# Everything You Need to Know About Sharding Presentation Transcript

View the recording of the presentation.

**Transcript**

I'm Dylan Tong, Senior Solution Architect at MongoDB and I'll be your host. Just to introduce myself briefly, I'm based in Palo Alto; my role in MongoDB involves working with customers on the field with different startups, enterprises, architect and exchange apps on MongoDB. Not surprisingly, sharding is frequently involved and I'm very excited to be presenting this talk. During this Webinar, if you have any questions please type them into the question tab in the chat window and I have my colleagues with me' Seth Rothman and Roger Dicaffa who will be answering those questions during the course of the webinar. As well as time permitting I will also answer the questions at the end of the presentation.

So let's take a look at the agenda for today. We're going to start off with an over view, talk about what is sharding, why and what should sharding be used for and then, I'm going to go take you guys through a process of building your first shard app cluster. I'll take you through the steps, the high level, some other things that you should do to ensure successful sharding. Finally we'll end things off with Q&A. Note that in this webinar I'm not going to require somebody with prior knowledge of sharding so for those who have some experience, thank you for your patience.

So what is sharding? Sharding is a means of partitioning data across your servers to enable several things. One is scale, used by modern apps to support massive workloads and data volumes, the other thing is geo-locality to support geographically distributed deployments for optimal user experience for customers across vast geographies. Also hardware optimizations, to get the best of both worlds in terms of performance versus cost and last but not least, to lower recovery times, to make your recovery times objectives feasible.

across these shards that reside across many physical servers.

So here is architecture diagram, we'll see this several times during the presentation. But just to give you an idea, this is a shard architecture in MongoDB so down here these are different shards. There's different nodes through familiar replica sets, primary secondary. And typically these nodes and these shards reside across many servers. Data is distributed across these shards and the loads distributed across the shards for you to scale up. And we'll definitely revisit this diagram later in the presentation.

And there's also… other key points is that MongoDB provides three types of sharding strategies which we'll be talking about; range, hash and tag-aware. Let's talk about these types starting with range sharding. So let's use an example here. Imagine I have a collection of data from sensor devices and I want to choose… I'm going to shard this data, partition this data based on some device id. So with range based sharding as the name applies, the shard key defines a range of values and then these values are then segmented sequentially into chunks. So if I had for example 5,000 devices, my chunks might look something like this. And if you're wondering if composite keys are supported, the answer is yes.

The other type of sharding as I mentioned before is hash sharding and you can think of hash sharding as a subset of range sharding. In MongoDB when you create a hash shard key, a MD5 hash gets applied on the key attributes. And ultimately this ensures that the data is randomly distributed. But as a result, the data is once again partitioned across MB5 hash range like as you can see down here. Third sharding strategy is tag-aware sharding, so the way you can think of tag aware sharding is it kind of adds another layer on top of range based sharding. So subsets of shards can be tagged and these tags can be assigned a sub range of a shard key.

So to use an example. Imagine I have lots of data that belongs to users that reside in 100 different regions within some country. And my goal here is to group up the data in two different zones. So I might say that the first 50 regions are considered party of the West zone and the rest is the East. I can define tag sub ranges something like this. And once I've done this, imagine I had something like four shards and I can tag these shards like so, so maybe shard one and two is West, three four in East, so on. And what happens here is all the data belonged to the West zone will reside in the first two shards and the data from the East zone in the other two.

So that's the different sharding strategies and now let's talk a little bit about how you apply these strategies to achieve geo-locality, hardware optimizations and lower recovery times. So you see this table in front of you trying to describe some minimal sharding strategy that's involved to achieve these different use cases around sharding. So keep in mind these are the minimal strategies, so for instance if your goal is to scale, you're going to use range or hash but it doesn't mean that you can't use tag-aware sharding in conjunction for other reasons.

the world's bestselling sports game franchises. And they have user and game data for millions of players stored within MongoDB, within the shard cluster.

Foursquare, a long time user of MongoDB it's a popular social mobile app with 50 million people will applied and MongoDB is their main database. And these starting to support foreign do check in and to support hundreds of thousands of ops per second. Other interesting things is HL, this is in the financial space. They're a quantitative investment manager based in London and Hong Kong. And they replaced a relational tick database with MongoDB to support 250 million ticks per second. And in the process, lower 40X lower costs. Now there's also a great webinar that they presented on that use case another interesting kind of impressive kind of [Indiscernible] [0:08:14] and performance story on shard cluster on MongoDB cluster.

As well as Car Flash another car dimension, they have a vehicle history database and it currently contains 13 billion documents before that's replicated across multiple data centers. So let's start talking about how do you achieve this type of scale on MongoDB. Give context to some people that are new to sharding, you must first understand how the how the architecture of a small MongoDB deployment looks like.

So this is a small deployment, this is just a replica set and for those who don't know, on a replica set you add your application, it talks to MongoDB through a driver. And what happens is the rights go to a primary and then that's replicated across to the secondaries. So with this deployment, it provides they high available HA but it's not scalable. So it has limitations with regards to rights. It's limited at the capacity of the primary, although rights go to that… this one node here and the reads it kind of depends. It depends on your consistency requirements, you need immediate consistency, it reads go to primary and once again your [Indiscernible] [0:09:31] by capacity primary node. If you're still okay with eventual consistency, then you're limited by the available replica set members.

But ultimately this is for HA and this is not for scaling. The scaling in MongoDB the primary mechanism is sharding. So you see here once again this is the diagram that we saw before. This is the shard architecture so let's talk about this is a little more detail. When you kind of look at the features in MongoDB and the shard architecture, we talk about the shard key and how you partition the data. Well, one of the things that makes sharding easy in MongoDB is just auto balancing capabilities. So what you saw there was taking this partition into chunks and MongoDB actually has the ability to auto balance and redistribute this data equally across the shards, that's important.

Another key component is that there's a query routing layer in the system for those who are familiar with it, the MongoS. And the idea here is that data based operations are transparently routed across the cluster through this routing proxy process. This is just software. Ultimately what this does is, it decouples the logic, it makes the development and operational on its own scaling simplistic. Ops can basically add charts, add capacity, scale up the system without any app dependencies and ultimately dev involvement.

compelling. So if we kind of look back at the traditional monolithic approach in some relational database, the typical kind of strategy is to scale up. So for high performance systems you're going to go for maybe a expensive database appliance and where eventually you're going to pay a very high price in terms of high capacity per dollar ratio. And at some point you're going to hit also a scale limit. And quite often to kind of get past that limit, you're going to have to come up with some clever workarounds or you're going to go through a complex or dev costly application that will sharding strategy.

That's obviously not an ideal route and one of the reasons why we see a lot of new databases supporting horizontal scalability capabilities like MongoDB. So when you take a scale-out approach, the idea now is that we can through sharding scale out to hundreds of thousands of services to achieve the scalability limits that are demanded by modern apps. And also just as important, each of these servers can then focus in optimizing the capacity per dollar. And so you get the best of both worlds on the cheaper scale limits, but also on infrastructure.

And the other interesting thing is you also have some elasticity. So you could scale down and reallocate resources if necessary. A good example would be if you're running some sort of games services and a lot of us must know that a game will over time lose popularity and ideally you would be able to scale down the system re-user resources as the game fades over time. Now if you have like a big appliance, most cases it's not going to be practical to repurpose it.

So that's kind of sharding for scale. The other usage that's very popular is sharding for geo-locality. Definitely very popular among many well-known consumer services and mobile apps. Adobe Cloud Services is a notable one that I'll mention. So the idea behind geo-locality is important from the perspective of ensuring quality user experience. The goal is to lower network latency for systems to operate across stack geographies. So to get some context, the network latency from the west to east Coast is typically around 80 milliseconds and that varies depending on the network and other things.

And you can see here from these statistics why that matters. From a… when you put things into perspective the importance of latency and its effect on user experience and subsequent user behavior is significant. The idea is for these large geographical services geo-locality I can lead to local reads and writes and we'll talk about that a little more. So let's take a look at an example, understand how tag-aware sharding can be architected to support a basically multi-active data centers.

So to start things off, we'll look at the simple scenario where we just have a replica set. And you might know that you can take a replica set and can deploy it across multiple data centers. So we got two datacenters West and East Coast. We've got a couple of clients on the West Coast; we've got clients on the East Coast. And you might know that doing local queries, local writes through the data center a reach is possible through things like the re preference feature like Nearest.

So we have a user's collection of all user profiles and then we can then shard that data on something like the user ID and some sort of region code. So I might have a tag West East and once again let's take that scenario where imagine I have like a 100 regions. Where the first 50 can belong to the West and the other to the East. So I'll kind of specify those tag ranges. Now once I've done that imagine I create three shards it might look something like this. The first shard is West and the other two East, this is one possible configuration.

Now with this… now with tag-aware sharding what you can then do is you got the primaries located in the East or tagged East and the East Coast users' user profiles are now basically… you can see them as being pinned to the shards on the East Coast. You can do basically updates to primaries on the East Coast and likewise the same thing on the West Coast. And same thing with the queries you can continue to do local queries, just like before the West Coast user can query the primary right here, but also the East Coast now has primaries in the East Coast data center.

It's available and the East Coast data center can query East Coast nodes. And both the West and East Coast clients, if for some reason the application desires to allow the West Coast users to query or access the East Coast user profiles, then you could do that by acquiring the secondaries that are from the East Coast shards that reside in the West Coast or vice versa; the West and the East. That's how you would support a multi-active data deployment with local reads and writes.

So let's take a look at the other scenario here. Let's talk about how we can optimize latency and cost through sharding. But before I do that, let's put that into a little bit of context. So we have some tables right here that kind of gives you the idea of the different types of storages and how they're different in terms of speed and cost. And as you can see here, between RAM, SSD and hard drives that we've got magnitudes of difference, both in speed as well as in costs. And with sharding, for example you can you can do like tier stores type architecture which will make it possible to get the best of latency versus cost ratio within the single shard cluster. And we'll talk about this a little more, but this table basically gives you an idea why it's so critical because there's such a big differences across these [Indiscernible] [0:18:39].

So let's use an example to understand how we can use sharding to optimize hardware. So we got a use case here. Imagine we have sets of data collected from millions of devices and this app here is… and use this data for real; time decision automation, real time monitoring historical reporting, so maybe predictive maintenance type application. There's two types of data. One data is metadata and this metadata contains information, that information that requires fast look up to drive real time decisions. In this case, latency is very low, so low latency.

And we have relatively a small data set and then we have time series data and it's a mix. We have recent data that has pretty low latency requirements. The data set is fair, in a 10 terabyte, nearing the 10 terabyte range and it's probably going to be used for some monitoring or real time reporting. And then there's a lot of other

requirements, as well as latency requirements. Let's see what we can do with this use case. How can we build like for example a tier storage with sharding? So let's look at these two collections and we got a meta collection, and let's say we might want to shard it by device ID. And then we have our metrics collection, there's time series data maybe all user shard key like device ID and time stamp.

So let's imagine now that I have a variety of servers, a mix of servers, some of high memory ratio, really fast cores. Another with kind of medium memory ratios, but what's nice it's got SSDs on those and we've got some storage optimized ones. A low memory ratio because we just got a lot of storage, servers and inner storage are hard disks.

So the idea here is that I can use tag-aware sharding, I could tag these according to the profiles. For example, the high memory, I might tag them as cache because this server is ideal for ultra-low latency with heavy workloads. The SSDs is flash and then these storage optimized ones along with tag is archive. I can do some filling. And once again I can specify my tags and my tag ranges, so as you can see here on my pin, my meta collection to the cache subset of shards and then I can have the time series data across the flash and archives shard. And in this case…

So the idea here is the data that is 90 within 90 days will reside in the flash and then beyond 90 days will reside in a archive. And if I wanted to… and what I can later do is, I can periodically adjust the ranges here and then have the balancer move the data from the flash storage to the archive storage as I change these ranges. This is the way I can tag it all… the archive type or tiered storage architecture and kind of get the best of both worlds across these different data.

So last but not least, let's take a look at the final application n of sharding and that's the lower restoration times. So let's use the example and understand how this works. So imagine I have an application bug that causes a lot of corruption, the data, the database needs to be rolled back to some previous point in time. So in this scenario imagine I'm using MMS and I've got a total of 100 terabytes of Snapshot storage.

So imagine the case where I have… let's say 10 shards, so then I would expect to have roughly 10 terabytes of snapshots per shard and ideas I need to possibly transfer and build these snapshots and transfer them in parallel. Now, imagine a scenario where I had more shards, 100 shards and I would get expect snapshot say roughly one terabyte. So the idea is then I can transfer these snapshots in parallel and potentially also generate… and potentially recover potentially 10 times faster. So that's the idea. More parallelism, no smaller shards, I can recover it faster.

Let's get on to the next part of the presentation and what we want to do here is, I want to walk you guys through some key steps of building and deploying your sharded cluster here. So we're going to go through the life cycle of sharding. So we'll start off by kind of creating an imaginary scenario. Starting with the product definition, here's our example use case. Imagine I want to build this predictive maintenance platform.

ingest the data, modify the data etcetera. There might be a kind of workflow rules engine in there where you can apply workflows and business rules. Publish subscribe to notifications and etcetera.

So the first question that comes to mind after I define the product is… the obvious question is do I need to shard? So sharding is something that definitely should be evaluated early on in general if you determine that sharding is needed within, let's say six to 12 months after deployment and the system should be designed with sharding in mind. So let's go through our example use case.

So do I need to shard? So we kind of look at thing from the scale standpoint, I think we can unequivocally say yes. When you just consider a few dimensions like throughput, I've got data from millions of sensors, updated in real time. I have some very low latency requirements that may require a lot of memory, high performance and I'm collecting a lot of data. No hints of sensors, very frequent, one terabyte of data collected and I want to retain let's say five years of data because it's [Indiscernible] [0:26:03] historical analysis interesting.

There's a lot of different dimensions which will require me to scale out, so yes sharding is very likely. And of course there's other things that I may want to consider, for example requirements on geo-locality or with restoration times, with optimizations opportunities and things like that. Okay let's move on.

So we decide that yeah, we want to shard. So the next step is critical and it's selecting a shard key. So try to keep in mind a few things why it's critical, the shard key determines how effective sharding will be. Without a good shard key, you will not get near in your scaling. Also the shard key attributes are immutable, so you need to keep that in mind and while you create a bad shard key, it's not the end of the world, re-sharding is non-trivial. So you want to get it right if possible the first time, because if you do want to change it, it does require repartitioning the data.

So the question is how do I select a good shard key? I want us to talk about that right now. A good shard key… there's kind of five properties that we're going to take a look at. So, if you need to remember just three, the ones in green are the most important. So let's consider some kind of the keys to understand what these five properties mean. So the first thing we'll take a look at is cardinality. So kind of going back to our use case, the sensor data, it's going to be deploys across a whole bunch of data centers. Well I'm thinking now it's going to deploy across a whole bunch of data centers, maybe a data center would be a good shard key.

Well actually that's a bad idea. While it's a viable shard tag, it's not a good shard key, the fact is poor cardinality. Let's say I had five data centers, it means that my data can at best partitioned into five chunks and meaning to the extent that I can scale out. You can create some jumbo chunks that cannot be split and that's a big problem. So, all right, let's use timestamps, a lot of my data has time series. So that's great, time stamps does have high marks on cardinality, however, it doesn't do well on the write distribution property.

What if I pass this up… we would call the hash shard key providing the passing it to a to a MD5 providing random distribution. Will this work? Absolutely. So I can hash the timestamp that solves the problem now. The hash shard key now randomly distributes my data or my writes across all the shards, great. But, the strategy falls short from the perspective of query isolation or target queries.

So unless your main access pattern is very simple, like a simple key value access pattern in your queries, a strategy like this will quite often lead to kind of scattered data queries. And these scattered data queries, the idea is that you're going to query not just one or two shards at a time per query, every query is basically querying all the shards and there's a lot of overhead. You have to use what we call scattered data queries. And this inhibits kind of linear scale, so you want to avoid this.

So in this case, queries in the system… if you choose a shard key like this will most likely involve range based queries. more than just simple key value. And likely it's going to be in context of the sensors, devices and in this case. the shard key that sensed the data for a single sensor is going to be lightly randomly distributed across a cluster. So when you retrieve that data, it's multiplied it's going to be scattered data so that's that.

You can fix that pretty easily you say, "Okay that's… you see now why I don't…" most of my queries are in complex devices and let's use device ID instead. So, assuming the majority of the queries are in context of a particular device them we expect that these queries will be targeted. So as a result, each of the queries are going to be targeted ideally to one shard or a few shards, and then they're distributed and well balanced, low overhead for merging and so on. So that's great that would be a good key.

And when you kind of compare like hash on timestamps and hash on device ID, turns out it's also good from the property or standpoint reliability. So what reliability looks at is kind of effect that let's say some fail over or loss availability event how that affects the rest of the system. So let's consider like the hash shard key on a time stamp. Let's say a node on the shard one went down, well what will happen among you guys? Do you know there's going to be a fail over and there's going to be a short loss availability during the fail over time.

Now the question is when that happens, do all or just some of the vices get affected? Now hash time stamp is the potential of all, whereas if you had a hash device ID, it's only some because the device that's either basically localized or a subset of the shard. So when one shard goes down, only some of the devices actually become unavailable during that fail over time. That's reliability.

Now it seems like wow this hash shard key is perfect and not quite. So where it fails in this particular case is the shard key also requires a local hash index and hash shard keys don't have good index locality. So in addition to use… when you use like in this case, when you use a hash device ID, each of the shards also needs a local index for the hash device ID. And if you might know that's well okay, if it's random and when I use this B-tree and next then have an equal or random chance of basically accessing any node in the index.

just because we need more RAM to manage these indexes with performance. Now could we use less RAM? Yes, it's possible if you had good index or better locality. Better locality in terms of how the indexes are accessed. So the idea is that there's certain indexes for stuff with the write balance index, for instance where you have a time stamp type data, maybe device ID and timestamp or just timestamp user write balance X is where quite likely you're going to be accessing for example the most recent data and you're going to be accessing the data, the range of data that's… write balance ideas some data in this case kind of visualize it to the right here and as well as all the nodes in the part of the tree that is due to write up the index. And basically you have better locality in indexes and in theory you don't really need necessarily to have the rest of the indexes in memory because you don't really access them. So in effect, there could be a little more efficient in terms of the resource requirements of the indexes, so that's the idea here.

Now if I didn't use the hash index and I didn't use the hash index and I did something like have a monotopic increasing component like a timestamp, like a composite shard key here, I can get some better locality. It turns out quite commonly if you have a shard key that has kind of a leading random component and a trailing monotopic increasing component, that this yields good results across all five properties.

So that's great. We've gone through how to select a good shard key here and we've now started development. We're staring building things. We have something prototyped and at some point we're going to start going through the cycle of performance testing. And this is kind of the stage where people start hitting their first major pitfalls. One common thing is someone's built a sharded cluster and they're ready to test their sharding strategy. And then the first observation is, "Wow there's a massive performance degradation."

The moment maybe initially they were just testing on a single replica set, but the moment they switched to sharded cluster, like wow, the performance just completely crashed. I'm writing not one server but I'm writing against a dozen and I'm getting worse performance than at one replica set and you're wondering why. Well, one common reason is due to not pre-splitting your data. So as a result, when people are doing their initial data load all the writes are actually going to one shard, one physical server.

And what happens is that initially there's only one chunk and the data's just going through that one server, the chunk rose and what MongoDB does is it dynamically splits and creates chunks on the fly, as it kind of realizes the data distribution. And once there's a certain number of chunks that's created then balancer kicks in and then migration happen. And migrations are expensive because you're moving data between servers. So you're in a situation where you're not using all the servers, you're just writing data, data is dynamically split and these expense from migrations are going on.

That's why sharding isn't giving you the scale-out that you're expecting. To avoid this, the best practice is to do some pre-splitting, so it's a simple, very simple in the case of a hash shard key. If you're using a hash shard key all you really need to do is basically when you create the hash shard key with a specified parameter numinitial

your own pre-split script that you run beforehand to create your chunks initially, and have those distributed across the cluster before you do your initial write load. But if you do these things, then your chunks will be created ahead of time, they'll be balanced ahead of time and when you start writing data to your cluster, they'll be distributed according to your sharding strategy.

Another thing as well is the query router, the MongoS; ideally you should run them m on the app server. You can run them anywhere you want but by running them on the app server, you can avoid extra network latency, extra network pops. So those are some good tips around sharding. So once we're starting performance testing and then we're getting to the pre-production phase, this is the time where you may have done this already. But if you haven't, you definitely should start thinking about capacity planning, once in sizing and the capacity planning perspective.

So let's try and define these two concepts. So when I'm talking about sizing here the idea is you want to understand what are the total resources required for your initial deployment? You need to answer the question what is the ideal hardware specs and the number of shards necessary?

Actually planning takes one step further and the idea is you want to be able to create a model to scale MongoDB for your specific application. And it starts… to answer the question how do I determine when more shards need to be added and how much capacity do I gain from adding these shards? And it provides kind of a long term methodology of scaling your system. So to answer the question how many servers around sizing or capacity planning, in three panel this kind of two strategies and there's pros and cons. One is you can go to for domain expert and the benefits of that, obviously it's a low level effort and it's something that you can do early in a project because quite often you need to do sizing before you even do development network.

So you want to be able to do this without any… what we were talking about, empirical testing or prototyping. So obviously this accuracy does range from high to low and it's inversely related to the complexity of the application, a simple app, something that a domain expert is familiar with and they'll be able to draw analogies based on their experience. But as it gets super complex, they're really… you've got to go for other strategies with empirical testing. You need prototyping, you need to run a test on the hardware, get an idea. And this is the most accurate method, and obviously the level effort is much higher. But the problem here sometimes it's not feasible if you need to do an early project analysis.

Ideally this is something that you should always at least be able to do because you should do performance testing at some point. A month ago this business [Indiscernible] [0:41:27] in great detail, I just want to give you an idea of what a domain expert, the processes that we go through. In general if this is a [Indiscernible] [0:41:34] you need to do sizing or some sort of in a basic part analysis. Early on, you can contact us, talk to us a little more about maybe engaging a solution architect would be a good way to deal with those requirements.

operations, the throughput, the latency, idea of the working set because those are the kind of things that domain experts we need to [Indiscernible] [0:42:28].

Have that information, the idea is if you're a domain expertise then you can start to estimate the total require resources. So when you look at things from a resource standpoint is really five dimensions that we'll look at. And just to explain things in brief, because definitely it's a detail expression. So from a RAM standpoint, what the expert typically will look at is, they have some approximation of the working set and the indexes. And that's kind of usually the base line for RAM. They'll adjust it basically based on latency or cost requirements. For very large clusters, also account for connection pooling, overhead and thread pool which is one megabyte per active thread.

IOPs is a bit more challenging and it does often require kind of using experience through other systems, sometimes it requires a bit of spot testing. One writes estimate strategies kind of just looking at the pick like throughput. And then you can do in terms of the read apps requirements you can do an estimate ratios, understanding the workloads, assume random IO. You should account for application, journal and log, but just a few things. Clearly this is one of the particular the message that is more difficult to estimate. Quite often some level of prototyping is ideal.

The other things are easier, like from the standpoint of storage. If you know the throughput, you know the document index sizes, you can then account for fragmentation, things like this, estimate storages, the card. The CPU isn't usually missing, because you very rarely is it the bottleneck and you can just make due what CPU is available on the commodity so as you end up choosing to use something that you normally need to look at because ultimately when you're looking at sizing, you're really choosing or identifying one of these dimensions that really drive the size of the cluster.

And the CPU is usually isn't one of them, usually it might be for latency RAM, maybe it's IOPs or maybe storage. The thing with network, normally this is pretty simple to estimate if you have approximations for throughput and document size. Basically this is how the domain expert as you kind of approximate collection need information, he can come over and that's the process that you can take in. As you gain experience that might be the process that can take for your early on in your MongoDB projects.

Another approach is sizing or capacity planning by empirical testing. This is definitely the ideal approach for accuracy. The good news is in a best practice world, you're definitely doing performance testing and this is something that you'll definitely will do anyways or you can kind of taking back on performance testing later in your project cycle. And for the empirical testing or performance testing, you can basically do sizing and capacity planning at the same time.

Now this topic alone is a webinar in itself and what I would do is basically reference to you from past webinars. But there's just definitely a high level of strategy involved and some of these webinars you're going to a lot of

you've built something amazing. It's naturally scalable but the big question now is, how do I manage this critical, business critical path? How do I manage hundreds of thousands of nodes?

So one of the important… one of the key tools to help you do this would be like using the database management tools out there like MMS. So we have a MongoDB Management Service and it provides things like automation, automated cluster provisioning, automate daily operational tasks like downtime upgrades. Imagine you got hundreds, thousands of nodes, how are you going to patch up that server? How are you going to patch up the releases that we make available every once in three months. And you've got centralized configuration management. Other important things is how do I get visibility into that mass of cluster.

So monitoring is going to be a key component available, MMS and do real-time monitoring and visualization of cluster health, alerting and things like that is going to be critical. And of course backup, you need a way to backup this sharded cluster. Especially a different piece from backing up a monolithic system. Do you have a system that can automate 25 snapshotting of a sharded cluster and also provide a way to do a point in time restoration for sharded clusters?

These are things that are supported by MMS, which is really key for you to operate a system at scale. So let's talk about MMS a little bit to give you an idea of how you're going to use it to help you in production. The first thing we'll look at is MMS Automation. And the ID here has got MMS, it's got tons of three different pillars. This management tool is actually available in two flavors. You can run it On-Prem, quite common for people who have requirements for On-Prem as backup. We can go for a hybrid of [Indiscernible] [0:48:25] like the SaaS, so all these ones are also available hosted.

So let's look at how automation works. So the ideas you've got server resources anywhere, public cloud is in your data center and you've got agents to install the across the server resources which you want to allow for provisioning. You can go into a web portal, you describe your cluster, so it's different from going into the Mongo shelf for those of you familiar with it or tweaking of [Indiscernible] [0:48:52] which really isn't scalable from an operation standpoint. You can just go on a GUI, configure things and then these agents will fall home, realize, "Hey these are instructions I need to build a cluster," and then they'll build up a cluster for your process server resources.

So ultimately, this is going to be a way for you to deploy and provision and then kind of on a day to day basis manage these clusters across in the public cloud, your data center, even your laptop from a development standpoint. Actually let's jump into the product because I want to show you this a bit, I think it's really key from… For those who aren't familiar this is MMS, this is the web portal. This is the host diversion. So what you see here is I got some hardware resources here that I've provisioned and they're under MMS control.

So if I jump in here, you can already see that I have a sharded cluster, three shards, three member replica sets that's under Sharding control that I deploy through MMS, it's also under monitoring controls. I can jump in here

my predictive maintenance question, of how predictive.

Then there's like diversion on MongoDB I'm going to run a… and then I have some servers with your [Indiscernible] [0:50:50] our prefix webinar and everything else is self explanatory. Sso I wanted to put them on this port ranges and I can enter them so and then I can then very easily just configure. I'm going to ask this and I can fix servers so I'm in the replica [Indiscernible] [0:51:17] or shards. I want three shards and then I can configure the replica set here, I just want to have two data baring those. An arbiter I can configure that very easily.

Then I'll call this my predictive, my predictive shards so I know what I'm doing. And there's a lot advanced options as well, I can resize OP logs through here and various configurations if I wanted to. So I'm going to apply that and once I've done this you can see it's qued up for change and I can jump into the server view, basically I can see this is the clusters and we deploy it across these three physical servers. This is where my existing cluster resides and if I wanted to as well I can kind of shift things around. So maybe I want the Mongo as actually want [Indiscernible] [0:52:03] my webinar or launch server so I can drag and drop that here.

So that's something I can do. Other things I can do like I say not just to provision service but day to day things. My cluster right here is right around 264, it actually just came out so it's going manually kind of doing a rolling upgrade process through hundreds of nodes, I can just simply [Indiscernible] [0:52:26], select 265 from 264 and apply the changes. And this is now this is how… then I can deploy an automated and roll out these changes very easily. So I got this demo cluster which I'm going to upgrade to 264, 265, you know I've got this shard cluster that I'm going to deploy and I just review, confirm, take it off and I let automation do its thing.

So it's going to run and it's going to do a rolling no-downtime upgrade on this cluster, it can deploy the sharded cluster, this new cluster you'll eventually see will also be added into modern control and everything is done for me very easily. And if I wanted to, if I want to hook this up to backup, there's MMS backup here, in this case I have a shardard cluster here on our MMS backup control. So this is something I can go control by basically how often snapshots are created from my retention schedule and then through here this thing form basically my cluster will then start having snapshots created.

And if at any point I want to restore any part of the cluster or the whole cluster I can go ahead and I'll retrieve these snapshots very easily. So if I want to do for example a point in time restoration of my cluster, my most recent one, I can go select a snapshot, I can specify a point in time and the way I can go ahead and select this [Indiscernible] [0:53:59]. This is basically where I can get a point in time restoration on my sharded cluster here then I can discrete the liberty snapshots through a push or pull method. And you know basically recover to some point in time. That's MMS backup.

Totally this gives you an idea on MMS, this is obviously very critical if you guys are going to use big MongoDB apps, it becomes more critical with the right tools enable just to scale operation. So that's MMS, actually that

Here's a sign up link you can sign up on our website definitely do this if you're interested in working. As well I want to let you know before we jump to the Q&A session, after the webinar there's going to be a survey and we will greatly appreciate if you provide some feedback to us and as [Indiscernible] [0:55:22] you have a chance to win MongoDB [Indiscernible] [0:55:24] along with the survey and feedback that you provide to us. If you ever want to contact me this is my contact information as well and I think so, we'll jump into the Q&A part of the webinar now. So we have a few questions.

I have a question from John here and I think I might have answered the question already. And the question here is can you give me a real world scenario of choosing a proper shard key? So the examples I gave you are real like device ID and the timestamp. Those are real, they were [Indiscernible] [0:56:10] how to select a shard key. Those are definitely real world scenarios. But if ever you guys have an actual application that extends to your shard that's another system we can work [Indiscernible] [0:56:27] provides ID timestamps is a real world scenario.

Other real world scenarios could be just as simple as just sharding on user ID that's very common as well. Perform management systems, just profiles, kind of key value access on one of these profiles just in contact where user ID you would [Indiscernible] [0:56:48] on that. I have another question here from… hopefully I'm pronouncing that name correctly Bros, the question is "Hi is it possible to have two dimensions sharding?" In Apple we have devices, not just on IT, devices are sending data to the DB. We would like to shard by the device ID and we would like to have data for the last three older data on hard drives.

So sharding or [Indiscernible] [0:57:28] ID and data value [Indiscernible] [0:57:30] for us, okay. So the… let me see if hopefully I'll rephrase the question in order to understand it. I think there's a couple of parts when we say can we have two dimensions of shardings, can we shard on maybe like a compensate. The answer is yes, I gave an example of [Indiscernible] [0:57:56] as a shard key where we have device and time dimensions, device ID and timestamps.

To answer your question around kind of pure storage architecture we kind of went over that earlier, so from that standpoint you can definitely do a multiple dimensions right one on dimensional device ID for example and timestamp. But also through [Indiscernible] [0:58:19] sharding you can also tag specific sub ranges to do content dimension of storage types, right. Flash, archive, and cash [Indiscernible] [0:58:34].

And if you have three months of data kind of like the example in my presentation you can have the data split between flash, in my presentation I gave you an example of how I split the data between flash and archive by specifying the range of the 90 days. So there's actually D blogs so if you actually search on Google for Tag Aware Sharding MongoDB you come up with some [Indiscernible] [0:59:06] to find a blog that is written by some of our consultant engineers on compute and storage. So if you read up on that that actually describes it at a very… a lot more detail on how they've done it.

vocal index on each of the shards. So the impact could be for index, right?

Each of your shards index then overhead associated with that. So if you were for example to create your own hash instead of going through our own hash index you can create your own let's say the underscore ID in MongoDB, there's already this primary index. Let's say if you already guarantee that [Indiscernible] [1:00:28] ID to underscore ID and you guarantee that there's already some [Indiscernible] [1:00:32] server hash then that's a way for you to avoid the extra index and that would work as well, right?

That's definitely an optimization, the obvious causes that the hash… using the hash index in MongoDB is more from a convenient standpoint because you can also use it to [Indiscernible] [1:00:48] the data as I described previously. So next question from Nick, "There are page [Indiscernible] [1:00:58] for indexes major concern ISPs, SSPs? That's a relative question, right? So RAM is obviously faster than SST and whether you end up page falling and going to SST or RAM that's a matter or your SLA or latency requirement, right?

It will matter if you're talking about, "Hey I need SLA of 98% are under two or one millisecond." Absolutely on hash [Indiscernible] [1:01:31] SST that still can be a challenge under high workload, low workload maybe that's fine. Under high workload that's probably the scenario where you need to run off RAM. But if you don't have that very high latency requirement then yes you can just run on [Indiscernible] [1:01:49]. And then there's another question from Nick as well follow-up, "When using less memory… that was associated with the previous question, do you think less memory that can… oh yeah.

Okay so it sounds like what if I don't have much memory and I can't fit the indexes in the RAM? Once again it's just a matter of your requirements. So if you end up paging off of SSTs it's going to take a performance hit it's probably not as fast as RAM. Whether that's okay or not is really dependent on your latency here. Another question from John, "For good locality on indexes would it be easiest to use apps collection?" So I'll have to say the answer is no and it's usually independent of CAT collections.

Generally for good locality t does depend on the type of queries, the right balances [Indiscernible] [1:03:06] a really good example of when we have a monatomic the increasing shard key… sorry monatomic [Indiscernible] [1:03:16] value like a timestamp and the queries are time related. It doesn't really matter if [Indiscernible] [1:03:26] or not so not really the CAT collection but [Indiscernible] [1:03:34]. Another thing to be aware of the CAT… for CAT collection just form the sharding webinars, you can know the CAT collections can't be sharded.

Follow up question from John or will that not work as expected shards, yes the CAT collections can't be sharded. [Indiscernible] [1:03:54] CAT collection. So I have a question from Tim, "What's the best way to figure out how many pre-allocate chunks to make?" Yeah so the best way to do that is one having a knowledge of [Indiscernible] [1:04:17] the range of… first knowing the range of values. Let's say you decide to shard on user ID and you know you have a million uses from one to one million, that's the first thing to know. Once

default it's 64 megabytes. So the idea is that no… understand the range of values and the total data volume. If you know the size of your chunk, a 64 megabytes then you can do the math for how many chunks to avoid dynamic splitting, to avoid migration. All right, so next question is from David, "What are your thoughts on choosing number on Mongo [Indiscernible] [1:05:31] one or are there [Indiscernible] [1:05:33]?"

So the general rule of thumb is that you run the Mongo Ss and app server and that works on because typically there's more application servers in MongoDB or MongoDB servers. So that ratio works out, if it doesn't, the general [Indiscernible] [1:05:59] probably share a ratio of one… what's the [Indiscernible] [1:06:03] rule of thumb one to one on Mongo S to Mongo Ds? And that's something that will also… that's a rule for generally work out, if it doesn't then in your performance test you can also review that if whether or not if there's a bottleneck for Mongo S.

If you're doing [Indiscernible] [1:06:22] and there's a bottleneck you can identify anything on the server, maybe it is related to the client side. It may add more clients [Indiscernible] [1:06:28] that could be a possible bottleneck, right? but in summary roughly one to one but to keep things very simple what people usually do is just deploy a Mongo S per app server and that kind of works out as well. We might take a scenario Mongo S will we now have a sufficient number of Mongo Ss. All right, a question from Ted, "Hi would you use sharding for multi tenancy?"

So that's definitely an option if not just like multi tenant designs in a relational database, there's a lot of options, do I have tenancy exfoliation at the schema level or the database level or the [Indiscernible] [1:07:15] level, those same strategies or decisions exist in MongoDB. To shard at… to basically shard for multi tenancies is a lot like sharding at the schema level. The answer is it depends, there's pros and cons, right? So one of the cons of that is that if you shard using that versus as the sharding at the database level, there's certain things like dropping a tenant a little more difficult because you actually need to delete the tenant data as opposed to dropping a database, right?

But the one benefit of sharding for multi tenancy is that it provides also kind of a transparent way for you to basic [Indiscernible] [1:08:10] on tenant as well. That is a little more difficult but sometimes in some cases database level where you may need to move databases around to balance the loads and things like that, whereas sharding will kind of lower I guess. So these are just pros and cons and it is very similar to very analogous to the same decisions of a tenancy to really [Indiscernible] [1:08:39]. So questions from Lucile, "What is the smallest configuration that you can have to saturate the shards to see how the latency test go?"

Smallest configuration to saturate the latency, okay, so that's a question, I think it's ideally for capacity planning. In a… there's no really… definitely once again it depends on the load and things, the number of servers and the workload and stuff like that. But in a nutshell what is the ideal methodology? You first [Indiscernible] [1:09:21] a replica set for a single shard and saturate that to identify the bottlenecks, kind of

another shard then you'll go through the same process, right? So that's how in your [Indiscernible] [1:09:50] and repeat that and that's how you would figure out basic methodology on how to… what you need to do to saturate your shards. There's unfortunately no [Indiscernible] [1:10:08], at least finer answer on beyond that process. I definitely recommend looking at the [Indiscernible] [1:10:14] planning webinar on that topic, that might provide you some direction on so and for how to identify bottlenecks and saturate things from a [Indiscernible] [1:10:29].

So hopefully that gives you… and so I have a question from John. What if we have an ID that isn't as well distributed as device ID? For example we have customer ID and some customers are much large than others? So that's a good point, so in that scenario this is where you do multi tenancy of that you may have some very big customers and very small ones. And those very big customers very well ideally that one customer ideally by shards. So this is the case where to get customer ID wouldn't be a good shard key from a standpoint of CAT analogy, right?

So how would you resolve that? It could be as simple as creating a composite key, maybe customer ID has something else would it make more sense? That's a starting point. But that's something that will need, what is the right shard key? You're going to basically go through that process that we discussed previously on how to analyze what is a good shard key, look at those five properties and you need to think of something else. Customer ID kind of feels out of a pat [Indiscernible] [1:11:55], you will need to find something else. I mean customer ID has something else that will satisfy those five properties. All right, so those are all the questions and I want to thank everyone for attending the webinar today.

You can see also under monitoring controls, so I can jump in here and I can see cluster level view of the system. If you put how to do analysis and I can drill in to the replica set and I can view all the different health metrics for this cluster as well. So let's jump back here, all right so if I wanted to go see I've just created my new… or in a process of deploying my new sharded cluster, what I can do is I can go into MMS and I can configure things through… I can [Indiscernible] [0:50:36] so it's going to be my predictive maintenance question, of how predictive.

Then there's like diversion on MongoDB I'm going to run a… and then I have some servers with your [Indiscernible] [0:50:50] our prefix webinar and everything else is self explanatory. Sso I wanted to put them on this port ranges and I can enter them so and then I can then very easily just configure. I'm going to ask this and I can fix servers so I'm in the replica [Indiscernible] [0:51:17] or shards. I want three shards and then I can configure the replica set here; I just want to have two data baring those. An arbiter I can configure that very easily.

Then I'll call this my predictive, my predictive shards so I know what I'm doing. And there's a lot advanced options as well, I can resize OP logs through here and various configurations if I wanted to. So I'm going to apply that and once I've done this you can see it's qued up for change and I can jump into the server view,

So that's something I can do. Other things I can do like I say not just to provision service but day to day things. My cluster right here is right around 264, it actually just came out so it's going manually kind of doing a rolling upgrade process through hundreds of nodes, I can just simply [Indiscernible] [0:52:26], select 265 from 264 and apply the changes. And this is now this is how… then I can deploy an automated and roll out these changes very easily. So I got this demo cluster which I'm going to upgrade to 264, 265, you know I've got this shard cluster that I'm going to deploy and I just review, confirm, take it off and I let automation do its thing.

So it's going to run and it's going to do a rolling no-downtime upgrade on this cluster, it can deploy the sharded cluster, this new cluster you'll eventually see will also be added into modern control and everything is done for me very easily. And if I wanted to, if I want to hook this up to backup, there's MMS backup here, in this case I have a shardard cluster here on our MMS backup control. So this is something I can go control by basically how often snapshots are created from my retention schedule and then through here this thing form basically my cluster will then start having snapshots created.

And if at any point I want to restore any part of the cluster or the whole cluster I can go ahead and I'll retrieve these snapshots very easily. So if I want to do for example a point in time restoration of my cluster, my most recent one, I can go select a snapshot, I can specify a point in time and the way I can go ahead and select this [Indiscernible] [0:53:59]. This is basically where I can get a point in time restoration on my sharded cluster here then I can discrete the liberty snapshots through a push or pull method. And you know basically recover to some point in time. That's MMS backup.

Totally this gives you an idea on MMS, this is obviously very critical if you guys are going to use big MongoDB apps, it becomes more critical with the right tools enable just to scale operation. So that's MMS, actually that tool, so we're just wrapping up on things and kind of touch on the key points I wanted to go through in this webinar. We went through a couple of final points, so if you guys want to get in contact with an expert advice for scaling for a limit of time you want to see the commercial relationship with MongoDB you sign up for a free one hour consult about scaling with our MongoDB engineers.

Here's a sign up link you can sign up on our website definitely do this if you're interested in working. As well I want to let you know before we jump to the Q&A session, after the webinar there's going to be a survey and we will greatly appreciate if you provide some feedback to us and as [Indiscernible] [0:55:22] you have a chance to win MongoDB [Indiscernible] [0:55:24] along with the survey and feedback that you provide to us. If you ever want to contact me this is my contact information as well and I think so, we'll jump into the Q&A part of the webinar now. So we have a few questions.

I have a question from John here and I think I might have answered the question already. And the question here is can you give me a real world scenario of choosing a proper shard key? So the examples I gave you are real like device ID and the timestamp. Those are real, they were [Indiscernible] [0:56:10] how to select a shard key. Those are definitely real world scenarios. But if ever you guys have an actual application that extends to

Perform management systems, just profiles, kind of key value access on one of these profiles just in contact where user ID you would [Indiscernible] [0:56:48] on that. I have another question here from… hopefully I'm pronouncing that name correctly Bros, the question is "Hi is it possible to have two dimensions sharding?" In Apple we have devices, not just on IT; devices are sending data to the DB. We would like to shard by the device ID and we would like to have data for the last three older data on hard drives.

So sharding or [Indiscernible] [0:57:28] ID and data value [Indiscernible] [0:57:30] for us, okay. So the… let me see if hopefully I'll rephrase the question in order to understand it. I think there's a couple of parts when we say can we have two dimensions of shardings, can we shard on maybe like a compensate. The answer is yes, I gave an example of [Indiscernible] [0:57:56] as a shard key where we have device and time dimensions, device ID and timestamps.

To answer your question around kind of pure storage architecture we kind of went over that earlier, so from that standpoint you can definitely do a multiple dimensions right one on dimensional device ID for example and timestamp. But also through [Indiscernible] [0:58:19] sharding you can also tag specific sub ranges to do content dimension of storage types, right. Flash, archive, and cash [Indiscernible] [0:58:34].

And if you have three months of data kind of like the example in my presentation you can have the data split between flash, in my presentation I gave you an example of how I split the data between flash and archive by specifying the range of the 90 days. So there's actually D blogs so if you actually search on Google for Tag Aware Sharding MongoDB you come up with some [Indiscernible] [0:59:06] to find a blog that is written by some of our consultant engineers on compute and storage. So if you read up on that that actually describes it at a very… a lot more detail on how they've done it.

So that's a good reference to [Indiscernible] [0:59:25]. And so let me jump to the next question here, hopefully I answered that correctly. The next question's from Dan, is there a performance impact to vetting the server hash or shard key as opposed to find your own key already hash? So the answer is yes. Sometimes the hash shard key there's require you also to create for shard keys in general since they're [Indiscernible] [0:59:55] vocal index on each of the shards. So the impact could be for index, right?

Each of your shards index then overhead associated with that. So if you were for example to create your own hash instead of going through our own hash index you can create your own let's say the underscore ID in MongoDB, there's already this primary index. Let's say if you already guarantee that [Indiscernible] [1:00:28] ID to underscore ID and you guarantee that there's already some [Indiscernible] [1:00:32] server hash then that's a way for you to avoid the extra index and that would work as well, right?

That's definitely an optimization, the obvious causes that the hash… using the hash index in MongoDB is more from a convenient standpoint because you can also use it to [Indiscernible] [1:00:48] the data as I described previously. So next question from Nick, "There are page [Indiscernible] [1:00:58] for indexes major concern

hash [Indiscernible] [1:01:31] SST that still can be a challenge under high workload, low workload maybe that's fine. Under high workload that's probably the scenario where you need to run off RAM. But if you don't have that very high latency requirement then yes you can just run on [Indiscernible] [1:01:49]. And then there's another question from Nick as well follow-up, "When using less memory… that was associated with the previous question, do you think less memory that can… oh yeah.

Okay so it sounds like what if I don't have much memory and I can't fit the indexes in the RAM? Once again it's just a matter of your requirements. So if you end up paging off of SSTs it's going to take a performance hit it's probably not as fast as RAM. Whether that's okay or not is really dependent on your latency here. Another question from John, "For good locality on indexes would it be easiest to use apps collection?" So I'll have to say the answer is no and it's usually independent of CAT collections.

Generally for good locality t does depend on the type of queries, the right balances [Indiscernible] [1:03:06] a really good example of when we have a monatomic the increasing shard key… sorry monatomic [Indiscernible] [1:03:16] value like a timestamp and the queries are time related. It doesn't really matter if [Indiscernible] [1:03:26] or not so not really the CAT collection but [Indiscernible] [1:03:34]. Another thing to be aware of the CAT… for CAT collection just form the sharding webinars, you can know the CAT collections can't be sharded.

Follow up question from John or will that not work as expected shards, yes the CAT collections can't be sharded. [Indiscernible] [1:03:54] CAT collection. So I have a question from Tim, "What's the best way to figure out how many pre-allocate chunks to make?" Yeah so the best way to do that is one having a knowledge of [Indiscernible] [1:04:17] the range of… first knowing the range of values. Let's say you decide to shard on user ID and you know you have a million uses from one to one million, that's the first thing to know. Once you've done that then the idea is that you want to pre split it so that once the million users of data is entered the nod that there's no dynamic split.

The way you determine that is you use the fact that you use this knowledge of how big this chunk is and by default it's 64 megabytes. So the idea is that no… understand the range of values and the total data volume. If you know the size of your chunk, a 64 megabytes then you can do the math for how many chunks to avoid dynamic splitting, to avoid migration. All right, so next question is from David, "What are your thoughts on choosing number on Mongo [Indiscernible] [1:05:31] one or are there [Indiscernible] [1:05:33]?"

So the general rule of thumb is that you run the Mongo Ss and app server and that works on because typically there's more application servers in MongoDB or MongoDB servers. So that ratio works out, if it doesn't, the general [Indiscernible] [1:05:59] probably share a ratio of one… what's the [Indiscernible] [1:06:03] rule of thumb one to one on Mongo S to Mongo Ds? And that's something that will also… that's a rule for generally work out, if it doesn't then in your performance test you can also review that if whether or not if there's a bottleneck for Mongo S.

will we now have a sufficient number of Mongo Ss. All right, a question from Ted, "Hi would you use sharding for multi tenancy?"

So that's definitely an option if not just like multi tenant designs in a relational database, there's a lot of options, do I have tenancy exfoliation at the schema level or the database level or the [Indiscernible] [1:07:15] level, those same strategies or decisions exist in MongoDB. To shard at… to basically shard for multi tenancies is a lot like sharding at the schema level. The answer is it depends, there's pros and cons, right? So one of the cons of that is that if you shard using that versus as the sharding at the database level, there's certain things like dropping a tenant a little more difficult because you actually need to delete the tenant data as opposed to dropping a database, right?

But the one benefit of sharding for multi tenancy is that it provides also kind of a transparent way for you to basic [Indiscernible] [1:08:10] on tenant as well. That is a little more difficult but sometimes in some cases database level where you may need to move databases around to balance the loads and things like that, whereas sharding will kind of lower I guess. So these are just pros and cons and it is very similar to very analogous to the same decisions of a tenancy to really [Indiscernible] [1:08:39]. So questions from Lucile, "What is the smallest configuration that you can have to saturate the shards to see how the latency test go?"

Smallest configuration to saturate the latency, okay, so that's a question, I think it's ideally for capacity planning. In a… there's no really… definitely once again it depends on the load and things, the number of servers and the workload and stuff like that. But in a nutshell what is the ideal methodology? You first [Indiscernible] [1:09:21] a replica set for a single shard and saturate that to identify the bottlenecks, kind of iterate on that until you reach the point for example that where the disk start saturating and your final bottleneck is your disk performance.

Once you've done that on that one shard then what you typically want to do is start scaling out, you'll add another shard then you'll go through the same process, right? So that's how in your [Indiscernible] [1:09:50] and repeat that and that's how you would figure out basic methodology on how to… what you need to do to saturate your shards. There's unfortunately no [Indiscernible] [1:10:08], at least finer answer on beyond that process. I definitely recommend looking at the [Indiscernible] [1:10:14] planning webinar on that topic, that might provide you some direction on so and for how to identify bottlenecks and saturate things from a [Indiscernible] [1:10:29].

So hopefully that gives you… and so I have a question from John. What if we have an ID that isn't as well distributed as device ID? For example we have customer ID and some customers are much large than others? So that's a good point, so in that scenario this is where you do multi tenancy of that you may have some very big customers and very small ones. And those very big customers very well ideally that one customer ideally by shards. So this is the case where to get customer ID wouldn't be a good shard key from a standpoint of CAT analogy, right?

Customer ID kind of feels out of a pat [indiscernible] [1:11:55], you will need to find something else. I mean customer ID has something else that will satisfy those five properties. All right, so those are all the questions and I want to thank everyone for attending the webinar today.

**Resources**

NoSQL Database Explained

MongoDB Architecture Guide

MongoDB Enterprise Advanced

MongoDB Atlas

MongoDB Stitch

MongoDB Engineering Blog

Referral Program

**Education & Support**

View Course Catalog

View Course Schedule

Public Training

Certification

MongoDB Manual

Installation

FAQ

**Popular Topics**

Comparing Cloud MongoDB Services: MongoDB Atlas vs mLab

Announcing MongoDB Stitch: A Backend as a Service for MongoDB