Open Source Database Infrastructure with Vitess

Shlomi Noach

PlanetScale

FOSDEM 2021

About me

Engineer at **PlanetScale**

Author of open source projects **orchestrator**, **gh-ost**, **freno** and others

Maintainer for **Vitess**

Blog at http://openark.org

github.com/shlomi-noach
@ShlomiNoach

planetscale

Founded Feb. 2018 by co-creators of Vitess

~45 employees

HQ Mountain View, remote team

Vitess

A database clustering system for horizontal scaling of MySQL

- CNCF graduated project
- Open source, Apache 2.0 licence
- Contributors from around the community

Agenda

Vitess architecture overview

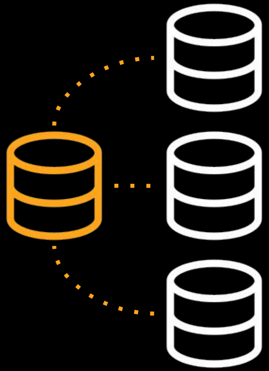
Database infrastructure; experimental and in development:

- Throttling
- Table life cycle
- Online DDL
- HA/failovers

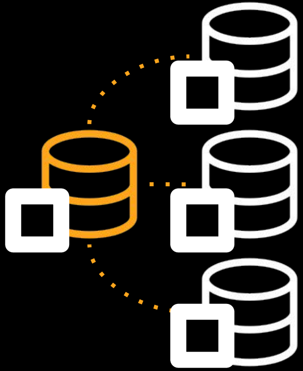
Vitess architecture basics

How the Vitess architecture enables transparent database infrastructure operations

Vitess architecture basics



Consider a common replication cluster



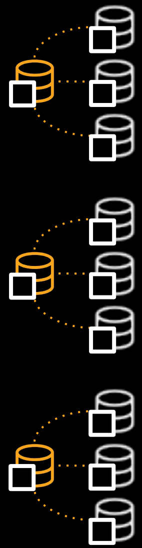
Vitess architecture basics

Each MySQL server is assigned a **vttablet**

- A daemon/sidecar
- Controls the **mysqld** process
- Interacts with the **mysqld** server
- Typically on same host as **mysqld**

Vitess architecture basics

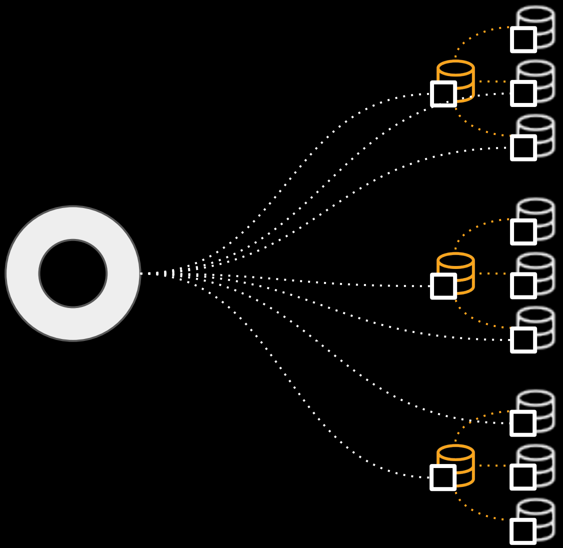
In production you have multiple clusters



Vitess architecture basics

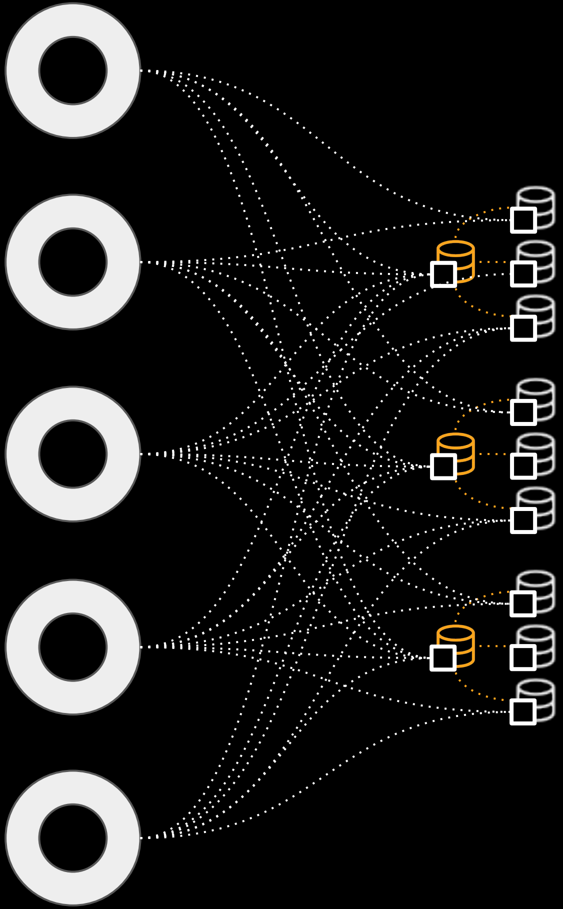
User and application traffic is routed via **vtgate**

- A smart, stateless proxy
- Speaks the MySQL protocol
- Impersonates as a monolith MySQL server
- Relays queries to **vttablet**s



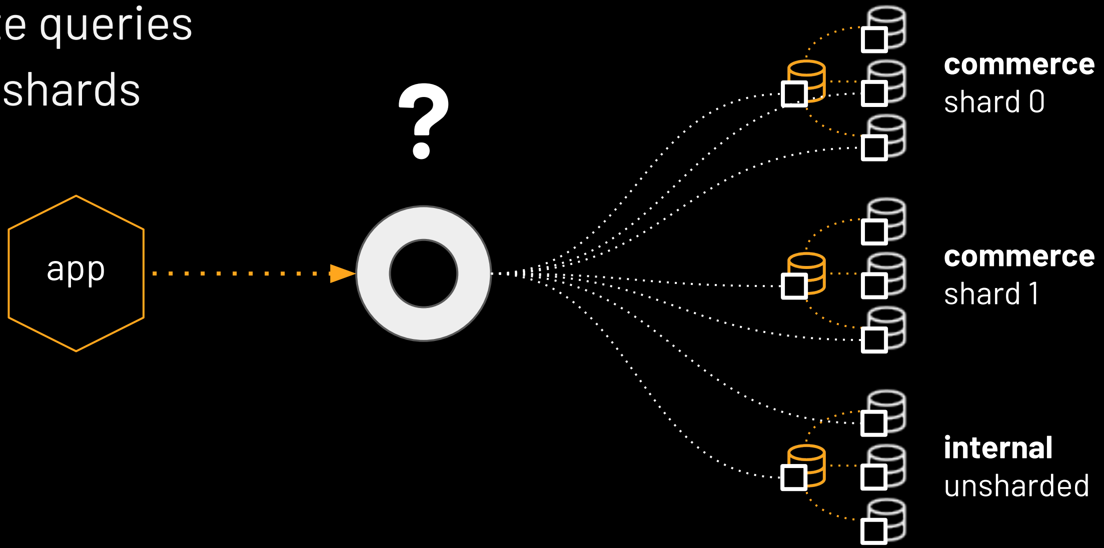
Vitess architecture basics

A vitess deployment will run multiple **vtgate** servers for scale out



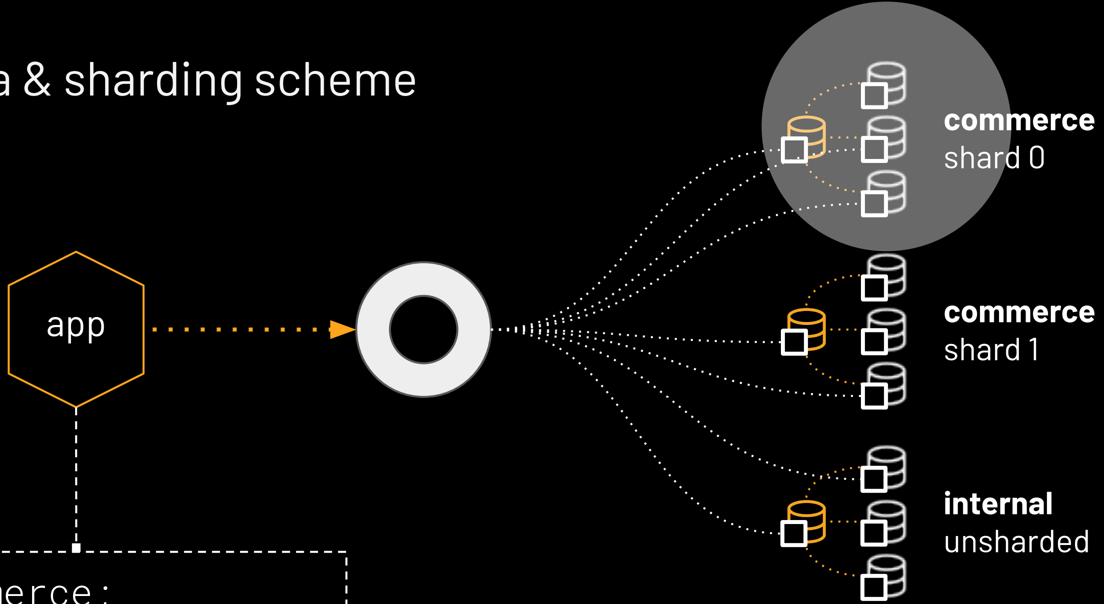
Vitess architecture basics

vtgate must transparently route queries to correct clusters, to relevant shards



Vitess architecture basics

Queries route based on schema & sharding scheme

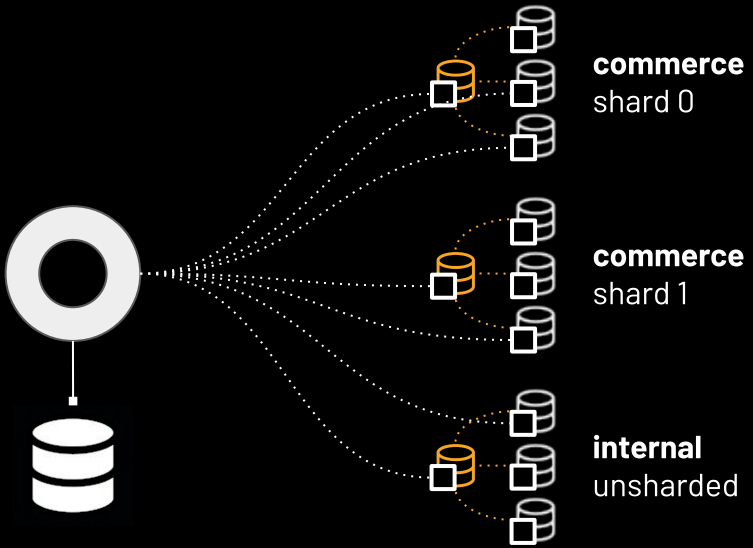


```
USE commerce;
SELECT order_id, price
  FROM orders
  WHERE customer_id=4;
```

Vitess architecture basics

topo: distributed key/value store

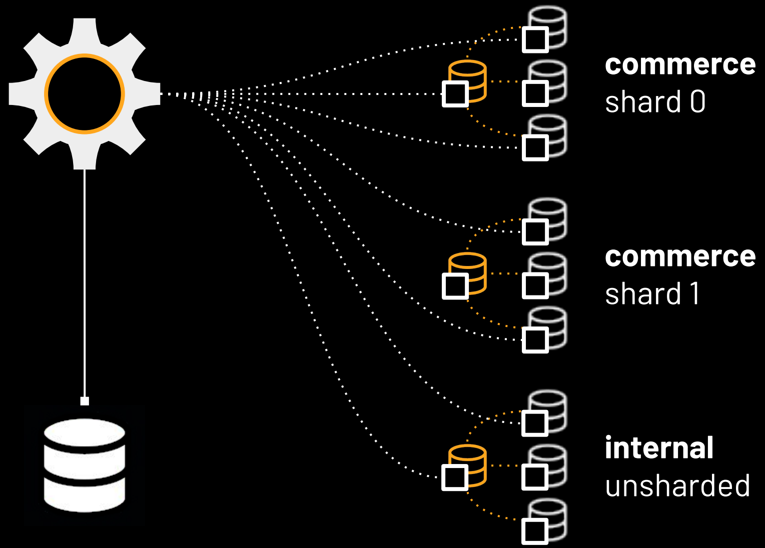
- Stores the state of vitess: schemas, shards, sharding scheme, tablets, roles, etc.
- etcd/consul/zookeeper
- Small dataset, mostly cached by **vtgate**



Vitess architecture basics

vtctld: control daemon

- Runs ad hoc operations
- API server
- Reads/writes **topo**
- Uses locks
- Operates on tablets



Throttling

Pushback for massive writes, maintain low replication lag.

Based on GitHub's **freno**, **github.com/github/freno**, a cooperative throttling service

Implemented in **vttablet**

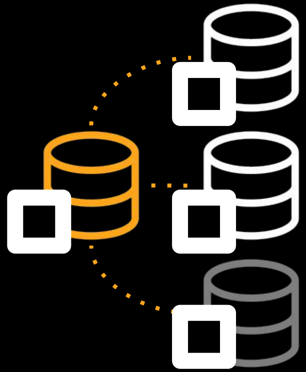
https://vitess.io/docs/reference/features/tablet-throttler/

Throttling

Based on replication lag

Vitess has an internal heartbeat mechanism, similar to **pt-heartbeat**, injecting **TIMESTAMP** records on the primary, read on replicas

Throttling

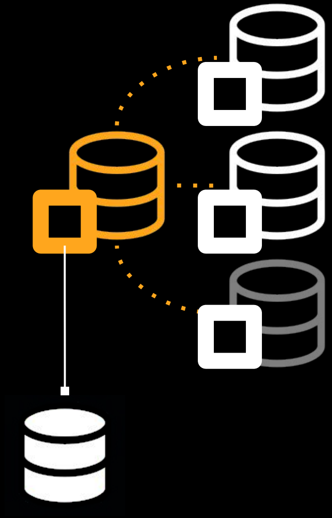


Vitess is knowledgeable about servers in a cluster:

- Primary
- Replica
- Non serving replica (OLAP)
- Backup servers

By default, vitess only takes into account lag on serving replicas. Override with:

```
vttablet -throttle_tablet_types=...
```



vttablet throttler

The primary tablet of each shard (MySQL replication cluster) polls relevant replicas for lag

Periodically consults **topo** for changes in replication topology and tablet roles

Serves HTTP API endpoint: **`/throttler/check`**

- Returns **`HTTP 200 OK`** when lag is good
- Other HTTP codes to pushback writes

vttablet throttler

Implemented internally:

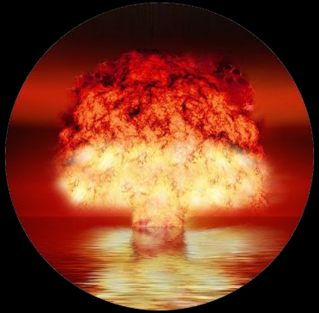
- Table lifecycle
- Online DDL

Ideas for the future:

- Enforce throttling for massives updates, e.g.:
  `UPDATE my_table SET new_colunm=price*rate`

Table lifecycle

An automated garbage collector for old tables



DROP TABLE here_be_trouble;

DROP TABLE alternatives

- **RENAME TABLE TO _something_else** for quick recovery in case of regret
- Purge table data, possibly with **SQL_LOG_BIN=0** Requires throttling, best avoid concurrent purges.
- Wait X days till table pages are evicted from buffer pool
- Actually **DROP**
- Potentially directly **TRUNCATE** on replicas
- Or use **BLACKHOLE** hacks

How do you automate/manage/track all these?

Vitess table lifecycle

A table can be in one of these states:

- *In use*
- **HOLD**: renamed and kept intact for X days
- **PURGE**: rows actively being purged
- **EVAC**: wait X days to evict pages from buffer pool
- **DROP**: ready for an actual DROP TABLE
- *Gone*

Vitess table lifecycle

Examples:

- **_vt_HOLD_6ace8bcef73211ea87e9f875a4d24e90_20210130093000**
  Table held intact until **2021-01-30 09:30:00**, then transitioned into next phase
- **_vt_PURGE_6ace8bcef73211ea87e9f875a4d24e90_20210131182000**
  Table is in purging process. It will transition into next phase once it is completely purged
- **_vt_EVAC_6ace8bcef73211ea87e9f875a4d24e90_20210207071500**
  Table remains in evac until **2021-02-07 07:15:00**, then transitioned into next phase

Purging tables

vttablet on primary is charged with purging table data

- Single table at a time
- **DELETE FROM <table> LIMIT 50** in iterations
- **SQL_LOG_BIN=0**
- Using *tablet throttler*, low priority requests

Vitess table lifecycle

With table name encoding scheme:

- The process is stateless
- Vitess auto-discovers relevant tables
- Will always do the right thing
- We do lose context to the original table

Vitess table lifecycle

Transition states controlled by:

```
vttablet -table_gc_lifecycle=<states>
```

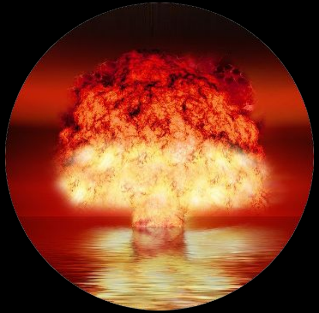
Examples:

- “`hold,purge,evac,drop`” (the default)
- “`hold,drop`”: keep intact for X days, then drop
- “`drop`”: just drop

https://vitess.io/docs/reference/features/table-lifecycle/

Online DDL

Schema changes made easy



```
ALTER TABLE here_be_trouble
ADD COLUMN i INT NOT NULL;
```

ALTER TABLE alternatives

pt-online-schema-change and **gh-ost**, adding operational complexity:

- External tools
- Remote login
- Discovery
- Accounts
- Scheduling
- Formalize, execute
- Throttling
- Tracking
- Interrupting

Operational complexity

Often outside ownership of the developers

Online DDL

Vitess’ architecture can own most of the complexity:

- External tools: executed by **vttablet**
- Remote login: not required
- Discovery: **vitess** knows the topology
- Accounts: **vttablet** can create on your behalf
- Scheduling: use **topo** to coordinate migrations
- Formalize, execute: **vttablet**, on primary server
- Throttling: using *tablet throttler*
- Tracking: via vitess infrastructure
- Interrupting: via vitess infrastructure

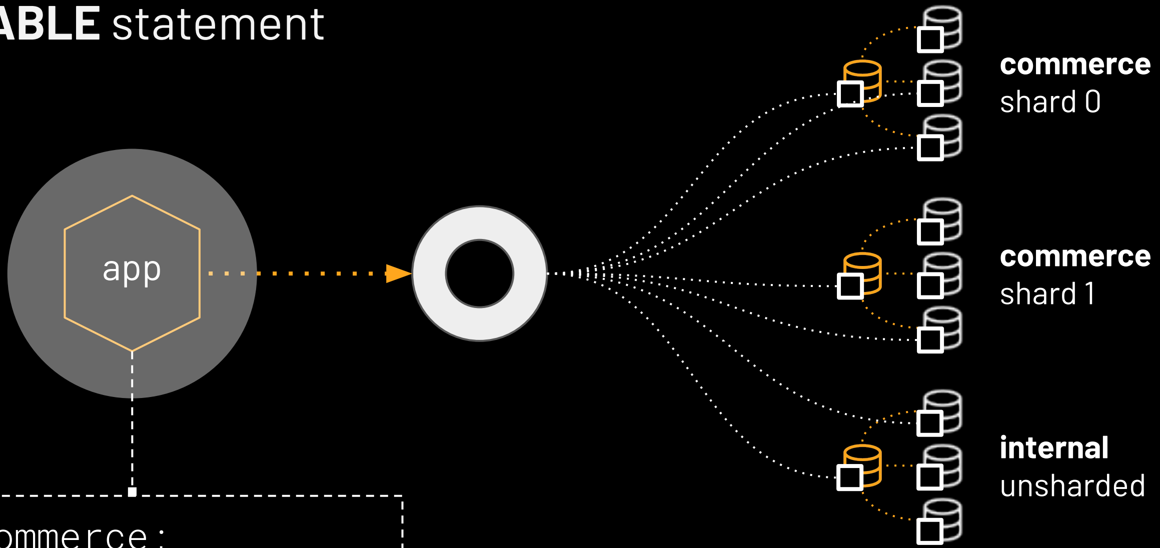
Online DDL

```
mysql> SET @@ddl_strategy=’gh-ost’; -- also ‘pt-osc’

mysql> ALTER TABLE no_problem
       ADD COLUMN i INT NOT NULL;
+--------------------------------------+
| uuid                                 |
+--------------------------------------+
| 7e9cd911_4b37_11eb_a80f_f875a4d24e90 |
+--------------------------------------+
1 row in set (0.01 sec)
```

Online DDL: flow

Application issues **ALTER TABLE** statement

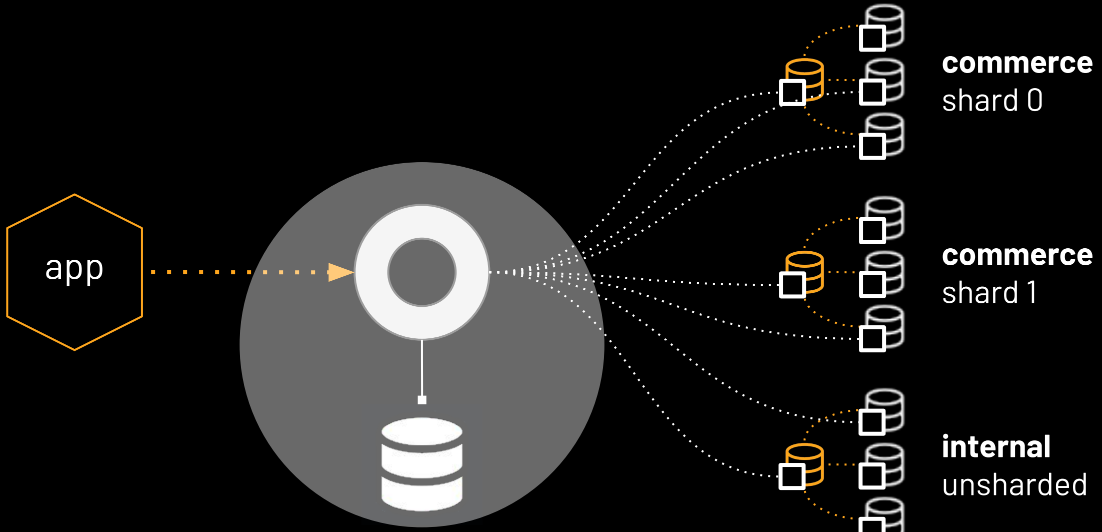


```
USE commerce;
ALTER TABLE orders ADD
COLUMN due_date
TIMESTAMP NOT NULL;
```

Online DDL: flow

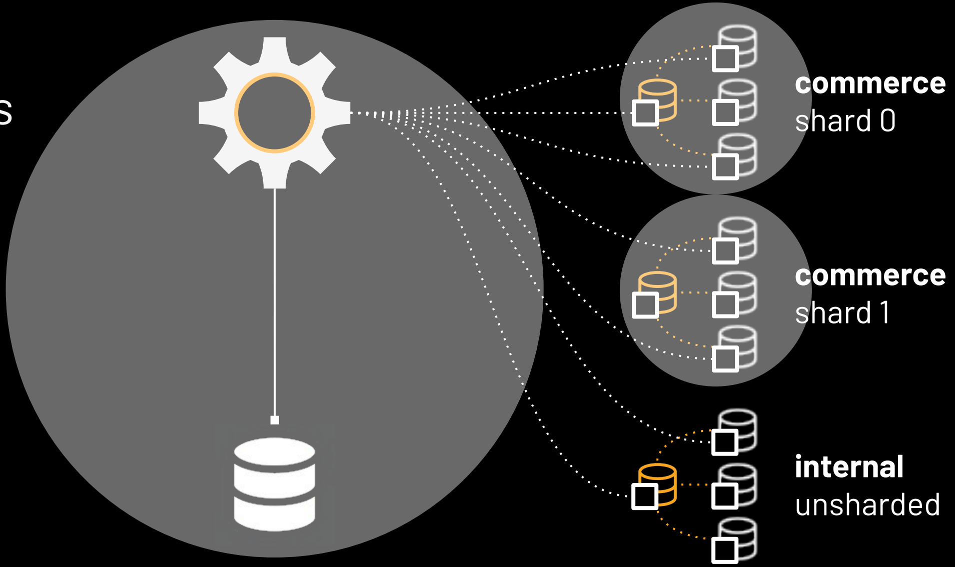
vtgate receives statement, but does not pass it on to tablets.

Instead, it notes the migration request in **topo**



Online DDL: flow

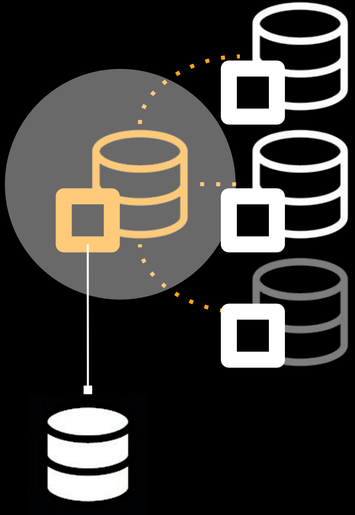
vtctld detects migration requests and ensures distribution to relevant shards



Online DDL: flow

vttablet on primary receives migration request from **vtctld**

- Persists internally
- Schedules
- Prepares script
- Creates one-off credentials
- Runs **gh-ost** or **pt-osc**
- Uses *tablet throttler*
- Tracks
- Cleans up
- Feeds artifact tables into the garbage collector



Online DDL: track, cancel, retry

```
$ vtctlclient OnlineDDL commerce show 8a797518_f25c_11ea_bab4_0242c0a8b007
+-----------------+-------+--------------+-------------+------------+--------------------------------------+----------+---------------------+---------------------+------------------+
|     Tablet      | shard | mysql_schema | mysql_table | ddl_action |            migration_uuid            | strategy |  started_timestamp  | completed_timestamp | migration_status |
+-----------------+-------+--------------+-------------+------------+--------------------------------------+----------+---------------------+---------------------+------------------+
| test-0000000401 | c0-   | vt_commerce  | demo        | alter      | 8a797518_f25c_11ea_bab4_0242c0a8b007 | gh-ost   | 2020-09-09 05:23:32 |                     | running          |
| test-0000000201 | 40-80 | vt_commerce  | demo        | alter      | 8a797518_f25c_11ea_bab4_0242c0a8b007 | gh-ost   | 2020-09-09 05:23:32 | 2020-09-09 05:23:33 | complete         |
| test-0000000301 | 80-c0 | vt_commerce  | demo        | alter      | 8a797518_f25c_11ea_bab4_0242c0a8b007 | gh-ost   | 2020-09-09 05:23:32 |                     | running          |
| test-0000000101 |   -40 | vt_commerce  | demo        | alter      | 8a797518_f25c_11ea_bab4_0242c0a8b007 | gh-ost   | 2020-09-09 05:23:32 |                     | running          |
+-----------------+-------+--------------+-------------+------------+--------------------------------------+----------+---------------------+---------------------+------------------+
```

```
$ vtctlclient OnlineDDL commerce cancel 2201058f_f266_11ea_bab4_0242c0a8b007
+-----------------+--------------+
|     Tablet      | RowsAffected |
+-----------------+--------------+
| test-0000000401 |            1 |
| test-0000000101 |            1 |
| test-0000000201 |            1 |
| test-0000000301 |            1 |
+-----------------+--------------+
```

```
$ vtctlclient OnlineDDL commerce retry 2201058f_f266_11ea_bab4_0242c0a8b007
+-----------------+--------------+
|     Tablet      | RowsAffected |
+-----------------+--------------+
| test-0000000101 |            1 |
| test-0000000201 |            1 |
| test-0000000301 |            1 |
| test-0000000401 |            1 |
+-----------------+--------------+
```

Online DDL: more than ALTER

CREATE and **DROP** statements can also participate in online DDL logic. Both go through **topo** and scheduled by **vttablet**, can be tracked, cancelled, etc.

In fact, **DROP** statements are modified to **RENAME** statements, e.g.:

```
mysql> DROP TABLE i_hope_nobody_uses_this;
```

Intercepted by **vtgate** and transformed into:

```
RENAME TABLE i_hope_nobody_uses_this TO
_vt_HOLD_b0d1fb34450a11ebb980f875a4d24e90_20210203094500;
```

Online DDL

Puts ownership back in the hands of the developers

- Zero dependencies using **gh-ost** on **linux_amd64** (comes with **gh-ost** precompiled)
- Auto retry in case of failover

Future work:

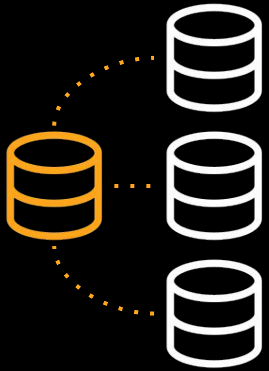
- Use **vreplication** instead of **gh-ost**/**pt-osc**
- Continuously migrate while resharding while reparenting

https://vitess.io/docs/user-guides/schema-changes/managed-online-schema-changes/

vtorc

Orchestrator integration

MySQL replication clusters



- Are not primitives
- Are not identifiable
- Only exist as meta information

But mean everything to us!

orchestrator’s approach

- Observe
- Accept reality
- Assign metadata such as *cluster alias*
- Detect failure, failover

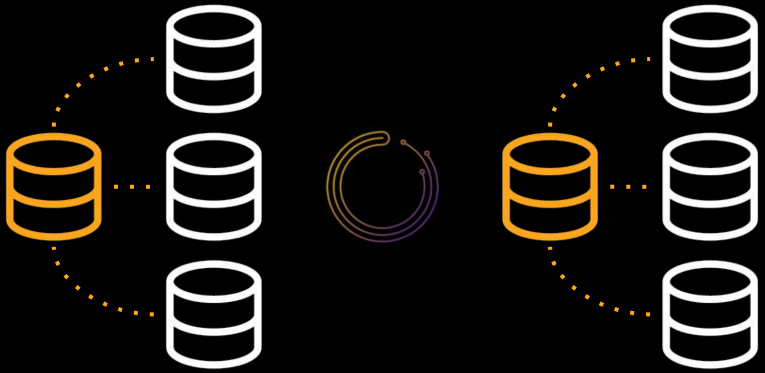
But otherwise does not know if the cluster meets your product expectation

Common example

After a split brain scenario and failover we end up with two distinct clusters. Which one is the “real” production cluster?

MySQL does not know

orchestrator uses heuristics based on failover history and bookkeeping



Vitess knows

Vitess keeps known schemas, shards, clusters, server roles, all in **topo**

It keeps a *state*

The old vitess-orchestrator integration

Works, until it doesn’t:

- Conflicting operations
- Conflicting opinions
- Too much information need to pass back and forth

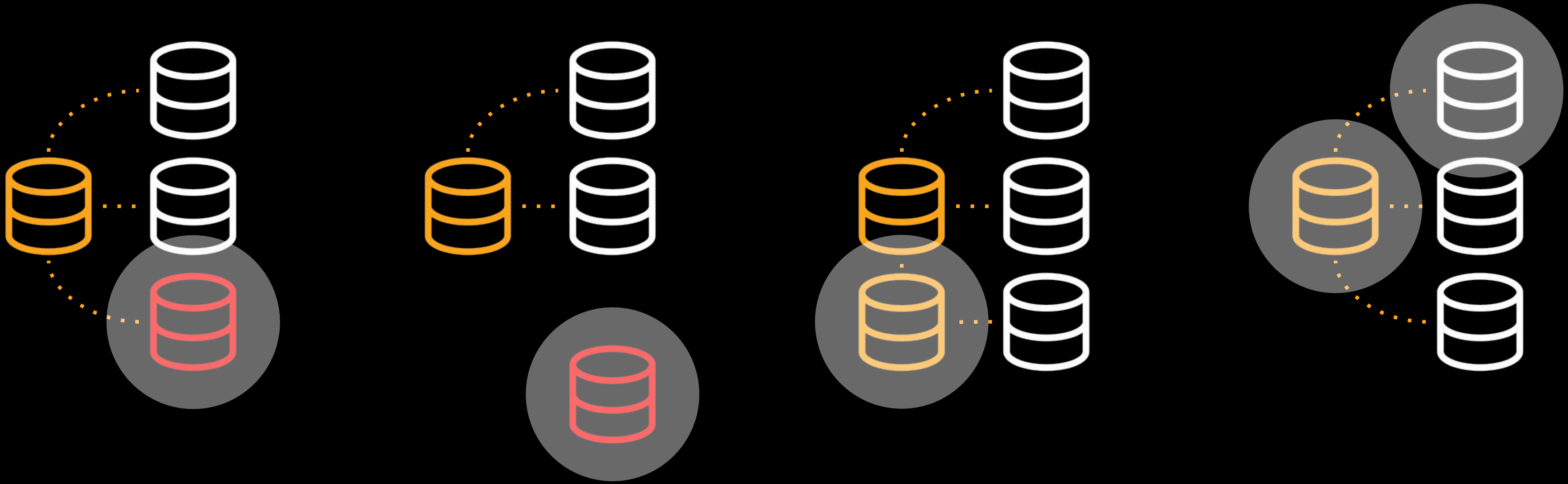
vtorc

An **orchestrator** spin-off, tightly integrated within **vitess**.

Has direct access to **topo**

Is goal oriented. Its mission is to make replication clusters converge to **vitess**’ expected state

vtorc scenarios, superseding orchestrator scenarios



vtorc

Work in progress

Future:

- Custom defined availability/durability rules (imply failover rules, semi-sync rules etc.)

Database infrastructure

Vitess becomes a database infrastructure framework in an attempt to reduce overall relational database complexity

Resources

Docs: vitess.io/docs/

Code: github.com/vitessio/vitess

Slack: vitess.slack.com

Thank you!

Questions?

github.com/shlomi-noach

@ShlomiNoach