EPISODE 22

[INTRODUCTION]

[0:00:00.3] JM: Different databases have different access patterns. Key-value, document graph, and columnar databases are useful under different circumstances. For example, if you're a bank and you have a database of customers and the transactions that they have performed, the ideal access pattern for aggregating the total amount of all transactions might be a columnar store.

If the transaction amounts are all in one column, it helps to have all of those columnar entries close together on disk. However, if you wanted to look at your bank as a social network and you wanted to be able to map how money flows between the different people who use your bank, you might want to map that data as a graph database. That would make it easier to query for the connections across the different users in the bank.

Cosmos DB is a database from Microsoft that allows for multiple data models and multiple well-defined consistency models. Today's guest, Andy Hoh, is a product manager at Azure Cosmos DB and he joins the show to describe the product. Microsoft unveiled Cosmos DB at Build, their annual developer conference, which is where I performed for this interview. It was a pleasure hanging out at Build in the podcast booth that they set up. So thanks to Microsoft for inviting me and setting up this awesome podcast.

[SPONSOR MESSAGE]

[0:01:29.3] JM: Flip the traditional job search and let Indeed Prime work for you while you're busy with other engineering work, or coding your side project. Upload your resume and in one click, gain immediate exposure to companies like Facebook, Uber, and Dropbox. Interested employers will reach out to you within one week with salary, position, and equity up front. Don't let applying for jobs become a full-time job. With Indeed Prime, jobs come to you. The average software developer gets 5 employer contacts and an average salary offer of $125,000. Indeed Prime is 100% free for candidates – no strings attached. Sign up now atindeed.com/sedaily.

Thanks to Indeed Prime for being a loyal sponsor of Software Engineering Daily. It is only with the continued support of sponsors such as yourself that we're able to produce this kind of content on a regular basis.

[INTERVIEW]

[0:02:45.1] JM: I'm here with Andrew Hoh who works on Cosmos DB. Andrew Hoh, welcome to Software Engineering Daily.

[0:02:50.1] AH: Yeah, thanks for having me. Excited to be here.

[0:02:53.2] JM: We're going to talk about databases and Cosmos DB, but let's do it with just the idea of the fact that many companies have different databases for different reasons and they often are putting two copies of the same piece of data in different databases. Why do they do that?

[0:03:12.2] AH: My experience, I've seen that people tend to use databases for a variety of different reasons. If they have certain latency requirements, if they have certain consistency requirements, if they have data that needs to be connected to certain analytics services, they'll find themselves duplicating data, putting it to a relational store, putting it to a NoSQL distributed database. They put it kind of all over the place. I think the typical example I always think of is they're individuals who like going with the hot storage and cold storage. I don't know if you're familiar with the pattern, but it's basically — It's not really the LAMDA architecture. I think LAMDA is more function-based, but this is — It's a component of the LAMDA architecture. It's purely based on, "I need really high performance but I don't want to pay that money of having tens of terabytes, almost at the petabyte scale, of data all on SSDs, all in memory," so I try to put as a day's worth of data, a week's worth of data inside my hot store and then the rest I put in the cold storage.

That architecture actually works fairly well just because you'll find yourself managing the budget much more easily. It's much easier to get approval for many managers when you have an architecture that leverages cold storage. You can also expire out data to save cost as well.

[0:04:38.3] JM: You talk about the hot storage in the cold storage. What are the costs to doing that?

[0:04:43.4] AH: The cost is now your double savings your storage. Essentially, you're saving different granularities. Talking a little bit about how even the team manages its hot and cold storage, you'll find that we keep about a week's worth of data in hot storage so that we can do quarries much more efficiently. If a customer comes in with a request to figure out why did this not work or something happened here. Can you give me some more insights? We have to do quick query that happens to go through within a second.

For that, we need to make sure that we it in a hot storage when we're doing quarries for much longer term data. Let's say someone is asking for their bill for the last three months, then we could go to colder storage, let a query run a little bit longer. Overall, we even go granularities. We make sure that we have data points for at the minute, at the second level for the hot storage, and then we expand it out to the daily, weekly level for colder storage.

This data pattern works very well just to optimize cost. In the dream scenario, a lot of people would like to put all of their storage into one database, one hot storage database, but in the end what we found was if you go to your managers or if you go to management, say, "I have a solution that's going to cost $100,000 per month, $500,000 per month," you'll find yourself not easily able to architect that without strong justification. I do see a lot of people double using the storage.

I'd also like to highlight, there's a lot of people tend to double use storage because they feel like with different databases you get different capabilities. For example, if I have data that requires a NoSQL database for the low latency reads and writes, like cosmos DB, which is a distributed database, you'llfind yourself using that for the latency. Then you might also find yourself storing the data into a SQL DB just so you can use a reporting services.

In SQL DB, it won't be identical data. What you'll probably do is get different granularities, probably do aggregates that will some up the data for daily metrics, put into SQL DB and now you can do different reporting services on it. Eventually, Cosmos DB will get a lot more integration and you'll find yourself using it for a lot different — Or you'll find yourself having a lot different analytic services attached to it for reporting reasons. With that, you'll find that you can keep all your data within Cosmos DB without having to replicate it to another database.

[0:07:10.5] JM: I want to about the world pre-Cosmos DB, because I think that illustrates better what Cosmos DB is. When I think about some of the most notable access patterns for different database types, you've got the classic row-wise access format, which is like a SQL database. You've got the document style format, which is like a Mongo database. You've got the columnar format, which I think Cassandra — Right? Cassandra is a columnar format where you can access an entire column all at once in a really efficient manner. You've got graph databases which can represent the — That's like Neo4j,0 which can give you efficient query times if you're trying to find all the connections between a social.

I think there's one more. There's a fifth — What's the fifth one?

[0:08:05.5] AH: You mentioned graph — Key-value.

[0:08:07.0] JM: Key-value, right. What's a key-value data? What's a popular key-value database?

[0:08:12.9] AH: The most popular key-value database, I would say it's simple storage kind of, like S3 for AWS, for blob storage key-value. Also, table storage, more key table storage where you have a single index attached to your actual data and you can easily access it from that.

[0:08:31.6] JM: Yeah. There's different applications where you want these different types of databases. If I'm Facebook, I've got people that are querying Facebook to find the friends that are closest in similarity to me. If I make that kind of query, sure, I want a graph database. If I want to query for the sum of all of the number of likes that I've gotten over the last year, maybe you want to query a Cassandra-like think because I can aggregate the sum of all of those likes overtime.

There is some overhead in maintaining all of these different models at once because if I'm building a graph, graph database representation of the world and I update that graph database representation, probably I also want to update the columnar database representation. Now, where we’re going with this is that Cosmos DB includes the ability to have multiple different models. Continue to talk about the pre-Cosmos DB world. How are people typically reconciling the differences between their different data models?

[0:09:49.7] AH: What we find is with Cosmos DB, we’ve revolutionize the world with different data models that can put into a single database. Databases are inherently very flexible, so you'll find that you can put in different data models within Cosmos DB and you'll find very good performance. I think when I think of a graph database, it has vertices, edges, and you're able to traverse the graph. Graph traversals can easily be put into the database. You can have your nodes being represented in however you want for most efficiency.

The concept of having a dedicated database specifically for you graph database, specifically for calmer types, is a little bit of it — It's a subset of culture that's come up within the last, I would say, 5, 10 years where everyone's looking for a specific database for their specific needs. When you have a highly optimized database, that's really efficient in doing basic crud operations; create, read, update, delete, and then you can add more functionalize on top for graph traversals for updating documents for extracting key-value simple index, returning the value.

You'll find yourself in a state where you’re able to use a database in many different ways that weren't previously thought possible. You can use it as key-value store. If I were to model my data in a way where I just have single indexes and single values and I just search on it with the simple query, that's all very efficient because a document database has indexes set up so that you can have your simple single indexes and your values, but document databases are a little because you got secondary indexes. You could have the nested data structure.

With that, we also support the secondary indexes and we have the auto indexing to make sure that you can access any part of the document. Graph databases, we can access vertices, we can access edges that are all modeled in a way within our database to make sure that it's extremely performant. Underneath, you'll find yourself that it shares a common JSON structure across the board. The native functionality for graph is graph traversal and we can add that logic inside the database to make sure that you can traverse an efficient way as well.

For large-scale aggregates, we have architectures supporting that as well to make sure that you do get benefits of a calmer store. The beauty of Cosmos DB is that because it's a fully managed service, all the implementation details, all the hassles of resource governance — Research governance is part of a multi-tenant system where we can make sure that there's no noisy neighbor problem. If I'm a user, user A, and there's another user, user B, and we're using the same tenant, we make sure that we have strict regulations so that you're not taking up all the CPU memory or eye outs for anyone else to experience downtime.

With the resource governance merge with a lot of the infrastructure and what we're building out, we can guarantee that you get this type of performance. You won't see much of the details in terms of how was this done. You won't have to worry about how do I set up my replica sets, how do I set up the global replication. A lot of this has taken into the Cosmos DB's hands because we want to make sure that all the hassles of setting up in the proper way so that you'd get this highly performing database that works well as a graph database, because the hardest thing is if you want to someone and said, "Hey, can you set up a really performant graph database," they need a large team to really understand how databases. Cosmos DB has a massive team that has a ton of experience of building databases, so we understand exactly what values you get from different databases and we're able to take the best parts and put into a single database.

As of right now, those optimizations are for specific database. If I choose Graph DB, you'll get a lot of the graph optimizations. If I choose the Mongo DB flavor for the Cosmos DB account, you'll get the MongoDB optimizations. If I choose the native document DB, I get the native document DB optimizations.

Eventually, we want to move to a model where we can share a lot of the optimizations across the board, but that's much further down the line where we have to really think about how we can share it without paying some type cost in terms of extra storage overhead, extra index overhead, or a lack of performance because there's conflicting matters in that aspect.

[0:13:59.6] JM: Just to be clear today, If I'm setting up a new social network and I want to make a graph database within Cosmos DB and then I also want that data represented in a row-wise format, like a SQL style format, I need to make two different Cosmos databases.

[0:14:17.9] AH: As of right now, you have to make a Cosmos DB account with the flavor of graph. At the creation time, you have to specify exactly what type of data model you want. Eventually, that might be going away where we're able to combine and merge it altogether. Work has actually started for that as well and that will happen further down the line.

[SPONSOR MESSAGE]

[0:14:42.6] JM: Ready to build your own stunning website? Go to wix.com and start for free! With Wix, you can choose from hundreds of beautiful, designer-made templates. Simply drag and drop to customize anything and everything. Add your text, images, videos and more. Wix makes it easy to get your stunning website looking exactly the way you want. Plus, your site is mobile optimized, so you'll look amazing on any device.

Whatever you need a website for, Wix has you covered. So, showcase your talents. Start that dev blog, detailing your latest projects. Grow your network with Wix apps made to work seamlessly with your site. Or, simply explore and share new ideas. You decide. Over one-hundred-million people choose Wix to create their website – what are you waiting for? Make yours happen today. It's easy and free. And when you're ready to upgrade, use the promo code SEDaily for a special SE Daily listener discount. Terms and conditions apply.

For more details, go to www.wWix.com/wix-lp/SEdaily. Create your stunning website today with Wix.com, that's W-I-X-DOT-com.

[INTERVIEW CONTINUED]

[0:16:08.0] JM: If I spin up a graph to Cosmos Graph DB and a Cosmos in SQL DB, it's not like there's some kind — You're saying there's not any kind of magical reconciliation that I could do where if I update something in the graph version, it'll just easily propagate to the SQL version. I still need to do some kind of maybe ETL job to pull the data from the graph database and put it into the SQL database, right?

[0:16:36.0] AH: Are you talking about the SQL syntax or API on top of Cosmos DB or are you talking about a SQL Server instance of a SQL database?

[0:16:43.5] JM: I’m talking about if I set up — I guess the thing I’m having trouble understanding, I get Cosmos DB is a place where I can have a hosted version of any of these database access and consistency models, whether it's graph or columnar or so on. What I'm unsure about is if I have a data set that sometimes I want to access it like it's a graph. Sometimes I want to access it like it's a columnar store. Sometimes I want to access like it's a document store. Does Cosmos solve that problem and unify those different data sources or does it just kind of present a way of hosting any of those different databases?

[0:17:25.3] AH: The question in the first option and second option where you talk about is it one or the other, it's a little bit of a combined solution to that just because when you use the underlying database technology that supports both, you'll find that it's actually fairly easy to expose all the APIs as well, if that make sense. It's not like if I save it as a graph database, I can actually still run a SQL query on top of it. I can go right now into the Azure management portal, do a select star from C, select different properties from C —

[0:17:58.9] JM: But it’s less efficient that way than the row-wise representing that.

[0:18:01.5] AH: It’s actually incredibly efficient as well, because the indexing is still the automatic indexing that was there in the underlying database infrastructure. For the graph aspect, there are a couple of additional functions on top that will let you do the efficient graph traversals for the key-value lookup or document lookups. We still have the automatic indexing so that you can still do the SQL star from C which is incredibly efficient. The indexing really hasn't changed. We index different parts of your document automatically. We allow you to configure it so that you can execute your own path as well. That underlying infrastructure is identical so you can still run efficient SQL queries on top of your graph database as well. You can do that right now provisioning a Cosmos DB account.

[0:18:43.6] JM: Is the underlying — The thing is if you set up a Cosmos DB database and say, “I want this to be like a graph database.” Or if you say, “I want this to be like a columnar database,” you have to these difference flavors, one of the five flavors. If I set it up as a columnar store and I give it a query that is optimal as a graph that would — If I gave it a query that would be best answered by a graph database, I'm not going to get the latency, the low latency that I would get from a graph database. I would get some degradation in the query quality because I'm accessing a columnar store with a query that would be best suited for a graph database.

[0:19:29.7] AH: Actually, you'll probably get the same performance, if not, better still. I think I guess I should elaborate a little further because a lot of the indexing that's done for the Cosmos DB objects can be shared across the board.

Quick example, if I were to run it as documents store and I want to extract values ABC, I simply have the automatic indexing that will index my document to actually index properties ABC and I could quickly go and find ABC. If it have as a graph database, now, let's say a simple graph operation is to traverse a neighbor, and so that extra metadata is actually saved within the database of that if I say, "I have vertex Mary and I want to find the neighbors of Mary." That can easily be done on the database level, but I can still say if I want properties ABC same as a document from Mary, that's still done in the same indexing patterns so I get it all back efficiently.

The way to think about it is Cosmos DB is leveraging a lot of flexible technology that can be shared across the board. It has a lot of resource governments as well as infrastructure technology that lets you take the benefits of indexing just because indexing is shared across, right? Any database — One of the most important things is how indexing is done. How do I index my values? How do I make sure — Then once the indexing is set, you now have to wonder, "There are certain operations that different databases are really good at." For a graph database, it's really good at doing traversals. For document store, it's really good at picking up values that might use secondary indexes. For key-value, obviously, its primary indexes.

Now, you can see that there's a common pattern where if I said of indexing structure that has the primary indexes, the secondary indexes, that indexes the graph, and then I have the additional support to have the transaction or the operation so that I can do graph traversals, I can do MySQL queries, I can do my MongoDB queries, now you've kind of built an architecture where you're able to support a lot of functionality across the board in an extremely performant way.

The shared automatic indexing that Cosmos DB has can be leveraged throughout, and you'll find we have a white paper out there we can read and kind of learn the details of how that's done too.

[0:21:44.2] JM: Sure. Then help me understand how do I choose what flavor, because if I can choose the flavor of the row store or the document store or the graph store, if it's all going to be indexed the same, why does it matter what flavor I choose?

[0:21:58.3] AH: I see. As of right now, I think the best example I could give is the Mongo DB API, MongoDB supports a BSON format. It's a binary JSON format. It supports a lot of additional data types that weren't there on the underlying destruction initially where we have to make some modifications to make sure it works efficiently.

As of right now, if I saved my data on my BSON data, it's saved with a couple of metadata. If I'd run MySQL query on this data it's going to look a little bit different because these are data types that were normally supported in the JSON schema of document DB API. And so we want to make sure that these type of confusions don't really happen until we make a lot of infrastructure to make sure that the simplicity and the understanding of the database reaches where we're trying to get to so that if I use different APIs and I mix-and-match that I won't see any unexpected behavior where I'm like, "What is that property? I don't really understand why is that there." That was probably there to make sure that the database is very efficient. Eventually we'll kind of hide, make sure that those details are natively in the database engine so that it's not exposed upfront and you'll be able to run your extremely efficient quarries, graph traversals, and MongoDB queries, SQL queries, all those.

[0:23:09.6] JM: Okay. If I'm a startup and all of my data has been written to MySQL or to Cassandra and I see Cosmos DB and I want to migrate to it, what's that migration path?

[0:23:25.4] AH: I see. The migration path will normally go through a typical — I've done a couple of migrations for a couple of customers where the process is in a couple of steps. First, you want to make sure that how much data you're migrating over. If you're a startup, it can mean that it's within 10 gigabytes, 100 gigabytes. With that, means that, "Can I afford downtime?" If I'm usually taking out a large amount of data from any database you'll find that it's taking up the CPU, the memory, the network bandwidth from your current production application. Determining that is the first step.

The second step is where we're really trying to help a lot of her users solve migrations by fixing the code redevelopment. There's a lot of code out there that's already used in MongoDB. There's a lot of code out there that's already using Gremlin, and so we want to make sure that we've open up a lot of familiar APIs so that people can leverage database infrastructure which we are already have without having to change your application.

As we start expanding a lot of these languages out, as we start supporting more drivers, as we start supporting more the models, you'll find that you'll be able to do migrations with a simple data migration rather than an application migration and the learning curve as well. There's a lot of people out there who aren't really familiar with document DBs unique syntax, and so we want to make sure that we open up a lot of their flavors so people can understand what the difference is between a document DB syntax versus — And they don't have to understand this, between Gremlin or MongoDB, and there are a lot of people who are already familiar with MongoDB which makes it a lot easier for them to use this Cosmos DB infrastructure without having to relearn something.

[0:25:01.8] JM: As I get tons and tons of data at my startup or if I'm even building an IoT application where I don't have very many customers but I've got tons and tons and tons of data. I, as the developer, need to make some decisions about how I'm going to spread my data around. Give a general explanation for how databases get sharted or partitioned or replicated and how that applies to what the developer has to do when using Cosmos DB.

[0:25:35.6] AH: Yup. Partitioning in general, it's not as magical as many people think. It's still technology. You still have to understand, when I'm partitioning any database, I need to understand how to partition my data. Let's take a typical example — Or maybe I should provide some context for partitioning. With any distributed database you're finding yourself scaling horizontally. Horizontally scaling means that you're not getting beefier better machines, rather what you're doing is you're scaling out and adding more partitions.

This is a nice infrastructure where you're trying to hit that terabytes, petabytes scale without having to get this single machine that has this much Ram, that has this much memory, that has this much a hard disk or SSDs, and you're finding that the cost is exponential when you're getting beefier machines. With distributed databases, what you're doing is you're adding more partitions and you're able to scale out that way in storage as well as throughput.

When you scale on in this effort you have to make sure that you choose a really good shard key or partition key. This key dictates exactly how will the data be separated across partitions. This is incredibly important. I think there has to be more emphasis in the database world about choosing a really good partition or shard key just because I've seen a lot of different users choosing a bad partition key which leads to one partition overflowing. All the request hitting one partition, and this is obviously not a scalable architecture. They're going down a path where — Let's say, typical example is I have customers' birthdays and I have an application where I'm sending a birthday message to whoever's birthday, like Facebook. Facebook will send you a happy birthday, or it's this person's birthday every time one of your friends has a birthday. I'm sure you're familiar with that.

Then, in partitions to find out — Partition my data by birthdays, I'm finding this — It's at least distributed well, but let's say a lot of people are born January 1st, now, all my request are going to the January 1st partition which is not really scalable. I guess a better example I would give is airline dates, when I'm booking a flight. A lot of people go home for Thanksgiving. A lot of people go home for Christmas, and if I'm partitioning my data by the actual date, you're finding yourself a lot of requests going to one partition or two partitions that hold the dates for Thanksgiving as well as Christmas.

You want to make sure that you just partition key that's evenly distributed so that your data can evenly distribute across X number of partition you provisioned. That's just in terms of storage. In terms of request, you want to make sure that let's say I have a new company that caters to food delivery in New York City. If I partition my database by cities, that's just the worst thing you could do just because everyone's in New York City. It's all going to go to one partition. Anyone from San Francisco will go to another one, but you're not really getting that throughput. You're not really able to hit request to every single partition, and so you want to make sure may be you do it by ZIP Code, may be you do by their first name and last name which is much work attributed.

You want to make sure that you really choose a partition key that spreads out the throughput, spreads out the storage, and you also want to make sure that the partition key is known so that you can direct your queries. Let's say I partitioned by first name, last name, but all my queries are who likes to eat Korean food? That's not my partition key, so I have no idea where that data lies, which means every time I send that query, it's hitting every single partition and that means every single machine is running that query to figure out, "Who likes Korean food?"

This is much more efficient if I say, "Give me all the people named Andrew, starts with And or whose last name is Hoh. Go find everyone there." Now you're going to find yourself hitting a subset of the partitions that have everyone's first name and last name, and this is a much more scalable architecture that will go to terabytes, petabytes, whatever you're looking for.

[0:29:26.0] JM: Does an application developer's approach to using Cosmos DB change if they are having a write heavy application or just a read heavy application with few writes?

[0:29:38.0] AH: Yes. If you have a super read heavy application, you do want to choose a partition key that allows for a lot of read heavy aspects to it. Let's say I know that one example would be people who read the news. One person will publish an article but there might be millions of viewers. If you know that it's kind of this type architecture, then you want to make sure that when you do a read, let's say, you're getting a list of all the news articles for the day, you can separate it out rather than doing a day by partition. You'd want to really make sure that you are not going to single partition for those reads, but it's separate across.

For a write heavy one, you want to make sure that all the writes are going across. Some of the other typical things that we see is when you do a write for a write heavy application, you want to make sure that's as fast as possible. You want to really make sure that when you're doing a single write that all your data is kind of — It's easily updatable, if that make sense.

In the typical database world, I think you're probably already familiar. You have data normalization. Are you familiar with data normalization? Data normalization is you're able to take objects, common part of objects and extract it out so that if you are updating that, you will need to update it once rather than for everyone. For example, let's say the city of New York changes its name. Let's say it's no longer New York City and they decided to name it just York City or something like that. Rather than going to every single person and changing their New York City, you can actually put an ID and reference a different document and change it once and so that every time you read, you'll have to read the person as well as the actual city or the ID for the city. For the write, it's only update that one object.

This is some of the things that you do need to think about when you're modeling your database to make sure that you can have this efficiency in terms of read applications and write up applications so that you don't need to do four writes every time for a write heavy application, or else that's kind of a waste on the database. You want to make sure it's catered towards that.

[0:31:48.7] JM: We're at Microsoft Build. You've been standing out in the Expo Hall for much of the day. You work on Cosmos DB. You work with the team. You and I know each other outside of Microsoft. We have mutual friends, but the past couple of days you've been just standing at the Expo Hall — People who haven't been to conferences, every software engineering conference, there are talks but there's also this giant expo hall where there are sponsors and — Since we're at Microsoft Build, it's like half external third-party sponsors and half of it is Microsoft talking about new technologies that have been announce at Build or things that have been popular over the last couple years or whatever.

Cosmos DB is one of the new things that was announced, so you are standing at a booth representing it, demoing it to people, talking to potential customers. When these people are coming and they are expressing interest in Cosmos DB o they're saying, "Yes, I'm going to use that." What is it that they're excited about?

[0:32:48.2] AH: Some of the things that they're really excited about is they're able to use the familiar APIs. I know that one of the biggest blocker for someone to use an application is that you have to learn something new, you have to really understand it and you need to pitch to your team and you have to tell them all to go buy a book, watch all these videos, and learn a new technology.

Especially in the database world where any migration is a combination of actually moving your data, it's actually changing the application, and it's teaching everyone. The last one which is also very hard is you need to go to management and say this is what we need to do and you need to explain how the value propositions kind of outweigh the cost of getting all these people to get on board.

[0:33:27.2] JM: You need to make sure that, now, all of your application endpoints still meet their SLA.

[0:33:35.3] AH: Exactly. With that, we've kind of — We're aiming towards unblocking one of the big blockers which is learning something new as well as changing your application code. So we found a lot of people were interested in our graph support, our Gremlin support, where they already know Gremlin. They're already familiar with graph databases and it's something that's much easier to grasp.

With that, our Mongo support which has been there for about a year now, I think it went GA last month, that's also exciting to people where they're able to use their familiar Mongo DB queries against a fully PASS service that has — For those who don't know, PASS is platform as a service, and that has all the benefits of allowing the Cosmos DB team to manage the upgrades, handle the 4am wake ups in case something goes wrong to handle all the SLAs, to deliver 99.99% availability, and all of that management kind of goes to us that we could free up time for one else.

Then the next thing that people are really excited about is the global distribution. A lot of people are looking to replicate, that they are seamlessly across many different regions. As many of you are familiar with, Azure has 30+ data centers. I believe as a cloud provider, probably the most in the market, and people are looking to actually replicate that data across continents to deliver that low latency as well as for disaster recovery scenarios. When your business is relying on your database and a natural disaster happens, you want to really make sure that you have another replica of your data so that if something happens, that could easily pick up as the write region, the read region to make sure that it could deliver all the request and have no downtime for any type of production service that needs to meet this 99.99 availability. We have SLAs on it, financially backed SLAs saying that we'll deliver the 99.99 availability, and it's financially backed so that if we don't and we offer metrics and the portal so that you can actually see it to make sure that you keep us honest.

[0:35:32.5] JM: What is your responsible working on the project?

[0:35:36.2] AH: I am a program manager for Azure Cosmos DB. My responsibilities have shifted over the time. I've been on the team for 2-1/2 years as full-time. I interned 3-1/2 years ago, so I was in the first batch of interns. I rejoined. We've been through private preview, public preview, we went through its GA. Now, with the Cosmos DB, we went through this GA as well.

My roles and hats within the team has changed from — I ran compliance at some points. I ran privacy security from the program manager standpoint. I ran some of the customer acquisition, so we all worked on getting a lot of big enterprise customers. I ran some of our development tools. A lot of the Azure integration, so how does it integrate with other Azure services. I worked on the Mongo DB API. Kind of a little bit of everything.

[0:36:29.8] JM: Do you like the variety?

[0:36:31.0] AH: Yeah, it's very unique in terms of getting this aspect of working on many different parts of a product because lots of products are very big. As you can imagine, if you work on a certain product within office that's way bigger, you'll find yourself working on a small scope. With a small scope, you're very focused in — Which is nice if you're looking to get a very deep deep dive into whatever technology you're working on. Since we worked on the technology all the way from — Before public preview and private preview, I got to see the development of the product. I got to see how we actually go and acquire customers. How we build a community. How do we make sure that our customers are happy? How do we deliver this kind of happiness?

I think the biggest thing is when you go to a customer, show them the database, you help them work with it and they're just ecstatic. They're ecstatic to go tell management that they want to move on to it, and that kind of gratification really goes a long way.

[SPONSOR MESSAGE]

[0:37:27.6] JM: Artificial intelligence is dramatically evolving the way that our world works, and to make AI easier and faster, we need new kinds of hardware and software, which is why Intel acquired Nervana Systems and its platform for deep learning.

Intel Nervana is hiring engineers to help develop a full stack for AI from chip design to software frameworks. Go to softwareengineeringdaily.com/intel to apply for an opening on the team. To learn more about the company, check out the interviews that I've conducted with its engineers. Those are also available at softwareengineeringdaily.com/intel. Come build the future with Intel Nervana. Go to softwareengineeringdaily.com/intel to apply now.

[INTERVIEW CONTINUED]

[0:38:18.6] JM: When you take a task like compliance or database security, that's something — You probably didn't study database security in college, but I'm sure there are plenty of people around Microsoft who are experts in database security. When your boss comes to you, or the director, or whoever it is, your higher up, and says, "Okay, you're on security now." Is the process of doing that — Is it like you just go around Microsoft and like ask as many people about security vulnerability. What's the process for that?

[0:38:52.3] AH: The processes is we actually have a top-notch security team for our data platform, and these individuals are the experts. They are the ones who'll go —

[0:38:59.3] JM: These are people who are under Azure.

[0:39:01.2] AH: Exactly. They're specifically for the data platform team, and I imagine there's other security things for different aspect of Azure, and these individuals would go and try to break your database no matter what. These are the individuals who are up-to-date on all the security flaws, when are they released, and they're the ones who'll make sure that we do critical updates.

My job and my role is to interact with them, fully understand and give them the tools to make sure that they can test against the database as well as make sure that we get to deliver a lot of the security requirements in time. Obviously, it's very tough just because coming straight out of college, I didn't really master database security. I think as we kind of move along, as the security flaw details get harder and harder, as people who are more familiar with it come in, I kind of shit it off more towards a MongoDB guy.

In terms of compliance, compliance was a little bit easier just because there really wasn't a lot of expertise in cloud compliance. That's still not something you can go into a database class in college and learn. That something that still very unique, and so when I came, I remember just reading hundreds and hundreds of pages on clouds compliance and different certifications. With that, it was much easier to understand exactly what people are looking for, exactly what different companies required and learning about the details of what makes ISO, HIPAA compliance, all of those. It was interesting. But yeah, I think the MongoDB API and growing the business is a little more interesting to me than compliance and security.

[0:40:36.2] JM: Tell me about those other things, because I think a lot of the people listening to this are just engineers and we haven't done many shows about program management, but I find it interesting especially when you work at a company like Microsoft and the products touch such a multifarious group of different types of customers. You really have to have all your bases covered when it comes to things like compliance and you can't release a half-baked thing in regards to security. Just tell more about like developing a varietal acumen and all these different things.

[0:41:12.3] AH: The best way to get started is always — Microsoft is such a big company, that there is an expert somewhere in what you are looking for, whether it's in the data platform team, whether it's in Azure, whether it's in Office, whether it's in Windows, there are experts throughout. These are people who have 10+ years' experience in the specific field. The best way to get involved is just going out, reaching out to a lot of them, asking for an hour or two, trying to understand — Hour or two is pretty short amount of time.

[0:41:41.5] JM: That's basically how you navigate in a big company regardless.

[0:41:44.8] AH: Exactly. Exactly, but now it's to learn. It's also good to network so that you'd get some more connections within the company, but it's good to learn and learn a little bit about what people have been working on, understand what they've been working and then use that knowledge to kind of go a little bit further. The truth is, working in almost like a startup within Microsoft, you're finding yourself still interacting with a lot of starter problems where you run into a certain issue, you have to resolve it and you're scrambling to find the resources, understand a problem, scrambling to find the resource to actually solve the problem. This is this is definitely reminiscent of the first year and a half working on the team. After that, it's kind of growing to a bit where we have specializations now. Where there's a lot of different members of the team who specialized in security, who specialize in resource governance to make sure that your database has this strict resource governance and no noisy neighbor problems.

People focus on queries. People focus on the data modeling or supporting the different models that you can just graph, key-value, and documents. With this specialization, it's a little bit deeper now where I'm diving a little deeper into the MongoDB API, which was very interesting just because Mongo has an incredibly rich language. Learning that took a lot of time to figure out the details and understand what makes the differences and kind of working towards that.

[0:43:08.2] JM: I want to talk a little bit more about database engineering stuff, then we'll wrap up. This fundamental question between consistency and availability exists in any kind of distributive system, certainly a globally distributed database, this is going to be the case. I think most of the listeners who are familiar with distributed systems episodes know about the CAP Theorem and there is this trade-off between if you want higher availability, you're probably going to have lower consistency. You might have one user that accesses the database and gets one set of data. Another user accesses the database, they get a different set of data, so you have this inconsistency. That can be okay for some applications.

What's something new that you learned about this trade-off between availability and consistency when you've been working on Cosmos DB?

[0:44:00.4] AH: One of the key pillars of Cosmos DB is this is where kind of the visionary of Dharma Shukla who's the founder of Cosmos DB came in where he found that the typical consistency levels which range is kind of binary, you have a strong and your eventual. There are a couple of others that you could put in between there that really capture different value propositions, so different needs.

As of right now, Cosmos DB supports eventual as well as strong, but we have three other flavors in between which is the consistent prefix session as well as bounded staleness, and if people are already familiar with eventual and strong, I could talk a little bit about the three others in between. Consistent prefix allows you to have a guarantee on the order of write, so that if I do write A, B, C, I will never have the replication across the nodes that go out of order where have C written first and then B, which means the way to think about it is you won't have an invalid state. I won't have a state where C and B and A haven't been committed, and so this is something that's a nice guarantee to have, and that's consistent prefix.

Next, you have your session. Session is a default for Cosmos DB and this is the one that really is used quite frequently because a lot of developers want to see their own right. Which a session allows a user to actually see their own rights. We don't guarantee for anyone else. Anybody else who hits the Cosmos DB database will find themselves, they could have a stale read, but the user is guaranteed to have their own read — They're able to read their own write, which means —

[0:45:30.5] JM: Read your own write. Got it.

[0:45:31.2] AH: Exactly — Which is awesome because if I'm coding against a database, I want to make sure that if I do a read I get to seed myself.

[0:45:36.8] JM: For sure.

[0:45:38.2] AH: The next one is bounded staleness. Bounded staleness is we put a cap on the delay of replication. Basically, I want to make sure that I never did a read that's staler than K prefix in terms of time or operations. I want to make sure that my read is never older than 100 writes, 150 writes, 200 writes. I want to make sure that my read is never older than 20 milliseconds, 300 milliseconds, 400 milliseconds, and this kind of caps off exactly what are able to see. Then you have your strong, it's terribly committed to all the replicas. It eventual where there's no real guarantee on ordering as well as there's real guarantee whether you're going to see a stale read.

This is really pivotal because there's a lot of people would just find that they need different type of consistency levels especially at the global scale to figure out what their application needs. For example, if I have a Facebook app where I'm writing to my comment section, I want to make that I could see my own comment. I want to make sure that if I write a comment, I see it in my news feed. There’s no real guarantee that anybody else who sees the comment needs to see it right now. There's no real guarantee that they can get a stale read that my comment is not there. I want the guarantee that if I write it, I can see it.

With that, you can actually use the session consistency to make sure that the individual who wrote that comment can see their own comments. Eventually, they’ll propagate to all the replicas globally where you’re able to see it across the board and anybody else who’s looking at your wall, you’re able to see exactly the comment.

This is a little bit of the nicest where you're able to have the trade-offs between consistency and availability and you can tune it to whatever you need.

[0:47:17.0] JM: Yeah. I think that’s’ really interesting that the fact that you've got these five consistency models and rather than asking the customer to understand the finer points of a distributed database, you just say, “Okay, here are the different consistency models we can offer you,” and you can give them examples like that comment example, the comments on Facebook. You could also say you don't want your comments to be out of order so maybe you want just the ordering consistency. What was that other —

[0:47:50.5] AH: Exactly. Consistent prefix.

[0:47:52.1] JM: Consistent prefix. You only want consistent prefix. It seems like a sophisticated sales model. Yeah, interesting.

[0:48:02.4] AH: It definitely gives a lot of power to the user so that they can choose. I know a lot of NoSQL and distributed databases have that mentality of you have either or strong or eventual. Now that we have a little bit more of a configurability, now, people can choose exactly what they need. This is a lot of the mentality of what Cosmos DB encompasses, you’re able to choose your own data model, so your own data type so that you can — Or models, so that you can run a graph database key-value column. You can choose what APIs you want to use; the MongoDB, the SQL like, the graph, and as well as the table storage. Then you can also kind of configure your consistency levels as well. This is the extremely flexible global scale low-latency database that Azure provides.

[0:48:49.7] JM: Great. What's in the future of the Cosmos DB?

[0:48:52.6] AH: The future in Cosmos DB is expanding out a lot of the different APIs to make sure that we capture the ones that people are looking for. We'll prioritize depending on all of our user's feedback, what they're looking for, add out more APIs, what people are familiar with. We'll drive the graph support as well as the table storage support to public, public, I guess GA, so general availability for those who don't know. Yeah, that's kind of all I know as of right now.

[0:49:22.2] JM: Great. All right. Well, Andy, thanks for coming on Software Engineering Daily.

[0:49:24.8] AH: Cool. Thank you.

[END OF INTERVIEW]

[0:49:25.8] JM: Thanks to Symphono for sponsoring Software Engineering Daily. Symphono is a custom engineering shop where senior engineers tackle big tech challenges while learning from each other. Check it out at symphono.com/sedaily. That's symphono.com/sedaily.

Thanks again to Symphono for being a sponsor of Software Engineering Daily for almost a year now. Your continued support allows us to deliver this content to the listeners on a regular basis.

[END]