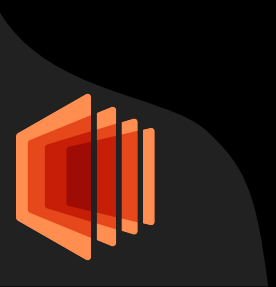

PlanetScale

Introduction to Vitess

sharding framework for MySQL

Matthias Crauwels
Enterprise Customer Engineer, PlanetScale
@mcrauwel



// Introduction

Who am I?

- Living in Ghent, Belgium 🇧🇪
- Bachelor Computer Science
- ~25 years Linux user / admin
- ~15 years PHP developer
- ~10 years MySQL DBA
- 3rd year at PlanetScale
- Enterprise Customer Engineer
- Father of Leander

About PlanetScale

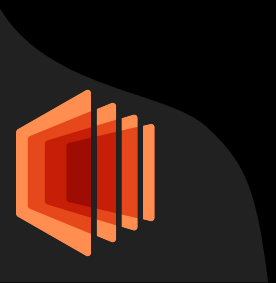
PlanetScale is the world's most scalable and reliable OLTP database. Built on Vitess, our cloud platform is trusted by some of the world's largest brands. We offer a range of deployment options including multi-tenant cloud, single-tenant, or bring your own cloud account with PlanetScale Managed.

PlanetScale was founded in 2018 by the co-creators and maintainers of the Vitess open source project. We are a globally-distributed company that values high performance, accountability, and integrity.

About Vitess

PlanetScale is powered by Vitess, the open-source database technology that was invented at YouTube in 2010 to solve the scaling issues they faced with their massive MySQL database.

Vitess went on to become open source as a CNCF project and continues to scale massive companies like Slack, GitHub, and more.



Vitess serves millions of QPS in production

slack

New Relic

Square

Flipkart

HubSpot

peak

Pinterest

shopify

nozzle

weave

GitHub

JD.COM 京东

Quiz of Kings

stitchlabs

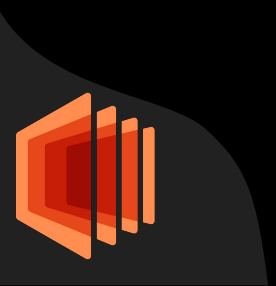
PlanetScale

// Introduction

Agenda

- Architecture
- MySQL Compatibility
- VReplication
- Online Schema Changes
- Automatic failovers

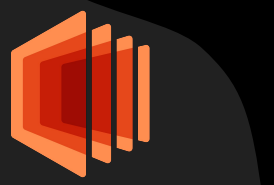
Architecture



// Architecture

What is Vitess?

- Cloud Native Database
- Massively Scalable
- Highly Available
- MySQL Compatible

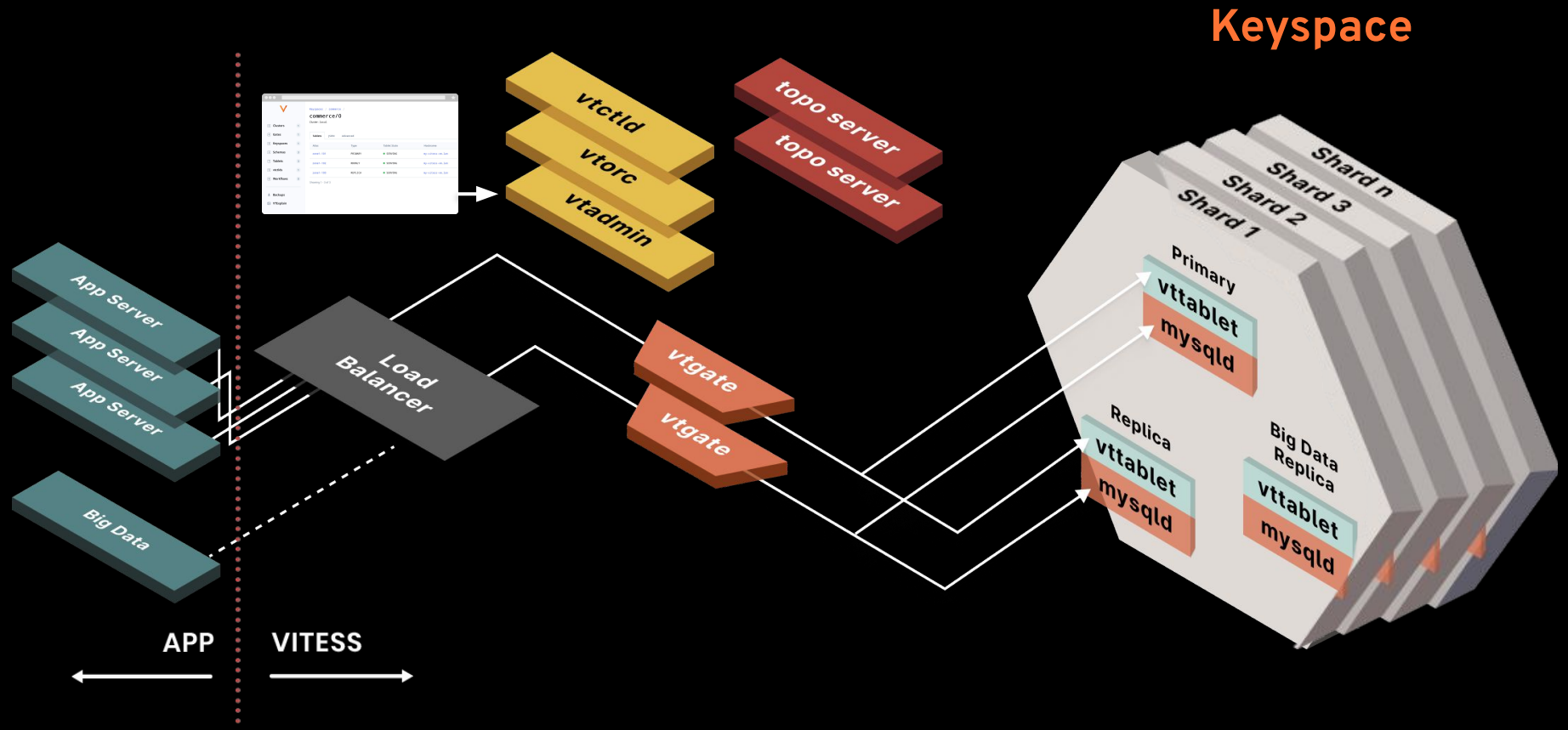


// Architecture

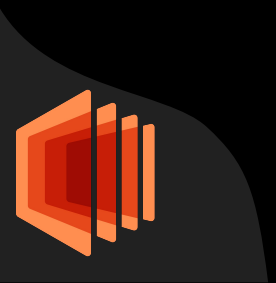
Concepts

- Keyspace
- Shard
- Topology server

Vitess Architecture



MySQL Compatibility



// MySQL Compatibility

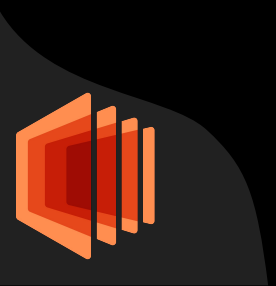
Vitess provides an illusion

- A single database (server)
- MySQL 8.0 and above
- A single, dedicated connection

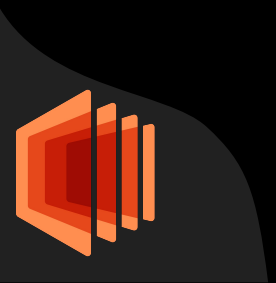
// MySQL Compatibility

Vitess needs to deal with

- Database frameworks
- ORMs
- Legacy code
- Third Party Applications



VReplication



// VReplication

Use cases

- Migration
  - Import data into Vitess
- (Re)sharding
  - Move data around
- Materialisation
  - Improve query performance
  - Materialise unsharded data on each shard

Import workflow

- Create an “external” keyspace with a vttablet pointing to an existing MySQL server.
- Create a Vitess keyspace
- Use the MoveTables command to create a VReplication Workflow to move data into Vitess
- Confirm that all data was migrated (VDiff)
- Switch read traffic to go to Vitess
- Switch write traffic to go to Vitess
- Full support for rollback in case of issues

// VReplication

Regular MySQL as source

- Binary logs are required
  - When copying from a replica GTIDs and `log_replica_updates` are required
  - `binary log format = ROW`
  - `binary log image = FULL`
- 2 phases
  - Copy phase
    - Long running select - vreplication_copy_phase_duration (default 1h)
    - Throttling
      - History List Length - `vreplication_copy_phase_max_innodb_history_list_length (default 1 million)`
      - Replication lag (source) - `vreplication_copy_phase_max_mysql_replication_lag (default 12h)`
  - Catch Up phase
    - After each chunk of the copy phase
    - Apply binary log event for the already copied rows
    - Avoids long catch up phase on large tables

// VReplication

Aurora MySQL as source

- No binary logs on Aurora readers
  - Set up a replica Aurora cluster doing binary log replication from the main cluster
  - Use the primary of this cluster as "replica" tablet
- Server UUID (GTID) is the same for all nodes on the source cluster
  - Stream survives a Aurora failover
- When you don't have GTID (enabling requires an Aurora reboot) repoint vreplication to the primary cluster before cutover
  - `STOP REPLICA` on replica cluster
  - `SHOW BINARY LOG STATUS` on replica cluster
  - ensure vreplication catches up (File/Position from the previous output)
  - `SHOW REPLICA STATUS` on replica-cluster
    - fetch `Source_Log_File` and `Executed_Source_Log_Pos`
  - stop vreplication workflow
  - `UPDATE _vt.replication SET pos = '...', tablet_types = 'PRIMARY' WHERE id = ...` on **target** cluster(s)
  - start vreplication workflow

// VReplication

Sharded MySQL / Vitess as source

- Shard by shard migration
  - Sharded source keyspace
  - Sharded target keyspace
  - Same shard definitions for both source and target
  - Requires vtgate flag `--enable-partial-keyspace-migration`
  - MoveTables stream per shard
    - Ability to copy in segments not to overload uplink
    - Ability to switch-reads and switch-writes individually
      - No "big bang" migration
    - Fully reversible migrations
  - `ShardRoutingRules`
    - to be able to switch traffic on a per-shard basis
  - Executed this with cluster up to 256 shards (not the hard limit)
  - Some manual cleanup at the end

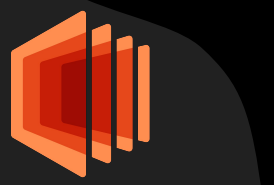
// VReplication

Multi-tenant migrations

- Multiple (identical) schema spread over 1 or more MySQL cluster(s)
  - Unique tenant identifier required in all tables (immutable)
  - Each source tenant is it's own external keyspace
  - Queries should include `WHERE <field_tenant_id> = ...` in all `SELECT/UPDATE/DELETE` statements
  - Target schema
    - Add `<field_tenant_id>` to PRIMARY KEY (required for uniqueness)
    - Add `<field_tenant_id>` to secondary indexes (as first field to use for filtering)
  - HowTo
    - VSchema add multi_tenant_spec to the top-level JSON

```
"multi_tenant_spec": {
      "tenant_id_column_name": "<field_tenant_id>",
      "tenant_id_column_type": "INT64"
  },
```

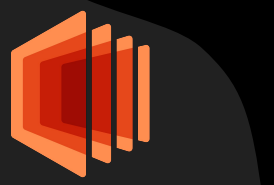
    - MoveTables specify `--tenant_id <value>` during Create
  - Merges 100s or 1000s of schema's into one (sharded) keyspace



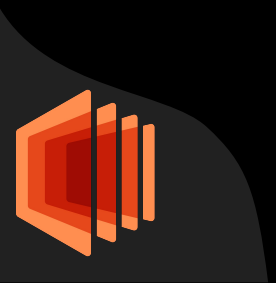
// VReplication

Multi-tenant migrations (2)

- `KeyspaceRoutingRules`
  - Introduced for multi-tenant migrations
  - All keyspaces have the same tables
  - Works very similar to normal `RoutingRules` and `ShardRoutingRules`
- When sharding the Vitess cluster on `<field_tenant_id>`
  - option to specify `--shards` option to MoveTables Create



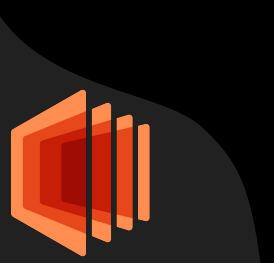
Online Schema Changes



// Online Schema Changes

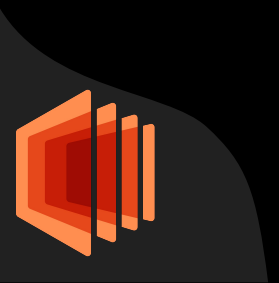
ALTER TABLE issues

- Blocking
- Resource greedy
- Not auditable
- Not interruptible



ALTER TABLE Workarounds

- gh-ost, developed by GitHub
- pt-online-schema-change, part of Percona Toolkit
- External tools
- Requires a lot of operational knowledge of the environment
- Takes control out of the developer's hands

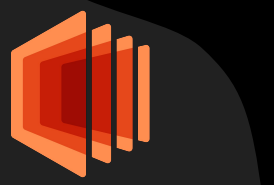


// Online Schema Changes

Based on VReplication

```
mysql> SET @@ddl_strategy = 'online';
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE table1 CHANGE id id bigint unsigned;
+--------------------------------------+
| uuid                                 |
+--------------------------------------+
| 4d8246f2_9801_11ed_a6ae_c87f5403e983 |
+--------------------------------------+
1 row in set (0.02 sec)
```

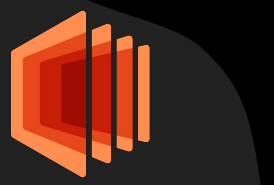
- Grab a coffee!



// Online Schema Changes

Based on VReplication

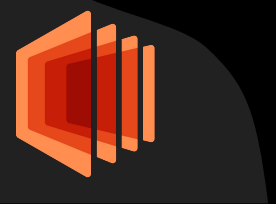
```
mysql> SHOW vitess_migrations LIKE '4d8246f2_9801_11ed_a6ae_c87f5403e983'\G

*************************** 1. row ***************************
...
     migration_uuid: 4d8246f2_9801_11ed_a6ae_c87f5403e983
    added_timestamp: 2023-01-19 14:58:08
completed_timestamp: 2023-01-19 14:58:18
   migration_status: complete
migration_statement: alter table table1 change column id id bigint unsigned
           strategy: online
...
1 row in set (0.00 sec)
```



But there is more…

- Migrations become fully reversible
- Because the VReplication stream could easily be reversed, the old table is kept up-to-date.
- Now a revert operation is as simple as

```
mysql> REVERT vitess_migration '4d8246f2_9801_11ed_a6ae_c87f5403e983';
+--------------------------------------+
| uuid                                 |
+--------------------------------------+
| 20f5337f_9826_11ed_a6ae_c87f5403e983 |
+--------------------------------------+
1 row in set (0.04 sec)
```



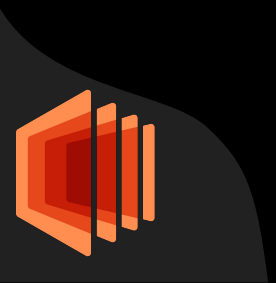
Declarative migrations

- No need to write `ALTER` statements anymore, just resubmit the `CREATE TABLE` statement and Vitess will figure out the difference and run the migration…

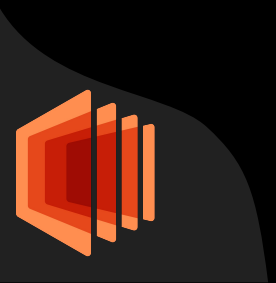
```
mysql> SET @@ddl_strategy = 'online -declarative';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE `table1` (
    ->   `id` bigint unsigned NOT NULL,
    ->   `data` varchar(512) DEFAULT NULL,
    ->   PRIMARY KEY (`id`)
    -> ) ENGINE=InnoDB;
+--------------------------------------+
| uuid                                 |
+--------------------------------------+
| c423f39b_982c_11ed_a6ae_c87f5403e983 |
+--------------------------------------+
1 row in set (0.02 sec)
```

Conclusion

- Schema changes are being put back into the hands of the developers!
- Easy to run
- Easy to revert
- Due to the robustness of VReplication, it can even survive a primary failover or other type of outage



Automatic failovers



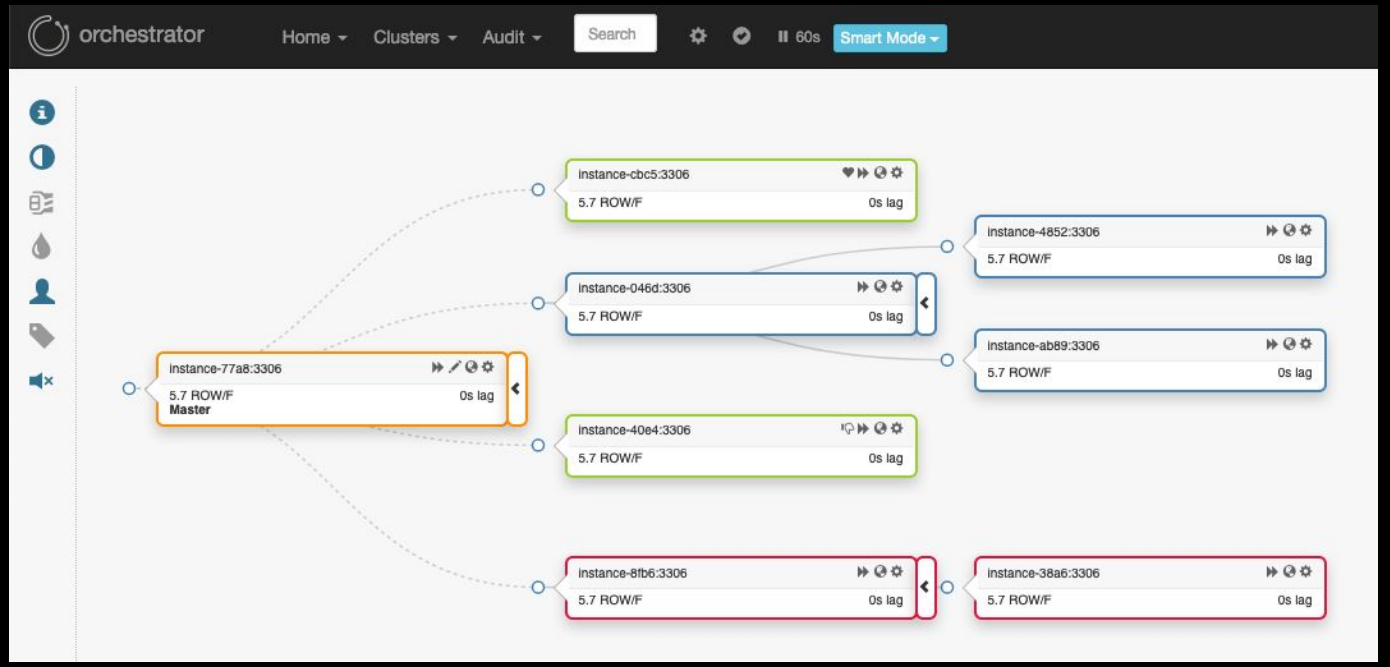
// Automatic failovers

Orchestrator

Orchestrator is open-source software commonly used as a MySQL Topology management tool. It's purpose is to observe MySQL replication topologies and potentially fix these topologies when failures are observed.

Core functionalities:

- Discover
- Visualize
- Monitor
- Refactor

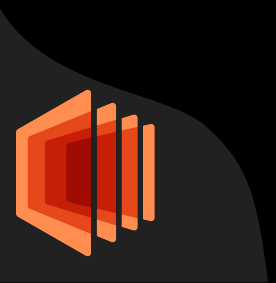


// Automatic failovers

VTOrc

- Vitess component based on Orchestrator
- Stateless, topology information is stored in the topology service
- Discovery is already taken care of by Vitess
- Refactoring will automatically update the topo
- Requirements
  - At least one replica
  - Optional: semi-sync replication

Questions?



Thank you!