**EPISODE 906**

[INTRODUCTION]

**[00:00:00] JM**: A new software application has simple requirements for a database. The database needs to be written to and read from. The database fulfills simple needs, such as storing user information and providing the application frontend with the necessary data to render a simple webpage of perhaps financial transactions, or blog posts, or rides in a ridesharing company.

As an application becomes successful, the database grows in size. The complexity of queries increases requiring more sophisticated logic in order to maintain performance. New databases need to be added to the overall system as users begin to have demands for advanced features, such as search or analytics.

Overtime, the requirements for a database expand into the need for a data platform. A data platform might include multiple databases, such as a NoSQL database, a relational database, a data warehouse and a search index. The relationships between these different data systems can vary in terms of consistency requirements, latency and scalability.

Andrew Davidson is the director of cloud products at MongoDB. In a previous episode, Andrew discussed the tradeoffs of scaling databases while maintaining high-performance indexing. Andrew returns to the show to discuss the emerging subject of data platform. As a growing number of companies have data requirements beyond that of a simple transactional database, Andrew's work has increasingly involved figuring out the best ways for developers to adapt these transactional systems to providing a wider set of functionality, such as search and analytics.

Full disclosure, MongoDB, where Andrew works, is a sponsor of Software Engineering Daily.

[INTERVIEW]

**[00:02:01] JM**: Andrew Davidson, welcome to Software Engineering Daily.

**[00:02:04] AD**: Thanks so much, Jeff. Good to be back. It's been a while.

**[00:02:07] JM**: Has been a while. In our last episode together, we spent a lot of time talking about the consistency properties of a database, the availability properties. We also spent a lot of time talking about what happens when a database gets big enough to start thinking about ETL jobs, alternative database solutions.

In this episode, I'd like to start on a similar topic. So when most applications start out, they have a single database, and that database has different tables for different purposes. An application usually has to expand at some point beyond that database into some kind of data platform. What are the reasons why a single database application might need to eventually build out a data platform?

**[00:02:56] AD**: That's a great question. I mean, I think there's a couple ways of answering it. I'll kind of go with three different ways. The first is as your application gets more advanced, you're essentially going to want to understand what's going on inside of it. So you're going to want to have those –Essentially, the global, for lack of a better word, the kind of God mode view where you can do analytics around what's happening in your application. Typically, you want to query your database to do that. But the challenge can often be one of not wanting to sort of disrupt your end customer experience while querying your database. In other words, you don't want someone having to wait while their page loads just because you're wanting to see this global view.

So people will often start pulling data out of their main database, their application backend, into more of a data warehouse to do that kind of thing. Now, I have a slightly biased point of view and that I'm the director cloud product at MongoDB. With our cloud platform MongoDB Atlas, what you would do is enable what we call analytics nodes, where essentially you have special dedicated replicas that give you workload isolation.

What you can do is have those heavyweight analytical queries target those special nodes, and they will disrupt the operational database portion of the workload. Therefore, it won't cause that person sitting there waiting for the site to load to have that slowdown experience. So that's great

sort of for the first – That's kind of a quick and easy way to just do reporting out of your application backend. But in practice, that might go pretty far for you. You can push down aggregations to do that, etc. But there might come a time in which you want to do like really heavy way analytical queries into your data that essentially require you to change your schema. In other words, the way your storing your data in your application backend, even with aggregations, might just not be optimized for the kinds of queries that you might traditionally want to run out of a data warehouse. And that's where folks will sometimes do those ETL jobs. They'll pull the data out of their online database, their application backend, and put it into some other system. Their other system might have the data optimized with a better schema, maybe even columnar, etc.

One of the things that we're seeing is a proliferation of kind of there's a lot of long-running kind of legacy data warehouses. There're some modern cloud native data warehouses. But what a lot of people are starting to do is really aimed to store a lot of that data that's kind of interesting to query every so often, but maybe not all the time. They'll store it directly in object storage, for example, in S3. And that's where things start getting interesting, because once the data is in S3, it's incredibly cheap, but it's kind of traditionally been hard to access that data. It's in a bunch of files in buckets. How do you make sense of that? That's where, recently in Atlas, we introduced in public beta, something called the Atlas Data Lake, which essentially allows you to query the data that you have in your S3 directly from MongoDB Atlas as though the data is MongoDB.

Essentially, you define virtual MongoDB collections that you map back to various data formats in your S3 bucket. So it could be JSON, CSV, VSON, AVRO, Parquet, whatever data format you structure your data in, you can query it as though it's MongoDB. And this becomes kind of an interesting use case for those long-running, occasional, or analytical, or maybe more explorative queries that are not worth necessarily running in an online database cluster.

**[00:06:16] JM**: When you say that data in S3 is hard to access, are you talking about the format makes it hard to access and the fact that it's in a remote location, or it's in a separate cloud provider basically, or not necessarily separate cloud provider, but it's in a separate system. It's in S3 and they're operating in Mongo and someplace, and maybe they're doing an occasional export of a file, like some kind of file that aggregates everything in the database. So are you

saying it's the file format, or is it the latency, or is latency not typically a problem with these kinds of reporting jobs that we're talking about?

**[00:06:56] AD**: It's a great question. On the one hand, with these kinds of reporting jobs, latency probably isn't the concern, but that doesn't mean latency doesn't matter, because even when you're trying to make sense of what data you've got and kind of exploring and running those initial queries, latency is always important, even just for the developer who was is writing those queries. But I guess what I meant by the challenged accesses is S3, for lack of better term, is kind of like "dumb storage". It's just the place where you can store a bunch of files. Traditionally, folks would use it more for binary storage, like images, movies, etc. Increasingly, we've seen this proliferation of folks storing lots and lots of text data in there. We've kind of seen folks use S3 for what they might've done with HDFS on-prem increasingly.

So all of a sudden there's this emergence of the data lake with lots and lots of rich text data in S3, but then the challenge becomes how do you create kind of the compute tier that can go and divide and conquer on performing queries into that data? There're various solutions out there, but basically they all have their drawbacks, etc. And until the Atlas Data Lake was launched, there was no solution for doing that that felt fluent to a MongoDB developer. Of course, the MongoDB developer, what they love about MongoDB is that the whole language, the mother MongoDB query language is one that kind of native to the object structure that developers live and breathe, the kind of JSON data model. So bringing that query language that's native to that rich structured JSON data model, or you can query very easily into sub documents, arrays. You can do aggregations on it. Bringing that to this context in S3 kind of unlocks this ability to do fluent querying on data that otherwise was a little bit dark from the perspective of the developer, if that makes sense.

**[00:08:40] JM**: So as we're talking about this reporting, which maybe we're doing it in S3. Maybe we're just doing it in a Mongo database that's a read-only replica. One trend in software engineering is that we have more and more applications where we want to do online processing of this kind of data. We want more rapid aggregations, because maybe we're doing some kind of newsfeed recalculation on the fly. We're building some analytics application where we want to do a large-scale aggregation very quickly.

So many times, these data platforms are becoming more and more online-based systems where you want a data warehousing kind of system that is almost at the application tier. Are you seeing that as a trend among people who get to a certain point with their main Mongo instance that they maybe have been writing things that they also need for analytics workloads and they started to say, “Okay. Maybe I need some kind of multi-model thing or some kind of in-memory system.” How have you seen the growth of those online large data processing systems?

**[00:09:53] AD**: Absolutely. I mean, great question. I think everyone is increasingly wanting to perform aggregations that are relevant to their end customers, for example, or to provide like global views that are customer-facing, and that therefore online real time. So the real time analytics, the ability to perform aggregations against the operational data store, which is certainly something that is pretty standard practice for MongoDB anyway, is totally a common use case. You might just – I don’t know. Let’s say you’re building a banking application. Showing each customer the key financial performance of their positions over the last 60 days, let's say, is totally canonical thing to do.

I think what habits and practices, you kind of show the operational view into stuff that's more recent, typically. And then there's some point at which you kind of no longer feel it makes sense to report. It’s sort of no longer relevant. It's long enough in the past and it’s really that older, colder data that probably moves into some more offline mode, like in S3. But for the recent stuff, absolutely.

In fact, native to the MongoDB Atlas platform, we have something called MongoDB charts, which basically lets you build those beautiful business intelligence dashboards and charts that kind of are native MongoDB data model. In other words, rather than forcing yourself to kind of compress these rich structured document schema down into a bunch of tables to build charts on it, MongoDB charts kind of has that awareness built in to do the unwinding of arrays and all the kind of richness of the Mongo aggregation languages built in.

And the really exciting thing about shards that just came out is that you can now embed shards in your end client experiences. So you can put a MongoDB chart in your application. You can of course have that chart be scoped to the appropriate end clients access. So you could have each customer see a chart from real-time MongoDB data that's easy to embed, looks really

good, but it's going to only show the data associated with that user so that the privacy is preserved. These are the kinds of things that absolutely are kind of up the alley of what you're saying.

But I think there's another angle on what you're saying. And kind of going back to your original question, which is that the other thing that's causing folks to move away from kind of a single database model is the proliferation of microservices. In other words, folks are moving to the model of breaking monolithic applications up into different constituent parts typically when there's different teams involved. And that model usually is a best practice, is a model where you want to have a different database, a different backend per microservice and have that service kind of own the data layer and provide access via APIs to the other services.

So as folks do more and more complex things involving these different services, there just becomes a proliferation of different schemas, different database, essentially, around all to power an end customer experience.

**[00:12:43] JM**: So I wanted to start with this discussion of data platform, because you're working on a lot of different things that fit into the scope of the database platform, the managed database system. And dealing with these high volumes of data kind of use cases is one area, but there's also search applications.

So another large data related search application is search. So if I'm building search on top of a database, I want to have applications, search applications that my users can search. That's a totally different query pattern than typical transactional workloads. What do you need to build on top of a Mongo database to have a robust search system?

**[00:13:30] AD**: Yeah, it's a great question. Basically, the really advanced search applications that are out there, the way they work is they've got this really complex indexes that enable the ability to provide those incredible results that fix the typos for you, that are language aware, that are intelligent about word order or word order missing certain terms, etc. That ability to do kind of true fuzzy search that we're all used to, because we all sit in front of Google so much every day.

Applications increasingly have to deliver that value. It's kind of like, in many ways, the front door into applications is often search. And it's traditionally been very difficult to deliver search into a database negatively, because those indexes necessary to deliver that capability are so intense. Essentially, there're so much data needs to be stored in such a unique structure that you would not want that structure to be on the critical right path from a consistency perspective of your database. That's why the consistency oriented databases, like MongoDB, traditionally not had the search kind of indexes built in, because MongoDB kind of wouldn't want to pay that write penalty effectively to maintain these very heavyweight indexes.

That's where there's this proliferation of an open source solution called Lucene, where everyone started building great search applications on top of Lucene, solutions like Solar and Elasticsearch, that basically enabled you to move your data out of your database, whether it was MongoDB or Oracle or something else into this Lucene-backed system and build these essentially eventually consistent indexes that can be queried.

But the problem for the developer became, one, you had to do a bunch of ETL from your database over to your search engine. Two, at query time, you have to essentially manage from a client perspective access to two different systems. So we figured why don't we figure out how to kind of deliver for MongoDB users in MongoDB Atlas a Lucene-based experience this just native to Atlas and that gives MongoDB developers an experience that basically means they don't have to leave the MongoDB query language.

So we've got in beta right now in MongoDB Atlas the ability to enable full-text search. You can do it essentially at the namespace or collection level in MongoDB. And when you do that, essentially adjacent to your MongoDB database, you're going to have on the backend a Lucene index that gets created that sits right there next to the MongoDB process on the backend of the Atlas cluster. And now within the MongoDB aggregation framework language, you can actually reach out to that search engine right there. It's eventually consistent, but it's really useful for when you want to deliver those queries that frankly were just very difficult to express previously in MongoDB, and it's all going to be write built-in. So there's no ETL any longer. It's just fully automated and it's with the same query syntax. Yeah. I mean, that's why we've done it. I think, increasingly, you can't build an app that doesn't have search, and we heard that from so many of our customers. That's why we built this.

**[00:16:35] JM**: The last conversation we had, there was a lot of discussion around indexing, and there is this tradeoff we explored between the indexing process and having a consistent index. Because every time you have a new entry that's added to your database, if you want to have that new database entry be indexed, the more indexes you have, the more updates to those indexes you need to make when adding an entry. Depending on how consistent those indexes are going to be.

You mentioned that the Lucene indexes is eventually consistent. What kind of frequency of updates to the search index is there and what is that update process look like?

**[00:17:24] AD**: Yeah. I mean, basically, the Lucene index can take advantage of our built-in capability that anyone can leverage by the way in MongoDB, which is change streams. You can follow changes inside of a MongoDB cluster using our capability called change streams. In fact, we have a capability in Atlas called triggers, which essentially are serverless functions that you can power based on those change streams. Maybe you want to push data to another service, etc.

But inside of Atlas, we can use these change streams to have that Lucene index be maintained very easily overtime. But there is sort of an initial build phase that needs to happen before that, the kind of ongoing maintenance can happen.

What it means is, effectively, the index could fall behind in a sense for a very write-intensive workload or a very write-intensive period of time, and that means that when you use a search engine like this, you have to be kind of okay with a model where you might have the search engine route you to a result that might be slightly older now.

So we're still on beta on this functionality. Anyone listening, we'd love your feedback. We'd love to get you involved. You can find the search right there in MongoDB Atlas today. We're still in beta on the syntax and kind of how customers are going to fully engage with this inside the aggregation pipeline, and we'd love to hear from you kind of really all the different ways you'd want to use this and all the scenarios that you'd like fully expressible in that syntax.

 

**[00:18:46] JM**: To talk more about the process of building a managed database platform, I'd like to get your experience from the two years that you've spent. Well, almost two years on this project. What's been the hardest part of building Atlas so far?

**[00:19:05] AD**: That's a great question. MongoDB be Atlas was launched in, I believe, basically over three years ago now, the June of more than three years ago.

**[00:19:14] JM**: So you've been working on it for three years.

**[00:19:15] AD**: Yeah. We've been out there for a while now, which is great, because we're finally at that point where MongoDB Atlas is experiencing this incredible growth at every kind of customer, from the sole developer building that next startup, to the big enterprise financial services and government type customers. It's great to have that full range and just sort of see the full spectrum of requirements across all of those very different types of customers.

Everything we do is kind of based on what we're hearing for our customers, but we always try and find that balance between. It's like I always literally imagine the solo developer trying to build that next great thing. We need to be delighting them as much as we're delighting the developer inside of that bank. We can't over optimize for either one. We just need to find this stuff that works for both of them.

So the biggest challenge to sort of building this whole thing out, I mean, I think part of our challenge has long been we want to deliver a truly global consistent experience for MongoDB that's elastic across the big three public clouds, so that's AWS Azure and Google Cloud. So a big challenge for us is that we need to be complete and total experts running a mission-critical application at incredible scale on all three cloud platforms. I think that's quite rare. Most customers are sort of adapting one primary cloud platform and are contemplating going multi-cloud. Whereas we're very much ahead of the curve there having to have that full expertise for true production mission-critical scale on all three.

I think just kind of keeping up with the incredible velocity of what's being introduced to those platforms, keeping up with the incredible velocity of what customers expect and are trying to do

in the platforms. It definitely is a challenge, but it's also kind of exciting to stay on the pulse of it and to really have that global view into.

**[00:20:59] JM**: You are the director of cloud products.

**[00:21:02] AD**: Correct.

**[00:21:03] JM**: Do you feel like it's mostly a management role? Is it like a software architectural role? What are you doing day-to-day?

**[00:21:10] AD**: That's a great question. I mean, it's a mix. Product management is kind of a unique role and that we typically refer to product management with these two concepts. There's inbound and there's outbound. And inbound is kind of all about staying on the pulse of what customers are saying and doing and feeling about your offering, and translating that back or kind of summarizing that back or providing bridges between that to the team, the architecture team that's ultimately defining – Essentially, the engineering group that's figuring out how to address what those customers need and turn that into something that we can actually action on and prioritize that against other things we can do. So that's the inbound side

I think a key detail is that from a product manager perspective, in a technical product, you want to have that kind of mind meld with the engineers. You certainly want to have credibility with them. But you also don't want to come with sort of, "Here's what we need to do solution mentality." It's more of a, "Here's what we're hearing from customers. Here's what they're trying to do, or here's some challenges they're running into, and then let's brainstorm. Kind of bring out the best of the architects you have. Brainstorm towards how we might solve these issues and then prioritize amongst those." The ones that we think are going to basically benefit the most people the fastest from an investment perspective. That's inbound.

Outbound product management, it's all about, "Okay. We've built something," or we're executing on a roadmap prioritize based on what I was just talking about. Once we have this new functionality or new capabilities we're bringing it to market  and helping our customers use it, it's all about just ensuring that everything that needs to happen for those customers to reach it, to understand it, to successfully leverage it are happening. So that could be anything from

ensuring that your documentation is solid, to ensuring that your customers to have a direct line for feedback, to running beta programs, to everything in between.

Yeah, in a director role, it's a mix of all those things, but you have people reporting to you who are also doing that, because you have a big enough platform that you have to divide and conquer on sort of all the different areas of the stack, essentially.

**[00:23:06] JM**: Let's get into talking about architecture and engineering. Describe the software architecture for MongoDB Atlas to the extent that you can describe it.

**[00:23:16] AD**: Yeah. I mean, it's a great question. So at a high level, MongoDB itself, a typical MongoDB database cluster, is a distributed system. Now, the way we operate MongoDB Atlas is every – Except for, let me put on the side for a moment what we call our shared tier, which is kind of like the starter tier for people just getting started. I'll talk about that in a minute, but first sort of the dedicated clusters, which is $60 a month and above. So pretty low price point to start out with. For dedicated clusters, we basically are deploying from Atlas dedicated MongoDB environments on dedicated cloud instances for each of those clusters, for each of those customers.

On the backend of that, effectively, every customer has one or more VPCs in the cloud provider of their choice. So if you deploy into AWS into a particular region, we're going to build for you a VPC on the backend in that particular AWS region. If you then deploy what we call the M10, kind of the smallest dedicated cluster, 60 bucks a month, we're then given to spin up three compute instances right across the three availability zones in that target region. Then we're going to, basically using our automation framework, kick off a distributed MongoDB cluster across those three nodes. So we need to ensure that essentially those three nodes can reach each other over the network. That's a critical thing. Then we need to ensure that those nodes have the ability to kind of push back to our cloud mothership all the monitoring necessary for us to really run this global system at scale and for everything to be automated.

So, effectively, we have a control plane that is sort of running centrally, not in the context of all of these dedicated customer environments, and that control plane is kind of the place where all of the monitoring and health information and the declarative goal states that the customers have

described that they want. What kind of cluster where? That control plane is where our application runs that really is responsible for sort of delivering all of that globally.

That application, it's written in Java. It basically goes out to all those cloud endpoints, whether it's AWS, Google Cloud, or Azure, and it hits all the necessary APIs and it starts all the necessary instances, etc., to enable this whole concept that I've just described. Once it started all of that, then it kind of passes the job off to our agents that run locally on those boxes on the backend of those customer clusters, and those agents will start the MongoDB environments

Essentially, from there, the customer can get that connection string and be off to the races. But if a customer, for example, were to come in and enable our VPC peering, or leverage our IP white listing capability, then we're going to be on the backend driving those API calls again. Whether it's kicking off a VPC peering connection from their own AWS VPC to the Atlas side VPC, or if it's an IP white list entry, we'll be simply updating those security groups on the firewall on the EC2 instances in that customer's environment. So if that makes sense, these are all of the things that we have to do in order to manage Atlas. Of course, there's a lot more that I'm not going into, and there's quite a bit there as you can imagine.

**[00:26:17] JM**: So Atlas was started before Kubernetes was popular, right? Three years ago, I'm trying to remember when exactly the industry centralized around Kubernetes. I think it was –

**[00:26:28] AD**: Yeah, Kubernetes kind of a emerged as the victor. I would say right around the time we were kind of launching Atlas. I remember the prior year, it was kind of like, "Is it Mesos? Is it Kubernetes? Is it Docker?" Yeah, probably the time in which the consolidation happened.

**[00:26:46] JM**: Container orchestration wars. We had a lot of shows about that. I was like a year and half into Software Engineering Daily, and that was like the most confusing time for me. I was, "Why are there so many of these things? I don't get it. Could somebody explain it?" and nobody could explain it, because they all had their like interests. They're like, "Oh, no! It's us. We're the best one."

**[00:27:04] AD**: Yeah, it was a crazy time. It was challenging for us too, because a lot of folks were trying to manage databases in containers at that time. Obviously, there's a lot of risk to

doing it, because they're mostly for stateless apps originally. But once we saw the kind of Kubernetes really kind of emerge as the standard. Yeah, that kind of made it clearer for everyone.

**[00:27:23] JM**: So what was the original architecture when you were in the midst of this stuff? Did you choose Mesos? Were you on Mesos, or did you even use a container orchestrator?

**[00:27:32] AD**: No. For Atlas, we didn't go down the container path. Just because for stateful applications, like databases, it wasn't – There just wasn't a need to.

**[00:27:39] JM**: Of course. Okay.

**[00:27:40] AD**: Because we're going with ultimately dedicated cloud instances backing our dedicated customer environments. We just didn't need that kind of thing as much. Now, we have, interestingly enough, make MongoDB something that you could can run with a Kubernetes operator using our ops manager. But that's really, in my view, more for like on-prem workloads. If you've got that private cloud on-prem and you're not able to leverage Atlas in the public cloud, you can now run MongoDB in Kubernetes. So we totally built that out. It's just something we don't use in Atlas, because it's really – We don't it. I guess my key guidance would be if you're in a public cloud and you're using Kubernetes, then almost certainly what you're doing is running your app tier in Kubernetes and then you're using databases delivered as a service plugged into them.

So that's certainly the way we see people use Kubernetes with Atlas, is you run Atlas for your database workload. You can have a consistent experience in any cloud, in any region, that way easily, and then you can have your Kubernetes application tier orchestrated for easy scalability of the app tier. And that will just connect to Atlas with a connection string.

In fact, we're in the near term going to release an open service broker API, which is kind of a Kubernetes infrastructure as code tie-in for Atlas. So you'll be able to, essentially, from a Kubernetes context, more or less request an Atlas cluster on-demand and get those bindings back. We're doing the same thing with Terraform, by the way. There's a nice Terraform provider

that a gentleman in the community created, and we're now building an official one as well for Atlas.

So folks, that's kind of a big trend we've seen. The cloud paradigm is just infrastructure as code. Give me a database on-demand. Snap my fingers and there it is declaratively. Whereas on-prem, you can actually go down that plumbing road of building your database into Kubernetes, which we can avoid at the public cloud.

**[00:29:21] JM**: Has Kubernetes made it easier to run MongoDB Atlas itself for you?

**[00:29:27] AD**: That's a great question. We don't run Kubernetes in the backend of MongoDB Atlas today. I think certainly in the future it might be something to look at more and more. But what Kubernetes has made easier for us is it's made it easier for our customers to adapt what they perceive as a multi-cloud or at least a multi-cloud ready architecture. Even if customers today are often still starting single cloud, a lot of our customers really value that sense that they don't feel locked into particular cloud platform. By adapting Kubernetes at the apps tier, there's these great synergies with Atlas at the data tier, where both make it really easy to have a consistent experience if you wanted to eventually move between cloud platforms. Even if you don't actually do the move, just kind of having the sense that you can, having that posture in terms of your relationship with your cloud platforms is something that a lot of our customers really value.

So it's been really interesting to see kind of Google make Kubernetes happen, and it's been really interesting to see this incredible growth of the Google cloud platform, and we see a lot of customers really on all three cloud platforms now going all in on Kubernetes for their app and then just connect into it from Atlas. So that's been a great accelerator for us.

**[00:30:42] JM**: Let's come back to that discussion of different cloud providers. Has the appetite for multi-cloud grown, and can you help me understand why companies want to be multi-cloud? What is the driving desire for multi-cloud?

**[00:31:01] AD**: Sure. Yeah. I mean, I think there're a number of desires. Some folks have come at it from a purely availability perspective. Some of those like the idea that if a true cloud

provider goes down, that they could be spread across multiple. That's something that is a great vision. A lot of folks in practice are not ready or prepared to build a truly multi-cloud architecture. So it's kind of more aspirational.

That sounds –

**[00:31:25] JM**: Wait. Hold on. Let's just pause on that one, because that's something I want to drill into. Is it even rational to do that, to have aspirational architecture like that?

**[00:31:36] AD**: It's funny you say that, because at a starting point it sometimes sounds like, "What are you really trying to achieve here?" If AWS U.S. East One is down, you can kind of chuck it up to basically the whole internet feels down, and therefore what are we really trying to achieve being the only ones who somehow stayed up in that paradigm, or whatever?

But there're a lot of local contexts in which this sort of does make more sense or it becomes more apparent. So, for example, if it's really important to you that you have multi-region availability, it's not so much that you care about multi-cloud availability, simply multi-region. But your country is one in which you need to keep the data in the country. Maybe it's a sovereignty thing, which is very common for a lot of use cases. If you're in a country that a particular cloud provider only has a single region in, which is very common, like maybe in the UK there's only one AWS region. Then all of a sudden you might say, "Look, I want to know there's an easy way for me to have like multi-region availability in the UK," and maybe you would look to the ability to use both the AWS UK region and the Azure and potentially Google Cloud UK regions just to solve that problem.

So if you're kind of like a bank in the UK and all of a sudden it starts making more sense from your perspective, if that makes sense. Because multi-region availability is something that – I think even in all intellectual honesty, most of our customers that I've talked to probably still are single region, probably –I don't know, probably just over half are single region, but a very large and growing minority of them are aspiring to at least be spread across multiple regions. And doing that in a context in which your cloud provider has only one region in your country, that can be limiting.

**[00:33:18] JM**: Well, you're talking about availability in the sense of up time. I think we could also think about availability in the sense that although we haven't seen this at scale, at least not for a while in the major cloud providers. There is the risk of data loss. We don't really know – I can't give you a proof that AWS is architected in such a way that there is zero tail risk of them losing some large number of S3 buckets, like permanently, right? Nobody could give you that proof, except maybe somebody at AWS.

So the idea of being multi-cloud just for the sake of data protection, I think that is a legitimate architectural move to make. I don't know if you have any thoughts on that. Then I also share your appreciation for the data provenance question. I think actually think that's a pretty nice opportunity for Atlas just in terms of being able to provide that data provenance solution. I think that's a really good reason for people to go with a very, very dedicated fleshed out database platform, is this data provenance idea, because that is – I mean, all signs point to that just being an increasing headache with GDPR and just other various national data policies that vary significantly.

**[00:34:44] AD**: Yeah, big time. I think it's one of these things where like GDPR – Great respect to the European regulators that I know obviously some people in our community are frustrated by it. But look, the reality is they're looking out from a privacy perspective for all of us. Setting a bar that hopefully all of us will benefit from. Seeing California follows suit is great. Yeah, we have to make it easy for developers to build modern applications that can adhere to these requirements. And as long as we do that, then the best of both worlds is fine. We can still build great businesses, apps, etc., on the tech side and we all get to benefit of this modern privacy stature.

Yeah, you're probably aware of our MongoDB Atlas global clusters, where in a single connection string, you can read and write from different parts of the cluster in different parts of the world and actually you basically embed the customer's location with a country code into your document schema. When you do that with these global clusters, the document literally goes to the portion of the cluster nearest to that country code and can sort of start small. Maybe you start with like just US and Europe. But then overtime you can add in, like Singapore, you could add Taiwan, you could add Mumbai to reach India. And all of a sudden as long as the cloud

provider has a region there, you can reach and keep data in one continuous cluster that can do global aggregations all with the people's data staying near them.

But, of course, data localities is only part of the equation. I think another key part of this privacy first approach is thinking about how we store our data. Literally, changing what it is that we feel comfortable just writing into a database. Even with all the great security that we have built into Atlas, the whole idea of Atlas is democratize it for app devs so they can move fast and always have that baseline of encryption at rest. TLS encryption of the wire, authentication, firewall. All those things are just always on in Atlas, and that's great, but there's so much more that can be done as developers. The responsibilities on your shoulders to – Even if the data tier is pretty secure, your app has to also be secure.

So one thing that we just launched it our 4.2 release is essentially field level client-side encryption, where you can encrypt certain fields in your documents, in your schema, and they'll be encrypted through your AWS KMS before the document actually goes to Atlas. Now when this happens, you give up some of the query ability. You can't do range queries on those encrypted fields any longer. You can do point queries on them, however.

But the advantage is you're now offloading to your KMS the encryption management that's happening in your application, and you can do all kinds of advanced stuff with this. You could potentially drop keys associated with particular users, which gives you kind of an easier story to erasure, etc., and also you basically are – You're ensuring that the confidentiality is preserved, frankly, specifically to your application. So that basically anyone at a lower level of the stack, including MongoDB, MongoDB Atlas, simply can't decrypt that data that you've put it.

So this this makes sense for kind of the highest data classification level, the stuff that needs to maintain the highest levels of privacy and confidentiality. Then, of course, for other kinds of data, you can use other capabilities in Atlas like to bring your own key for encryption key management or just rely on our built-in encryption at the storage tier, which is fine for many use cases as well.

**[00:38:01] JM**: That example of the data locality. As you were talking about that, I was thinking about that would be a hard system to test. So if you're thinking about something like data locality and you want to test a data locality-based solution for AWS, you've got to set up the right

kinds of measurement systems. You need to set up, I mean, some kind of – I'm not even sure what that testing infrastructure would look like. But with the data locality stuff as an example, maybe could you tell me about how you think about testing and certifying that the database platform is doing the things that you want it to do.

**[00:38:47] AD**: Yeah. I think that make sense. So I'll tell you how the backend of MongoDB Atlas global clusters work. They leverage a capability in MongoDB sharding called zone sharding. And the advantage of that being the case is this is kind of a well-understood part of the open source community MongoDB that you can totally understand what's happening. In other words, there's no magic there. But what we do is this very opinionated form of zone sharding where essentially we map those two letter ISO country codes to a particular zone in the cluster, and you choose which zones you're in, like which regions you're actually in. And then you actually in the Atlas UI as the end user of the global cluster, you can actually – We will, by default, just map those country codes to the nearest zone and you can explicitly override those mappings however you see fit. So it's all sort of transparency how it's working. And then from there, MongoDB sharding handles, kind of ensuring that that data is stored in the right zone within the cluster.

Now, using this in practice for it to really be sensible to really deliver at any customer experience that gives them those sort of CDN style regional latencies for operational data. for that to be possible with MongoDB Atlas, there's a lot you need to do above Atlas, above the global cluster. You need to have a global load balancer. You need to have a global application tier. Otherwise, it's sort of a bit of a moot point. Although perhaps some application, just storing the data at rest in the right locations alone very valuable.

But what we really build this with the assumption that customers would use this to deliver in-region latencies as well as sovereignty for that data. But because it's using sharding, there's sort of no magic there, and it's something that people can understand. People who understand the MongoDB architecture, which obviously there's a lot of documentation out there about how it works.

**[00:40:35] JM**: There are other techniques that you may need to employ to get the most important data into a place where it is most easily accessible. So data locality is obviously one

mechanism. There're also different forms of caching and getting different parts of data warm and leaving other parts of data that are less likely we touched cold. And there're different kinds of storage tiers, you could put in different locations. Do you have any other examples of this warm versus cold data tradeoff that you make in Atlas?

**[00:41:15] AD**: Great question. I mean, at a high-level – So MOngoDB on the backend, the storage engines is called Wired Tiger. Wired Tiger is a very advanced storage engine. It basically does the management of what stays in-memory internal to itself. But you can sort of imagine that it's one of these things were the last used page is what eventually will be evicted from cash in general. It's hard to come up with a better memory management solution than that, frankly. It's obviously a big problem though.

So a lot of folks wonder, “Do I need to use a caching solution adjacent to MongDB?” But of course doing that has a lot of downsides. You sort of need to query again to systems. You have to do the ETL. What we typically find is that most people who assume they need that, that's because their experience comes from a more legacy database offering. Then with MongoDB, for data that’s in-memory, you get caching latencies for that data. If it’ not in-memory, there’ a good chance that you didn't need it as quickly anyway, because it wasn't recently touched.

But there are certain classes of data where maybe you want like a small subsidy of your data to always be snappy to return, even though that data is seldom access. Meaning, it may not always be in-memory, and that's where people can use various strategies to kind of put a background thread that is always touching that and keeping in-memory, etc.

But I think Atlas performance is always excellent, because it is always SSDs on the backend. But really what happens in practice is folks, it's not that they're saying, “I want more performance. It’s that they’re saying, “I want to pay less to have less performance for data that very rarely gets touched.” And that I think brings us full circle to this notion of the Atlas data lake on S3. Increasingly, customers – If your dated that’s older than 60 days has essentially 0.01% access rates, kind of like scrolling down on your Facebook newsfeed. Nobody's going past some level of history there. It's totally reasonable to not want to keep that online on SSDs and pay for that in Atlas.

And that's where we also see another use case online for the data lake, would be to just move that data to S3. And when that customer of yours scrolls far enough down in there newsfeed, effectively, that takes longer to load, and that's something that our users all expect anyway, because they get it. I mean, they're kind of programmed to get that from their Facebook experience.

So giving you that consistent MongoDB experience for data that was once online and has now moved offline into S3 is something that we see as a use case for Atlas data lake. Now, I will note that we don't, today, in Atlas data lake, sort of move the data from an online Atlas cluster to S3 for you. You got to do that yourself. I think it would be interesting to kind of explore that space and get customer feedback around what kinds of patterns would really help make all that easier overtime, and maybe eventually we will do some that for you. I think that would be very likely.

**[00:44:07] JM**: Let's talk a little bit about security. Security is obviously a core component of what somebody would want out of  a database system. What are the features that you've built around security and what have been some of the difficult engineering problems in implementing those security features?

**[00:44:27] AD**: Sure. I mean, there're millions of people out there who will download a piece of software and sort of turn it on and use it, really, taking simply the shortest path to using it. Because that's the case and because we wanted to ensure that no Atlas customer should ever be able to shoot themselves in the foot. We made a very hard decision early on that we're simply not going to allow our customers to disable the core bedrock security capabilities.

For example, you just can't disable authentication. Occasionally, a customer will ask like, "Why? I don't want to use authentication for X,Y and Z reason, or maybe I don't want to use authentication, because I don't want to have to have that even just the time required when you're creating a connection to authenticate. There could be folks who don't want to have to build an app that's comfortable with a little bit of time to do an authentication handshakes." And that's one of these things where you have to – When you're building a cloud database offering. You have to have that hard point of view of just don't give in on that particular one, because the customer will regret it in the end. I think our customers broadly do in the end appreciate that.

The same goes with TLS. I mean, authentication is fairly easy to enable in a database if someone is self-managing. Whereas TLS, where you started to do TLS certificate managements and all the rest. That's something that self-manage MongoDB users really do struggle with, and that's just not an easy one. You have to be domain expert. There are many easy ways to frankly do it wrong. Make a mistake.

So in Atlas, by essentially using TLS, require TLS, ultimately having the certificates be coming from the trusted certificate authority that can be validated based on the standard bundle on any operating system. That again just democratizes having ironclad encryption over the wire. Maybe this sounds obvious to everyone, but not everyone even knows what TLS is. TLS is basically the modern acronym for what we used to call SSL, the standard changed transport layer security. TLS is kind of the new nomenclature for using new protocols.

Sometimes people will still refer to it as SSL. But, basically, it's the thing in your web browser that says HTTPS. But, for us, it's at the data tier. So you shouldn't use a database that doesn't use that, and obviously with Atlas, we just require it and make it easy for folks. I think that's a huge democratizing capability.

Then with encryption at rest with optional layers above, like I mentioned, you can bring your own key optionally for that, or you can optionally now encrypted before the data comes into the database. That's another key one. Then there's the network level security. With Atlas, by default, the firewall doesn't allow any network access in, and you have to explicitly choose the white list particular IPs or to white list private IPs and leveraged peering. Building all of that and making it easy to set up so the customers don't have to figure it out on their own I think is just a game changer for application developer security.

**[00:47:16] JM**: All right, well let's begin to wind down the conversation. What will people want out of a database platform in five years?

**[00:47:23] AD**: I think in five years, folks will expect that they can just basically turn on the database, connect to a connections string an auto-magically everything just works. You don't have to think about sizing. You don't have to think about indexing. You're still going to probably have to have some level of thought around your schema, but the hope would be that you kind of

throw a schema in there and then you throw a workload pattern at that and the database will do whatever it can its darndest to automatically size and scale and provision and index so that you basically have as good of a performance experience as you can, at least for the amount you're willing to pay for it.

I think kind of we're seeing more and more of that. We're seeing more auto scaling capabilities. Atlas has various forms of auto scaling. We're seeing more index guidance. Atlas will do index suggestions based on the query workloads you've seen. But I think we're going to see a lot more of that, and customers are just going to basically say, "I want to have no cognitive overload, and I just write an app and the database just handles the rest."

**[00:48:24] JM**: Final question. I've been doing some shows on I guess what you'd call low-code tools. So there's this growing number of a low-code tools that engineers could use, non-engineers could use. There are things that look like spreadsheets. There are things that just look like drag-and-drop tools. Just various forms of whatever is this low-code trend.

There's also obviously a growing number of APIs that do complex things. We've got the Twilios, and the Stripes and all the other things in the API economy. Are we going towards a place where your backend is entirely just a little bit of system that interfaces with APIs and low-code tools and maybe have a few like functions as a service? But are we seeing the disappearance of backend software?

**[00:49:23] AD**: I think that's an interesting question. I think we are seeing more software, more data, more customers expecting digital experiences than ever before, and as a result, all of this is growing superfast. So everything is growing, even backend software is growing while we're introducing growing low-code front-end experiences.

So I think this is a new space, because, basically, the expectation is that people who are not software developers should be able to build business processes more so all the time. So that's what low code is all about in my view, but there's still so much sophistication that needs to go into whether you're building one of those platforms. The people who build those platforms, they need to have a database running on – We see many of them building them on top of Atlas in fact, but if you're building a non-low-code app, a.k.a. like true brand defining app for your

customer, for your company, true differentiator for your business, then you're going to have to really think about it end-to-end. But, yeah, I think we're seeing a shift towards, "I want to offload anything that's not differentiating. I shouldn't have to rebuild an SMS sending solution. I'm going to use Twilio for that. I shouldn't have to build a PCI-ready credit card processing solution. I'm going to use Stripe for that. And I shouldn't have to build a massive database management capability in the house. I can use MongoDB Atlas for that," etc.

We're absolutely seeing everyone move to higher level of abstractions including business types who are moving from spreadsheets into these low-code solutions. But I think developers, there're only more and more developers as well and they won't be using the low-code solutions. They'll be building those customer critical – Because developers are so expensive. They're such a key resource. They're going to be focused on the most brand-defining, business-critical experiences. If they're not, then there's probably something problematic in that organization.

**[00:51:09] JM**: Andrew Davidson, thank you for giving me your thoughts on something fairly unrelated to database. But I'm sure that impacts your work. You get to know these trends and think about how it –

**[00:51:18] AD**: Oh, no. All the time we think about this, because the amazing thing about being at the data tier is that we have a global visibility into what everyone's doing everywhere. The whole market is using general-purpose databases like MongoDB for everything you can imagine, from IoT, to banks, to dating apps, to some of the largest games. So, yeah, it's great to kind of get that exposure and to see the trends. Yeah, I appreciate the questions.

**[00:51:40] JM**: All right. Andrew, thanks for coming on the show.

**[00:51:42] AD**: Thank you, Jeff. Have a great rest of your day there.

[END]