How to migrate a MySQL Database to Vitess

Liz van Dijk - @lizztheblizz

planetscale

Who am I?

Liz van Dijk - Solution Architect

- Howest University > Percona > PlanetScale
- New to Vitess, MySQL has been my world for 8+ years
- liz@planetscale.com
- @lizztheblizz

PlanetScale

- Founded in February 2018
- Venture backed: a16z, SignalFire
- 30 employees mostly in Mountain View, CA

What is Vitess?

Vitess

Cloud Native

Massively Scalable

Highly Available

Based on MySQL

Vitess Stats

Started
2010

CNCF
Graduated

200
Contributors

9,500
Stars

v5.0
(4 Feb 2020)

18,000
Commits

1100
Slack Members

1200
Forks

Key Adopters

YouTube

slack *

JD.京东.COM

Square *

Pinterest *

HubSpot *

nozzle *

GitHub

New Relic

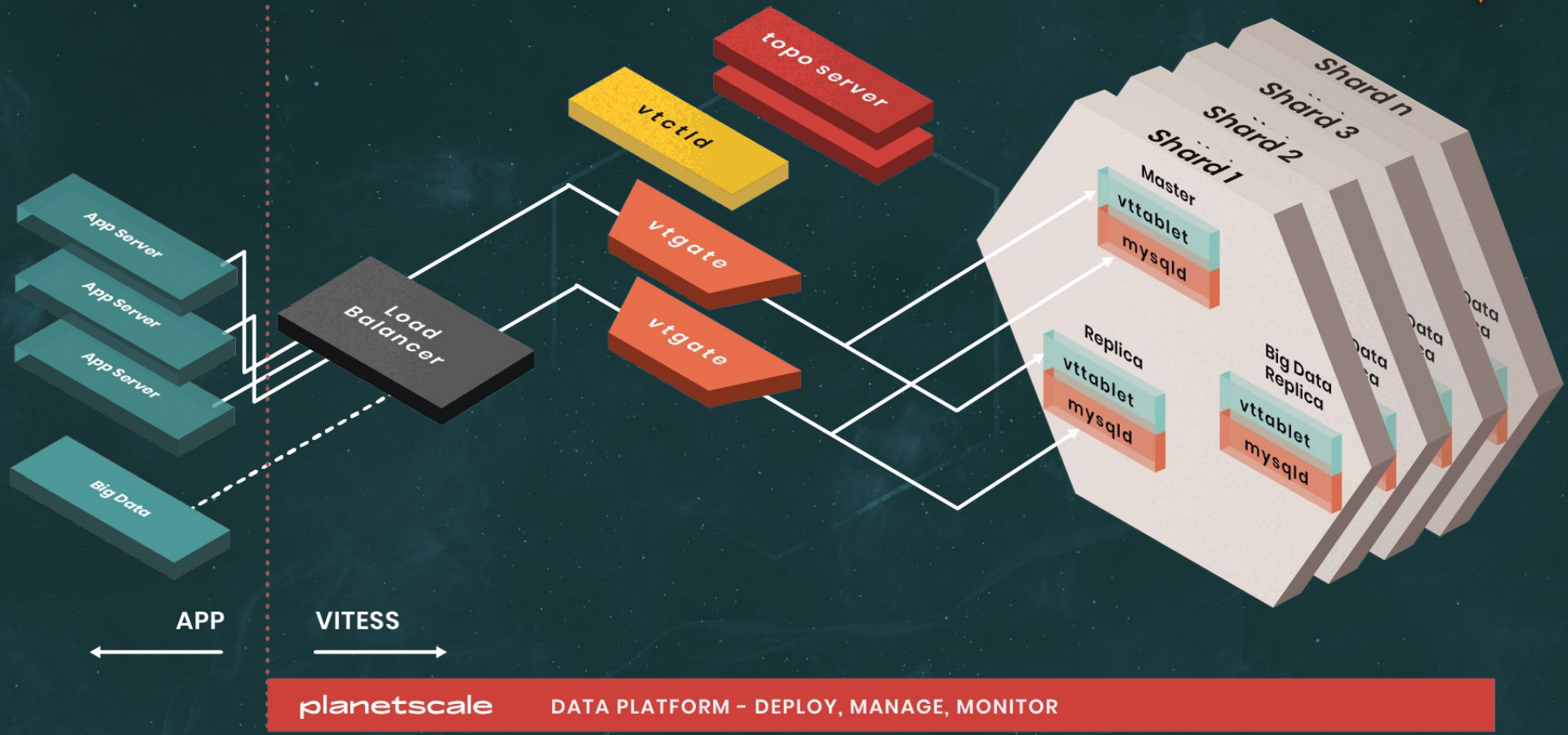
Flipkart

peak GAMES

stitchlabs

* Employing active project maintainers

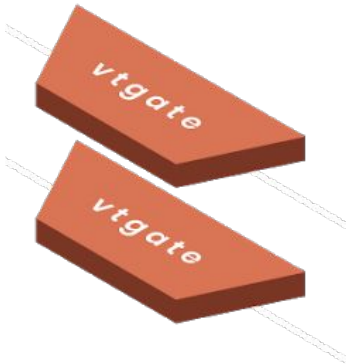
Architecture



planetscale DATA PLATFORM - DEPLOY, MANAGE, MONITOR

VTGate

- VTGate is a stateless proxy, and the entry point into the cluster
- Can be connected to and presents the cluster as a monolithic database
- Interprets SQL and supports Vitess-specific hints



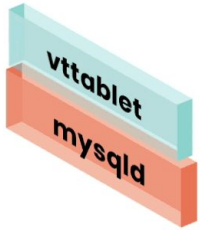
https://vitess.io/docs/concepts/vtgate/

Keyspace & Shards

- Keyspace is an analog to what we call a *logical database*.
- Keyspaces consist of one or multiple Shards.
- Shards contain one or more replica tablets, of which one will be elected as master.
- Adding shards compartmentalizes risk.

https://vitess.io/docs/concepts/keyspace/
https://vitess.io/docs/concepts/shard/

Vitess Tablet

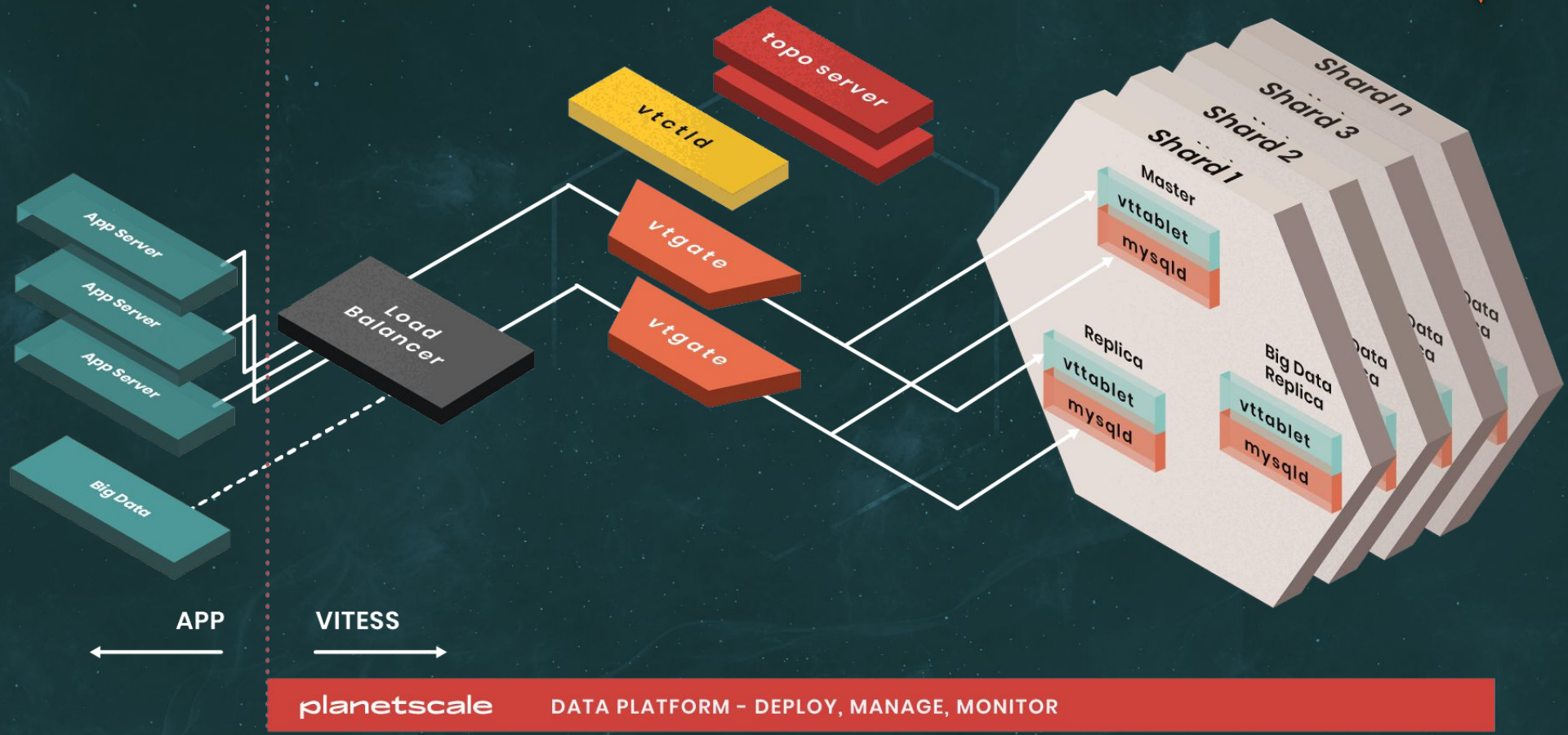


- Most basic “worker” unit of a Vitess Cluster
- MySQL Server may be any flavor
- VTTablet is a *sidecar process*
- Tablets can fulfill multiple roles
  - Master
  - Replica
  - Analytics Replica

https://vitess.io/docs/concepts/tablet/


DEMO

Architecture



planetscale DATA PLATFORM - DEPLOY, MANAGE, MONITOR

Migration Plan (Dev/QA)

1. Test Vitess for Query Compatibility

2. Test Application for added Latency

3. Backout if any issues discovered

Query Compatibility

- Getting started with **`vtexplain`**
- If you can, use a normalized query list from Prod
  - If not, grab a representative sample using the Slow Query Log with **`long_query_time`** set to `0`
  - Use **`pt-query-digest`** to extract a normalized list of queries
- Other required ingredients:
  - **`vtexplain`** - a stand-alone program that utilizes much of the parsing and routing logic in Vitess
  - **`schema.sql`** - your schema design for the Vitess cluster
  - **`vschema.json`** - the same you would use with an actual cluster, with sharding and vindex instructions

vtexplain in a nutshell

Schema.sql: (in keyspace **ks1**)

```
CREATE TABLE `foo` ( `id` int(11), `data` int(11), PRIMARY KEY (`id`)) ;
CREATE TABLE `bar` ( `id` int(11), `data` int(11), PRIMARY KEY (`id`)) ;
```

Two simple tables each with a an **id** column and a **data** column that we shard on.
So we have two shards: **ks1:-80** and **ks1:80-**

```
vtexplain  -schema-file  schema.sql     \
           -vschema-file vschema.json   \
           -shards 2                    \
           -sql 'SELECT foo.id FROM foo JOIN bar
                 ON  foo.data = bar.data
                 AND foo.data = 7'
```

output:

```
----------------------------------------------------------------
SELECT foo.id FROM foo JOIN bar
      ON foo.data = bar.data
      AND foo.data = 7

1 ks1/80-: select foo.id from foo join bar on foo.data =
bar.data and foo.data = 7 limit 10001

----------------------------------------------------------------
```

The above output indicates that the query was sharded, because it is sent only to **ks1:80-** , where the value **7** would be hashed according to the vindex clause. If the query instead said "AND **foo.data** < 95" the output would show a scatter result, with the query being sent to both shards.

vschema.json:

```
{
  "ks1": {
    "sharded": true,
    "tables": {
      "foo": {
        "column_vindexes": [
          {
            "column": "data",
            "name": "hash"
          }
        ]
      },
      "bar": {
        "column_vindexes": [
          {
            "column": "data",
            "name": "hash"
          }
        ]
      }
    },
    "vindexes": {
      "hash": {
        "type": "hash"
      }
    }
  }
}
```

vtexplain Summarized

- Detects unsupported SQL syntax
- Reports ambiguous query constructs
- Validates vSchema by predicting sharding behavior of all queries

vtexplain further example

Initial query:

```
SELECT   pref_codes.id
  FROM   pref_codes
  WHERE  pref_codes.subject_id`  = 9
  AND    pref_codes.location_id` = 12204
  AND
   (reserved_at is null
      AND not exists
        ( SELECT 'x' FROM reject where reject_code_id =pref_codes.id )
   )
  ORDER BY pref_codes.id LIMIT 40;
```

Output:

```
unsupported: cross-shard correlated subquery (scattered subquery was attempted)
```

Altered to succeed and produce a sharded query where **subject_id** is the sharding key

```
SELECT  pref_codes.id
  FROM  pref_codes
  WHERE pref_codes.subject_id  = 9
  AND   pref_codes.location_id = 12204
  AND   pref_codes.reserved_at IS NULL
  AND   pref_codes.id NOT IN
          ( SELECT reject_code_id
          FROM   reject
          WHERE  reject_code_id =pref_codes.id              -- references upper query
          AND    subject_id     =pref_codes.subject_id      -- references upper query
      )
  ORDER BY pref_codes.id LIMIT 40;
```

Output: (a sharded query)

```
1 ks1/-80: select pref_codes.id from pref_codes where pref_codes.subject_id = 9 and pref_codes.location_id =
12204 and pref_codes.reserved_at is null and pref_codes.id not in (select reject_code_id from reject where
reject_code_id = pref_codes.id and subject_id = pref_codes.business_id) order by pref_codes.id asc limit 40
```


DEMO

Query Compatibility (2)

- Even if everything looks perfect in vtexplain, start by building out a Dev/QA environment
- Typical Gotchas
  - Some applications have user-generated queries that are hard to predict
  - Unexpected performance regressions
  - Third party plugins/connectors (CDC, analytics, etc.)

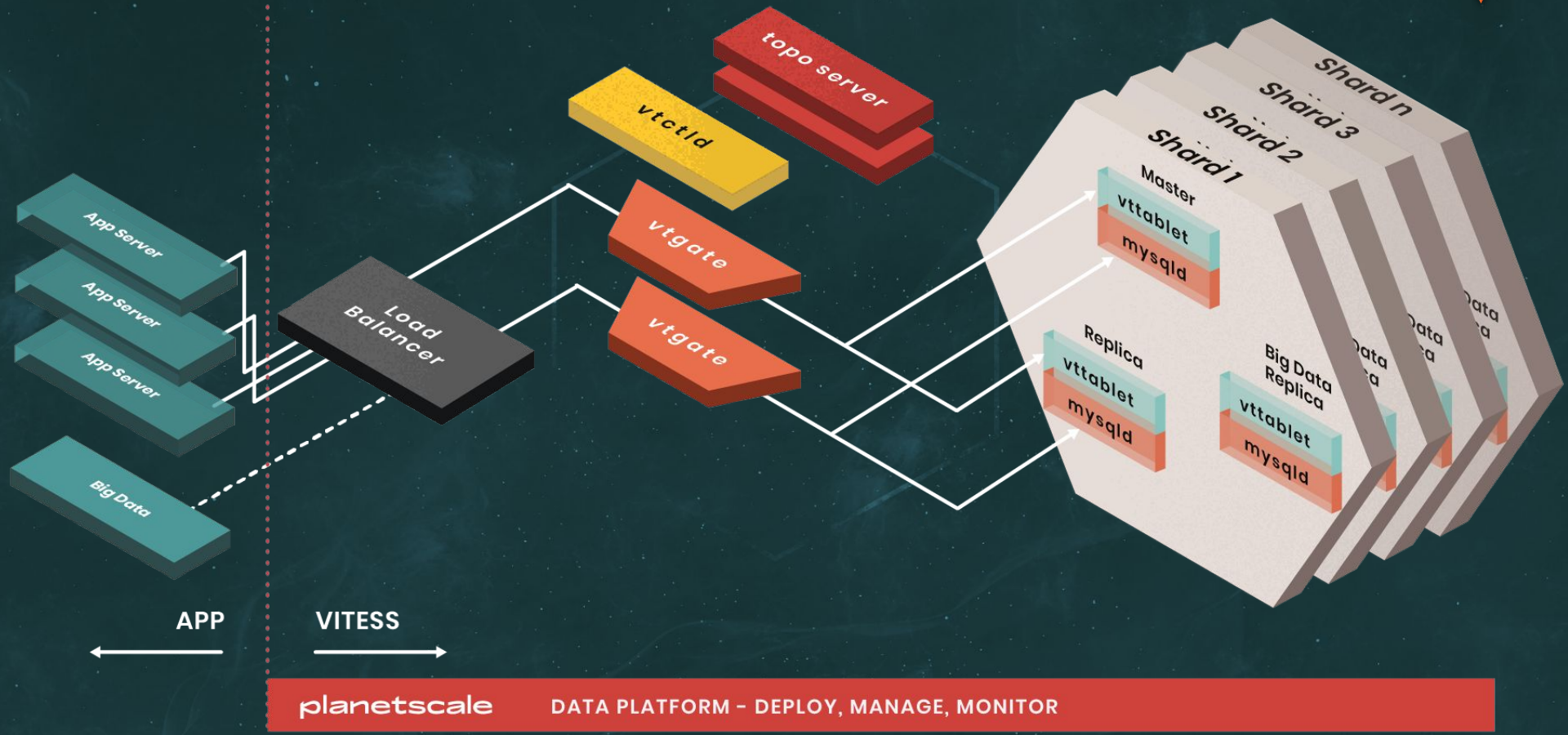
Added Latency

- Vitess requires one more network hop than MySQL (VTGate Proxy)
- Simple back of napkin math: +1-2ms on each query
- Should be within tolerable threshold for most Apps
  - Edge cases are N+1 pattern, typically not well designed apps.

Backout Plan

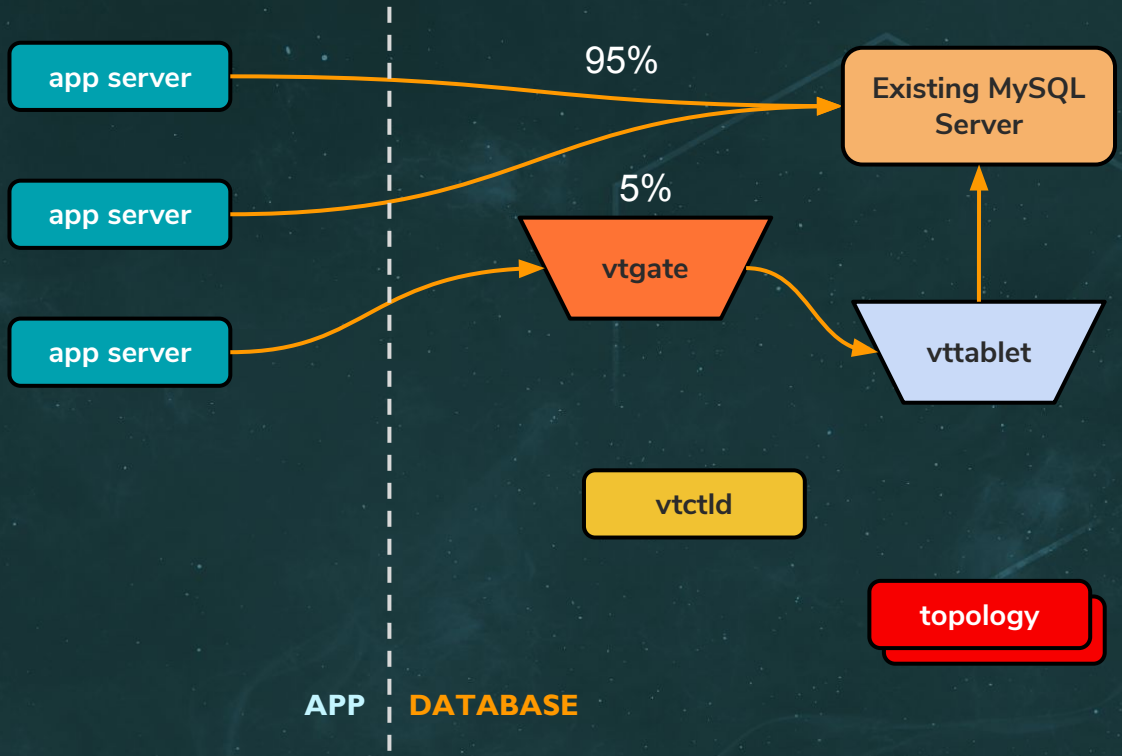
- We've verified in Dev/QA that our App works with Vitess
  - No observable errors or problems
- Good ops practice is to use a Canary
  - Migrate just 5% of our traffic to Vitess
  - Rollback if any issues
  - Works great with Vitess!

Architecture

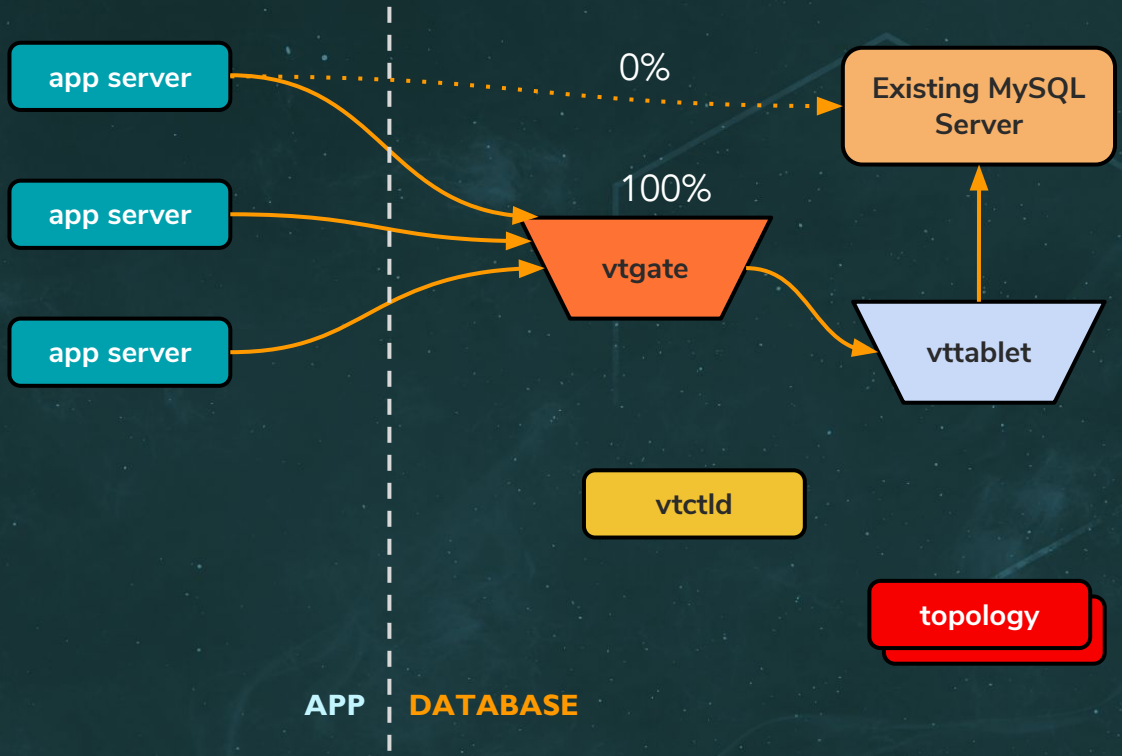


planetscale DATA PLATFORM - DEPLOY, MANAGE, MONITOR

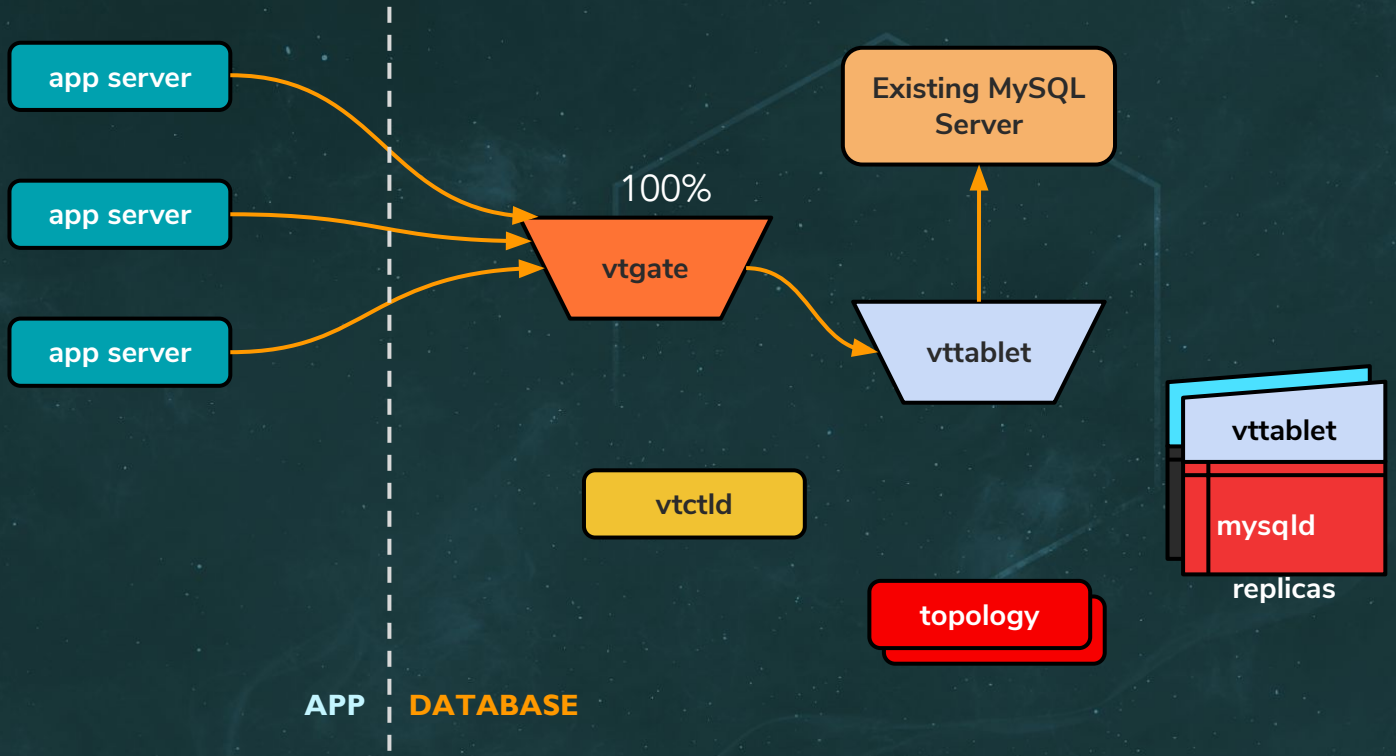
Canary Deployment: Phase 1



Phase 1 Completed!



Phase 2: Add a New Tablet



Phase 2: Table Migration

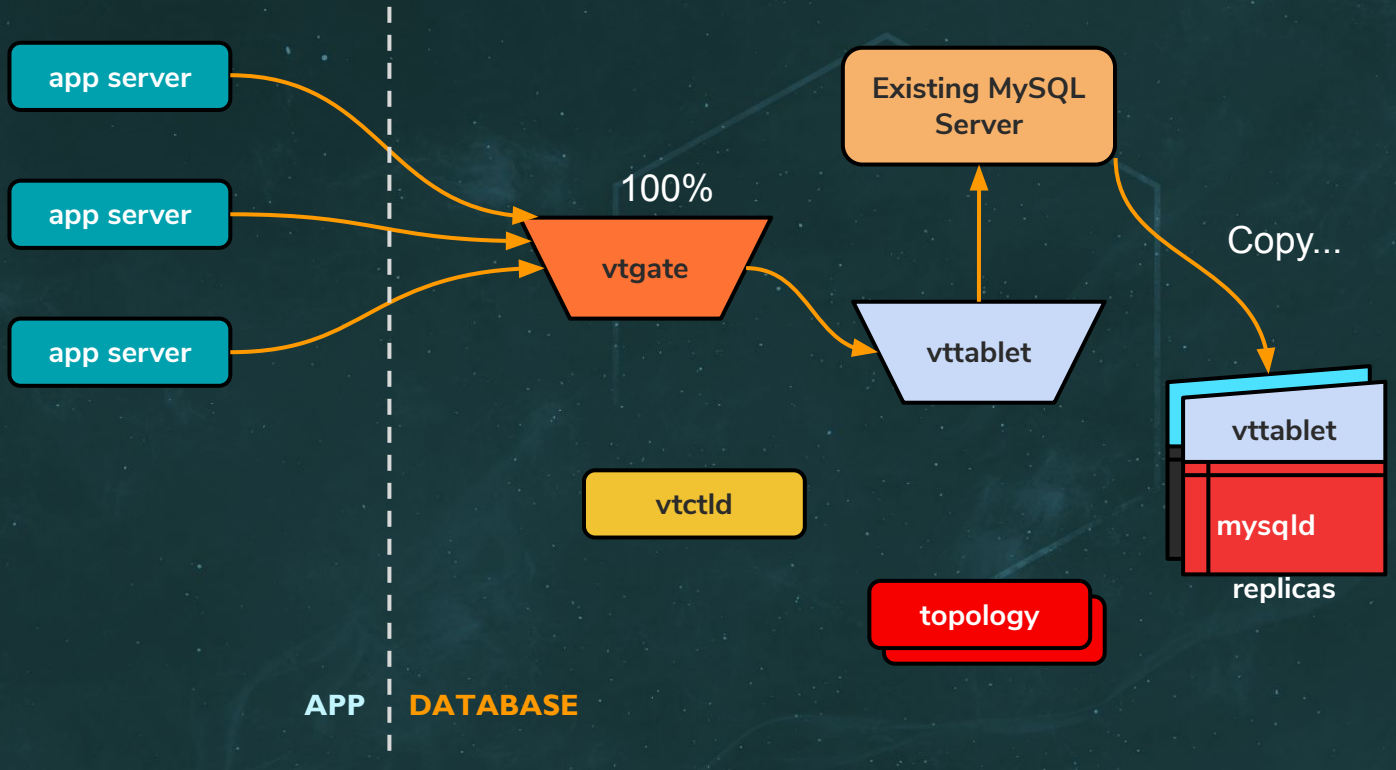
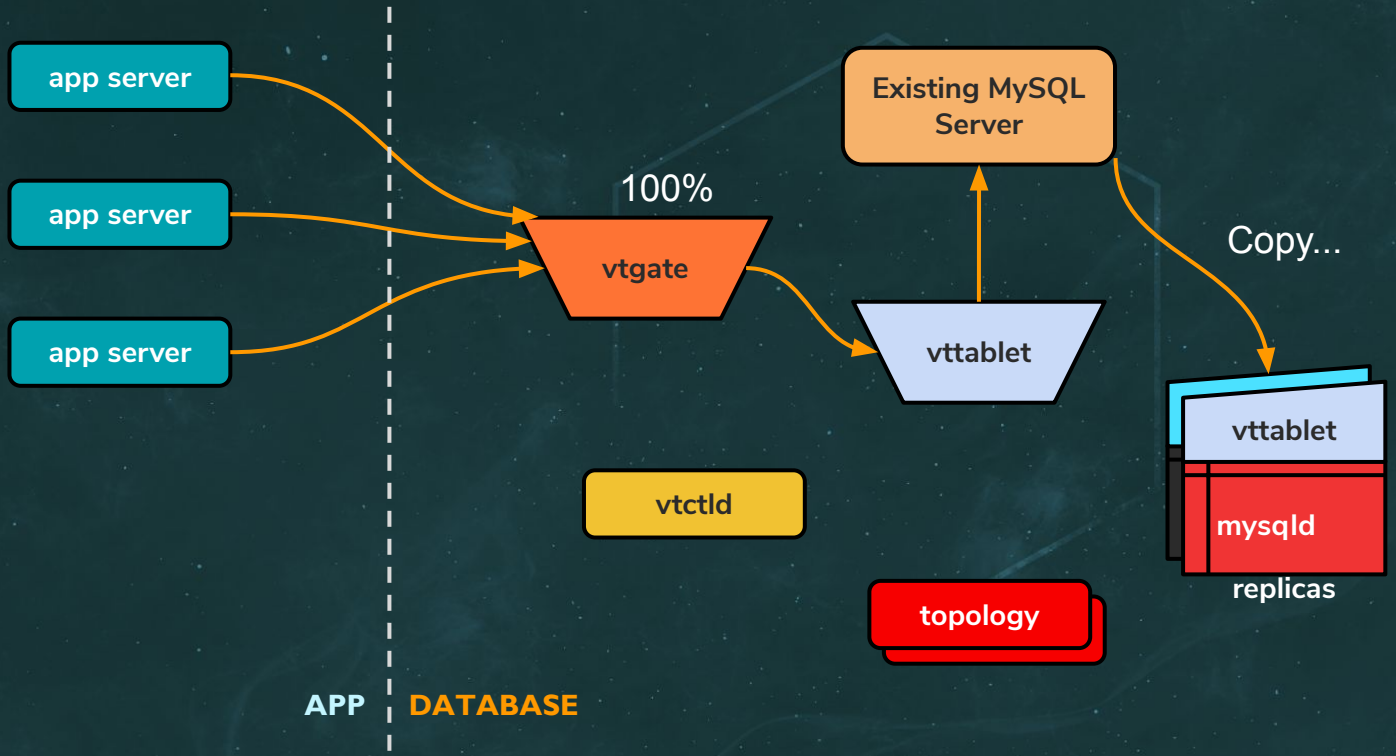


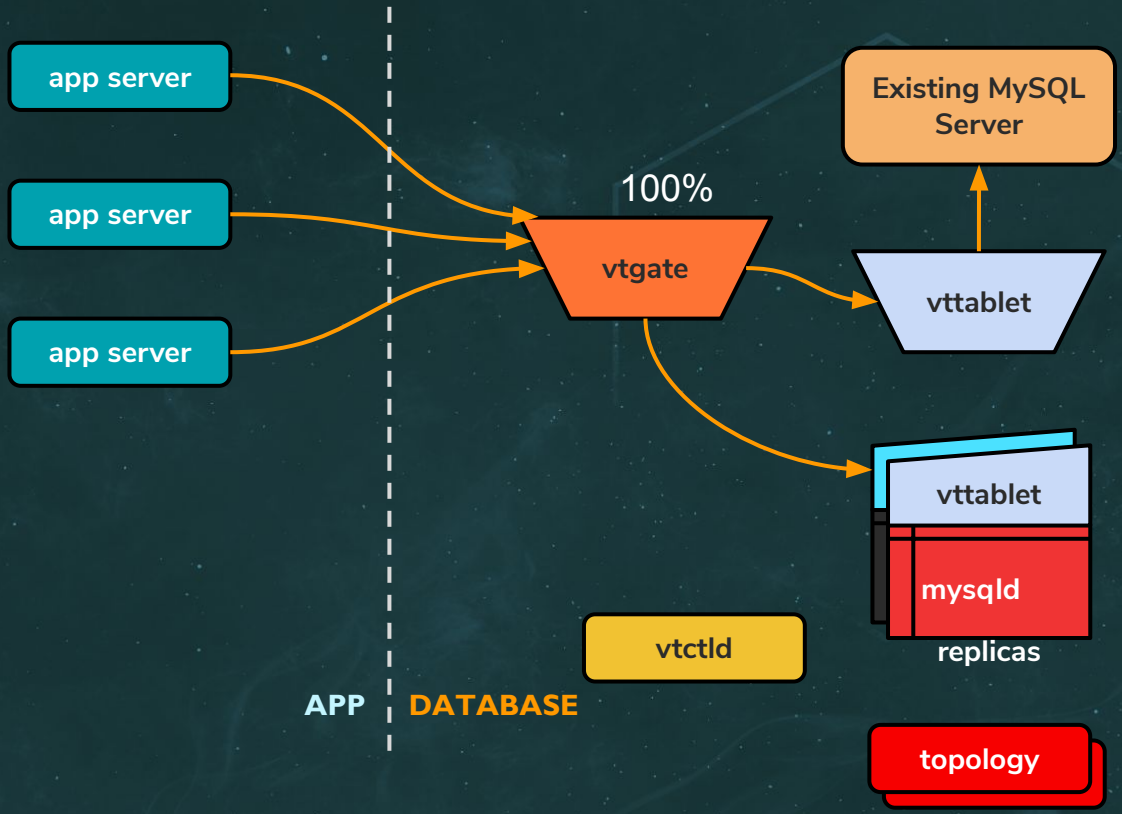
Table Migration

- Based on a feature called vReplication
- Requires your source MySQL Server to have enabled:
  - Binary Logging with GTIDs
  - Row-based Replication
  - Matching Character Set (utf8)
- Copy phase is completely online
- Final cut-over will take a couple of seconds of blocking

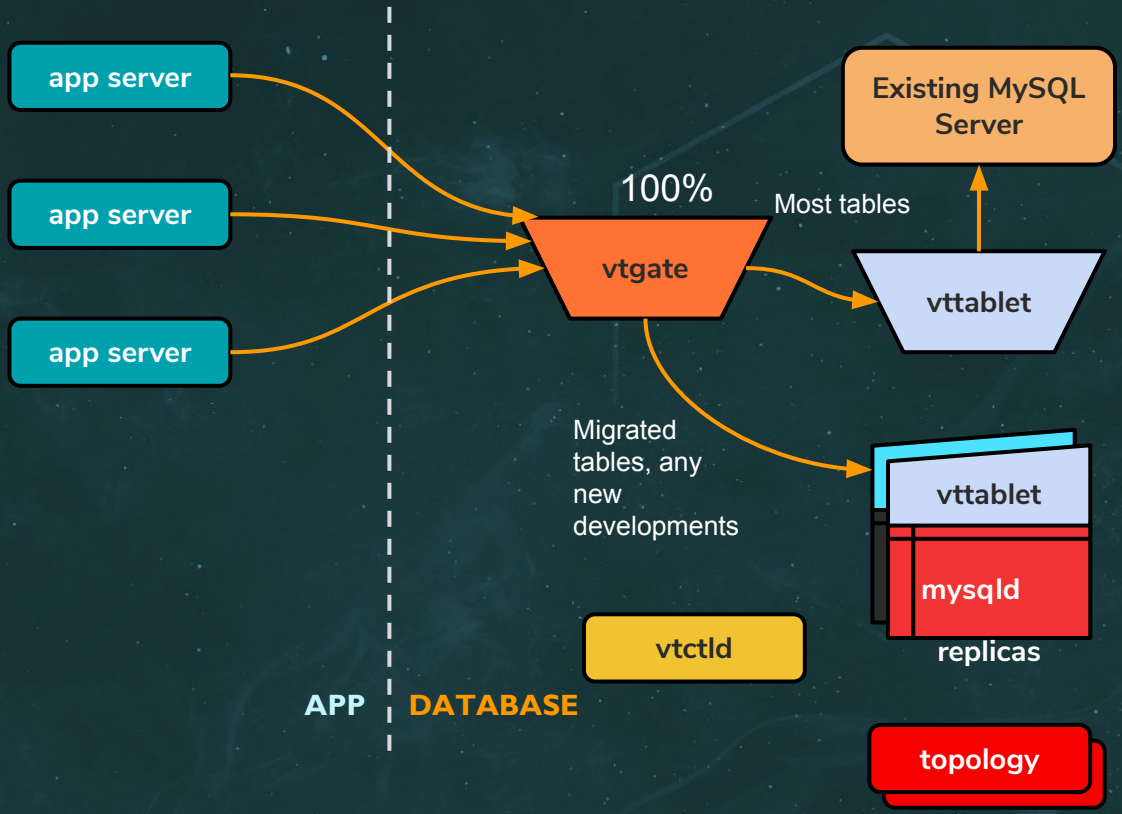
Phase 2: Completion



Phase 2: Completion



Phase 2: Completion



Our Setup

- Legacy MySQL is treated as one unsharded Keyspace
  - In Vitess terminology “The tablet server uses an externally managed MySQL”
- New Vitess Tablet is a single unsharded Keyspace
  - We could have just as easily migrated to a sharded keyspace
- We can still join queries between tablets in each keyspace
- It is recommended to keep updates contained within a single keyspace

Vitess User Guides

- Great way to become familiar with Vitess!
- https://vitess.io/docs/user-guides/

Questions?

Vitess Website: vitess.io

Vitess Documentation: vitess.io/docs

Slack Community: vitess.io/slack