EPISODE 586

[INTRODUCTION]

[0:00:00.3] JM: YouTube runs a large MySQL database to hold the metadata about its videos. As YouTube scaled, the database was sharded, and applications within YouTube had to write queries that were aware of the sharding layout of that database. This is problematic, because it pushes complexity to the application developer. An application developer shouldn't have to be aware of how a database is laid out among different nodes. The developer should be able to issue a query and have the cluster simply return the data.

Vitess is an open source system for scaling large MySQL databases. Sugu Sougoumarane is the co-creator of Vitess, and he started creating it at YouTube. Since YouTube is owned by Google, Vitess was able to leverage the Borg cluster manager developed at Google. Once Kubernetes came to market, it became more viable to make Vitess available to open-source developers.

Sugu joins the show to talk about the scalability problems that YouTube's database infrastructure encountered, and the motivations for building Vitess. This was a great conversation for anybody that's interested in distributed systems, distributed databases, Kubernetes. I really enjoyed it and I hope Sugu comes back on the show at some point in the future.

[SPONSOR MESSAGE]

[0:01:28.0] JM: Every team has its own software and every team has specific questions about that internal software. Stack Overflow for Teams is a private secure home for your teams' questions and answers. No more digging through stale wiki's and lost e-mails. Give your team back the time it needs to build better products.

Your engineering team already knows and loves Stack Overflow. They don't need another tool that they won't use. Get everything that 50 million people already love about Stack Overflow in a private secure environment with Stack Overflow for Teams. Try it today with your first 14 days free. Go to s.tk/daily.

Stack Overflow for Teams gives your team the answers they need to be productive, with the same interface that Stack Overflow users are familiar with. Go to s.tk/daily to try it today with your first 14 days free. Thank You Stack Overflow for Teams.

[INTERVIEW]

[0:02:41.1] JM: Sugu Sougoumarane is the CTO of PlanetScale Data and one of the creators of Vitess. Sugu, welcome to Software Engineering Daily.

[0:02:48.9] SS: Thank you.

[0:02:50.0] JM: Today, we're talking about Vitess. I want to start with some of your experiences at YouTube. You were an engineer at YouTube from 2006 to 2018. That's 11 or 12 years, it's a really long time. Tell me a story about the difficulty of scaling YouTube in the early days.

[0:03:09.7] SS: Oh, so actually when YouTube was started we were just on one single database, one MySQL database. We called it main. All that I was there. The first time we grew out of it, we did what we call as vertical sharding, which is basically move some tables out of that table and split that into separate databases. Then we soon realized that that's not going to scale. Then we did our first resharding, which was 100% manual and scripted.

That actually took us until about I believe 2010. In 2010, that system started falling apart, which is basically when myself and my co-creator of it that’s Mike Solomon, decided to that we are up against the wall and it's just getting worse every day. Unless we do something about this, we are going to be in deep trouble.

We actually took ourselves out of firefighting, and then said, “Okay, let's think about all the problems that we face today, and see if we can solve them in a unified fashion,” which is how Vitess was born.

[0:04:22.2] JM: When I think about the engineering challenges of YouTube, the first thing I think is actually streaming video. I don't think of the databases being that challenging. I mean, you're just adding a video, and I imagine the database is just the title of the video, a pointer to where the URL of the streaming mp4 file is. Why is the database a complicated problem at YouTube?

[0:04:50.3] SS: Video is also a challenging process to stream, but the solutions are easily solvable with relatively simple architecture, because you can copy the video all over the world, and you can build some caches in front of it. Your video serving scalability problem itself is solved that way much more easily. The metadata, even though it is much smaller, the main problem is that if you are on something like MySQL, you have only one machine that it lives on. How much smaller can it get? I mean, YouTube is still, has still a large number of users and videos with billions of videos at YouTube. It was not going to fit on one database. Then you have to still figure out a way to scale it.

The challenge is because the metadata is transactional, you cannot just spread it out and copy it. It's actually data that changes and is updated by the users. You have to make sure that it's updated correctly. Whereas, the video once it's created, you don't update it. You just copy it and then serve it. That's what makes scaling transactional data more difficult.

[0:06:01.8] JM: We're talking about changing the – if a user wants to change the title of their own video, or if they want to add some tags to it, or if they want to comment on it, all of these things are metadata associated with that video database entry. With that metadata database, tell me a little bit more about the scalability issues that you were encountering.

[0:06:26.6] SS: One scalability issue that we faced was the read traffic. That was easily – we had an easy way out of it was to basically replicate that data into replicas and direct our reads to them. The downside of reading from a replica is that it is lagged and you may not read up-to-date data. We actually employed a very interesting technique to handle the fact that we are reading from a replica.

For example, a user goes and updates a profile and that data then gets replicated, but then the first thing that people do after they update their profile is to actually reload the page to see if it took. Then almost always, they don't see their page updated, because it's not gone to the replica yet. We need to cheat there, where you read from a replica and if it's your own profile, then we send the reads back to the master. If you are reading somebody else's profile, then that can be read from the replica, because people are not checking on something they just updated.

We had to do some tweaks like that in the application. Then that actually got us a couple more years of scalability. At the end, once your writes cannot scale, then you are up against the wall. You have to reshard. That's the only way out in terms of scalability.

[0:07:46.7] JM: Before we get into the solutions that you built at YouTube with Vitess, I feel we should review a little bit about scaling SQL and NoSQL databases. SQL databases have a defined schema, NoSQL databases have a less defined schema. How do the scalability properties of SQL and NoSQL databases compare?

[0:08:14.9] SS: Yeah, there's some history there, like the one thing that you would notice is they call it NoSQL, not a key value store. There's cultural significance in that name, it is because it was actually a revolt against SQL databases that actually caused NoSQL to be born. That is because I think around 2000s or so when the internet started to grow immensely, people wanted relational databases to solve the scalability problem for them. I believe they failed in that. The people got mad and said, “Well, if you're not going to solve this, I'm going to do something that will solve this for myself,” and they went to NoSQL.

I think took it too far on this side and gave up some very, very interesting and important properties that are needed for data stores, especially transactions and secondary indexes. Those are the things that NoSQL databases typically don't have, and they actually end up making the application more complex.

[0:09:19.0] JM: The scalability properties of SQL versus NoSQL, why is there some advantage in the scalability of the NoSQL databases, or at least if we go back to that point in time where NoSQL databases started becoming more popular?

[0:09:37.2] SS: Yeah, so that is actually a whole category of applications that a NoSQL database can satisfy. The main restriction is that the NoSQL database doesn't allow you to define secondary relationships. Once you don't have that, it makes the scaling very easy, because you can just keep on sharding your data and there's no secondary relationship to keep between them.

Scaling NoSQL became extremely easy, and also because NoSQL has such a simple API, it actually became very easy for people to learn. SQL you have to go through a course, you have to go through training, you have to understand how to use indexes, so all that complexity went away and that made NoSQL extremely popular, because it was so easy to use. Also, the downside is that there is only beyond certain level of complexity, you would have to start building those things back in your application.

[0:10:40.7] JM: When you say that term secondary relationship, do you mean in a SQL database you have foreign keys to access richer data than the certain table that you're looking at, but in NoSQL you just cram all the data into the same document?

[0:10:57.0] SS: There's actually two kinds of secondary indexes. One is the foreign key that you mentioned, which allows you to relate one table with another. There is also the simpler case, where somebody – like I am a user, I create an account for myself. I access that account either through my user ID, or it could be my e-mail address. If I said, I enter my user ID, that becomes your primary key, but then what if I say I only know my e-mail address, I don't know my user ID.

Then a key value system cannot really answer that question. In a database, you just create a secondary index on the other column, and then you can instantly search a user by their e-mail address, for example.

[0:11:41.6] JM: Right. Got it. Now there is also the term eventual consistency. Can you explain what the term eventual consistency means when we're talking about SQL versus no SQL databases? I guess, I think most people know what eventual consistency means, but maybe you could explain how it applies to scaling NoSQL versus scaling SQL databases.

[0:12:02.1] SS: Actually eventual consistency is an approach that actually applies to both SQL and NoSQL databases. Prior to MySQL, the bigger mainframe-based databases, there was only one database and you had to read from that. There was no question of eventual consistency.

MySQL introduced replication and they made it very fast and efficient, which means that you could actually – the data you've written on a master database, you can go and read it from a replica. Because the replica is lagged, they call that eventual consistency, because the replica was going to eventually catch up to it. NoSQL database has embraced this very early on, and eventual consistency got associated to NoSQL databases more than traditional relational databases.

[0:12:52.4] JM: When we're talking about the problem specifically to the MySQL infrastructure that you were facing at YouTube, we turned back to these YouTube issues, what were those specific scalability problems?

[0:13:06.9] SS: It is just a volume of transactions, the volume of traffic, the number of writes basically exceeded what a single disk on a machine could handle. Also, as we added features the amount of information that you wanted to store with each user that also kept growing. Basically, that's two different problems. One is the capacity of the disk that the amount of data you can store on a single machine, it has a limit based on the physical capacity of a disk. The number of IOPS you can perform is limited by the hardware bus itself.

Those limits now are when we actually faced these problems at YouTube, these were we thought were unique to YouTube. Now we are seeing even smaller companies face – hit these limits very quickly.

[SPONSOR MESSAGE]

[0:14:03.8] JM: Azure Container Service simplifies the deployment, management and operations of Kubernetes. Eliminate the complicated planning and deployment of fully orchestrated containerized applications with Kubernetes.

You can quickly provision clusters to be up and running in no time, while simplifying your monitoring and cluster management through auto upgrades and a built-in operations console. Avoid being locked-in to any one vendor or resource. You can continue to work with the tools that you already know, so just helm and move applications to any Kubernetes deployment.

Integrate with your choice of container registry, including Azure container registry. Also, quickly and efficiently scale to maximize your resource utilization without having to take your applications offline. Isolate your application from infrastructure failures and transparently scale the underlying infrastructure to meet growing demands, all while increasing the security, reliability and availability of critical business workloads with Azure.

To learn more about Azure Container Service and other Azure services, as well as receive a free e-book by Brendan Burns, go to aka.ms/sedaily. Brendan Burns is the creator of Kubernetes and his e-book is about some of the distributed systems design lessons that he has learned building Kubernetes.

That e-book is available at aka.ms/sedaily.

[INTERVIEW CONTINUED]

[0:15:39.1] JM: I thought that when YouTube was getting big, people had been running large MySQL clusters for a long time. Why hadn't MySQL scalability been solved by that point in time?

[0:15:53.1] SS: Anybody that actually ran MySQL at massive scale actually are companies that had huge engineering then. Many of them were actually people who knew MySQL inside-out. They were very smart engineers and they actually solved this problem through application sharding basically. They actually sharded their database and then change the application to handle the fact that this is now sharded.

It wasn't really solved by a formal system, it was just solved by each company by applying their own techniques. Each one of them solved it differently. Facebook solved it one way, Twitter solved it another way. Every big company that you see has a different way that they solve the problem.

[0:16:40.1] JM: I see. What we're going to get towards with Vitess is the tests solved it in a way that you just – all of the scalability problems are abstracted into the database itself, and so you don't have to write application specific logic, because why would you have to? You wouldn't want to write application-specific logic to scale your database. You would want all that to be in the database, which is why when you were faced with this scalability issue at YouTube, you took a step back and you basically – it sounds like you took almost a spiritual journey into the world of databases thinking very deeply about how to solve this problem, and you came out with the solutions in Vitess.

I think when I was reading about Vitess, the typical MySQL solutions prior to Vitess, they required expensive hardware, is that right? There was not a solution that just used commodity hardware?

[0:17:38.1] SS: That's correct. You start off with commodity hardware, and because you cannot trivially shard an application when the database is growing, people typically scaled MySQL up almost like a mainframe. I hear of some Vitess users that migrated out of their existing MySQL hardware, some of them were saying that their hardware costs like a $100,000. That's how big of a machines they grew into before they said, "Well, this is not going to work out in the long run." Then they decided to go the sharding way.

[0:18:11.0] JM: You wanted to solve these problems in open source, but there were some tight coupling between the MySQL infrastructure and the YouTube-specific infrastructure. It sounds like you had done your own application-specific sharding at YouTube?

[0:18:26.9] SS: Yes, we did. The first sharding that we did was application. When we actually thought of Vitess, we were not really thinking about coming up with the generic way to solve the sharding problem. We thought we will just expose our way of solving sharding and have other people adopt it, but it became obvious that that was not going to be the right approach. That the only approach that people would like to use is one where the application remains agnostic of sharding.

[0:18:58.6] JM: Database infrastructure often starts out with masters and replicas, as soon as you are going to, if you decide you want to have backups, you have a master and you have some replicas. Then eventually you shard your database, because it becomes too much data to be on any single node. Then each of the shards has a master and replicas. Why doesn't this strategy just scale to infinity? Why do you have to build something specific? Why do you have to build application sharding? Why can't you just infinitely scale this model of masters, replicas and shards?

[0:19:34.0] SS: That is essentially how all companies ended up with. Anybody who was able to shard and use replicas, they pretty much were able to scale indefinitely. The good example would be both Facebook and Twitter. This is exactly what they did. Once you can do that, once you have made the application absorb the complexity of sharding, then you can scale indefinitely and forever. As you know, Facebook has huge QPS, same with YouTube. The place where it becomes complexes is, and they start to add newer features. Every new feature has to carry this burden of sharding.

[0:20:14.6] JM: Can you just give an example of application-level sharding?

[0:20:19.8] SS: Oh, an application-level sharding, I should probably take YouTube as an example, since I've been there. In YouTube, we decided to shard by the user, which means that if I as a user come in and create an account, we decide that you live on that shard. We made another decision, which is the videos that the users create also live with the same shard as the user.

That has some advantages and some disadvantages. One advantage is get me all the videos that this user has, becomes a very inexpensive query. When you start to go into what are all the videos that this user liked, that becomes now a cross shard query, because a user might have liked videos across many other shards. Which means that the application when it says, “Oh, get me all the videos for this user,” it knows that I need to figure out which shard this user lives in and then I have to send the query there.

Then if it decides, “Oh, I want to see all the videos that this user liked,” then I have to either have secondary information about where the videos are, or send this query to all shards and then fetch that information to return to the application.

[0:21:30.4] JM: That complexity is pushed to the application developer. The application developer themselves has to be aware of the sharding schema, in order to figure out how to correctly query the cluster database?

[0:21:44.4] SS: Exactly. Every user has to know what data lives together and what data does not. They have to change their query to take care of that. There are bugs all the time, because somebody forgets something else.

[0:22:01.0] JM: Right. Not to mention, if you ever wanted to change the clustering layout, you would just destroy every application developer who has written against that schema.

[0:22:11.5] SS: Yeah, it’s impossible. It’s impossible. Very much not done. It's never ever done. Once you've decided to shard one way, you are stuck with it.

[0:22:20.4] JM: Yeah, interesting. Okay, well let's get into Vitess. Vitess is a way to abstract away these scalability problems, so that the developer and the database operator do not have to think about them. Describe some of the high-level goals of Vitess.

[0:22:38.5] SS: Vitess actually has three major goals. Historically, the first goal that we started with was just to protect MySQL, because that's what we were suffering from the most, because we are already sharded. Solving sharding was not our primary goal. We were more interested in making sure that bad queries written by the developers don't take the database down. That's one major goal of Vitess.

The second one, was we thought the application somehow knows where to send the query, and how does the application know where to send the query was the main question that we asked ourselves. We realized that there is direct correlation between the application knowing where to send the query and the where clause that the application used to send the query. Then you reverse that relationship and you say, if you look at the where clause, I can figure out where to send the query and the application doesn't need to tell me. Was when we got the inspiration that we can do transfer and sharding, where the application does not need to know where to send it. That which is the second major feature of Vitess.

The third one is something that got forced on us. What happened was when we first built Vitess, we were actually on bare metal, on frame. YouTube had their own data centers. In 2013, we were required to move into the Google Cloud. That was quite a challenge, because Vitess being an open source project, and Google is an ecosystem that has its own APIs and infrastructure, and many of those things don't have equivalents in the outside world. We had to make a decision about whether to stop Vitess as an open source project and fully internalize it, or whether we can make some trade-offs and still keep it as open-source while figuring out how to run it in the Google Cloud.

We ended up going for the second option, which is we'll see how to make this work and see how far we can go. We had a few challenging times, but we managed to succeed to run Vitess in Borg, which is Google's internal cloud, and build interfaces, for example outside we have GRPC and inside we have stubby. There are mappings for each of the features like that.

Which became Vitess's third biggest advantage, which is being able to run in any cloud environment and non-cloud environment, because of all this versatility that we built in the software.

[0:25:08.9] JM: That set you up quite well for today, where we have Kubernetes being widely accessible. Don't want to get ahead of ourselves. Let's say I've got a MySQL database. It's rapidly growing out of control. How do I start using Vitess?

[0:25:27.4] SS: There are a few approaches that Vitess users have used. The one that I generally encourage people to do is to just deploy with us on top of your database. Vitess can run like a middleware. You deploy Vitess like an application that uses the database, and then you can go back and forth between Vitess and your database. You could for example trickle some queries to Vitess and see how it handles it.

As you gain confidence, then you can over time migrate entirely all your traffic into Vitess. Once you've done that, you can go underneath and split tables, reshard your database and the application will remain mostly agnostic of what is happening underneath.

[0:26:12.7] JM: When Vitess is deployed, it attaches a component called VT tablet on top of each MySQL shard. I've got this sharded MySQL database. This VT tablet is a component that controls the interfaces between Vitess and the MySQL shard. The VT table interfaces with the MySQL Daemon on each of those shards, and VT tablet also interfaces with VT gate, which is a piece of middleware. VT gate communicates with all of those VT tablets, so that the VT gate can orchestrate things between the tablets. Then when an application makes a query to my Vitess cluster, that query hits a VT gate, and then VT gate communicates with all of those MySQL shards via the VT tablets. Is that an accurate description?

[0:27:07.6] SS: That's accurate. The only addition I would add is we present the architecture that way where the VT tablet is attached to over MySQL. That is actually not necessary, so you could actually view Vitess as a VT gate, plus a bunch of VT tablets that connect to an existing my set of MySQL instances. That's the alternate way to look at this architecture, where then Vitess appears as a pure middleware.

If you are going to be wanting to use Vitess to also manage your own databases, then it is better to attach the VT tablet with MySQL and see the VT tablet and MySQL as a single instance.

[0:27:47.9] JM: Describe what happens when a query is issued to Vitess.

[0:27:53.5] SS: Yes, that's a beautiful story that I love to tell, which is the life of a query. The application would connect to one VT gate through a load balancer. Typically, there will be multiple VT gates behind that load balancer, and those VT gates are essentially stateless, which means that you can scale them up and down based on increasing, or decreasing load. The first thing that VT gate does is parse your query. That's one unique thing that we did in Vitess, a very early decision. They said, we are going to build a parcel, because we have to be able to understand the query if we are going to do anything intelligent with any system.

VT gate first parses a query, and it has to make a few decisions. One is which table are you trying to read from? Based on that information, it makes a decision about which VT tablet, or which physical database that table lives in and that's the decision number one. The decision number two is, is your table sharded? Is your database sharded? If it is sharded, I have to figure out which particular shard I have to send the query to. For that it looks at the where clause.

Of course, it's actually I'm talking about a very simple use case where you are reading from just one table and you're not doing a complicated join, or anything. Once that is figured out, it also has to make a third decision, which is are you trying to do a consistent read? Do you want this data to be read from a master, or do you want it to be read from a replica, because you don't require consistency? Then it makes that third decision and uses these three decisions that it made to direct the query to one of the VT tablets.

The VT tablet does its own parsing. The VT tablet was written more to protect MySQL, so all the features that I talked about where people write bad queries and stuff, those are all handled by VT tablet. VT table for example would parse your query and say, “Hey, this looks an unbounded read, I'm going to add a limit class, make sure that you don't end up scanning the entire table and blowing up the database.”

It adds those protection artifacts to your query and sends it to MySQL. Finally MySQL executes your query and then returns the results. The results are then returned back to VT gate. If VT gate had sent this query to multiple VT tablets, because your where clause was required you to do a scanned query, mitigate combines the results and sends them back as if you issued the query to a single database, even though underneath the query might have been rewritten in multiple parts. That's the full life story of a query.

[0:30:39.9] JM: What about a write to Vitess?

[0:30:43.5] SS: The write to Vitess performs happens more or less the same way, because an update or an insert, or a delete. They all have where clauses. It uses the same mechanism to figure out where to send a query. Then the only difference is that this time it knows that you are writing, which means that it asks VT tablet to open a transaction on its behalf before writing. This can actually go across multiple shards, because you may issue a begin and then you may issue two or three write statements, and some of those write statements can go to different VT tablets. A transaction is opened in each VT tablet on behalf of this one logical transaction that the application is running. At the end when the application issues a commit, the commit is then transmitted to all VT tablets that got involved in that transaction.

[0:31:37.7] JM: Is there some serialization required in order to make sure that your data doesn't get read in inconsistent fashion?

[0:31:47.9] SS: No. That's one limitation of Vitess, which we call – Vitess gives you read committed consistency, which means that when you perform a read while these transactions are being written, whatever is committed is what you read. That's a limitation of the database. It would be nice to have what they call as repeatable read transaction, but that is currently available only if you are within one shard. If you are going cross-shard, then you get a read committed consistent view.

However, this is something that I've not covered in many talks. We are talking to the Facebook engineers who developed MyRocks, which is an alternate storage engine in ODB. That engine allows you to request a specific snapshot of data that is not the latest. That feature can be used to provide MVCC, which is the alternate name for repeatable reads. We do intend to eventually make that feature available once we deploy Vitess on MyRocks.

[0:32:54.9] JM: To be clear, if two reads and a write got issued at time 0, then it's possible that the first read could get processed and then the write gets processed, and then the second read gets processed, and then the two reads, despite being both issued at the same time could return different data.

[0:33:16.5] SS: Correct. That would that would be the case in ODB. In MyRocks, what happens is that when you issue a read, it gets associated with a timestamp. Then you say that when once you get that timestamp, the subsequent reads that you send to other databases, you basically say, do not give me data that is newer than this timestamp, which means that you will get a cross-shard consistent view of all your data as of your first read. That's because the MyRock storage engine is an append-only engine, which means that it has all previous snapshots of your data. It can go back and read. You can request as data as of certain time and it can give you that data. Whereas in ODB, overwrites the data, so you do not have an older snapshot to request.

[0:34:08.5] JM: Would you say that you probably shouldn't use Vitess today to maybe manage banking transactions, or something else where you need explicit serializability?

[0:34:21.0] SS: Ironically, I would have said that, but ironically square cache is a major user of Vitess. I can see why and how, because I have also been in PayPal from the early days. It is surprising, the only time you require reads to be consistent is when you want to perform a transaction. At that time, a snapshot consistent view is actually not sufficient. What you actually want is a locked serializable read, and that is possible in any database even across shards.

A serializable read, where you do select for update, because I intend to change this row, I don't want this row to change when I update its balance, is something that is available with Vitess also. It's not actually as important a feature as it is made out to be. Many people can live with read-committed consistency most of the time.

[0:35:18.3] JM: Okay, I think I understand. If I started using Vitess and my database keeps growing and growing, eventually it's going to need to reshard. This is something that would be impossible to explain over a podcast, because it's just probably a little bit too diagram necessitating. Let's talk a little bit more abstractly, why is resharding a complicated problem?

[0:35:49.0] SS: The resharding is complicated, because I remember having this argument about after we resharded at YouTube, what we ended up doing was it becomes complicated, because data is now all over the place. You have masters and you have replicas. Not just replicas locally, you also have replicas across various data centers. You now have such a complicated topology. How do you make sure that you haven't made a small mistake in your script and accidentally sent the data to the wrong shard, or to the wrong replica? How do you ensure that replica isn't accidentally connected to the wrong master?

This big mesh that you have built, you multiply that complexity by secondary system that you are bringing and they all should be correctly connected to the source system, is a huge nightmare and almost impossible to do manually, which is what makes the Vitess resharding so exciting.

[0:36:55.2] JM: The database engine, or I guess Vitess takes care of resharding in a way that is entirely transparent to the database operator, is that right?

[0:37:06.3] SS: Yes. Yeah, the database operator has a little bit of control and decision-making to do. They for example have to decide how they want to split the data. They may say, “Oh, this shard is hard. Maybe instead of splitting it two-way, we will split this one three-way, whereas the other shards will split them two-way. Those levels of decisions are made by the operators, but once those decisions are made they just have to bring up these target shards and then push a button and then at the end your resharding is completely done.

[SPONSOR MESSAGE]

[0:37:47.9] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plugins. Use the value stream map to visualize your end-to-end workflow. If you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

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

You can check it out for yourself at gocd.org/sedaily. Thank you so much to ThoughtWorks for being a long-time sponsor of Software Engineering Daily. We're proud to have ThoughtWorks and GoCD as sponsors of the show.

[INTERVIEW CONTINUED]

[0:39:10.0] JM: In order for the Vitess system to be able to coordinate these queries among different shards and abstract away all the complexity of this sharded infrastructure, the VT tablet has to be, I believe has to be aware of a lot of information. The VT tablet has to be aware of what information is on different shards, what the scheme of the database is, perhaps what other VT tablets are doing. Can you talk about the implementation of the VT tablet, or maybe the relationship between the VT tablet and the VT gate as well? You could just give me a little bit more information on the data layout among these components.

[0:39:58.9] SS: The VT tablet itself doesn't concern itself with other VT tablets. Its only job is to take care of the MySQL that it is attached to and serve queries from it. We are actually thinking of adding some relationships across VT tablets, but most VT tablets just operate by themselves. Their only job is to sort queries –

[0:40:22.4] JM: Right. I'm sorry. I had it backwards. I'm sorry, I had it backwards. The VT gate, VT gate is what I meant to say. The VT gate is again the middleware that communicates with the VT tablets that are attached to the MySQL Daemons. I'm sorry about that. I should have said the VT gate needs to be aware of some of this information, is that right?

[0:40:40.0] SS: Yes. Yes, correct. This is something that we believe we did some really innovative stuff here, which is when we designed how VT gate is going to handle sharding, we set aside what traditional sharding was considered to be done. Like sharding is a concept that came out of NoSQL databases. The obvious question is always what is your sharding key? The moment you say, "I'm going to shard," you ask what is your sharding key?

What we did was actually use, start with the relational database as a source and extend those concepts into sharding. That led to a few architectural changes. For example, the entire sharding topology or layout is actually described in something that is similar to a database schema. In a database, you would say these are my tables, these are my indexes, and you talk about some table layout if you are in mainframe database. Well, in MySQL they are always simple files.

In the same way, when you describe you use a similar language to describe your sharding at the VT gate level. We actually call it the V schema, which stands for the Vitess schema, and then you say these are my tables, these tables are in these key spaces, these tables are – these key spaces are sharded. When you talk about sharding, you also have a formal language that you use to describe how the table is sharded, which means that – which also extends database concepts, so you have something like primary of index, which is very similar to a sharding key, but there are differences, and there are secondary indexes, and there is also the lose concept of a foreign key, which is very, very useful when you want data to actually be co-located so that your joints can be more efficient.

That information is actually external to the VT gate engine, so it's actually an input, which means that very different organizations are very, very completely different schemas and they all work fine with the same VT gate engine.

[0:42:48.4] JM: One of the ideas that you wanted to deal with better on Vitess was connection pooling. Can you explain what connection pooling is?

[0:42:57.6] SS: Oh, connection pooling is actually the first problem we had to solve, because that was what was hurting YouTube the most. It solves actually two problems. One is what we call as the thundering herd problem, so there are thousands of clients connected to a MySQL database. If that database goes down – this is at YouTube scale, huge QPS. These thousands of clients are pounding this database and suddenly this database goes down. All these thousands of clients are now going to go migrate to a new database and issue connection, new connection requests and start pounding it.

This is something that MySQL is not geared to handle very well. What happens in such cases is that these thousand connections when they go hit the other database, they essentially take the other database down and it becomes a vicious loop. Before Vitess, we had to actually write some throttling logic to prevent this thundering hurt from taking down databases.

The connection pool, what it does is it comes from the fact that to max out a MySQL database, you don't really need many connections. If you have about 24 connections or so, you can you can max out a MySQL database with just that many. What we did was we created 24 connections to MySQL and multiplexed all client requests through those 24. What happened was if there was – if a repair had happened, where you suddenly had to migrate your data to another MySQL, then you only open 24 connections, so those thundering herd requests will not take down your MySQL.

The other problem that it solves is when there are huge spikes that can happen, once in a while somebody becomes very popular and there's a huge increase in traffic. When that happens, the overload is something that MySQL is not very good at handling. In this case, again the same connection pool will make sure that MySQL is maximally utilized, but not overloaded, which actually ends up serving all your traffic much faster than it would have otherwise served if you had just sent all those queries directly to MySQL. Those are the two big problems that connection pooling solved for us, and it actually brought us about two years of runway, just that one feature. It's a huge feature.

[0:45:28.5] JM: Very cool. I want to put this in broader context. I saw you at KubeCon and in Copenhagen, which is a great conference. I had a really good time there. Kubernetes is making it a lot easier for people to take control of distributed infrastructure. Vitess recently became part of the CNCF, which is this collection of cloud native products, it's like – not product, projects, which is like the Apache Foundation, or the Linux Foundation where you've got these collection of open-source projects. What is the relationship between Vitess and Kubernetes?

[0:46:07.0] SS: Vitess and Kubernetes. Vitess actually was Kubernetes ready before Kubernetes was born, because we got lucky. because the effort that we did to make Vitess run on Google Borg was painful when we did it, but it paid off big time when Kubernetes was announced.

[0:46:28.0] JM: It sure did.
[0:46:32.0] SS: That is the relationship that we have. We probably were the first ones to blog about being ready for Kubernetes. It's ironic, because people were saying, “Oh, it's impossible to run databases in the cloud. It's impossible to do it.” Says, “What are you talking about? We can run Kubernetes. No problem.”

[0:46:48.4] JM: Why did people say that?

[0:46:50.0] SS: I think the biggest fear they have is the fact that the instances are ephemeral, that they can be taken down by the software, is something that they are very, very uncomfortable with.

[0:46:59.7] JM: For anybody who has not dealt with EC2 instances on a regular – or any cloud infrastructure instances, they die. They go away, like they disappear unexpectedly. Because deep failures do happen in the cloud all the time.

[0:47:14.5] SS: Yes, yes. We had our pain points when we deployed Vitess on Borg. The first two years were very rough. We used to get tens of pages per day when we started off. We had to go back to a few backups and restore them and recover data and stuff in the initial stages. Then we learned our lesson about how to make sure that you can run on the cloud without losing data, and with full uptime. That was a good lesson that we learned when we move to Borg.

Weird things happen. Borg would for example take down your instance, then bring up another database on that same host and port, for example. You have to build defenses against that, where you constantly keep identifying, making sure that you are tightly talking to the right database. All these things had to be written to be able to – to become comfortable running in the cloud.

[0:48:15.6] JM: Do you have to make some hash the database, to make sure it's still the right database that you think you're –

[0:48:23.3] SS: Yeah, we chase our request itself to say that every request contained the database name that I'm talking to. That's the first thing that a VT tablet validates. Is the key space name matching? Is the shard name matching? Okay, I'm going to let you execute the query.

[0:48:39.6] JM: Yeah, well and how does the experience with – so Borg, for people who don't know is the thing that came before Kubernetes at Google, or Google still runs on Borg; the cluster management system there. Kubernetes took a lot of the learnings from Borg and omega and open sourced it as a way to manage infrastructure. How does the experience of running a distributed database against Kubernetes compared to running one against Borg?

[0:49:08.6] SS: I would say Borg is still functionally way ahead of Kubernetes. There are still things that Kubernetes needs to build and grow to reach what we could do with Borg when we were inside – I mean, what Vitess can do in Borg. For example, on Borg we run tens of thousands of nodes, just Vitess nodes, these are just VT tablets. They are spread all over the world in many data centers. The entire deployment is fully automated; no human touches anything when Vitess gets – when our updated Vitess gets deployed into Borg, it just all these happens automatically.

Functionally, Borg is way ahead of Kubernetes, but Kubernetes is getting there. I would say deployment help ability to deploy flexibility in how you deploy, and also cross, world-cross datacenter support are areas where I think Kubernetes would need to catch up.

[0:50:11.7] JM: One of the things that I was paying attention to at KubeCon was the storage interfaces on Kubernetes. How important is that to you and Vitess?

[0:50:23.4] SS: Yeah, I think – I could be wrong, but I feel Kubernetes needs to spend more time studying databases, finding out what is the common element between them, common trait, being able to recognize what a master is, being able to recognize a replica, being able to figure out load balancing between master and replica. Those are things that are common traits that exist across most data stores. Those things if they somehow supported by Kubernetes, I think it would be nice.

They have this concept of the operator, which helps you define those things, but I think those things if you went the operator route, I feel every data store person would end up replicating a lot of that logic that they would write for their operator, which I feel is something Kubernetes can do.

[0:51:20.4] JM: What are the debates that are being had right now around storage on Kubernetes?

[0:51:26.3] SS: Kubernetes actually hasn't started tackling this problem at this point. I think they are still from based on what I see in terms of discussion, they are still trying to resolve how block storage is unified. I think once they get that gate put that behind them, I have a feeling the next thing that people will start to focus on is data. At this point, it's brushed off as okay, just build an abstraction and just use this as a black box. But sooner or later, I think people will start to realize that some standardization would help.

[0:52:04.7] JM: Okay, so just to give people some color here, there is a desire to be able to run databases and other stateful long-lived workloads on Kubernetes. In order to do that, there needs to be some standard – because of that, because the reason for that is because if you can do that, then you would be able to migrate database workloads to any place that runs Kubernetes. If you had an open source database that ran on Kubernetes, you could migrate your database to wherever your Kubernetes instances, as opposed to today, where you often have cloud provider specific APIs into the database that you want to work with. Is that would you say is accurate?

[0:52:48.5] SS: Yeah. The one cool thing is the meta controller demo that I showed at KubeCon was developed by Anthony. He developed it on GCE. All I did was just use his script and run it on AWS and it just came up. He was amazing.

[0:53:05.7] JM: You've got a chance to become a database that runs on Kubernetes, that is extremely portable. That seems like the huge opportunity for Vitess.

[0:53:15.5] SS: Yeah, yeah. I was I was so excited when I saw that he wrote the operator and he demoed it on GCE, and all I did was run his script on AWS. It just came up and just equally well. Then once it comes up, then it's just Vitess, you just use it.

[0:53:32.2] JM: The implication here is if somebody has a MySQL database on a cloud provider that they feel is more expensive than, or is more expensive than what they want to be paying for, they could spin up a Kubernetes cluster on a cheaper cloud provider, put Vitess on that Kubernetes instance and copy all their data from the MySQL database on that other cloud provider to Vitess, and then –

[0:54:01.5] SS: They don't even have to do that. We actually built a proof of concept, where we brought up a Vitess cluster on AWS and replicas on GCE.

[0:54:13.5] JM: Okay.

[0:54:14.1] SS: This was part of the planet scale, because some people were asking for cross-cloud solutions across RDS and GCE. We did that and perform the reparent from AWS to GCE.

[0:54:26.6] JM: A re what?

[0:54:27.7] SS: Reparent, which means that, so you bring up a cluster in AWS and you bring up some replicas in GCE and connect them up, which means that the replicas are up to date. They are only a few seconds behind. A reparent essentially means that I'm completely transferring traffic over from AWS to GCE. The master traffic all writes, everything moves over atomically. That's a cool demo that we have.

[0:54:52.0] JM: Yeah, and so is this – do you have a model in place for – so your company plans scale data. I know you're doing something around Vitess. I actually did not look at all into what your company is trying to do though. Maybe you could tell me about what the companies is planning to do.

[0:55:07.9] SS: PlanetScale is going to do three things. One is basically run Vitess as a service entirely for you. You basically say it's similar to, you go to Amazon, you want RDS, you come to PlanetScale and you say, “I want Vitess.” Then you push a button and then it brings up a Vitess cluster for you and then you start using it, and then we'll take care of scaling it as needed, resharding as needed, monitoring pages. All that will be set up for you. That's one thing that PlanetScale will do.

It also plans to help those enterprise customers that are on bare metal, on prem that would like to run with Vitess and would like to have support and software. The software that we build to run Vitess in the cloud, we will make them available to enterprises, so that they can benefit from the same automation that we built.

The third one is there is a group of companies that just want to run Vitess by themselves, and they just need basic support. For them, we'll provide training courses, so that they know how to get started with it.

[0:56:17.4] JM: Awesome. Okay, well I will be continuing to cover that as your company develops. I had to ask just one last thing, because I'm sure you get asked this question all the time, but before YouTube, like you said, you were at PayPal. Before PayPal, you were at x.com, which is the first company that, or the second company, I guess Elon Musk started. What was your experience with Elon Musk like back in the day?

[0:56:41.4] SS: Oh, Elon is an inspiring person. There's an interesting interview story that I had, was when I went to interview to x.com, I thought it was a bank, so I showed up in a suit. Elon was like, “What the hell are you wearing?” Yeah, he’s just inspiration. If he starts talking, you just sit and just keep listening to him.

[0:57:09.8] JM: Okay. All right. Well Sugue, I want to thank you for coming on Software Engineering Daily. It's been really fun talking to you about Vitess.

[0:57:15.0] SS: Yeah, and thank you very much.

[0:57:16.3] JM: Okay. Wonderful.

[END OF INTERVIEW]

[0:57:20.9] JM: If you are building a product for software engineers, or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an e-mail jeff@softwareengineeringdaily.com if you’re interested.

With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers. I know that the listeners of Software Engineering Daily are great engineers, because I talk to them all the time. I hear from CTOs, CEOs, Directors of engineering who listen to the show regularly. I also hear about many newer, hungry software engineers who are looking to level up quickly and prove themselves.

To find out more about sponsoring the show, you can send me an e-mail or tell your marketing director to send me an e-mail jeff@softwareengineeringdaily.com. If you’re a listener to the show, thank you so much for supporting it through your audienceship. That is quite enough, but if you’re interested in taking your support of the show to the next level, then look at sponsoring the show through your company.

Send me an e-mail at jeff@softwareengineeringdaily.com. Thank you.

[END]