EPISODE 1062

[INTRODUCTION]

[00:00:00] JM: For many applications, a transactional MySQL database is the source of truth. To make a MySQL database scale, some developers deploy their database using Vitess, a sharding system build on top of Kubernetes.

Jiten Vaidya and Anthony Yeh work at PlanetScale, a company that focuses on building and supporting MySQL databases sharded with Vitess. Their experience comes from working at YouTube, which has a massive, rapidly growing database for storing the information about videos on the site. Sharding is not the only database problem that YouTube faced. Availability was another issue. At YouTube, the database operators want YouTube MySQL cluster to be resilient to the failure of an entire data center.

Similarly, a developer deploying an important MySQL database to the cloud wants their database to be resilient to the failure of an entire cloud provider.

Jiten and Anthony join the show to talk about their work building multi-cloud support for MySQL and their process of deploying a consistent MySQL database in Azure, GCP and AWS.

If you enjoy this show and find it useful, you can help us out by subscribing. You can become a paid subscriber at softwaredaily.com/subscribe and it's $10 a month or $100 a year, and you get access to all of our old episodes without ads. That's over 1,300 episodes, and there's lots of content in there on anything that you're learning right now. Also, at software.com, you can find question and answer and all kinds of other content relating to this episode which can help you augment the knowledge that you're going to learn from today's show.

[SPONSOR MESSAGE]

[00:01:50] JM: Today's show is sponsored by StrongDM. StrongDM is a system for managing and monitoring access to servers, databases and Kubernetes clusters. You already treat

infrastructure as code. StrongDM lets you do the same with access. With StrongDM, easily extend your identity provider to manage infrastructure access.

It's one click to onboard and one click to terminate. Instantly pull people in and out of roles. Admins get full auditability into anything that anyone does. When they connect? What queries they run? What commands are typed? It's full visibility into everything. For SSH, RDP and Kubernetes, that means video replays. For databases, it's a single unified query log across all database management systems. StrongDM is used by admins at Greenhouse, Hurst, Peloton, Betterment and SoFi Control Access.

Start your free 14-day trial of StrongDM by going to softwareengineeringdaily.com/strongdm. That's softwareengineeringdaily.com/strongdm.

Thank you to StrongDM for sponsoring Software Engineering Daily.

[INTERVIEW]

[00:03:13] JM: Jiten and Anthony, welcome to Software Engineering Daily.

[00:03:16] JV: Thank you for having us.

[00:03:18] AY: Good to be here.

[00:03:19] JM: I'd like to start by talking about distributed SQL databases. Describe the market for distributed SQL databases. Who are the different constituents that want a distributed SQL database?

[00:03:32] JV: Any enterprises that are interested in having their data distributed across different geographical regions either for the disaster recovery case or for having their data close to where the data needs to be would be good candidates for using distributed SQL databases.

[00:03:53] JM: The biggest relational database constituents that I'm aware of are MySQL and Postgres. These are the older database providers that have just been around for a really long

time or database systems I should say. Tell me about the MySQL and Postgres markets have respectively matured.

[00:04:14] JV: MySQL is the open source relational database that is used the most by a really large margin for the current install base, but I think the rate at which Postgres is growing is higher. A lot of new folks when they are deciding which open source database to use seem to be choosing Postgres. One of the reasons for this could be the fact that MySQL, even though it's open source and even though Oracle has been a really good caretaker of the open source project for a couple of years after Oracle acquired Sun, there was a perception that this could lead to vendor lock-in using MySQL, even though it's open source, could lead to vendor lock-in, and I think that's what sort of started MyPostgres becoming more popular.

[00:05:04] JM: What about some of the newer options? There's Spanner, there's Aurora, there's CockroachDB, there's YubaByte. How did these compare to the older relational databases?

[00:05:18] JV: From my understanding, Cockroach and YugaByte are based on Spanner's model. The fundamental tradeoff that Spanner makes is that it will give you cross-data-centered durability at the cost of 99% percentile latency, and Spanner as it runs on Google Networks can pull it off because Google has by far the best networks amongst cloud providers and it also has the hardware for the atomic clocks that they use for global timestamp service. Doing the same thing on public networks, which is what Cockroach and YugaByte are trying to do is hard. What we do instead, let's just say that the tradeoffs that we make are different.

[00:06:01] JM: Cool. Well, I'll be looking forward to maybe diving into a few of those. The Spanner paper, it does include I think this use of – What was it? Like an atomic clock and then trying to have the same consistency model without atomic clocks just on regular public cloud infrastructure. You're saying that pretty difficult.

[00:06:20] JV: It's possible to be correct, but it's not possible to be correct and have a high 99 percentile latency. That's the point that I'm trying to make. If you are trying to use a database like that as a backing storage system of record for anything that is user-facing where latency is important, it doesn't become a good fit.

[00:06:45] JM: How has Kubernetes affected the market for these distributed database solutions?

[00:06:54] JV: In general, what Kubernetes has done is Kubernetes has sort of emerged or rather is emerging as an operating system for compute resources, right? Whether those are a bunch of servers running in your own data center or whether there are a bunch of VMs in a cloud provider. Kubernetes gives you the opportunity to create this layer on which you can deploy your services fairly uniformly irrespective of where these computing resources live in the cloud or on your data center and so on, and that's immensely useful.

But Kubernetes, the way it was developed, is fantastic for stateless services. The advantages that I just described work really well for deploying Stateless services across your computing resources wherever they are. The trouble comes when you want to run a stateful service, like a database on Kubernetes. Vitess and our solution is we built our solution database as a service, PlanetScaleDB, on top of Kubernetes, and the reason that we could do it is because we use Vitess at Google, on YouTube's databases on Borg, which is the orchestration system that Google uses inside Google, which is a predecessor to Kubernetes and the blueprint for Kubernetes.

[00:08:11] JM: We talked at some length about the architecture of Vitess on a pervious show, the sharding architecture. Can you just give an overview of that architecture to refresh us so we can talk through some of the details and then we'll eventually get to discussion on multi-cloud?

[00:08:28] JV: There are two main binaries in the Vitess cluster, VTGate and VTTablet. VTGate is a stateless proxy. You can start many of them behind a load balancer, and VTGate has a full MySQL parser built into it as well as it supports to MySQL binary protocol. It presents itself to an application as a MySQL server, right? Behind the VTGate, what you have are database clusters, and each database cluster represents a shard in what we call a key space. A key space with a single shard is identical to a standard MySQL cluster with one master and how many more replicas you want.

Each MySQL D in such a cluster, masters and replicas, gets a minder process, that is the second process that I talked about, which is VTTablet. Let me just quickly tell you the life of a

query. The application sends a query to a VTGate using the MySQL binary protocol. VTGate parses that query, looks at the work logs, figures out which shard the query should go to. If it's a transaction, it sends it to the master. If it's a query where it's okay to have eventual consistent reads, it sends it to one of the replicas. If it's a query where you want to have the read after write consistency, it sends it to the master.

When I say master or replica, the queries being sent using GRPC from VTGate to the VTTablet associated with it. VTTablet again parses that query, and the reason for doing this is that it does a lot of things to protect the underlying MySQL D. It does things like connection pulling. It adds timeouts to the queries and transactions. It does [inaudible 00:10:11] protection. It even adds limit clauses to your query if it things that you're going to do a full table scan using different criteria.

In general, a lot of things that a conscientious DBA would do who's monitoring a database, VTTablet already does it so that your underlying MySQL D continuous to function well and it's virtually impossible to write a query of that for a system which is managed by Vitess. All that metadata about how a particular key space is sharded and the cluster topology about which host and port the MySQL Ds are running or VTTablets are running, all that cluster topology information and information about how a key space is sharded is stored in what we call a topology server, and the topology server can be either a quorum of ETCD, a quorum of ZooKeeper or a quorum of Council. Because we needed to support inside Google, Chubby, which is Google's internal lock server, we put that behind an interphase. In the open source, we can support multiple different lock server implementations for our topology server. That's sort of the high-level architecture of Vitess.

[00:11:28] JM: The topology server helps with the management of a query coming in and needing to know which shard to be routed to through the VTGate, and the VTGate is the system that routes the query to a specific shard or a set of shards?

[00:11:48] JV: That's exactly right. Each VTGate connects to the global server and caches the information about the cluster topology as the sharding information. We have extended just the way a relational database has a schema, a sharded Vitess database has what we call a V schema, which is a JSON file where for every table in a key space, you tell us which column do

you want to shard by and what sharding function you want to use, because we allow you to shard on numeric columns as well as non-numeric columns. If you're using a numeric column, the sharding function would typically be a hash. If you're using [inaudible 00:12:24] binary, we have a function called lose MD5 Hash.

One of the coolest things about Vitess is that these sharding – We never save the sharding key for every row. We calculate it using the value in their column as well as the sharding function. Because of this, we can write custom sharding functions and everything else, the query routing, as well as re-sharding and all of that is continuous to work even if you write custom sharding function. Because of this, we can support GDPR really well or location-aware clusters.

[00:12:54] JM: One thing you mentioned earlier was that Kubernetes is not great at handling stateful workloads and that some of your work around Vitess has improved the viability of running stateful workloads. What about that architecture that you've described generalizes to being able to handle state management more effectively?

[00:13:20] JV: In general, what you need, basically when you are running in an orchestrated environment like Kubernetes, the fundamental difference between environment like that and the old way of running databases is that you cannot pick the longevity of your master for granted. In Kubernetes, your master park and go away with very little warning very quickly, whereas in the old way of doing things, you run your databases as pets, right? They're on an exotic hardware, somebody gets paged. The master goes down and so on. But here your master [inaudible 00:13:54] can go away anytime.

To deal with that, what you need is you have to have a really good story around master to replica failover. We have built that into Vitess. Vitess has very clear workflows that work very well for both plan as well as unplanned failovers from master to a replica. Similarly, once you failover a master to a replica, the second thing that the system needs to have is service discovery. Your applications need to know now where to send their writes. Because of the VPGate layer and because we maintain this information in ETCD, we have built a really fantastic service discover mechanism so that your application just is connecting to VTGate. VTGate in turn now figures out what is the new master, because that information has changed

in the topology server and it gets notified when that particular node changes and it knows that, “Oh! This is the new master now.”

In fact, VTGate actually participates in the workflow for failing over a master to a replica and it actually buffers transactions while the failover is happening so that the application doesn’t even see errors if the failover happens within a certain threshold. As I said, the second thing is a service discovery, which we have built in there, and VTGate, which is stateless proxy, which is sort of VTGate [inaudible 00:15:14] server implement that.

The third one is observability. A system like this, you really need to figure out what’s going on really well so that you can manage parts coming in, going out, masters failing over and so on. You need to understand the system really well. So you need to have really good observability. That was built into Vitess also. Every VTTablet, every VTGate has ports where telemetry can be script. We also support Prometheus on the slash metrics endpoint and it’s a very, very observable system which allows you to run it well in an orchestrated system.

[SPONSOR MESSAGE]

[00:15:58] JM: When you spend your spare time learning, you can accelerate your career. O’Reilly lets you learn through high-quality books, videos, courses and interactive experiences. O’Reilly content has been built over decades. They’re a trusted source of effective technology education. If you’re an individual leveling up on your own, you can use O’Reilly to chart a course for your career goals. If you manage a team or a company, you can get access to O’Reilly’s career development resources for your whole organization.

Go to softwareengineeringdaily.com/oreilly to explore O’Reilly’s e-learning experiences. You can build the skills you need to future-proof your career. Check out softwareengineeringdaily.com/oreilly, and thank you to O’Reilly for being a partner with Software Engineering Daily for many years now.

[INTERVIEW CONTINUED]

[00:16:59] JM: Can you describe in some detail how Vitess handles durability across a sharded MySQL cluster?

[00:17:08] AY: Yeah. As Jiten was talking about, one of the challenges of running a database or a stateful service inside Kubernetes, you can't rely on your pod to be sticking around for a longtime and you may be shut down this pod short notice because maybe the node is going to be going under maintenance, and that might be separate that doesn't know what even the node is writing. It's just saying I need to take this node down because I'm responsible for it. You need to be able to react quickly to shut this down and move it somewhere else, and that's difficult for a stateful service.

One of the tricks that we use in public cloud environments is we use things like remote attached storage or a remote block devices. If we are the master for example and are told we need to move to a different node, we can quickly shut things down, sync it to disk and then detach the disk and reattach it to a different node and then continue where we left off. It's even faster if you do, like Jiten was saying, failover in between replicas and why you're doing this so that you don't have the master gone for some amount of time.

[00:18:10] JM: So this is giving us more explanation for how a sharded MySQL cluster using Vitess works on top of Kubernetes, and the story there is already pretty durable if we're running a database on AWS. Like we're using sharded MySQL database on AWS. What's the advantage of deploying the same database to multiple clouds and having a multi-cloud database deployment?

[00:18:41] JV: There are a couple of reasons why people want to do that. One is obviously disaster recover. We were talking to a customer in Japan who were on a single cloud, and that region went down for them. Other regions in, say, North America, etc., were available, but the region in which they were running went down. In their geographical region, other computing resources were in one region and one cloud and they went down hard for a matter of hours and the amount of money their business lost, they said that we would have loved to have our resources in another cloud provider in the same geographical regions so that we could have continued to serve traffic while this outage happened. That is one sort of an obvious reason.

Again, because of Kubernetes, it's easy to migrate your stateless services quickly, because Kubernetes gives you this uniform API on which you can deploy your application servers, which are stateless by and large. But databases are – Data has gravity, right? Actually, being able to deploy your databases in a way where your masters and replicas where single database are distributed across cloud providers, gives you the option of failing over to a region in the same geography into another cloud provider if one full region goes down for a given cloud provider. That's an obvious disaster recover use case.

The second use case, vendor lock-in scenario, right? If you are talking to a cloud provider and if you are completely using everything that they provide, the advantage that something like Kubernetes gives you, you are forfeiting it by being locked-in to vendor-specific services. If you just start, most of your resources in cloud provider one, which start one or two replicas. In cloud provider two, you have the option of migrating out with one click. When you're negotiating with them for your next year's contract and so on, it's really nice to have that option. Vendor lock-in is the second reason why this would be useful.

[00:20:52] JM: It's impossible to have a single Kubernetes cluster that stretches across multiple clouds and you have the databases managed in these different clusters, or this database is managed across the same cluster, or are you talking about each single cloud has a Kubernetes cluster that is managing a complete copy of the database?

[00:21:14] AY: Yeah. To the first question, can you run a single Kubernetes cluster across clouds or even across regions? The answer is you technically can, but it's not going to perform well-enough to actually use. Kubernetes itself wasn't designed with that kind of latency in mind between, for example, the kubelets and the master API server.

The Kubernetes project itself recommends please don't do that. Only run in a local data center type environment for one Kubernetes cluster. What they then say is run a separate Kubernetes cluster in each different data center. Then they say invoke some kind of magic to connect your application across Kubernetes clusters.

Unfortunately, the story from the upstream project, it never really got far enough to say, "Here's exactly how you connect all your stuff together." A lot of people have come up with their own

bespoke solutions and we ended up having to do that as well. There is hope in the future they're starting to work on putting more effort into like Federation V2 to try to say, "Here's the one true answer," but I think the Kubernetes project is not there yet to say there's one answer for how you connect across Kubernetes clusters. That was one of the big challenges we had to work on.

[00:22:16] JM: In each cloud, let's say I've got a database. I want to put it in the three major clouds. I want to put it in Azure. I want to put it in AWS. I want to put it in Google Cloud. In this case, in the multi-cloud architecture that you guys have been working on, is it the case that I have the exact same database each of which has a Kubernetes cluster running in each of these respective clouds?

[00:22:41] AY: It depends on what you mean by exact same. The data itself per shard is going to be replicated, so the extent of maybe there's some replication delay. Other than that, the data itself is the same. Things that are not necessarily the same are how many replicas do I run in this region versus that region or maybe entire shards, for example, you might say this shard is a data locality in Europe and I want to only run replicas in Europe for that shard. You can also make those differences across different regions.

[00:23:10] JM: If I have this multi-cloud database set up, how do the read and write properties of that database change when you go to multi-cloud? Am I reading and writing to all of the different cloud provider instances or is there kind of a leader database instance in one of the clouds that I select?

[00:23:30] AY: Each shard has one leader, which is the MySQL master. That's where all writes go. Then reads, as long as you're able to tolerate whatever replication like we have from just sending data across the world, if you're able to have a little bit of latency or a replication lag, then you can do reads locally from whatever region your application is in. Reads will look pretty much the same. Writes will have higher latency if you're trying to write across the world obviously. The way that we address that when we're doing this at YouTube was minimizing the number of round trips you have to do across the world. It's inevitable you have to do at least one, but the worst case is if you're saying the application code still runs in, let's say, Europe and is doing a round trip for every query to the database in a different continent. That would be

however many queries you do to handle one web request, you have to multiply the latency times that.

What we did at YouTube was we said, “Let’s redirect write requests at the application level across continents.” That’s one [inaudible 00:24:29], and then we run the application code in the same region as the master so that all those multiple queries that the application does are happening really fast within the region and we ship the result back across continents.

[00:24:43] JM: I guess I’m not understanding completely. If I have a database deployment where I have decided that I want my database to be in AWS, and GCP, and Azure, am I designating one of those specific clouds, or are all of those active databases?

[00:25:01] AY: They’re all active for reads all the time, and at any given point, you get to specify, “I want my masters to be in this particular place,” and you can change that to say, “Oh! Maybe this provider is suffering. I need to move to a different region.” You say, “Please put my masters over there instead.”

[00:25:16] JV: Right. Let’s take a concrete example. You designated AWS US East, GCP US West, and Azure Asia as the three regions across which you want to distribute the database, right? Let’s say that you decided that this database is going to have one master and five replicas. You said that my master and one replica, when you start, is going to be in AWS, two replicas in GCP, two replicas in Azure. This is how the six replicas, one master and five read replicas are distributed.

Anytime that you want to say that instead of using the replica in AWS as a master, I want to instead to failover to GCP. It’s going to be a matter of whatever the replication lag is. Typically, in Vitness, because we keep shards at 250GB, even though cross-cloud and cross-PC latencies, we are still talking about milliseconds to seconds of replication lag, which means that within millisecond to a second, you can failover that master to be now in GCP. All VTGates that you are connected to now know that that is the master and they start sending the writes to the master in GCP. Wherever your applications running, whichever cloud and however they’re connecting to this database, the application doesn’t need to know where the master is. The

VTGates have figured out where the master is and now the writes start going to the master in GCP rather than the master in AWS.

The key concepts to keep in mind is that each database cluster has one master at a given point and N-number of replicas. Any of those replicas can become a master, and Vitess takes care of this failover for you behind-the-scenes.

[00:27:03] JM: Got it. Okay. I think I understand the happy path at least, which is that in each cloud provider, I've got a Vitess deployment. One of the replicas is a master and then I select one of my cloud providers to be the cloud provider that accepts writes to the database, and then in a different write, there's some model for communicating the writes to the other cloud providers and all of the cloud providers can accept reads. At that point, let's talk about the write. If a write happens to the cloud provider Vitess cluster that I have selected as my write acceptor, how does that write make it to the other database instances?

[00:27:50] JV: All the other database instances are using MySQL binary replication to replicate from the master every change that is happening to the master. That means that if you just dig down, you have to have some way for the packets from a pod, which is running in a Kubernetes cluster in cloud provider one to go across to the replica part, which is running in Kubernetes cluster 2 in cloud provider 2.

[00:28:18] AY: Yeah. Cilium, we mostly use for just doing policy enforcement. Cilium's job here is to restrict what the pods can communicate to each other, which is very important for example when we're running a multitenant service. We are restricting tenants at the network level and Cilium was the piece that allowed us to extend that network policy API in Kubernetes to be working across Kubernetes clusters.

As far as the actual replication at the MySQL level, as Jiten said, the hard part of connecting multiple clouds together was you need to be able to have each individual pod, which is maybe a replica pod in one cloud be able to directly connect to the IP of a master pod, which might be in a completely different cloud provider. That required building a sort of flat network across all of our various VPC in each cloud. To do that, we've used either VPC peering were available or standard VPN tunnels.

[00:29:12] JM: Could you explain what Cilium does?

[00:29:15] AY: Yeah. Maybe to start with just like network policy in general, what we mean by that is the ability to say let's say this process running at IP 1234 is allowed to talk to process 4321 or IP 4321 and nothing else. You could define that as an example policy.

The Kubernetes network policy API is a built-in API that takes that to another level of saying, "Now you can write your policies in terms of Kubernetes concepts like labels and pods." You can instead of writing out IP addresses, you define your policy as a pod with labels matching the selector A equals B. It's allowed to talk to let's say only other pods that have the same label A equals B. An example we would use for just say tenant equals X.

A Kubernetes network policy is this API that lets you express your network level restrictions in terms of Kubernetes API concepts. Kubernetes itself doesn't even give you an implementation of that API within one cluster though. Before we went across cluster, we used Calico as our plugin to implement the network policy API, and that worked well within one Kubernetes cluster. But as soon as you are trying to have pods talk to each other from different Kubernetes clusters, you now are lacking enough information to make this policy decisions. For example, if pod A in cloud 1 says I want to talk to this IP address of pod B in a different cloud, the policy system doesn't know anything about the pods in the other Kubernetes cluster. So how it can make that decision?

This is what Cilium does for us. Cilium implements the network policy API, but it implements it in a way that works across Kubernetes clusters, and really all they're doing is doing some selective mirroring of information from one Kubernetes API to the Kubernetes API in different clusters so that now each Cilium running in each cluster has enough information to say, "Oh! I can accept this connection because I know that's the IP of pod B in that other remote cluster," and now it has the information.

[00:31:05] JM: Does Cilium help you manage the changing IP addresses? Because you don't have static IP addresses because of the dynamic nature of all the change that could be going on across your Kubernetes clusters?

[00:31:18] AY: Yeah, in the sense that for policy, because you can now write your policy agnostic to these changing IP addresses, it lets you write your policy in a way that will always be enforced no matter how the IPs change.

[00:31:30] JV: The changing IP addresses, that is managed by Vitess's own service discovery, as I told you earlier, right?

[00:31:36] AY: Yeah. There are some cooperation between – It's like service discovery all the way down. We have different layers. Vitess knows how to do sort of point-to-point for tablet-to-tablet. Vitess already records tablet A has IP 1234, and other Vitess processes can look up in topology, "Please give me the address of tablet A," and it will give that out.

We also rely on another feature of Cilium, which was global service discovery. That's another feature that because the federation story is not fully there for Kubernetes, anything that relies on like the Kubernetes service object API is not going to work across Kubernetes clusters. Cilium, as an additional feature on top of their policy implementation, that gives you a way to make the Kubernetes service API work across Kubernetes clusters.

We also make use of that at, for example, the layer of connecting services together, not to point-to-point, pod-to-pod but saying, "If this pod wants to talk to the ETCD lock server, that's a servers that's load-balanced across many pods, and Cilium global service has allowed us to do that across Kubernetes clusters.

[SPONSOR MESSAGE]

[00:32:45] JM: Scaling a SQL cluster has historically been a difficult task. CockroachDB makes scaling your relational database much easier. CockroachDB is a distributed SQL database that makes it simple to build resilient, scalable applications quickly. CockroachDB is Postgres compatible, giving the same familiar SQL interface that database developers have used for years.

But unlike older databases, scaling with CockroachDB is handled within the database itself so you don't need to manage shards from your client application. Because the data is distributed, you won't lose data if a machine or data center goes down. CockroachDB is resilient and adaptable to any environment. You can host it on-prem, you can run it in a hybrid cloud and you can even deploy it across multiple clouds.

Some of the world's largest banks and massive online retailers and popular gaming platforms and developers from companies of all sizes trust CockroachDB with their most critical data. Sign up for a free 30-day trial and get a free T-shirt at cockroachlabs.com/sedaily.

Thanks to Cockroach Labs for being a sponsor, and nice work with CockroachDB.

[INTERVIEW CONTINUED]

[00:34:07] JM: Tell me more about the networking challenges for implementing a multi-cloud database.

[00:34:13] JV: We used AWS and GCP support VPN gateways and BGP routing. Pairing AWS and GCP was fairly straightforward, but Azure doesn't support the BGP protocol. We couldn't use the VPN gateways. As a result, we had to manually set up the routes, create firewall rules and firewalled the traffic from a manually provisioned IP to the classic VPN. It was basically M by N matrix of like we have three cloud providers and four regions in each cloud provider all paired with each other.

[00:34:46] JM: Could you explain what VPC pairing is?

[00:34:50] JV: VPC pairing is basically – It's a tunnel that allows you to route traffic from one VPC and one cloud provider to another VPC and another cloud provider.

[00:34:59] JM: VPC meaning virtual private cloud. The communication between two clouds, between getting to two VPCs to talk to one another, can you explain in a little more detail why that was difficult?

[00:35:10] JV: It's easier to route packets within the network boundary of a cloud provider, but now you are sending packets across cloud providers, and that's why you need VPN gateways and BGP routing.

[00:35:24] JM: I see. Did you have to write your own VPN gateway to route the traffic from one cloud provider to another?

[00:35:31] JV: No. AWS and GCP support the HA VPN gateways, but Azure doesn't. For Azure, we needed to set up the routes manually, created firewall rules and firewalled the traffic from a manually provisioned IP to the classic VPN.

[00:35:46] JM: How else do the different cloud providers differ from one another significantly?

[00:35:52] JV: Their Kubernetes implementations differ. The maturity of their Kubernetes implementations – By Kubernetes implementations, I mean their hosted Kubernetes services have different maturity. Some of them work as advertised and some of them we have to put in scaffolding to make sure that we are working around their flakiness.

[00:36:13] JM: These are like the Kubernetes as a service platforms on the different cloud providers.

[00:36:18] JV: Yeah. GKE by far is the best I think EKS is pretty good too. AKS, we have problems with.

[00:36:25] JM: How does the failure domain of Vitess change when it goes multi-cloud? One of the previous episode, we talked about how Vitess running on Kubernetes in a single data center can recover from failures and handle failures. But I'm sure the failure domain becomes much more complex when you have multi-cloud scenarios. Talk me through some of the failure cases that you've solved for.

[00:36:57] JV: Right. Vitess conceptually, a failure domain is a cell, and we map the cell to a Kubernetes cluster in a region in a cloud. In total, we have 16 different cells, and each cell

corresponds to a Kubernetes cluster in a region in a cloud. We have 12 different cells. Four regions and three cloud providers, so 12 cells.

Conceptually, cell is the failure domain in Vitess, and the same is true in this multi-cloud, multi-region world. Even the cloud providers tend to think of a region as a failure domain. It's a set of computing resources. I mean, within regions, we have availability zones, which is yet another failure domain, which is slightly more granular. That's how we deploy when we are deployed within a single region. But when you're deployed across regions or when you're deployed across cloud providers, we tend to think of a region in a given cloud as a unit of failure domain.

Anthony, do you want to add anything?

[00:38:00] AY: I would add that even in the cross-cloud or multi-region case, we do still define each availability zone within region as its own failure domain. For example, if you tell us to launch N replicas in region 1 cloud 1, we're going to do our best to spread those out across the availability zones within that region.

[00:38:19] JM: Maybe talk through the resolution of a failure that could occur. Let's say like I'm doing a write to the database instance on the cloud that's accepting writes. So I do a write and somewhere in this write, the entire cloud provider fails. What's the process of recovering from that failure? If that cloud provider has received the write, how do the other clients or how do the other database instances on the other clouds identify that a failure has occurred?

[00:38:56] JV: There are two different types of failure scenarios that we recognize. One is planned failure and one is an unplanned failure. What I mean by planned failure is that you know that this region is going to go away at a given point in time for maintenance of for whatever reasons. That scenario is handled pretty in a straightforward fashion where you failover your master out of that region into either another region in the same cloud provider or into a region in a different cloud provider.

The second, which is more problematic is if it's a catastrophic failure. With very little warning, a whole region goes away. Your master was in a Kubernetes cluster which was in that region. Let's see what all will happen, right? First is that the write itself will fail. The second thing will

happen is that we have a global ETCD cluster with members in different regions, and if there was a member region when I say ETCD server I'm talking about the topology server.

The member in that region would go away is the second thing that might happen, but because there are multiple members which create that quorum, they will still continue to respawn. Our operator detects that the master has gone down. Anthony will be able to explain it better than I do.

[00:40:07] AY: Yeah. Picking up where Jiten left off, he made sure to talk about the fact that we still have the ability to read and write to this ETCD topology server, and that's important for ensuring that Vitess can do this automatic failover in a safe way while taking account for the fact that one region might be unresponsive. There're a couple of different tradeoffs you can make as a user. You can choose different tradeoffs.

One, for example, is you could say I want to do a SQL semi-sync across clouds or across regions. If you do that, then you can know that before you were told any transaction was committed, it has been replicated to a different region. If you sign up for that setting, it means that you won't lose any transactions that reported to clients as committed.

The downside of that, the reason it's a tradeoff is because that means every write you have to wait for the round trip to go to a different region and come back and report that it was acknowledged. The other option that a user would have is to say, "I'm going to roll the dice and say if an entire region goes down, I will abandon any transactions that got orphaned in that region and repairing somewhere else while saying if that region comes back up, I'll kind of take a look at what transactions were orphaned and replay them as necessary." This roll the dice choice is actually the way that we chose to run at YouTube. We only did semi-sync within a given region or a given cell and we said, "If it ever happens that we have to continue somewhere else, we will figure out how to replay transactions."

During the time I was there, and this has never happened, and I think Sugu also has said it didn't happen while he was there, because what we saw much more frequently was it's not that an entire region suddenly disappears. It was more that you have degradation in a region ramping up, getting worse and worse.

What we did is we just had policies in place to say as soon as the degradation gets passed some threshold, we will preemptively shift our masters over to a different region, and that might be slow because maybe there's network packet loss or things like that, but we kind of babysit that, wait for it. Then once we're out of that region, we say, "Okay, we're safe. We have avoided the possibility of transactions being orphaned." Those are two different tradeoffs you can make.

[00:42:21] JM: In a condition where databases get out of sync, you might need to have a voting procedure to determine a quorum. Can you give an example of when that might occur and how databases would communicate in that scenario to resolve an inconsistency?

[00:42:43] JV: The way we run our system, we don't use voting in a quorum to figure that out. MySQL has this concept of a GTID or global transaction ID. As Anthony described, you have a master in multiple replicas and different replicas might have replicated to a different extent. Also if you have semi-sync replication turned on, what that means is that before the master says that something was successfully committed, the data associated with that particular transactions is guaranteed to be on a replicas somewhere else in its relay logs.

All that needs to happen is that it needs to get over the network and it needs to get written to the relay login. It doesn't need to be materialized into the target database. The latency required for that is lower than actually replicating into and materializing into the database.

Long story short, what we need to do is to talk to all the replicas and figure out which replica has progressed to the highest GT ID. Which replica has GT IDs, which is as close to the master as possible and then choose that as the new master. This is something that is done post-factor after the master has gone down and not for every transaction. That's how we award a split-brain with semi-sync turned on. We don't pay the cost of figuring this out for every transaction. We just do this at the time of failover with semi-sync replication turned on. Does that make sense?

[00:44:14] JM: It does, and I'd like to change the conversation to actual production deployments. Have you put the multi-cloud database into production for any particular users? Who are the people that would want to switch to this more aggressively from just a single data center?

[00:44:33] JV: We have trials. Nobody is actually using it in production yet, but I was telling you about this company in Japan who's interested in it. We are talking to them. They have created an account. I don't exactly know at what point their trial is. Somebody like them would be very interested in it. We also have some conversations going on with government agencies, early conversations who are interested in this, and we just rolled this out about two weeks ago, early days yet.

[00:45:04] JM: Zooming out, talking about the business more broadly, you've had some large customer deployments that you've worked with. You've with Slack and Square and HubSpot and a lot of other companies, and the overall business of PlanetScale is around this control plane that you can sell and license. How does the control plane that you sell compare to what the open source Vitess model contains?

[00:45:38] JV: Let me just clarify that Slack, Square, HubSpot, who have done who have done the case studies with CNCF is for open source Vitess. We do provide support to Slack and Square, but HubSpot, we don't have a relationship with.

What you said is exactly right. I mean, we do want to provide Vitess support for any company that wants Vitess support, because we want Vitess as an open source project to be as successful as possible and we are here to help the community. But that's not how we are going to scale the company. We have built is basically the technology that we have built is this operator, which allows us to create an API on top of Kubernetes that allows us to run databases safely in single or multiple Kubernetes clusters, right? That is the IP that we have built.

We have actually taken our operator and we have made a subset of that open source. We will be doing an announcement about that, etc., pretty soon, and it's a pretty full-fledged open source PlanetScale operator that will allow you to run Vitess well and safely in the Kubernetes cluster. It just doesn't have some features related to teams and cross-cluster federation and so on that Anthony talked about that our PlanetScaleDB operator has. The operator version that we use to run our own clusters, we call that PlanetScaleDB operator.

What we are going to scale the company on is this database as a service that we are running as PlanetScaleDB. What is coming is multiple ways to deploy PlanetScaleDB. One of them we call BYOK, which is bring your own Kubernetes, where you will be able to create custom regions by giving us access to your Kubernetes cluster and you could use our control planet, but when you say deploy, the actual parts that's get started in your Kubernetes cluster so that their data never leaves your Kubernetes cluster.

We will also have a premium version of PlanetScaleDB where we create a certain account for you. We can associate it with your billing account and we run a version of PlanetScaleDB in a certain VPC for you so that you are not in a multitenant environment and you could pair it with your VPC so that all your traffic is private and so on, but we still manage it for you. We carry the pagers and so on. Two or three different ways of deploying the PlanetScaleDB, and that's how PlanetScale as a company will grow and that's our commercial plan.

[00:48:10] JM: In that commercial strategy, just closing off on the business strategy. Vitess could be used to run databases that are not MySQL. You could have spent this engineering time on maybe making Postgres run with Vitess, for example. Strategically, why did you do the work on building multi-cloud support for MySQL before implementing a different database on Vitess?

[00:48:44] JV: Multi-cloud support, it's sort of an expression of all the power that Vitess has. I don't think that there is anybody else out there, there's any other technology that would allow you to run multi-cloud databases as reliably, as high-availability that we can't. We have done this at YouTube, 20 data centers, 256 shard databases over 30,000 nodes distributed all around the world. Vitess is capable of delivering that scale, and Vitess is really, really good at running databases on Kubernetes, single shard or multiple shards.

That is our core strength and we decided to build on top of that. Postgres, a lot of people ask us for Postgres compatibility and Postgres Vitess on to of Postgres, and that's definitely something that – I mean, if the community asks and if the community is willing to contribute the engineering time, we would definitely start doing work on that. We have actually figured out how much work it is and where it will need to be. Right now, there is a whole universe of MySQL compatible databases that we can go after. We do not want to dilute our focus in time as PlanetScale. As

Vitess open source project, if the community wants that, Sugu would be very happy to supervise a project like that.

[00:50:04] JM: Awesome. Well, guys, I've been really enjoying this conversation and I look forward to talking more in the future.

[END OF INTERVIEW]

[00:50:20] JM: This episode of Software Engineering Daily is sponsored by Datadog. Datadog is a cloud monitoring platform built by engineers for engineers enabling full stack observability for modern applications. Datadog integrates seamlessly to gather metrics and events from more than 400 technologies including cloud providers, databases and webservers. Easily identify slow running queries, error rates, bottlenecks and more fast with built-in dashboards, algorithmic alerts and end-to-end request tracing and log management from Datadog.

Datadog helps engineering teams troubleshoot and collaborate together in one place to enhance performance and prevent downtime. You can start a free trial of Datadog today and Datadog will send you a free t-shirt. Visit softwareengineeringdaily.com/datadog to get started. That's softwareengineeringdaily.com/datadog.

[END]