Migrating Massive Aurora and MySQL Databases to Vitess Kubernetes Clusters with Near-Zero Downtime

Matthias Crauwels, Rohit Nayak

PlanetScale

Vitess is a scalable, distributed database system built around MySQL

CLOUD NATIVE
COMPUTING FOUNDATION
GRADUATED PROJECT

What is Vitess?

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

Works With

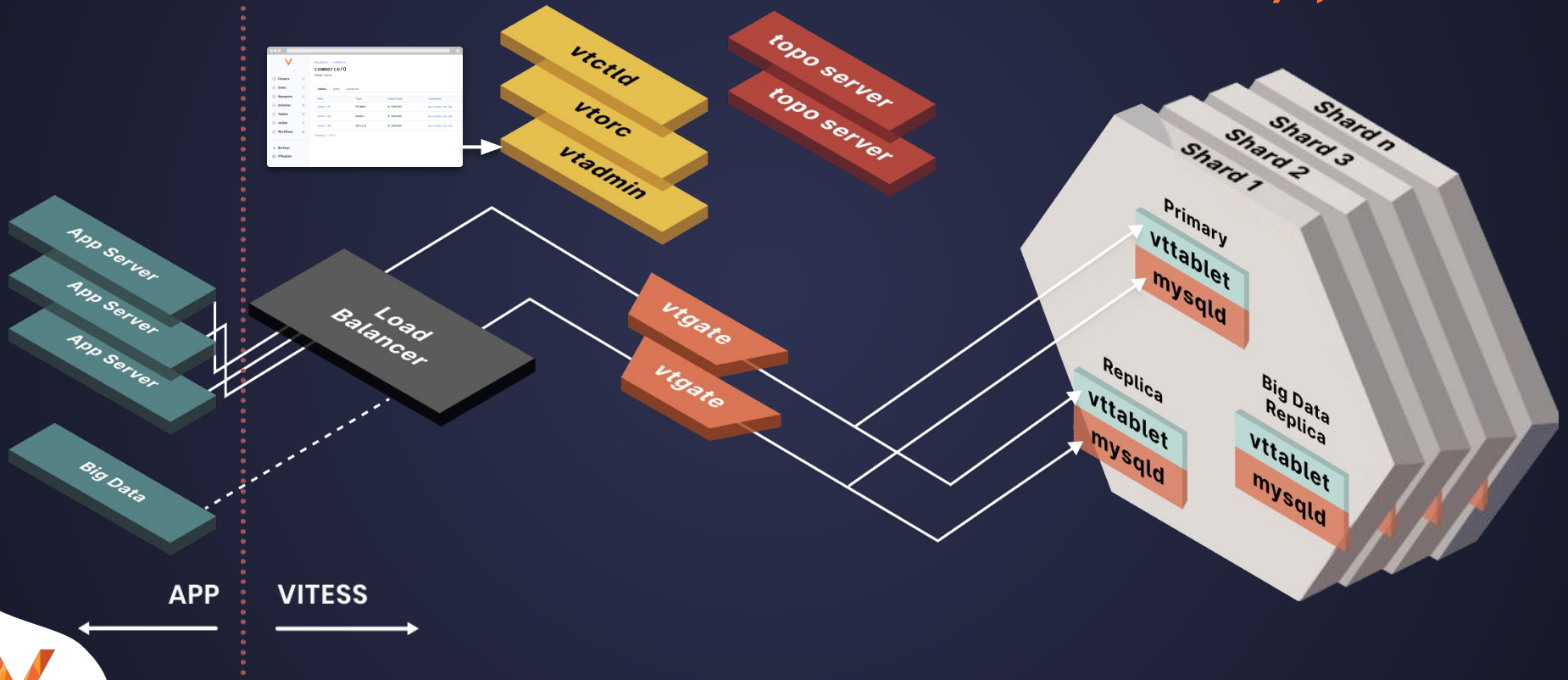
- Database Frameworks
- ORMs
- Legacy Code
- Third-Party Applications

Logical Database

- Many Physical Databases
- Query Routing
- gRPC Clients MySQL protocol
- Single Connection

slack
New Relic.
Square
Flipkart
HubSpot
peak
Pinterest
shopify
nozzle
weave
GitHub
JD.COM 京东
Quiz of Kings
stitchlabs
PlanetScale

Architecture



Vitess

@vitessio

Cloud Native by Design

- Microservice-oriented design
  - Decoupled components
  - Containerizable
  - Independent vertical and horizontal component scaling
- Horizontal and vertical sharding
  - Leverage commodity hardware
  - Leverage cloud scaling
  - Massively scalable
- MySQL replication and semi-sync
  - Resilient to storage unavailability
- Cluster management
  - Automated failovers (`vtorc`), backup/restore
  - Resilient to ephemeral instances
- Kubernetes integration (Vitess Operator)
  - Auto-scaling, self-healing
- Global Availability, cloud platform agnostic

Why Migrate

- An example, user is on RDS
- Why migrate out
  - Limited by vertical scaling
  - Manual sharding
  - Too expensive (storage/vm/data transfer)
  - Lack of control over MySQL configuration/upgrades
  - Vendor lock-in
- Why Vitess
  - MySQL compatible
  - Horizontal sharding
  - Commodity hardware
  - Online DDL for schema changes
  - Cluster management

Step 1: Setup Vitess

- Create production Vitess cluster
  - Sharding configuration
    - 2 shards, primary + 2 replicas
  - Provisioning
    - topology (etcd)
    - vtgates
    - vttablets → local mysql servers
    - vtctld/vtorc
  - External keyspace
    - unsharded
    - vttablets → RDS

Step 2: MoveTables

- Create MoveTables workflow
  - `MoveTables Create`
  - Source = External Keyspace, Target = Vitess Keyspace
  - Routing Rules → RDS
  - Denied Tables on Target
- Point App to Vitess cluster

Step 3: Switch Reads

- `MoveTables SwitchTraffic –tablet-types replica`
- Primary traffic continues to be routed to RDS
- Monitor
  - compatibility
  - warm caches
  - performance

Step 4: Switch Writes

- `MoveTables SwitchTraffic –tablet-types primary`
- All traffic routed to RDS
- Reverse replication to RDS
  - `MoveTables ReverseTraffic`
- Monitor Write Performance

Step 5: Complete

- `MoveTables Complete`
- Deletes Reverse Workflow
- Point of No Return
- Deprovision RDS

Migration Workflow

- Target streams from source vttablets (replica/primary)
- Starts with a Copy phase
  - One table at a time, in batches
  - On Source: Take consistent snapshot, streaming select
  - On Target: Bulk insert into target
  - State maintained in a sidecar database.
  - Between tables/batches, stream binary logs, with dmls for copied ranges
- Move to Running (binlog streaming) phase until cutover

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

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

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

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

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

Numbers

- Thousands of vreplication imports to PlanetScale in self-serve mode
  - 64TB to 16 shards
  - millions of QPS across multiple Vitess clusters
- Copying multi-TB datasets to sharded Vitess in hours
  - observed speeds over 1 billion rows per hour
- Multi-tenant migration with multiple 100-thousands of tenants ongoing
  - multiple (large) Aurora clusters into one sharded keyspace
  - 250TB to 128 shards
  - ~9 million tables per source Aurora converging into 90 sharded-tables
  - online DDL going from weeks to hours
- Shard-by-shard migration for over 400 shards completed successfully
  - https://planetscale.com/case-studies/cash-app
- More case studies on https://planetscale.com/case-studies/

Thank you!

rohit@planetscale.com
matthias.crauwels@planetscale.com