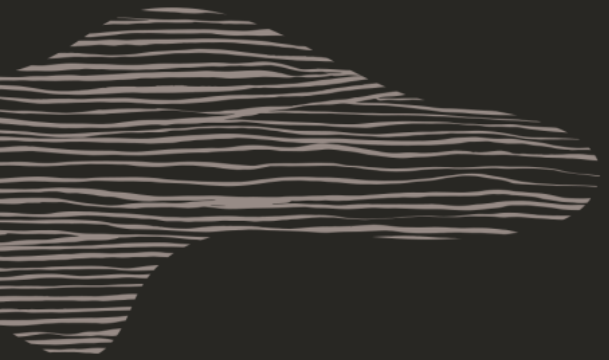
MySQL Database Architectures

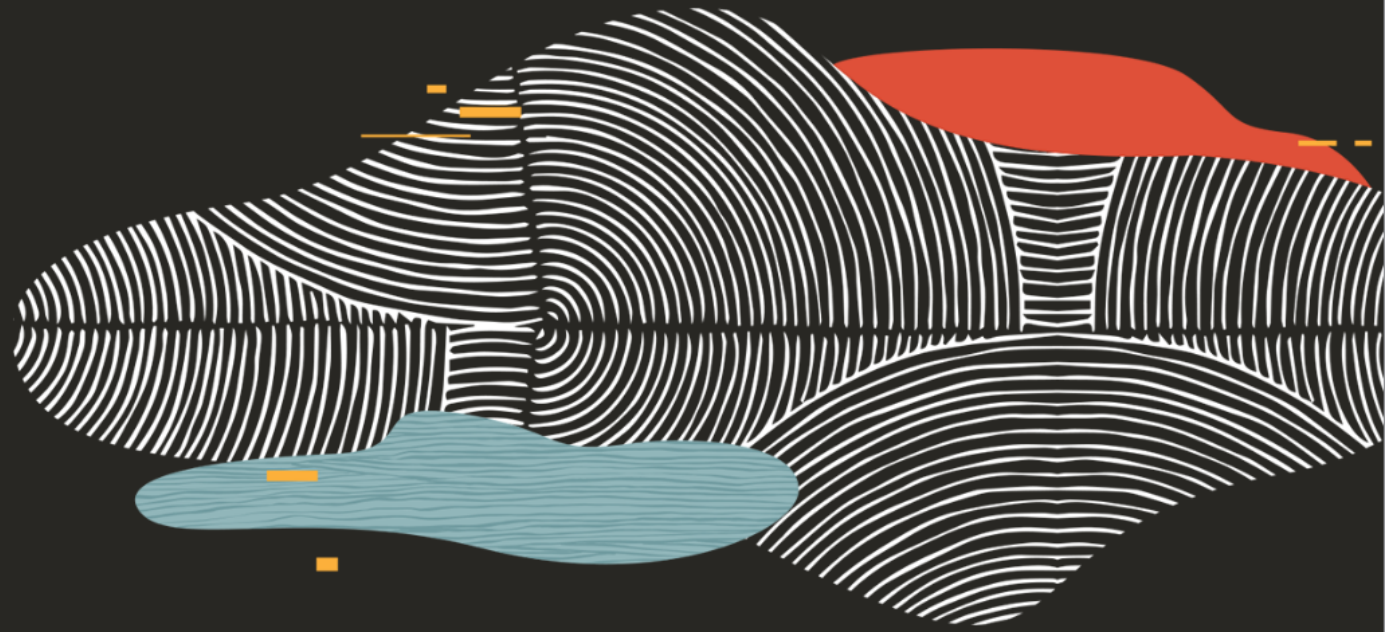
MySQL InnoDB Cluster

MySQL InnoDB ReplicaSet

Kenny Gryp
MySQL Product Manager


ORACLE®





Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purpose only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied up in making purchasing decisions. The development, release and timing of any features or functionality described for Oracle´s product remains at the sole discretion of Oracle.

Past, Present & Future

Past, Present & Future

In The Past - MANUAL

- Setting up Replication topology was done manually, taking many steps
  - including user management, restoring backups, configuring replication...
- MySQL offered the technical pieces, leaving it up to the user to setup an (always customized) architecture
- This requires technical components ... bringing lot's of work for DBA's and experts, who spent their time automating

Past, Present & Future

In The Past - MANUAL

- Setting up Replication topology was done manually, taking many steps
  - including user management, restoring backups, configuring replication...
- MySQL offered the technical pieces, leaving it up to the user to setup an (always customized) architecture
- This requires technical components ... bringing lot's of work for DBA's and experts, who spent their time automating

2016 - MySQL InnoDB Cluster

- Group Replication: Automatic membership changes, network partition handling, consistency...
- Shell to provide a powerful interface that helps in automating an integrating all components
- InnoDB `CLONE` to automatically provision members, fully integrated in InnoDB

Past, Present & Future

In The Past - MANUAL

- Setting up Replication topology was done manually, taking many steps
  - including user management, restoring backups, configuring replication...
- MySQL offered the technical pieces, leaving it up to the user to setup an (always customized) architecture
- This requires technical components ... bringing lot's of work for DBA's and experts, who spent their time automating

2016 - MySQL InnoDB Cluster

- Group Replication: Automatic membership changes, network partition handling, consistency...
- Shell to provide a powerful interface that helps in automating an integrating all components
- InnoDB `CLONE` to automatically provision members, fully integrated in InnoDB

2020 - MySQL InnoDB Replicaset

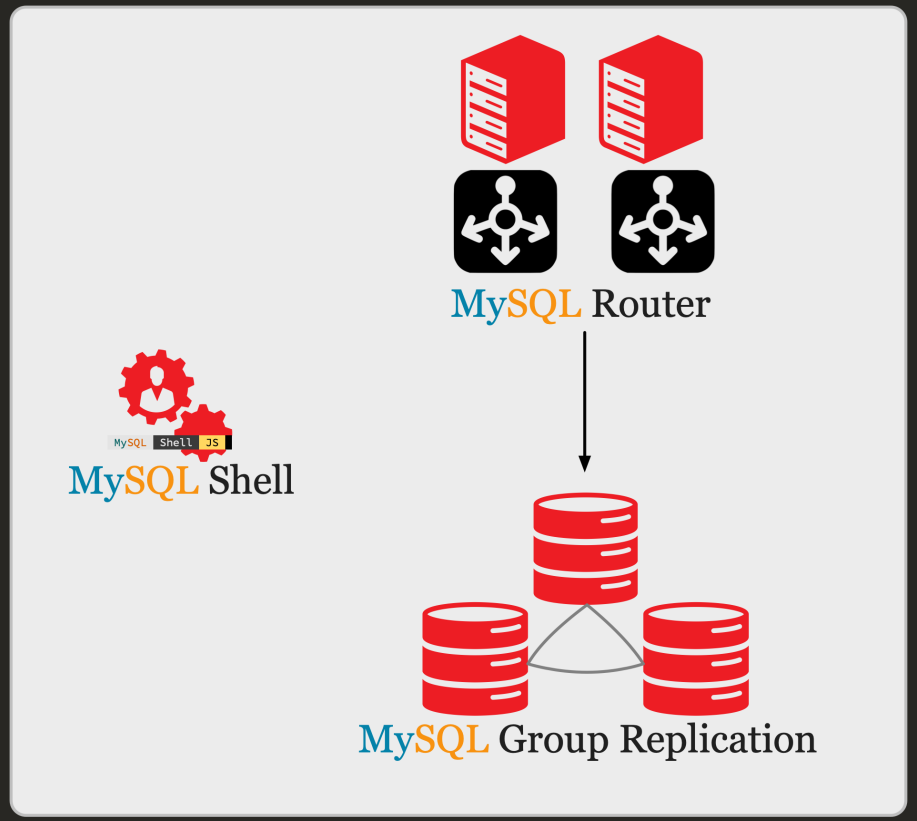
- 'classic', 'asynchronous' Replication based Solution, fully integrated

MySQL InnoDB Cluster

"A single product — MySQL — with **high availability** and **scaling features** baked in;
providing an **integrated end-to-end solution that is easy to use**."

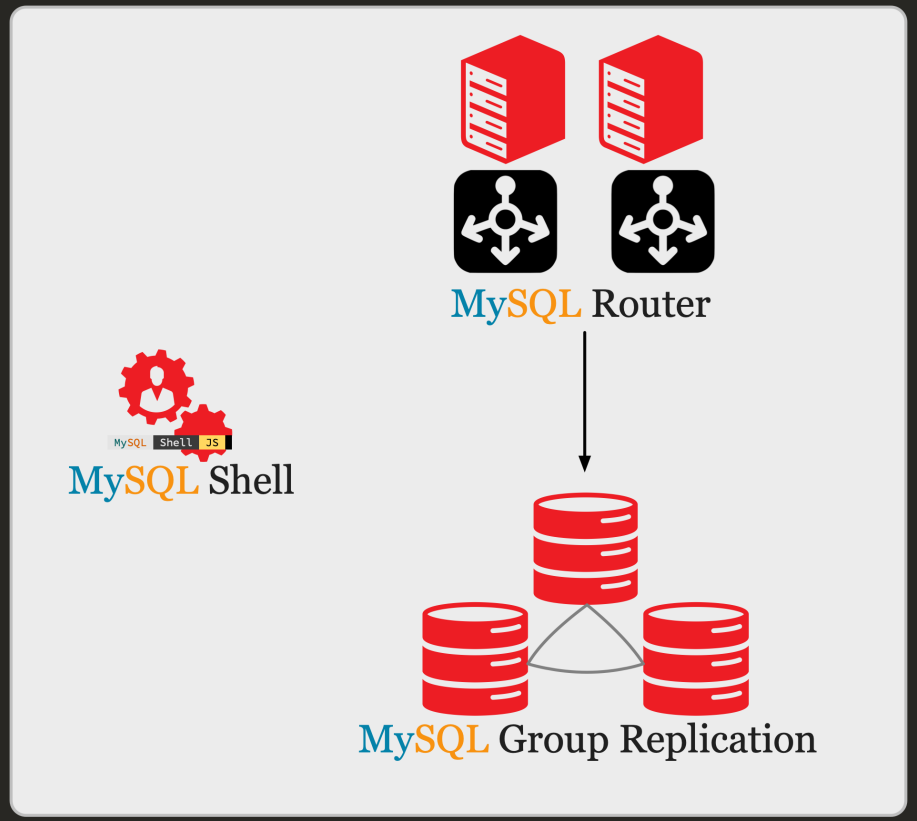
MySQL InnoDB Cluster

"A single product — MySQL — with **high availability** and **scaling features** baked in;
providing an **integrated end-to-end solution that is easy to use**."



MySQL InnoDB Cluster

"A single product — MySQL — with **high availability** and **scaling features** baked in;
providing an **integrated end-to-end solution that is easy to use**."



Components:

- MySQL Server
- MySQL Group Replication
- MySQL Shell
- MySQL Router

MySQL InnoDB Cluster - Goals

One Product: MySQL

- All components developed together
- Integration of all components
- Full stack testing

MySQL InnoDB Cluster - Goals

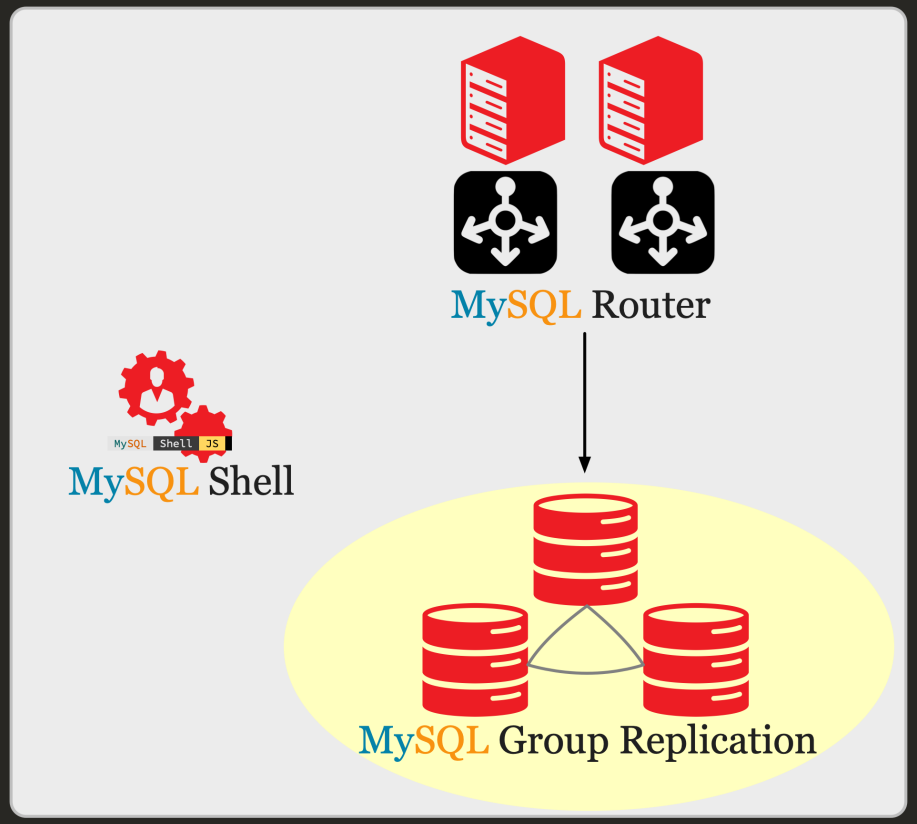
One Product: MySQL

- All components developed together
- Integration of all components
- Full stack testing

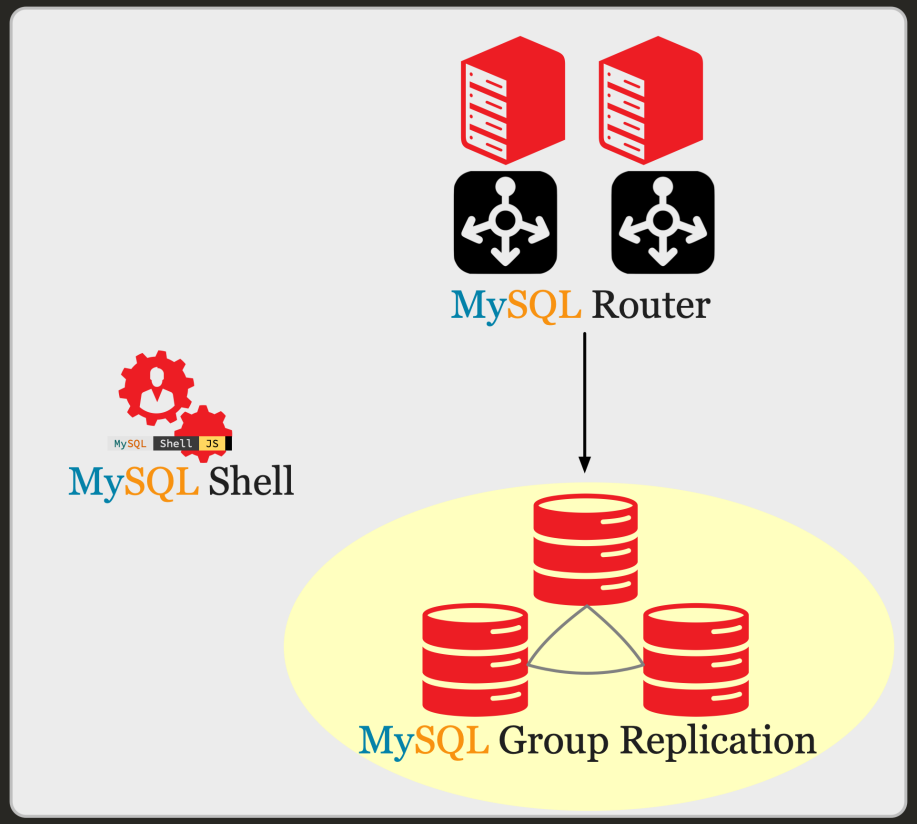
Easy to Use

- One client: MySQL Shell
- Integrated orchestration
- Homogenous servers

MySQL Group Replication



MySQL Group Replication



High Available Distributed MySQL DB

- Fault tolerance
- Automatic failover
- Active/Active update anywhere (limits apply)
- Automatic membership management
  - Adding/removing members
  - Network partitions, failures
- Conflict detection and resolution
- Prevents data loss

MySQL Group Replication

- Implementation of Replicated Database State Machine
  - Total Order - Writes
  - XCOM - Paxos implementation
- Configurable Consistency Guarantees
  - eventual consistency
  - 8.0+: per session & global read/write consistency
- Using MySQL replication framework by design
  - binary logs
  - relay logs
  - GTIDs: Global Transaction IDs
- Generally Available since MySQL 5.7
- Supported on all platforms: linux, windows, solaris, macosx, freebsd

Business Requirements

Recovery Time Objective (RTO)

How fast should the service recover from a failure

Recovery Point Objective (RPO)

How much data loss can the service lose from a failure

Business Requirements

Recovery Time Objective (RTO)

How fast should the service recover from a failure

Recovery Point Objective (RPO)

How much data loss can the service lose from a failure

Types of Failures

High Availability: Single Server Failure, Network Partition
Disaster Recovery: Full Region/Network Failure
Human Error: Little Bobby Tables

How Much

- None
- few seconds
- minutes
- hours
- day
- ...

MySQL Group Replication - Use Cases

Consistency: No Data Loss (RPO=0)

- in event of failure of (primary) member
- Split brain prevention (Quorum)

MySQL Group Replication - Use Cases

Consistency: No Data Loss (RPO=0)

- in event of failure of (primary) member
- Split brain prevention (Quorum)

Highly Available: Automatic Failover

- Primary members are automatically elected
- Automatic Network Partition handling

MySQL Group Replication - Use Cases

Consistency: No Data Loss (RPO=0)

- in event of failure of (primary) member
- Split brain prevention (Quorum)

Highly Available: Automatic Failover

- Primary members are automatically elected
- Automatic Network Partition handling

Read Scaleout

- Add/Remove members as needed
- Replication Lag handling with Flow Control
- Configurable Consistency Levels
  - Eventual
  - Full Consistency -- no stale reads

MySQL Group Replication - Use Cases

Consistency: No Data Loss (RPO=0)

- in event of failure of (primary) member
- Split brain prevention (Quorum)

Highly Available: Automatic Failover

- Primary members are automatically elected
- Automatic Network Partition handling

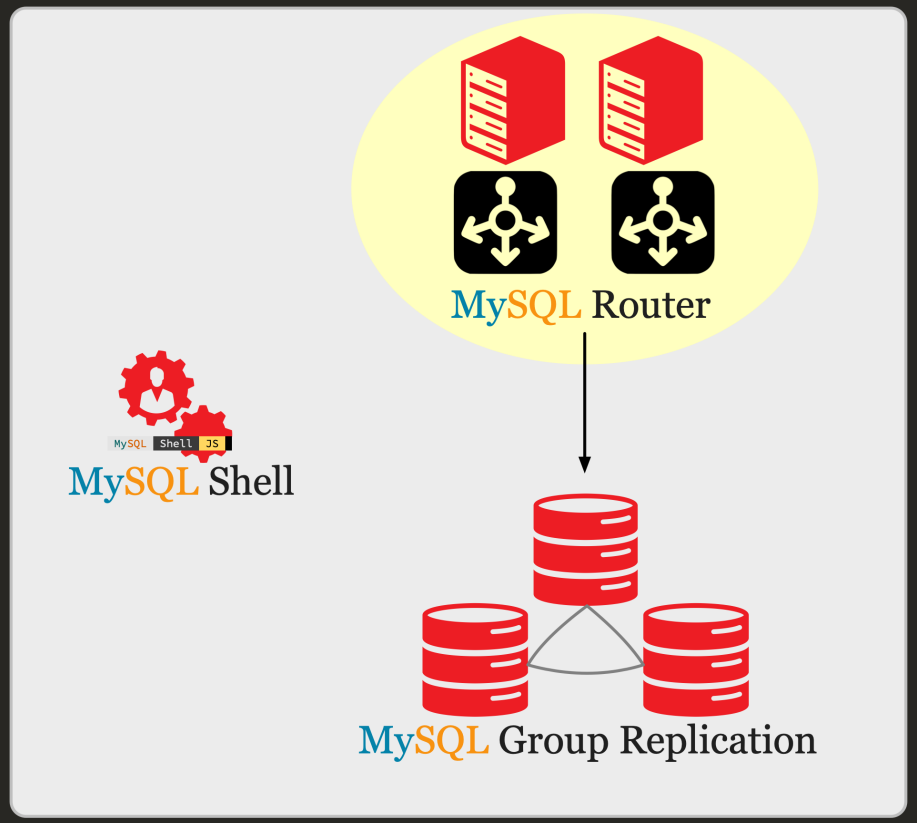
Read Scaleout

- Add/Remove members as needed
- Replication Lag handling with Flow Control
- Configurable Consistency Levels
  - Eventual
  - Full Consistency -- no stale reads

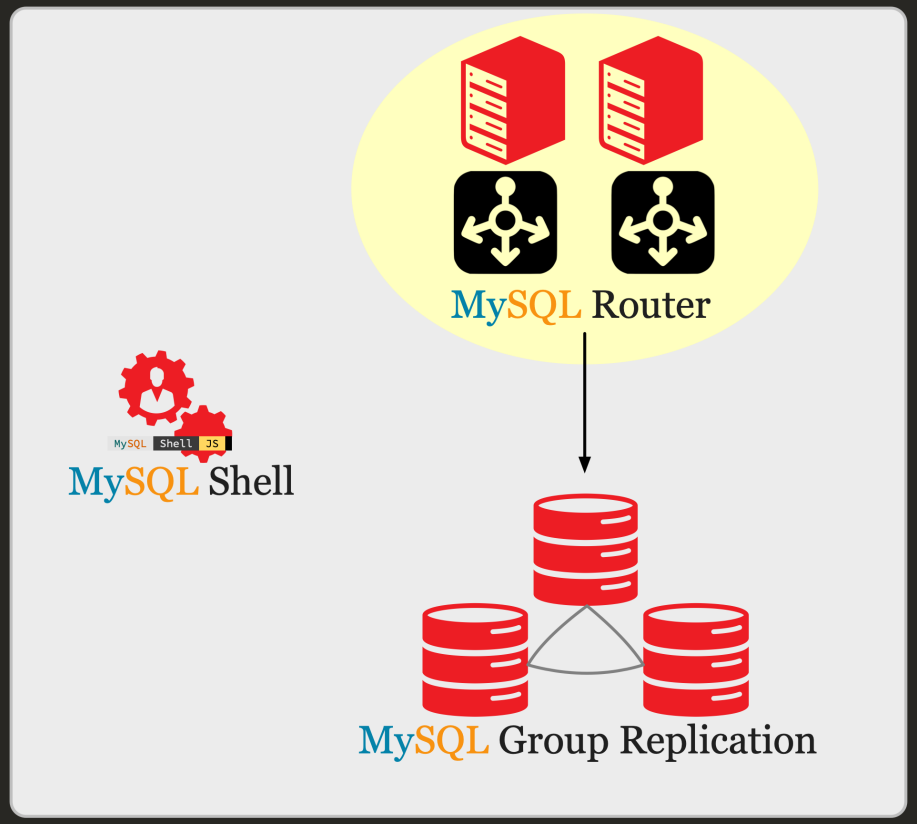
Active/Active environments

- Write to many members at the same time
  - ordered writes within the group (XCOM)
  - guaranteed consistency
- Good write performance
  - due to Optimistic Locking
    (workload dependent)

MySQL Router



MySQL Router

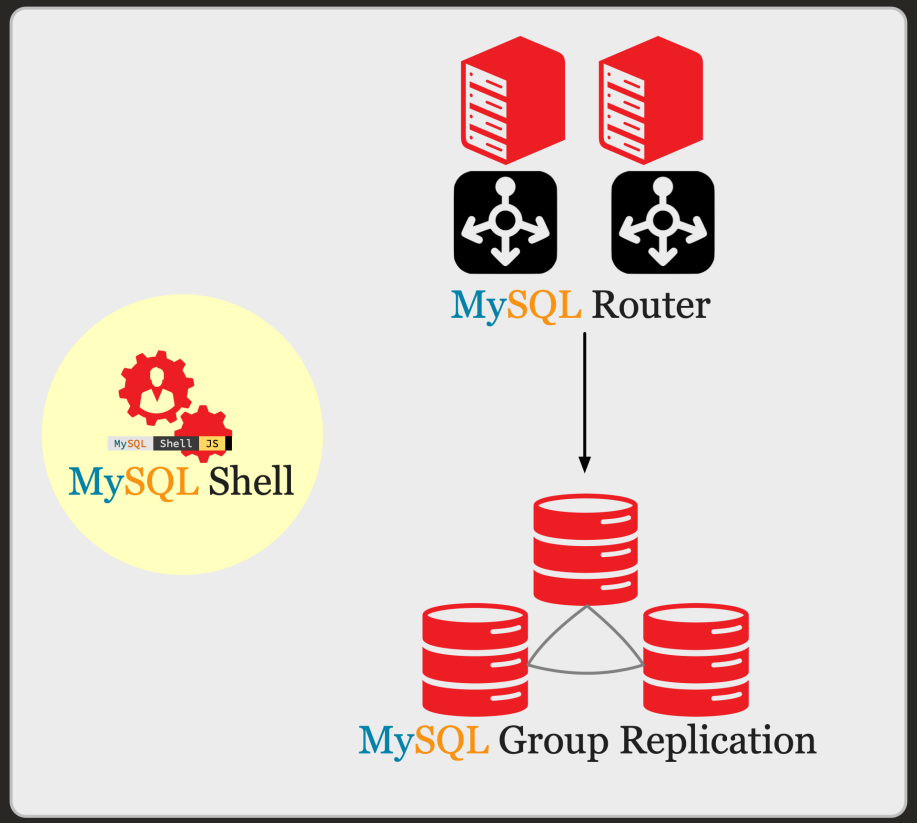


Transparent Access to Database Arch.

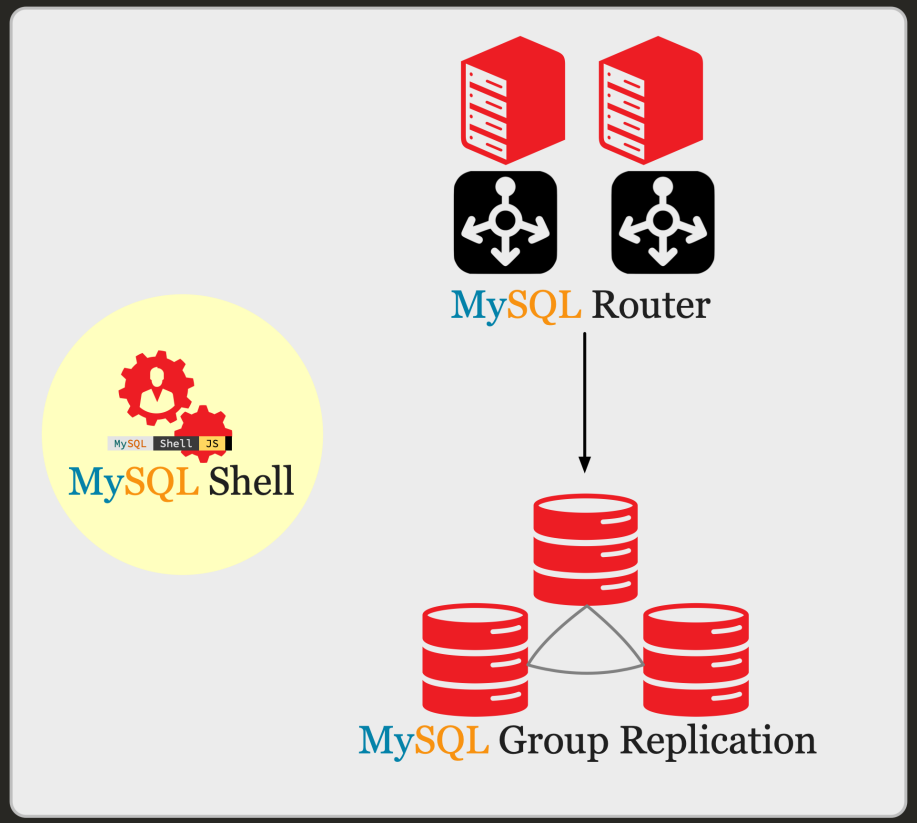
"provide transparent routing between your application and back-end MySQL Servers"

- Transparent client connection routing
  - Load balancing
  - Application connection failover
  - Little to no configuration needed
- Stateless design offers easy HA client routing
  - Router as part of the application stack
- Integration into InnoDB Cluster & InnoDB ReplicaSet
  - Understands Group Replication & Replication topology
- Currently TCP Port each for `PRIMARY` and `NON-PRIMARY` traffic

MySQL Shell



MySQL Shell



Database Administration Interface

"MySQL Shell provides the developer and DBA with a single intuitive, flexible, and powerful interface for all MySQL related tasks!"

- Multi-Language: JavaScript, Python, and SQL
- Naturally scriptable
- Supports Document and Relational models
- Exposes full Development and Admin API
- Classic MySQL protocol and X protocol

MySQL Shell - Easy to Use

MySQL Shell - Easy to Use

MySQL InnoDB Cluster

```
mysql-js> \c admin@mysql1
mysql-js> cluster = dba.createCluster('cluster')
```

MySQL Shell - Easy to Use

MySQL InnoDB Cluster

```
mysql-js> \c admin@mysql1
mysql-js> cluster = dba.createCluster('cluster')
```

Configure server to add later:

```
mysql-js> dba.configureInstance('admin@mysql2')
```

MySQL Shell - Easy to Use

MySQL InnoDB Cluster

```
mysql-js> \c admin@mysql1
mysql-js> cluster = dba.createCluster('cluster')
```

Configure server to add later:

```
mysql-js> dba.configureInstance('admin@mysql2')
```

Add server to the Cluster:

```
mysql-js> cluster.addInstance('admin@mysql2')
```

MySQL Shell - Easy to Use

MySQL InnoDB Cluster

```
mysql-js> \c admin@mysql1
mysql-js> cluster = dba.createCluster('cluster')
```

Configure server to add later:

```
mysql-js> dba.configureInstance('admin@mysql2')
```

Add server to the Cluster:

```
mysql-js> cluster.addInstance('admin@mysql2')
```

Bootstrap MySQL Router

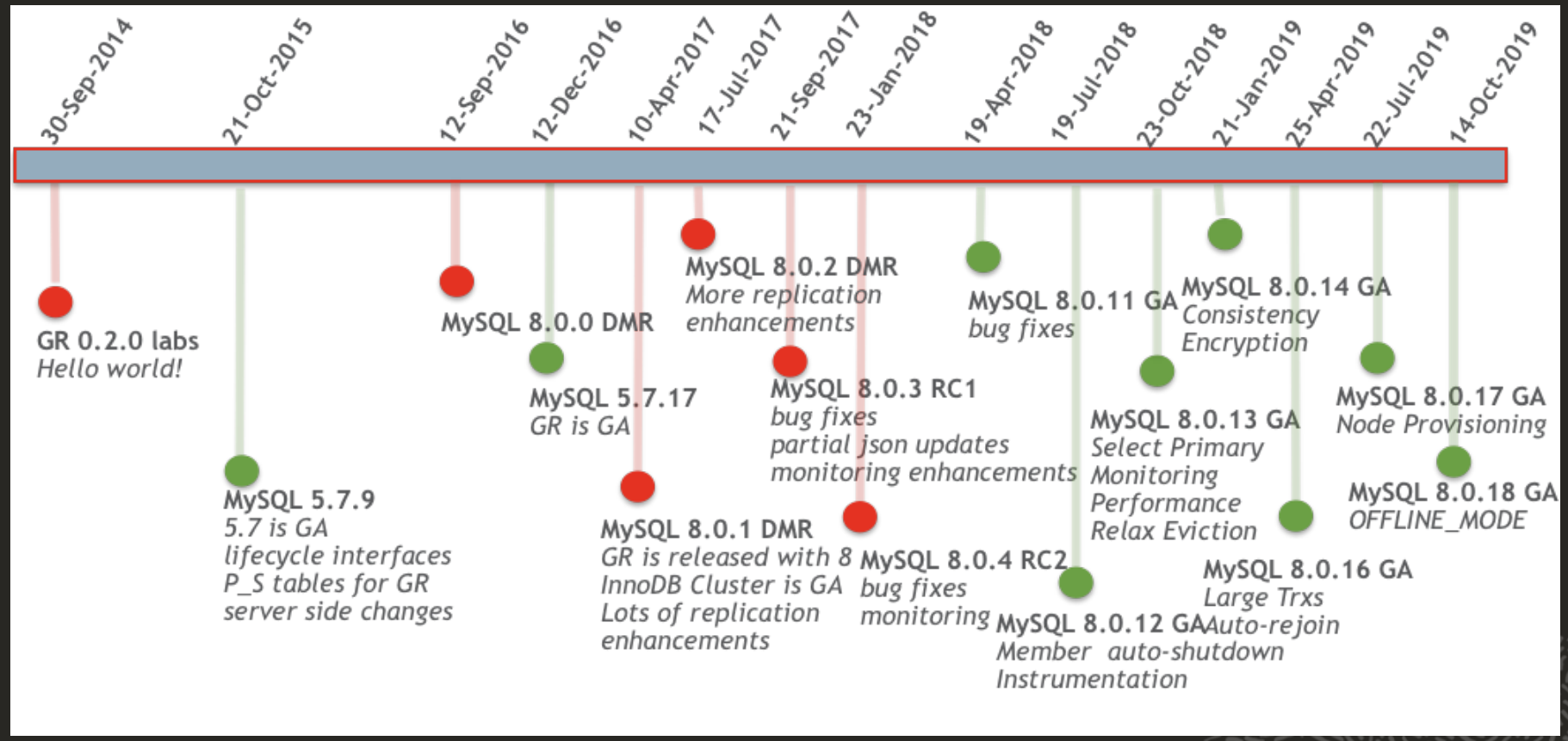
```
$ sudo mysqlrouter --user=mysqlrouter --bootstrap
$ sudo systemctl start mysqlrouter
```

MySQL Shell - Easy to Use

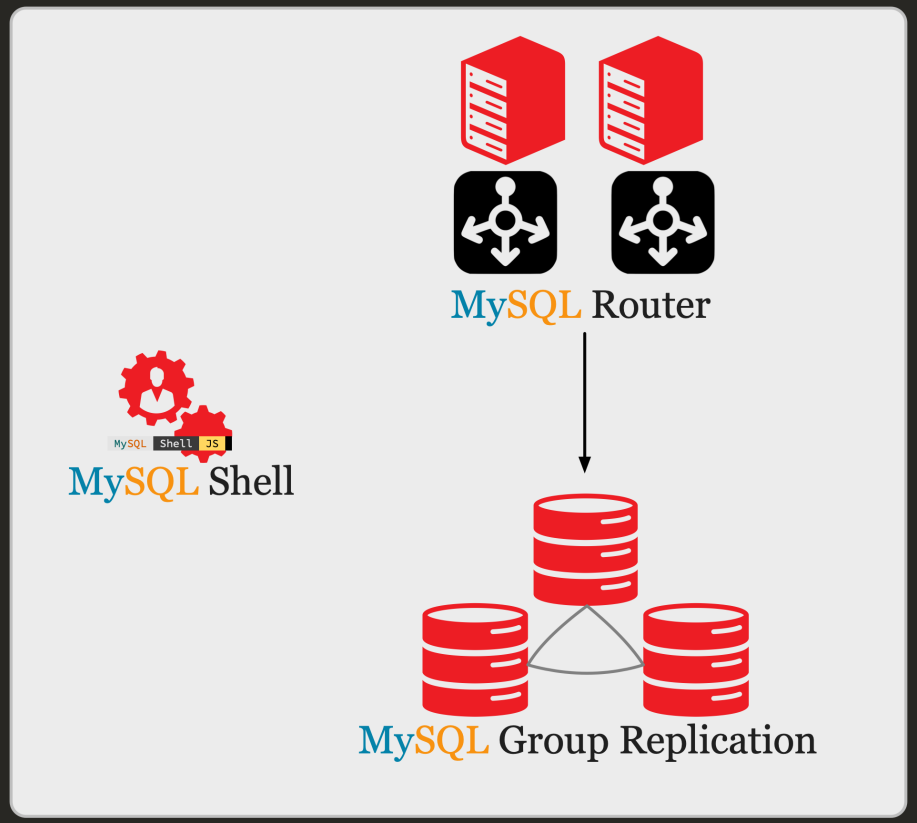
Check the Cluster status:

```
mysql-js> cluster.status()
{
    "clusterName": "cluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "mysql1:3306",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can
                       tolerate up to ONE failure.",
        "topology": {
            "mysql1:3306": {
                "address": "mysql1:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "mysql2:3306": {
                "address": "mysql2:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "mysql3:3306": {
                "address": "mysql3:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            }
        }
}
```

MySQL InnoDB Cluster - Evolving



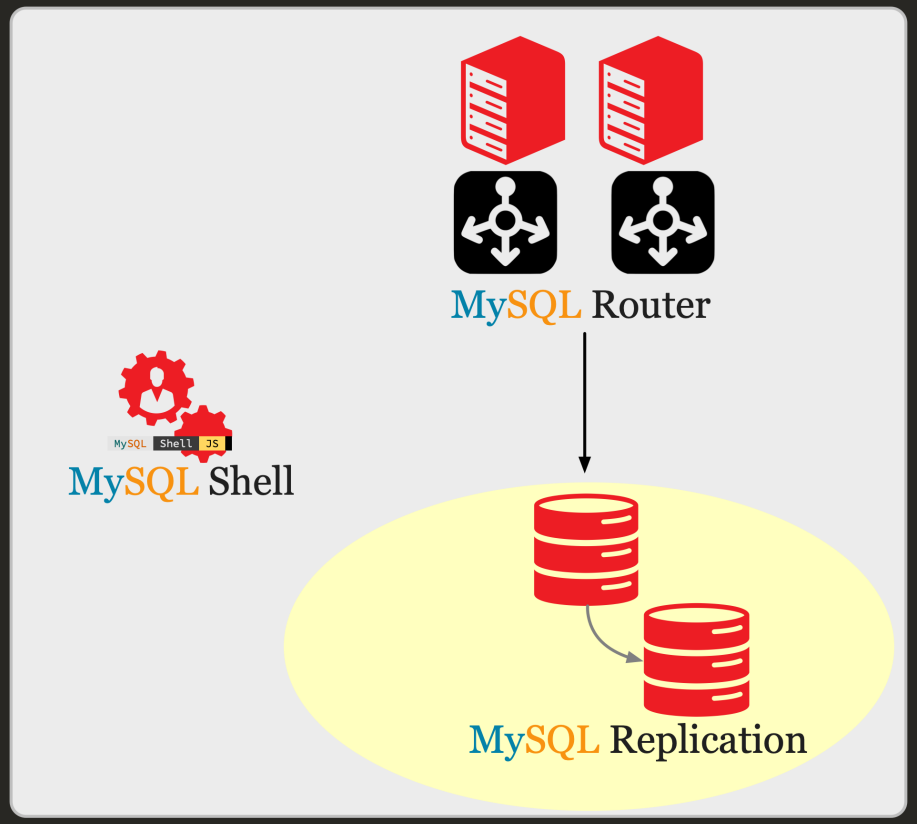
MySQL InnoDB Cluster - Adoption



Many of our customers have adopted InnoDB Cluster!

- **Fully integrated MySQL Router**
  - Automatic Routing
- **Ease of use with MySQL Shell**
  - Configuring, Adding, Removing members
- Group Replication Architecture
  - **Providing Consistency**
  - Automatic Failover
  - Network Partition Handling
  - No data loss in case of failure
  - Automatic Member Provisioning (`CLONE`)

MySQL InnoDB Replicaset



Introducing MySQL InnoDB ReplicaSet!

- 8.0.19 Feature!
- **Fully integrated MySQL Router**
  - Automatic Routing
- **Ease of use with MySQL Shell**
  - Configuring, Adding, Removing members
  - Automatic Member Provisioning (`CLONE`)
- Replication Architecture:
  - (manual) Switchover & Failover
  - (asynchronous) Read Scaleout
  - 'Simple' Replication architecture:
    - no network/hardware requirements
      - **Providing Availability** on PRIMARY when issues with secondaries or network

MySQL InnoDB ReplicaSet - Features

Past

- Restore a backup to provision a member

MySQL InnoDB ReplicaSet

- Automatically provisioning new members: InnoDB `CLONE`

MySQL InnoDB ReplicaSet - Features

Past	MySQL InnoDB ReplicaSet
• Restore a backup to provision a member	• Automatically provisioning new members: InnoDB `CLONE`
• Configure Replication Users • Configure Replication	• MySQL Shell Automatically configures users & replication

MySQL InnoDB ReplicaSet - Features

Past	MySQL InnoDB ReplicaSet
• Restore a backup to provision a member	• Automatically provisioning new members: InnoDB `CLONE`
• Configure Replication Users • Configure Replication	• MySQL Shell Automatically configures users & replication

• Manually configuring, adding removing servers in Application, MySQL Router (or other proxy)	• Integrated MySQL Router load balancing
	  ◦ Only need to bootstrap Router
	  ◦ Router is stateless, adapts to topology changes

MySQL InnoDB ReplicaSet - Features

Past	MySQL InnoDB ReplicaSet
• Restore a backup to provision a member	• Automatically provisioning new members: InnoDB `CLONE`
• Configure Replication Users • Configure Replication	• MySQL Shell Automatically configures users & replication

• Manually configuring, adding removing servers in Application, MySQL Router (or other proxy)	• Integrated MySQL Router load balancing
	  ◦ Only need to bootstrap Router
	  ◦ Router is stateless, adapts to topology changes

• Manually or relying on external tools to make topology changes	• Easy to use manual switchover/failover

MySQL InnoDB ReplicaSet - Features

Past	MySQL InnoDB ReplicaSet
• Restore a backup to provision a member	• Automatically provisioning new members: InnoDB `CLONE`
• Configure Replication Users • Configure Replication	• MySQL Shell Automatically configures users & replication

• Manually configuring, adding removing servers in Application, MySQL Router (or other proxy)	• Integrated MySQL Router load balancing
	  ◦ Only need to bootstrap Router
	  ◦ Router is stateless, adapts to topology changes

• Manually or relying on external tools to make topology changes	• Easy to use manual switchover/failover
• Use additional monitoring tool log in on all machines to check topology status	• See status of the topology through MySQL Shell `status()`

MySQL InnoDB ReplicaSet - Features

Past	MySQL InnoDB ReplicaSet

MySQL InnoDB ReplicaSet - Features

Past

- complexity: user is responsible for the full configuration of every component and it's settings

MySQL InnoDB ReplicaSet

- Shell configures Server, Router, Replication in a standardized best practice setup, prevents mistakes

MySQL InnoDB ReplicaSet - Features

Past	MySQL InnoDB ReplicaSet
• complexity: user is responsible for the full configuration of every component and it's settings	• Shell configures Server, Router, Replication in a standardized best practice setup, prevents mistakes
• every setup is a customized setup	• Standard Solution -- Supported & QA'ed by Oracle

MySQL InnoDB ReplicaSet - Features

Past	MySQL InnoDB ReplicaSet
• complexity: user is responsible for the full configuration of every component and it's settings	• Shell configures Server, Router, Replication in a standardized best practice setup, prevents mistakes
• every setup is a customized setup	• Standard Solution -- Supported & QA'ed by Oracle
• A lot of manual steps and additional software required, always customized and often overengineered by MySQL DBA's	• Easy to use, even for MySQL beginner

MySQL InnoDB ReplicaSet - Requirements & Limitations

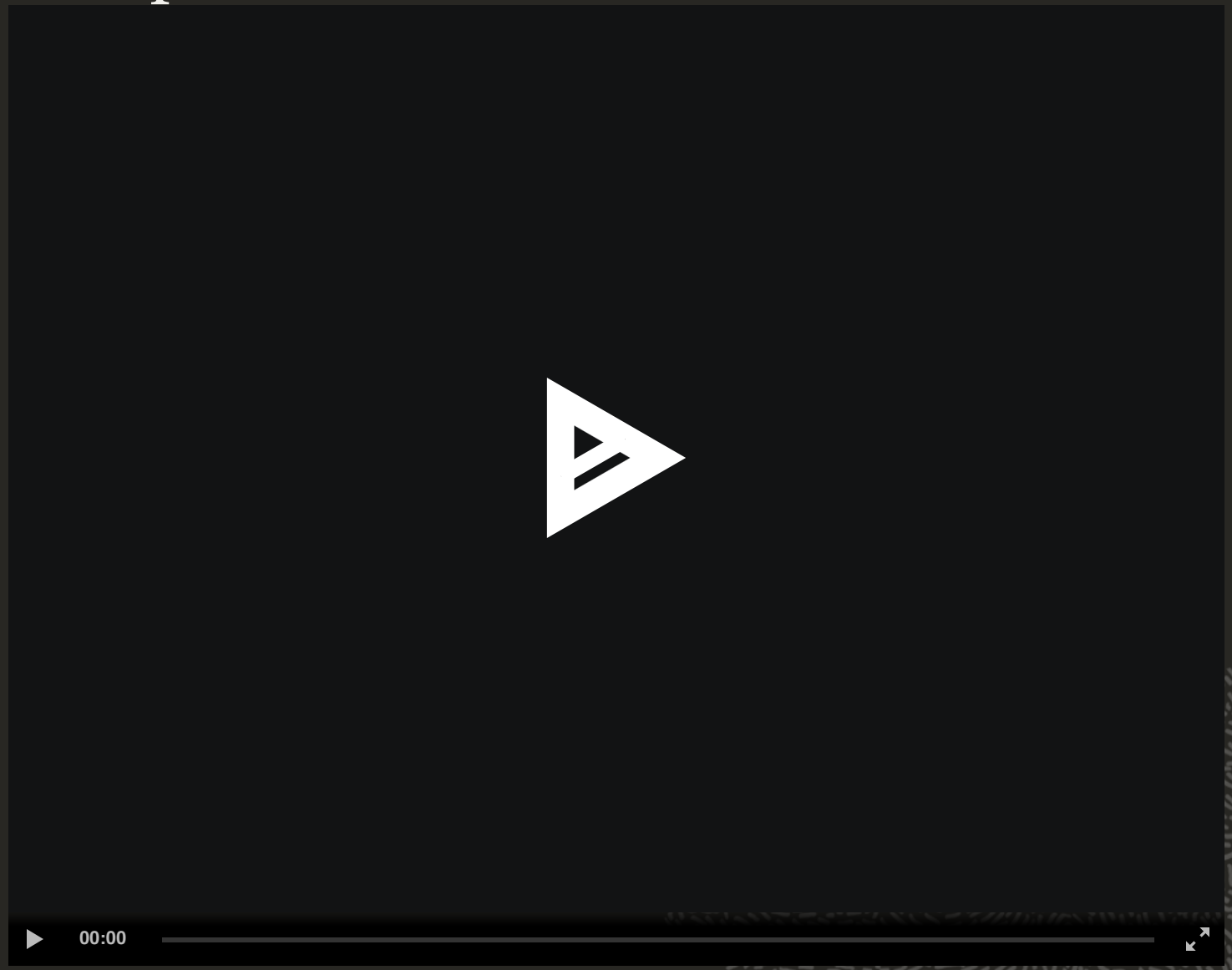
Requirements:

- MySQL 8 (`SET PERSIST`!)
- GTID

Limitations:

- Manual failover
- No multi-primary as such topology cannot guarantee data consistency
- All secondary members replicate from primary

MySQL InnoDB ReplicaSet - Demo



MySQL InnoDB ReplicaSet - MySQL Router Integration

- `--bootstrap` to automatically configure
- Metadata schema contains Replication Topology
- Health checks are performed to all members
- Switchover to new primary (`setPrimaryInstance()`):
  - Shell performs topology change, configures `super_read_only` and updates `view_id` in metadata
  - Router automatically sees higher `view_id`
  - and will redirect primary traffic to new primary
- When the Primary is lost: `forcePrimaryInstance()`
  - Shell performs topology change, updates `view_id` in metadata and invalidates lost primary
  - Router will notice topology change by higher `view_id` in metadata (if network allows)
  - and automatically connect to new Primary

MySQL Shell - Easy to Use

MySQL Shell - Easy to Use

MySQL InnoDB Cluster

```
mysql-js> \c admin@mysql1
mysql-js> cluster = dba.createCluster('cluster')
```

MySQL InnoDB ReplicaSet

```
mysql-js> \c admin@mysql1
mysql-js> rs = dba.createReplicaSet('replicaset')
```

MySQL Shell - Easy to Use

MySQL InnoDB Cluster

```
mysql-js> \c admin@mysql1
mysql-js> cluster = dba.createCluster('cluster')
```

Configure server to add later

```
mysql-js> dba.configureInstance('admin@mysql2')
```

MySQL InnoDB ReplicaSet

```
mysql-js> \c admin@mysql1
mysql-js> rs = dba.createReplicaSet('replicaset')
```

```
mysql-js> dba.configureReplicaSetInstance('admin@mysql2')
```

MySQL Shell - Easy to Use

MySQL InnoDB Cluster

```
mysql-js> \c admin@mysql1
mysql-js> cluster = dba.createCluster('cluster')
```

Configure server to add later

```
mysql-js> dba.configureInstance('admin@mysql2')
```

Add server to the Cluster

```
mysql-js> cluster.addInstance('admin@mysql2')
```

MySQL InnoDB ReplicaSet

```
mysql-js> \c admin@mysql1
mysql-js> rs = dba.createReplicaSet('replicaset')
```

```
mysql-js> dba.configureReplicaSetInstance('admin@mysql2')
```

```
mysql-js> rs.addInstance('admin@mysql2')
```

MySQL Shell - Easy to Use

MySQL InnoDB Cluster

```
mysql-js> \c admin@mysql1
mysql-js> cluster = dba.createCluster('cluster')
```

Configure server to add later

```
mysql-js> dba.configureInstance('admin@mysql2')
```

Add server to the Cluster

```
mysql-js> cluster.addInstance('admin@mysql2')
```

Bootstrap MySQL Router

```
$ sudo mysqlrouter --user=mysqlrouter --bootstrap
$ sudo systemctl start mysqlrouter
```

MySQL InnoDB ReplicaSet

```
mysql-js> \c admin@mysql1
mysql-js> rs = dba.createReplicaSet('replicaset')
```

```
mysql-js> dba.configureReplicaSetInstance('admin@mysql2')
```

```
mysql-js> rs.addInstance('admin@mysql2')
```

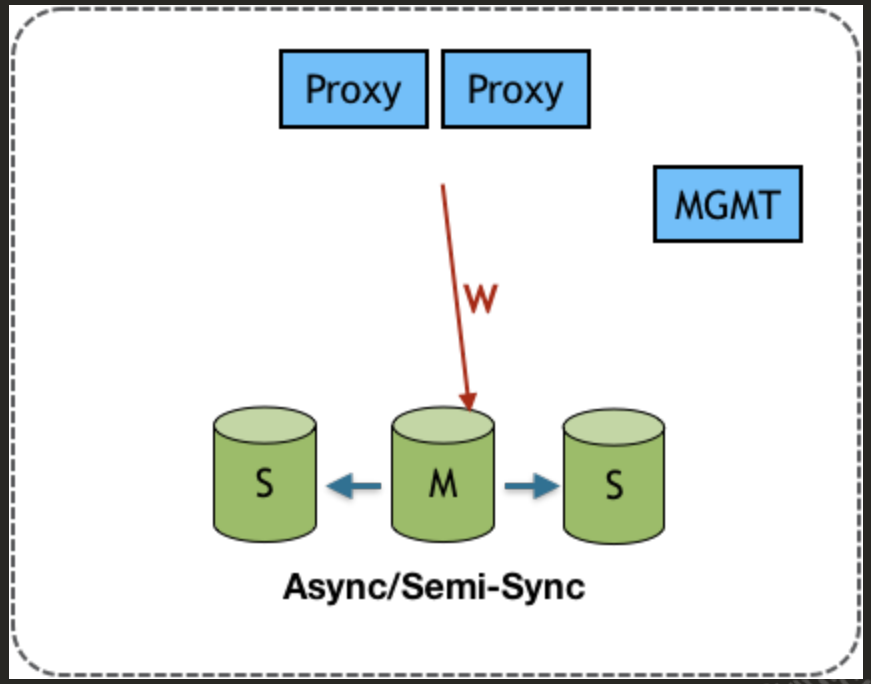
```
$ sudo mysqlrouter --user=mysqlrouter --bootstrap
$ sudo systemctl start mysqlrouter
```

Demo: https://mysqlserverteam.com/introducing-mysql-innodb-replicaset

MySQL InnoDB ReplicaSet - Manual Failover

By design chosen to be Manual failover

- Having external monitoring processes decide failover can cause a lot of false positives.
  - External tool decides, if the tool has issues: even bigger issues.
  - Split brain issues
  - Majority of production deployments are configured with Manual Failover, which increases Uptime!
- When automatic failover is needed:
  use MySQL InnoDB Cluster instead of Semi-Sync



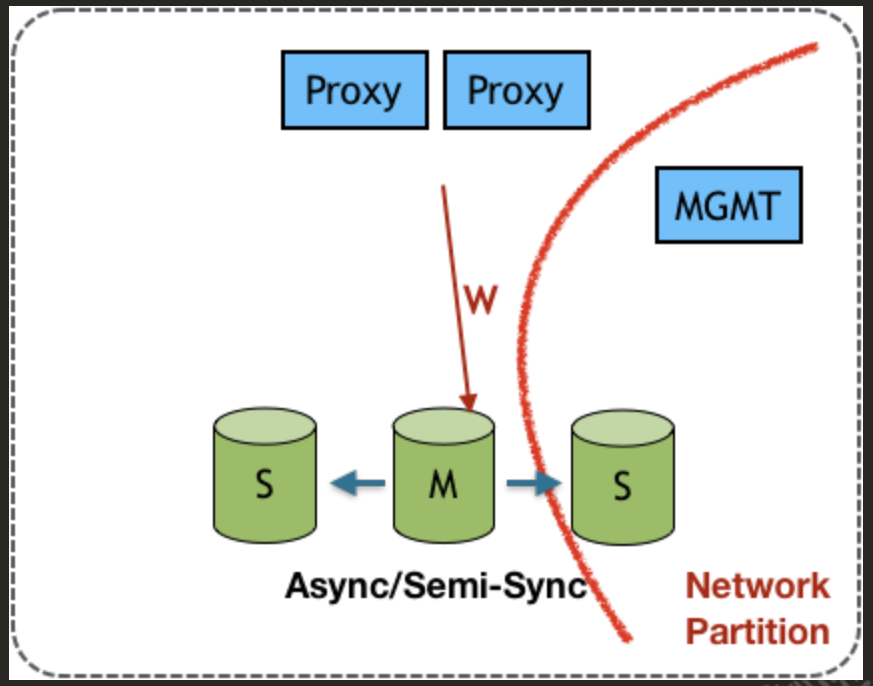
MySQL InnoDB ReplicaSet - Manual Failover

Example:

- 1 Primary, 2 Secondaries, 2 Proxy hosts (App servers)
- 1 Management server: monitoring and making Topology changes automatically

Problem: Management server & replica server is partitioned from all other servers

What will happen?



MySQL InnoDB ReplicaSet - Manual Failover

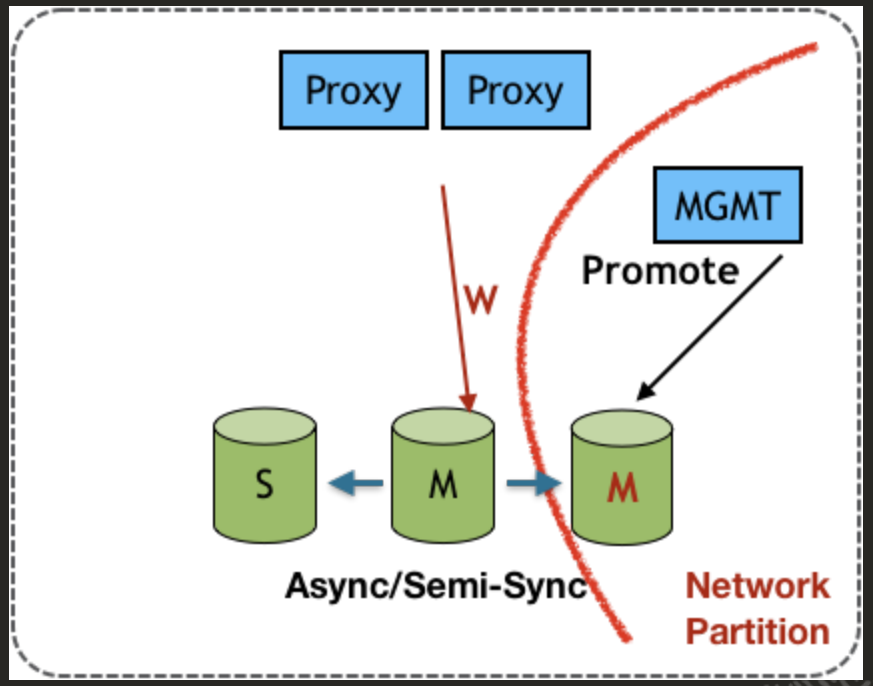
Example:

- 1 Primary, 2 Secondaries, 2 Proxy hosts (App servers)
- 1 Management server: monitoring and making Topology changes automatically

Problem: Management server & replica server is partitioned from all other servers

What will happen?

- The management host might promote the secondary causing split brain



use MySQL InnoDB Cluster over Semi-Sync

- With semi-sync, all members have to ack (prevent split brain), causing downtime with every small glitch

Which Solution fits me?

MySQL InnoDB Cluster

MySQL InnoDB ReplicaSet

Which Solution fits me?

MySQL InnoDB Cluster

- Single Primary, multiple Secondaries
- Multi Primary mode (write to all)

MySQL InnoDB ReplicaSet

- Single Primary, multiple Secondaries

Which Solution fits me?

MySQL InnoDB Cluster

- Single Primary, multiple Secondaries
- Multi Primary mode (write to all)

- Read Scaleout with configurable consistent reads

MySQL InnoDB ReplicaSet

- Single Primary, multiple Secondaries

- Read Scaleout with stale reads

Which Solution fits me?

MySQL InnoDB Cluster	MySQL InnoDB ReplicaSet
• Single Primary, multiple Secondaries • Multi Primary mode (write to all)	• Single Primary, multiple Secondaries
• Read Scaleout with configurable consistent reads	• Read Scaleout with stale reads
• All members partake in consensus (write performance is as fast as it's slowest node)	• Secondary members only get stream of changes (binlog), does not impact Primary

Which Solution fits me?

MySQL InnoDB Cluster	MySQL InnoDB ReplicaSet
• Single Primary, multiple Secondaries • Multi Primary mode (write to all)	• Single Primary, multiple Secondaries
• Read Scaleout with configurable consistent reads	• Read Scaleout with stale reads
• All members partake in consensus (write performance is as fast as it's slowest node)	• Secondary members only get stream of changes (binlog), does not impact Primary
• Stable network is required: Network glitches cause stalls/membership changes • Uneven amount of members (3-5)	• No network requirements: glitches in network do not impact availability • 2 or more members

Which Solution fits me?

MySQL InnoDB Cluster	MySQL InnoDB ReplicaSet
• Single Primary, multiple Secondaries • Multi Primary mode (write to all)	• Single Primary, multiple Secondaries
• Read Scaleout with configurable consistent reads	• Read Scaleout with stale reads
• All members partake in consensus (write performance is as fast as it's slowest node)	• Secondary members only get stream of changes (binlog), does not impact Primary
• Stable network is required: Network glitches cause stalls/membership changes • Uneven amount of members (3-5)	• No network requirements: glitches in network do not impact availability • 2 or more members
• No data loss in case of member failure (Recovery Point Objective = 0 (RPO)) • Consistency	• During planned switchover: No data loss, split brain is handled • During manual Unplanned failover: up to user

Which Solution fits me?

MySQL InnoDB Cluster	MySQL InnoDB ReplicaSet
• Single Primary, multiple Secondaries • Multi Primary mode (write to all)	• Single Primary, multiple Secondaries
• Read Scaleout with configurable consistent reads	• Read Scaleout with stale reads
• All members partake in consensus (write performance is as fast as it's slowest node)	• Secondary members only get stream of changes (binlog), does not impact Primary
• Stable network is required: Network glitches cause stalls/membership changes • Uneven amount of members (3-5)	• No network requirements: glitches in network do not impact availability • 2 or more members
• No data loss in case of member failure (Recovery Point Objective = 0 (RPO)) • Consistency	• During planned switchover: No data loss, split brain is handled • During manual Unplanned failover: up to user
• Automatic Failover in case of failure (Easier to achieve a lower Recovery Time Objective (RTO) with stable network/servers)	• Manual switchover/failover only

MySQL Database Solutions

MySQL InnoDB Cluster

MySQL InnoDB ReplicaSet

http://www.mysql.com

http://dev.mysql.com/doc

http://www.mysqlhighavailability.com

http://www.mysqlserverteam.com

Kenny Gryp
MySQL Product Manager