storage automatically if it's not accessed.

But then let's say that those photos were called but then for some reason someone comes back and says, "I want to pull all those photos back or I want to use something with them." We will then automatically move it back into a frequent access tier, where you may a little bit more for the storage, but you don't have to pay per access at that point. So we automatically move storage between these frequent and infrequent access tiers based on what we actually see is being done with that storage.

[00:14:35] JM: What you're describing is the advantages of a cloud provider that has matured and has built primitives in the past that they can now leverage to offer finer-grained access patterns, finer-grained products, and you can also compose these different products together. So the idea of composing together a managed relational database with a managed bucket storage system provides a lot of options for people who are building stuff.

What are some other storage patterns that we could consider? Because one that I've personally used a lot is the relational database, or object database+ bucket storage system. What are some other kind of synergistic patterns between different storage classes?

[00:15:23] KM: We find actually a lot of customers, when they first get started with AWS or first get started with the cloud, what they're trying to do is really move – Make as few changes as possible and just migrate an application they run maybe in an on-premises data center. They just want to move it as easily as they can to the cloud and then give themselves the opportunity overtime to iterate and replace components of that with something that's more cloud native.

For example, when we looked at our Elastic File service, which provides an NFS compatible mount, or now we have FSX, which is our managed Windows file storage. Actually, it is a Windows file server that we just fully manage. So you can imagine on a customer, in most on-premises data centers, they're using either sort of SAN block storage, or a NAS, NFS, or Windows compatible file storage. So using something like EFS or FSX gives them a really quick way to just move that application into AWS. Then overtime, for example, there are earlier application where we then built a photo sharing capability on top of it. That would be a perfect use case to that add object storage on top of it. Because [inaudible 00:16:32] combine these things however they like. We don't limit how they can combine the different storage options.

Another one that I'll throw out I'll mention is you mentioned a relational database, and so a lot of our relational database is run on top of our block storage, but then we also have our DynamoDB database service, and that has its own storage engine as well that it runs on. DynamoDB, like

S3, was really designed for incredible scale, millions of TPS, potentially, through something like DynamoDB.

So, again, we see a lot of customers come to us and say, “I have – I’m using NFS. I’m using a MySQL database. I just want to move it,” and we say, “Great. That’s the pattern. Relational database, use our RDS service, use Elastic File service.” But then overtime, we really encourage looking at things like DynamoDB and S3, where they have even less burden of having to manage and provision storage, and typically the costs are lower as well when they do that.

[SPONSOR MESSAGE]

[00:17:41] JM: On Software Engineering Daily, we’ve had several shows about the future of technology education. We believe that boot camps are an efficient, cost-effective way to become trained for the tech industry whether you want to be a programmer, a data scientist, or a designer.

Flatiron School can teach you the skills you need to build a career that you will love. Flatiron School has immersive programming courses on JavaScript and Ruby, everything you need to become a full stack developer. If you’re interested in becoming a data scientist, Flatiron School has courses on Python, SQL and machine learning.

You can learn in-person or online and you can find everything you need to get started by going to flatironschool.com/sedaily. Flatiron School has options to save money on the program, such as gender diversity scholarships and income share agreements. Flatiron School also helps the students who graduate find a job. Every graduate is paired with a dedicated career coach so that they can find a job or their money back.

The complete details are at flatironschool.com/terms. Flatiron School is a cost-effective way to start working in a tech industry. Learn more at flatironschool.com/sedaily.

[INTERVIEW CONTINUED]

[00:19:11] JM: Because you are working at Amazon and you're an engineer, you have seen not just the patterns that people can build with these different services, but you also understand what is required to construct one of these services. What are some of the computer science problems that the cloud storage systems that you've seen are solving underneath the API surface that the developer, the external developer, is working with?

[00:19:43] KM: I think that one of the biggest problems that is solved by using cloud services, I mentioned it earlier, but one of the biggest problems it's solved is the ability to have three data centers and the attendant increase in durability and availability that you get by a service that automatically replicates data.

With S3, when you put an object into S3, before our web server even comes back and says, "200. Okay, that data is stored." Before that even goes back, it's already been replicated into three data centers. So even if at that very moment there was a failure in one data center, there's copies in two other facilities. So your data remains durable. Same thing with availability, where one data center fails, the system automatically shifts the frontends so that you're not impacted, or the impact is very minimal while that reconvergence happens. So those are incredibly hard computer science problems, and frankly just not solvable if you're only operating in one facility, and it becomes a pretty significant cost to go to additional facilities.

The other thing with our storage services is to achieve the low cost points that we do requires real innovation and application of fundamental CS concepts to how do we distribute that data in a very cost-effective way while maintaining the availability and durability that's required. Those were some fun problems to solve. Again, using a service like S3 or Glacier, those problems are solved for you.

[00:21:16] JM: Explain what Glacier is.

[00:21:20] KM: Yeah. Glacier is our archival storage service and it's designed for asynchronous access. So today we launched a flavor of Glacier, S3 Glacier Deep Archive. With Glacier Deep Archive, we really view it as a replacement for customers who run tape libraries as one use case. There's many use cases for it, but one in particular is if you run a tape library on premises, the cost point that we have with Glacier Deep Archive from everything we've looked at is it's less expensive than running a tape library on-premises and dealing with all of the hassle of running a tape library.

So just like a tape library though, it's asynchronous access. So when you need to access an object, if you're using S3 standard for example, it's milliseconds to get your object back. When you get an object In Glacier, we have three different retrieval speeds with expedited providing one to five minutes to get an object back, but then you can also save money and access it with our standard retrieval speed. That's 3 to 5 hours, and bulk is 5 to 12 hours.

[00:22:21] JM: So it's worth noting that what you get by increasing the latency at which you're going to store your data is lower cost.

[00:22:33] KM: Dramatically lower cost, yes.

[00:22:35] JM: Dramatically lower cost. That can be achieved by, like you said, tape is one. Tape is just very cheap per bit that you're storing data. Overtime, we might have even things like DNA. Maybe we can store – I've done some shows about storing data in DNA. Not very efficient quite yet, but the promise is that would take up a significantly less space than tape if you could store a lot of data in a molecule of DNA.

[00:23:05] KM: Sure. DNA is incredibly dense, yeah.

[00:23:07] JM: It is.

[00:23:07] KM: Storage, yeah.

[00:23:08] JM: It is. So what are the different physical mediums that are at the cutting edge of low cost storage?

[00:23:17] KM: Well, there's a lot of research in a lot of areas. You mentioned DNA. Certainly, there's a lot of research around that. Another medium that people have used for low cost storage in the past is optical media. So think DVDs. That's another medium. Then you have tape magnetic media. Those are I think probably the three that are notable in the field. There's a lot of research on different technologies, but I'd say those are the three notable ones.

It is challenging to get to scale on new media and get to a scale point where it becomes – When you think about cost effectiveness and you look at how much investment has been made in magnetic media over decades, it's incredibly hard to beat some of those cost points. But there's certainly attempts to do so in other domains.

[00:24:10] JM: What are the requirements for building something like a Glacier storage system relative to the requirements for building something like S3 itself?

[00:24:22] KM: Well, with Glacier, as I said, is it provides asynchronous access, and we use that asynchronous access to optimize the way we read and write data from –

[00:24:31] JM: Can you define that term asynchronous in this context?

[00:24:33] KM: Yeah. Asynchronous just means that, in contrast, S3 provides synchronous access. You call S3, say get, and then within milliseconds the object storage is coming back. With Glacier, when you know that you want to get an object back, you call the S3 API and say, "I want to restore that object, and then we notify you when the object is ready to be accessed." So it's asynchronous and that you initiate a request, and then sometime later we come back and say, "Okay, it's ready to be retrieved," and based on the retrieval speed you select.

So the benefit of asynchronous access is that we can optimize the way that we store and retrieve data in ways that make it more efficient cost-wise for us to store that data. To do that, there's obviously then systems we have to build that are receiving request doing the prioritization and the ordering of those requests and then accessing it on underlying media. With Glacier, we have additional opportunities again, because it is asynchronous. One to five minutes is our expedited retrieval space. That gives us time form other optimizations to really lower the cost point.

[00:25:44] JM: So what you're talking about today is Deep Archive. So can you compare Deep Archive to just the version of S3 Glacier that you had before that?

[00:25:56] KM: Sure. Deep Archive is about 75% less expensive. So it's actually just about a dollar per terabyte month to store data on Deep Archive. So it costs less. It also has longer retrieval times. So what we heard from a lot of customers is that they have workloads where they have a lot of data, but they really don't ever intend to retrieve much of it, but they want to hold on to it. They have to or they want to hold on to it. It could be anything from medical records, or legal records, or media archives that are very infrequently accessed.

We, with Deep Archive, our fastest retrieval time is 12 hours in contrast with Glacier, which 12 hours is actually the slowest retrieval time. With Glacier, you can retrieve data in one to five minutes. So for customers who are willing to store data knowing that they can provide us 12 hours to receive that data once they request it, they can save 75% on the storage cost.

So it really goes right back to what we're talking about earlier with storage hierarchy. We have a number of customers that want to use something like Deep Archive for analytics data that goes back much further into the past. But if they decided to start doing an analytics job, they would then just begin retrieving the data, where data that is fresher, data that is less in the past, they would retrieve first and start processing that data. Meanwhile, they're making request to pull some of the older data back. By the time they are ready to process it, then that data is available too to process.

[00:27:29] JM: Tell me about het product development process. Did this start with just the idea that – Maybe we should try to think of a way that like make things even cheaper, or did it start with like, "Okay. We figured out we can store data in DNA." Just give me kind of an idea of the product lifecycle and how you took it from ideation to general availability.

[00:27:53] KM: To launch, yeah. Well, 90% or so of the features on AWS's roadmap are directly set by what customers tell us they want, and this was no exception. This was one where we heard from customers that they liked Glacier, they liked the price point, although when they compared it to running tape libraries on premises, they said, "It's still a little bit more expensive than running a tape library on-premise. Can you bring that cost down to something where there's no decision for me. If it was truly cheaper than tape library on-premises, I would just move my data to you and be done."

Because for many of our customers, actually, they look at everything they have in their data center and they just go rack by rack saying, “How am I going to get rid of this rack?” and they get to the tape library and they say, “Well, I need something at this price to do it.” So that was the main feedback it receives, and then we turned around and we looked at what we had with Glacier and we realized that we had a lot of the raw building blocks with Glacier, a lot of the key services that allow us to, as I said, allow us to receive request and order them correctly. We had a lot of the raw building blocks and there were some additional things we could do to drive the price point even lower. So that’s what we did and developed. So it was entirely based on what customers told us they needed.

[00:29:10] JM: Can you get there entirely from changing storage mediums, or do you also need better compression systems? What else can you do? I mean, I guess thinking from more of a creative standpoint, the product gives you relative to just regular Glacier, “Okay, we have this much more latency that we can play with.” So it’s sort of like, “What can we do in that time period? We can shift it around between different storage mediums. We can change the compression ratios we’re working with.” What are the other axes that you can explore there?

[00:29:45] KM: Well, it gets a little bit into some of the secret sauce so to speak of Glacier. But I’d say that we took a very hard look – When we built Deep Archive, we took a very hard look at the cost structure, as we do for all of our services. We get very specific about what is our cost structure look like, and we go line-by-line saying, “What can we do to pull this down?” Some of that involves hardware optimizations and things that we can do in kind of purpose-building certain hardware for the use case. Others of those line items relate directly to how our software interacts with this service, with the hardware.

So it was a bit of an all of the above type approach of really just going down line-by-line saying, “What we can do to further optimize the cost here?” and we were able to achieve some nice savings, and that’s what resulted in this price.

[00:30:36] JM: As a GM, how do you develop the mental model for how you should be exploring those different areas of the stack? How have you personally found ways to be productive when you have to be such a full stack cognizant tie?

[00:30:54] KM: Well, part of it is we organize our teams in ways where we have certain parts of our team that are responsible for certain line items, essentially, and a cost model and other teams are responsible for different line items. So we can scale the organization by having teams that focus on specific aspects of the problem and are therefore pretty focused on what do they need to do to drive their one or two cost line items down. Of course, I look at everything, and [inaudible 00:31:29] looks at everything when we're pulling a product together to see where it puts us.

[00:31:33] JM: So what are some use cases for Glacier and what are some use cases for Glacier Deep Archive? Can you describe maybe how those two storage tiers contrast?

[00:31:45] KM: For Glacier, we have a number of customers that do media archives with Glacier, and so that is essentially any kind of large photo or video archive or audio archive. For example, when you think about how much video is created by every cable network, every broadcast network, all the movie studio. It's an incredible amount of data every year. Obviously, for those archives, the amount of data that they need on any given day is actually governed by, basically, humans. How much can a single human process and consume in one day if they're pulling together clips or a new broadcast segment, or whatever they might be doing. So media archives is certainly a big use case we have in Glacier.

We also have a number of customers who use Glacier for slightly aged analytics data. So anything that typically we see 30+ days. Typically, if customers are doing big data analytics, they are processing the last day's data, last week's data pretty quickly and using that to drive kind of real-time decision making. But then once it's a week or two weeks or a month old, they aren't accessing it as frequently, yet they still want to keep it around so then they'll shift it into something like Glacier, where it's obviously there for them if they want to pull it back.

Then you take that to the next step, and those same customers then say, "Okay. Once that data has aged even further, maybe it's three month's old, or even a year old, I still want to keep some of that data around."

Actually, we've heard from many customers who say, "At a dollar per terabyte, am I might just not delete some of these data, where before I had to to save money. I might actually just keep it around and have it available so that I can do a year over year comparison, for example, or go back and reprocess that data if I've developed a new ML algorithm and I want to go train it against even older as well, pull it back to do that kind of processing."

That's really where Deep Archive fits, is data that customers – Many customers come to us and they say, "I don't actually ever intend to retrieve this data, but I can't delete it." It might have been a healthcare image, where healthcare providers are required to store data, typically, based on the life of the patient plus some amount of time. So that might be 100 years that they would be storing that data for. Same thing with financial records or legal records where you just have to maintain that data for long periods of time.

Then other customers have intellectual property that they've developed that when they think of Deep Archive, it's kind of a backup for it. It would be in maybe a second or third copy of their data. You can think of folks who generate original content, or kind of core intellectual property for companies. They look at Deep Archive as a way to protect that data, because they intend to keep it forever.

[00:34:38] JM: I could also it being really useful for this machine learning providence problem, where you have this like super complicated tree of decisions that your machine learning model is making. If you're doing something like issuing a loan to somebody, you have to be able to explain why a machine learning model made a given decision at a given point in time, and doing that involves like tracing back how the model looked at a particular time, which can be super data-intensive, and you're very rarely going to need to do that probably. But when you have a court case against your loan officiating service, well, you're going to need to retrieve that data.

[00:35:18] KM: Right. That data suddenly becomes the most important data to find. Absolutely. We've certainly heard that use case as well, where, yeah, where customers say, "Again, I don't intend to ever need to pull this back, but if it's needed, then I'd better be able to go to retrieve it. So I want to park it somewhere at the lowest possible cost, and also know that I can get it back."

Again, as Deep Archive is three data center had done and it's 11 9s durable. That data, unlike a lot of customers we talked to have existing tape archives, where they rotate tapes to off-premises storage, and that can be a very frustrating process for them to actually have to have a human figure out which tape they have to get back and load it and, "Oh, maybe that was the wrong tape," and it's a lot easier with Deep Archive. They issue the request and that data comes back.

[00:36:03] JM: What SLA do you guarantee with the Deep Archive?

[00:36:07] KM: So we have an SLA for availability. The expectation we set for retrieval time is 12 hours with our standard retrieval speed, and then 48 hours is bulk. So you save even more money if you're willing to wait up to 48 hours typically, then you'll get even lower price to retrieve it.

[00:36:24] JM: That's still pretty fast. 48 hours is still pretty fast.

[00:36:27] KM: Right. If you're involved in litigation, for example, where you need to pull it back. I mean, typically, the timelines for that are weeks that you have to sort of pull that data together. Sure, 48 hours for those kinds of use cases seems perfectly reasonable.

[00:36:41] JM: Do you think you can go – Can you go to deeper archive? Can you get –

[00:36:46] KM: The biggest problem is naming it actually. We don't really know what we would call it if we did.

[00:36:50] JM: Very cold Glacier. AWS Pluto Glacier.

[00:36:54] KM: Right. That's right. We could. Certainly, we don't hear a lot of customers asking for even slower than 48 hours at this point. If there's an opportunity, we certainly would explore it.

[00:37:05] JM: Would there be significant savings at this point, or do you think it's diminishing returns after 48 hours?

[00:37:08] KM: I think we're getting pretty close to diminishing returns at 48 hours, but my team loves the challenge. If I say, "Hey, let's find even more ways to save money and set a different retrieval time," then I think they would love that challenge. Again, it's mostly driven by what we hear from customers.

[SPONSOR MESSAGE]

[00:37:31] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

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

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

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

[INTERVIEW CONTINUED]

[00:39:23] JM: It's another topic I don't know if you know anything about, but there was this paper that came out of Berkeley recently about the storage options for serverless computing, and basically the fact that if you're doing something with Lambda functions, for example, AWS Lambda, then there are some challenges with storage today.

For example, communicating messages between two Lambda functions, or having a Lambda function read from storage. I know you've been working on AWS Glacier, so this may not be your cup of tea at all, but do you have any idea what the ideal storage mediums for serverless functions might look like?

[00:40:05] KM: Well, we think that S3 is a pretty good option for a lot of Lambda function use, and that it's API driven. It can be all accessed to the AWS SDK. I think that there are certainly some use cases that might be suited for something like an NFS chair that's connected to a Lambda function. I think it really does depend on the workload and what customers are trying to accomplish with Lambda functions. If they're building a brand new workload, we would strongly recommend they consider using S3 and DynamoDB and Lambda together. I mean, those are all – If you think about the spectrum from self-managed all the way up to fully managed, S3, DynamoDB, Lambda, are the extreme managed. These are fully managed services. With Lambda, you drop in your code. With DynamoDB, you store your data in a table structure and can run queries against it. With S3, you have a blob store to be able to store larger blobs of data that you want to be able to access by URL.

So, I mean, I really think the combination is actually pretty powerful. That said, there are customers who, for sure, look at Lambda to help them process data in a file system. For them, sure, it would be valuable to have a file system available in their Lambda function to be able to do that just to be able to work on things that they might already have.

For example, if customers are bringing a file system in and they want to use Lambda to process it, then that would make sense to do. For a truly kind of net new workload, I think customers would be – Certainly, internally, the way we think about building services, we look to the extreme to say, "Can we use a fully managed service?" Then we only back off from that if we have really good reason that we don't think we can for some reason.

[00:41:56] JM: You're very heavily on the storage side things, but it's pretty clear just looking over into the compute side of things that there's quite a gradient developing there as well in terms of stateful workloads, and stateless workloads, and, as you said, "When do I need a file system for my Lambda function, and when don't I? Can my function disappear miraculously and can I restart it, and is that a problem?" What's your perspective on the "compute hierarchy?"

[00:42:28] KM: Well, I think sort of what I said earlier. When we start to go build these services internally, our initial preference would be how do we build this in a way where another service is responsible for maintaining as much of it as possible? I'd rather not actually have to have my engineers worry directly about catching their instances or keeping the software up to date. I'd rather have that be managed for me and be something that's just done automatically and know that it's happening, because it's something that my service provider, Lambda, or perhaps one of the container services, Fargate, is doing for me.

Yeah, I think for new workloads, the default position should be to try to go as far as you can in a managed service. Again, when customers are bringing workloads in that they already are running, there's certainly some use cases where they are looking to have more direct control or the application requires multiple components that are running in close proximity. So those are some use cases where going more towards the self-managed routes makes more sense. The default position, I should say, is to try to have it be managed as much as possible.

[00:43:43] JM: We completely agree with you, but do we have the same spectrum of questions around durability of this service? If I want to spin up a service, if I can tell the cloud provider, "Yeah. I mean, it's okay if this service falls over occasionally," then I'm going to get cheaper compute, because you can schedule it on to spot instances.

[00:44:08] KM: With a spot instance, for example. Yeah.

[00:44:11] JM: Or like really low-quality hardware perhaps. The discount dustbin hardware, we could schedule stuff on to their.

[00:44:19] KM: Yeah. We wouldn't think of it as discount dustbin, but I do think there's so many workloads that can benefit from our spot instance capability, and it's not about quality. It's really just about time shifting. Do you need to have a thousand cores this very second, or is your workload one where you can have 100 cores now, and maybe it'll have 200 cores in 10 minutes and it'll vary based on what's available.

But there are lots of workloads that actually can run in that kind of environment and certainly save money doing so if they're willing to essentially time shift when the results are available. Similar to how we think about Glacier, in terms of asynchronous access versus synchronous access.

[00:45:00] JM: Would it make any sense to have like a spot market for storage?

[00:45:04] KM: I think the challenge with storage is that we – The concept of deleting data without a customer explicitly telling us to delete the data is, I think, because shivers down our spine. So I think the idea the bot storage market is that you'd have to be willing to essentially delete data when you need this to reclaim the space. I mean that's, in essence, what's happening with spot, is you can use as much capacity until the moment that it's needed for someone who's paying more for it, and then those instances will be shut down.

I think the moral [inaudible 00:45:37] in storage would be deleting data that is marked as spot data. Yeah, it gives me shivers down my spine thinking about it. If it's a real need, we'd explore it for sure.

[00:45:47] JM: Well, I would just think of it like I guess the spot storage vision that I have will be just like kind of what you described earlier with the thing where you say, "Just detect – Monitor this to piece of data, and if nobody is accessing it, just go deeper and deeper and deeper."

[00:46:02] KM: Right. So we do have an S3 intelligent tiering, which automatically saves customers money based on how frequently it's accessed. Certainly, we're hearing a lot of positive feedback from customers about intelligent tiering, because again it's just one less thing they have to think about, and yet they save money if they put the data in intelligent tiering.

So we're certainly exploring other things we can do with intelligent tiering along that idea of it being intelligent. What else can we do with that to automatically save money? For sure, we're looking there.

[00:46:34] JM: So one of the cool things about AWS Lambda was it was one of the most concrete examples of – In doing this show, I've seen a lot of examples of cloud provider technology being utilized to develop new design patterns. Things that look like new computer science design patterns coming out product, as supposed to like computer science whitepapers.

Have you seen any interesting application architectural design patterns recently that seem emergent and just things that are being built with cloud provider tools?

[00:47:11] KM: I think the big pattern that is still very early, but there's a lot more there, is just around more of an event-driven flow-based model versus – And this is where Glacier I think fits really nicely, because one of the other things we launched actually at Reinvent last year was restore notifications. So the idea is you could have a Lambda function say, "Okay. I need this data from Glacier. Send the request the Glacier." Then S3 Glacier will send a notification. Once that data is available, that could go right back to a Lambda function to kind of continue the processing.

I think if customers and application builders start thinking more about applications as event-driven and where the events are not necessarily my code calling another piece of my code, it's really my code calls whether it's SQS, or Kinesis, or out to S3 to initiate an action. Then once that action is completed, it flows back into the next part of my code. That seems like a really interesting pattern just to – That seems like a really interesting pattern.

The other thing I would say about that pattern is that it leads to a world where there's much better inspection and introspection capability over what's happening. That I think is another pattern, which is really underdeveloped in the market today, and I've seen more and more people realize that if I'm going to run a cloud application that's designed to scale to very large scales, I really need the ability to introspect to see what's going on.

If I'm seeing extra latency, where is that latency coming from? So you think about AWS tools like X-Ray, and CloudWatch, and there's capability and certainly more to be built to help provide that introspection. But when you kind of decompose the application from being a bunch of code running in one JVM, for example, and you decompose it to, it's a little bit of Lambda code that runs, that fires off some actions and then those actions come back. There are so many points of inspection that you can have there that we can provide, AWS can provide, that actually a lot of it could be "free". You get a lot of that monitoring kind of just built in.

[00:49:21] JM: Yeah. Just to wrap up, any ideas for future products within the storage space that you're excited about?

[00:49:27] KM: We talked a little bit about intelligent tiering. I think that's an area where we're certainly talking –

[00:49:31] JM: It seems like there's a lot of depth to that product.

[00:49:33] KM: There's a lot that can be done there. So talking to a lot of customers about what they like, what else we can do to make that storage even more intelligent, so to speak. I think that everything we talked about with Lambda and with compute, we're definitely looking hard at what can we do to make those integrations even easier.

Another thing we launched at Reinvent last year with S3 Batch Operations, and that's the idea – I mean, we have customers that literally have billions of objects, or multiple billions of objects, and they want the ability to process all of that data, or change metadata across all billion objects. S3 Batch Operations makes that possible, and we're sort of thinking about what are some next iterations we could do on that.

[00:50:17] JM: A couple other dynamics I find interesting in the storage space are you have these applications these days where it's like, "I'm using my phone and I want to have as much data on my phone as possible for a given application, because the applications could be more buttery smooth if I'm operating it on my phone. Obviously, I can't store everything on the phone."

Is that a space that you might play in on the AWS storage team, or do you feel like that's more like Appsync, or somebody on the mobile?

[00:50:50] KM: Certainly, Appsync is probably where the focal point when you think about that. But we don't think of it as being strong silos, where I can't go look at something else in that space. I think that there's been some customer feedback around that, and maybe something we spend more cycles on. But I think with Appsync, that's a pretty good product for helping customers sync the data they need today and sort of have a coherent view. I mean, that's part of the challenge with caching data locally, is the coherence, right? So Appsync definitely helps with the cache coherency type problem, but there's certainly more we could do there.

[00:51:29] JM: Yeah. So there was this theme that seems to be emerging. Problem like the tiered storage that you described, it seems like an opportunity for machine learning. You have all these different signals for types of data and how frequently data is being accessed and the types of data that are being accessed. You don't have to talk about particular applications of machine learning, but do you have a broad perspective for how machine learning could be applied to making storage more efficient?

[00:52:02] KM: I certainly think it's true that proper application and machine learning in storage could help customers save money. Kind of to the point earlier about memory hierarchies, if you can make better predictions about what data should be in which part of the hierarchy at a point in time, you will likely find opportunities to save money.

I'd say we've taken a look at that a bit when we were building intelligent tiering, and what we found with intelligent tiering is that the single most important question, or the single most important signal about whether data would be accessed was has it been recently accessed? So that's why we built an intelligent tiering initially to be – Really, just based around how frequently or what was the last time that this data was accessed.

[00:52:45] JM: Don't need much machine learning.

[00:52:46] KM: You don't need much machine learning for that actually.

[00:52:49] JM: [inaudible 00:52:49].

[00:52:51] KM: Right. But that said, machine learning, particularly if it can be tied into more business intelligence around the data to say, "We know that this piece of data is about this patient, or this record, or this tax return," whatever that kind of additional metadata is. That's where I think you'll find more optimizations are possible with machine learning than sort of generically looking at an object and just the basic characteristics of an object. Certainly, I think there's opportunity in that more application-specific domain.

[00:53:26] JM: Yeah. Kevin Miller, thanks for coming on the show. Great talking.

[00:53:28] KM: Yeah, thanks for having me. It was great.

[END OF INTERVIEW]

[00:53:33] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use, and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plug-ins. Use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on-the-fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations. You can check it out for yourself at gocd.org/sedaily.

Thank you so much to ThoughtWorks for being a longtime sponsor of Software Engineering Daily. We are proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]