ORACLE

MySQL

Disaster Recovery Solutions

MySQL InnoDB ClusterSet

Kenny Gryp

MySQL Product Manager

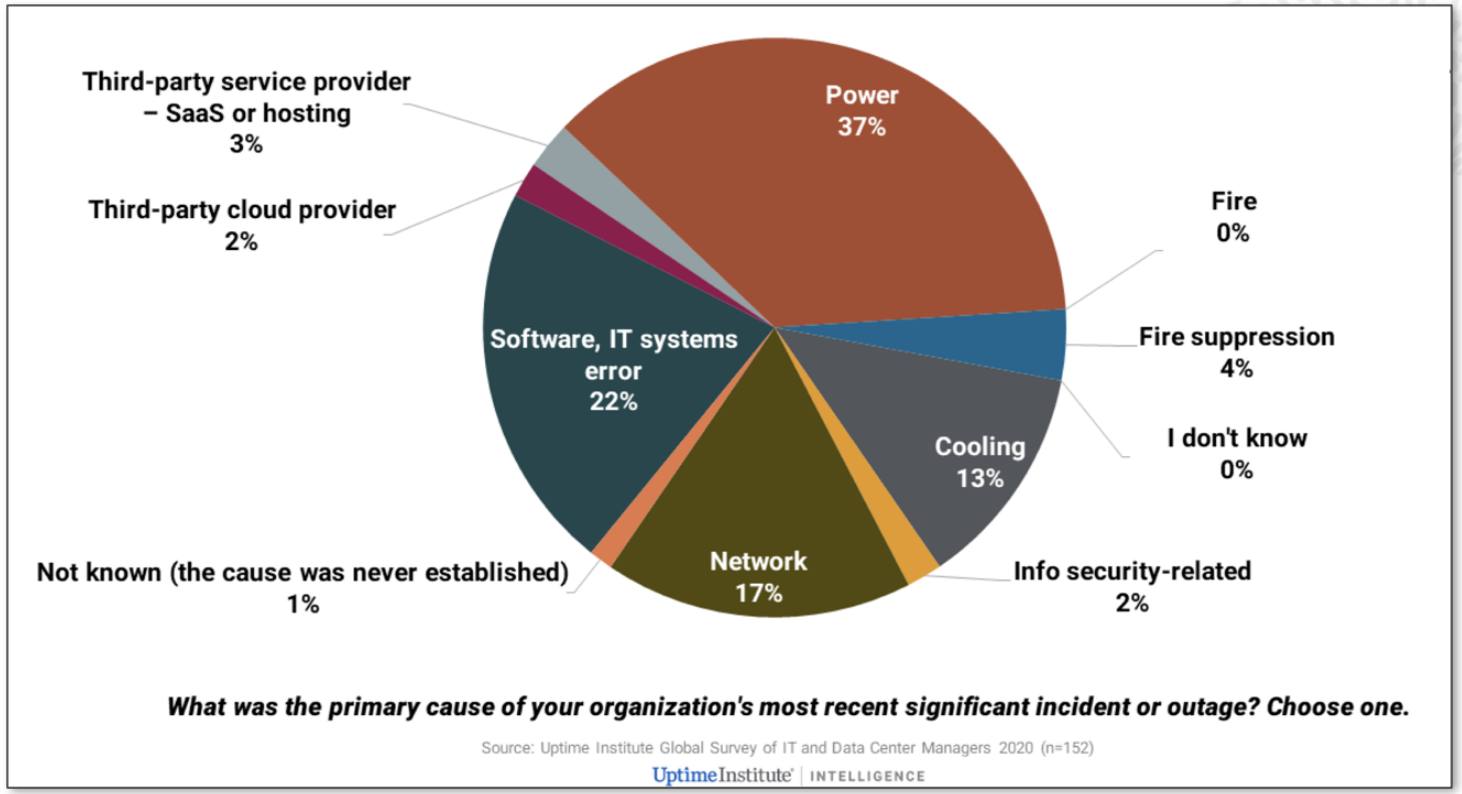
Miguel Araújo

MySQL Software Engineer

Safe Harbor Statement

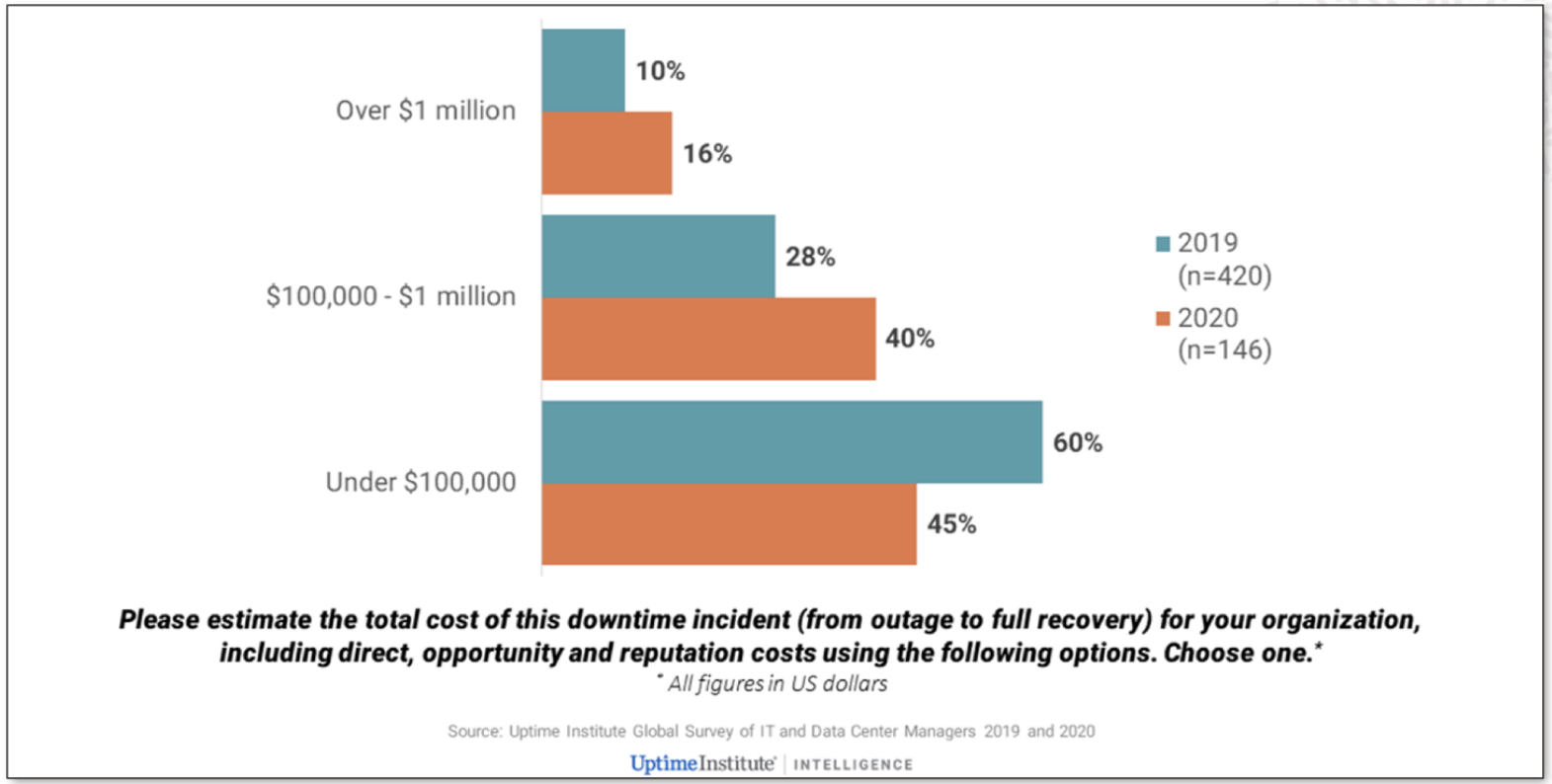
The following is intended to outline our general product direction. It is intended for information purpose only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied up in making purchasing decisions. The development,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

IT Disasters & Outages: Primary Causes



On-site power failure is the biggest cause of significant outages

IT Disasters & Outages: Costs are Rising



Over half who had experienced an outage costing more than $100,000.

IT Disasters and Outages: Examples

DELTA

5-hour computer outage cost us $150 million. The airline eventually canceled about 1,000 flights on the day of the outage and ground an additional 1,000 flights over the following two days.

OVHcloud

Millions of websites offline after fire at French cloud services firm. The fire is expected to cost the company more than €105 million.

BRITISH AIRWAYS

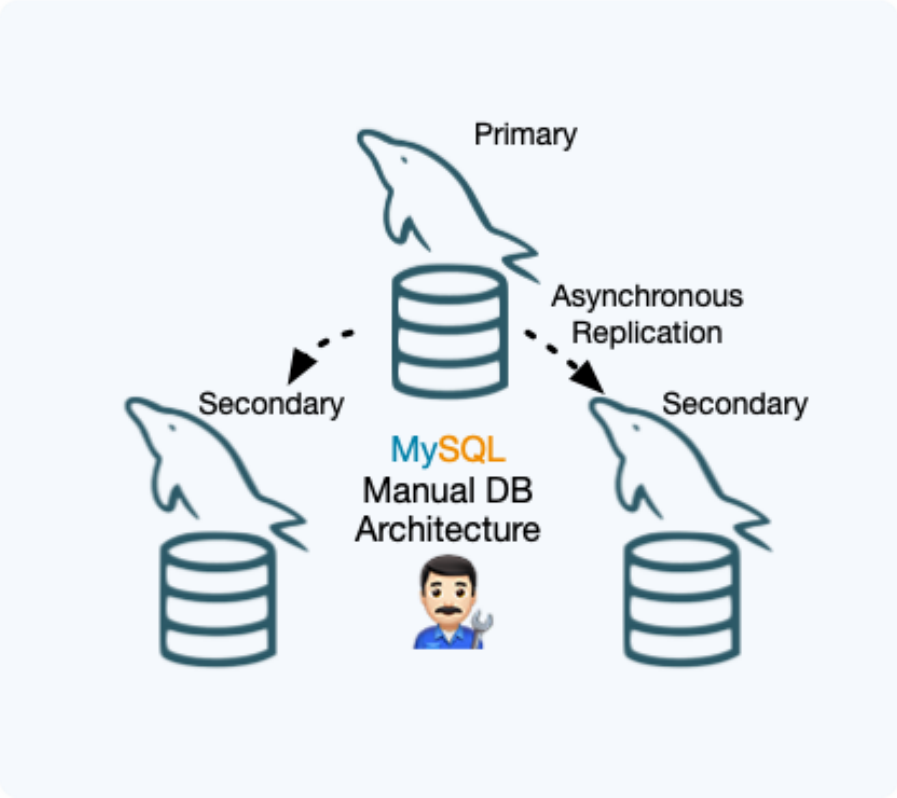
Tens of thousands of passengers were stranded in cities around the world due to cancellation of about 130 flights and the delay of 200.

HSBC

Millions of bank customers were unable to access online accounts. The bank took almost 2 days to recover and get back to normal functioning.

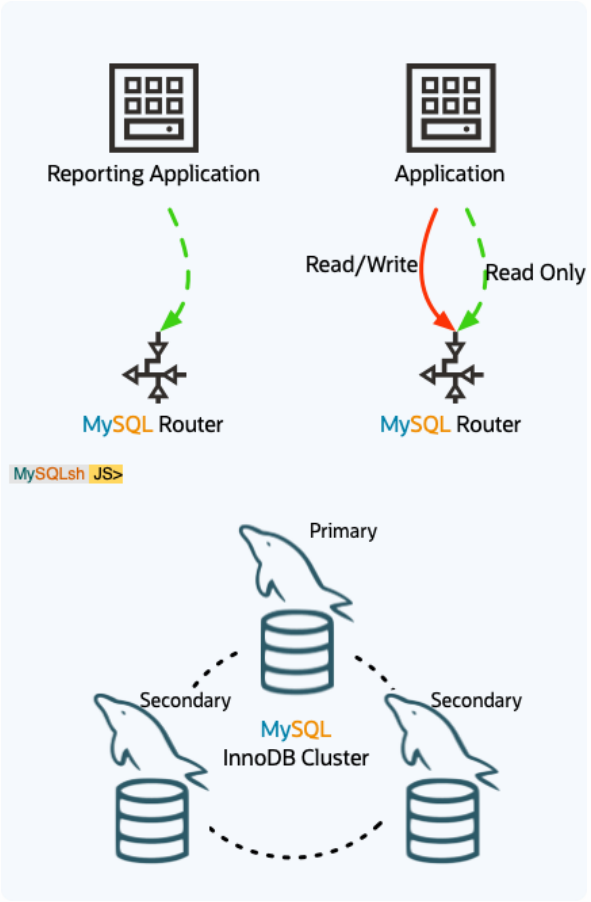
Past, Present & Future

'Past' - Manual



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

Present - Solutions!

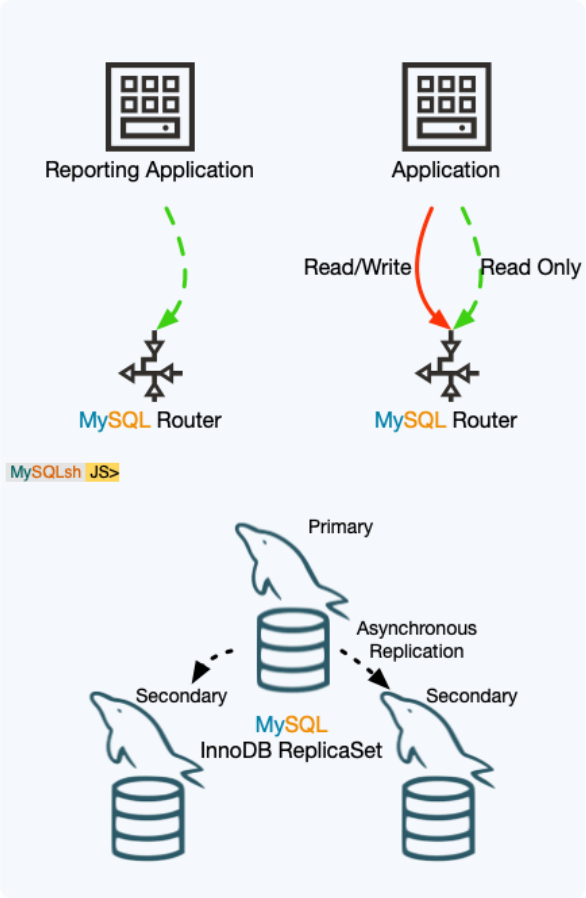


RPO = 0
RTO = seconds (automatic failover)

2016 - MySQL InnoDB Cluster

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

Present - Solutions!



RPO != 0
RTO = minutes (manual failover)

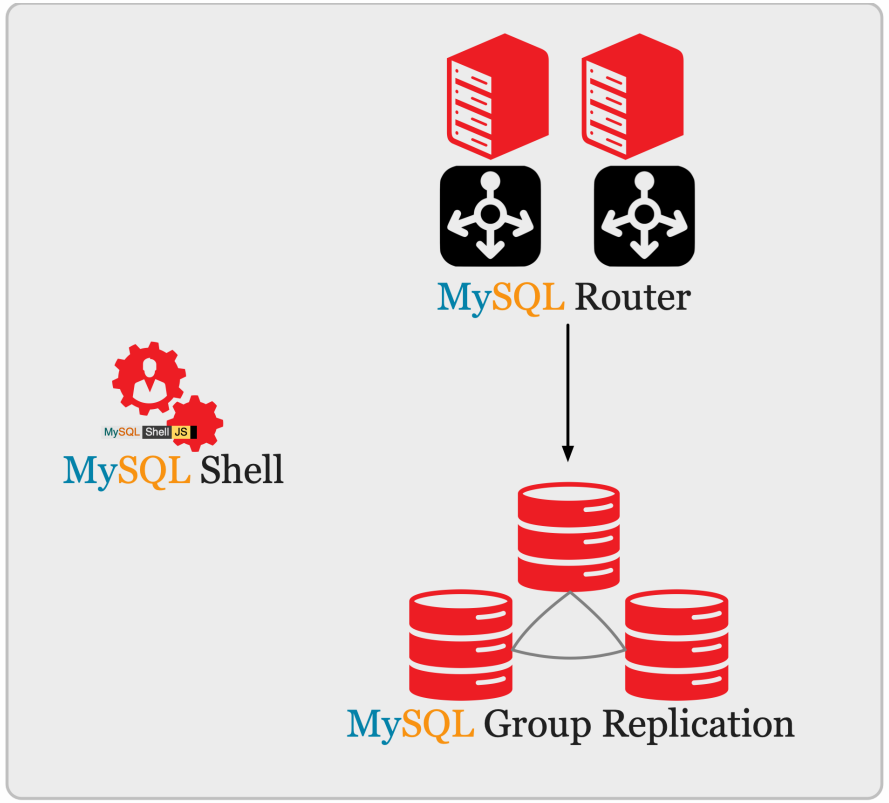
2020 - MySQL InnoDB Replicaset

- 'classic', 'asynchronous' Replication based Solution, fully integrated
- MySQL Shell
- MySQL Router
- MySQL Server

MySQL InnoDB Cluster

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

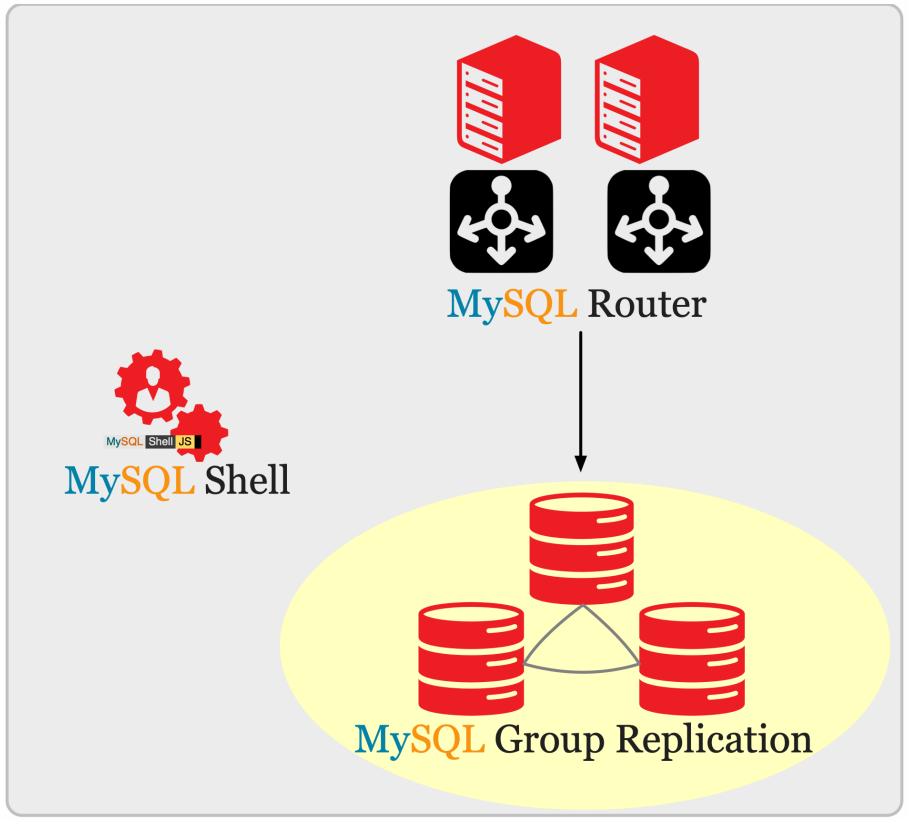
One Product: MySQL

- 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



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

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

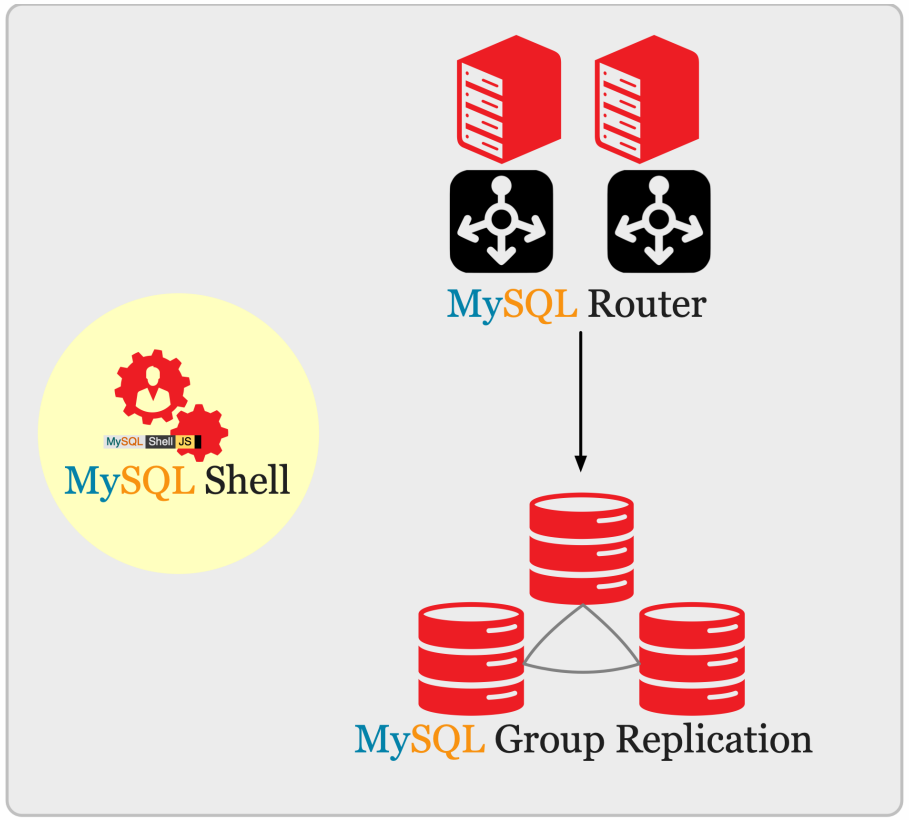
Read Scaleout

- 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 Shell

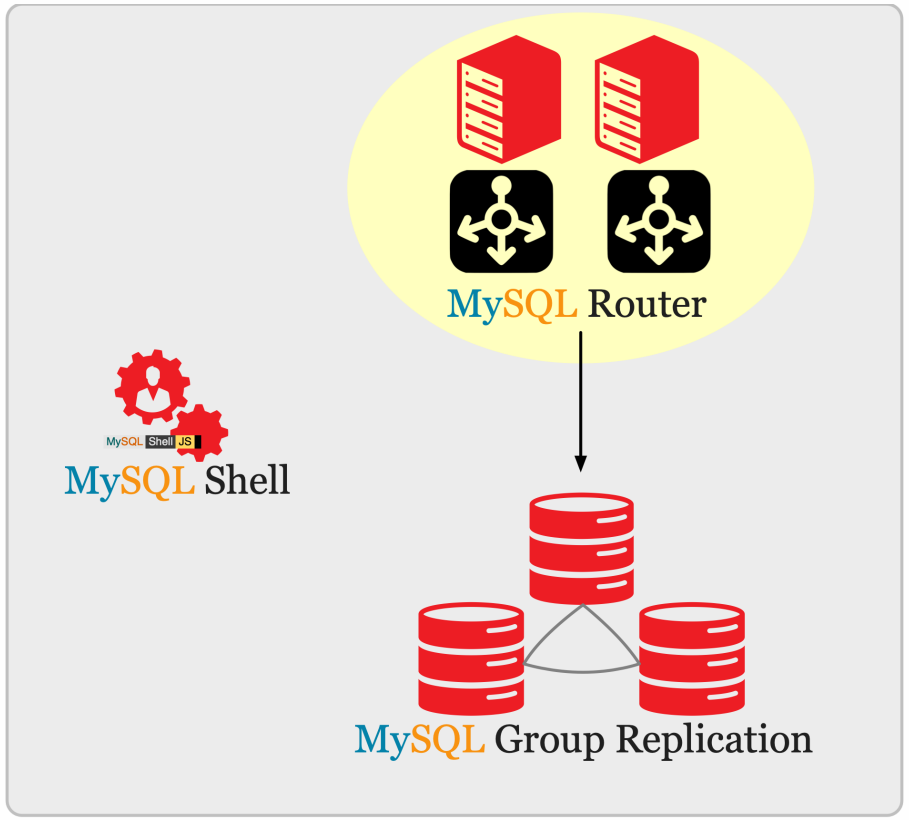


Database Administration Interface

"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 Router



Transparent Access to Database Arch.

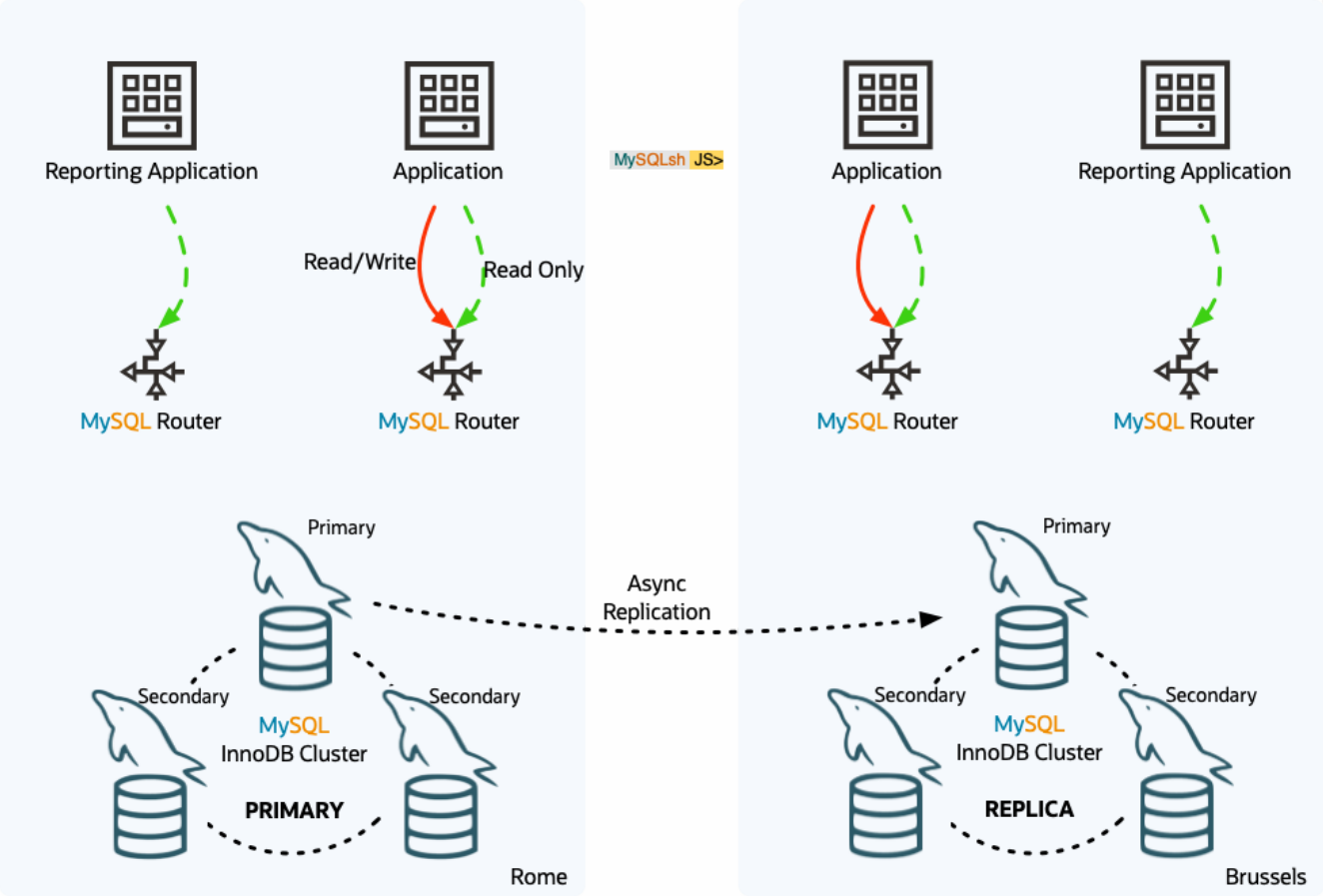
"provide transparent routing between your application and back-end MySQL Servers"

- 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 ReplicaSet/Cluster/ClusterSet
- 2 TCP Ports: `PRIMARY` and `NON-PRIMARY` traffic

MySQL InnoDB ClusterSet

MySQL InnoDB ClusterSet

One or more REPLICA MySQL InnoDB Clusters attached to a PRIMARY MySQL InnoDB Cluster



High Availability (Failure Within a Region)

- RPO = 0
- RTO = seconds (automatic failover)

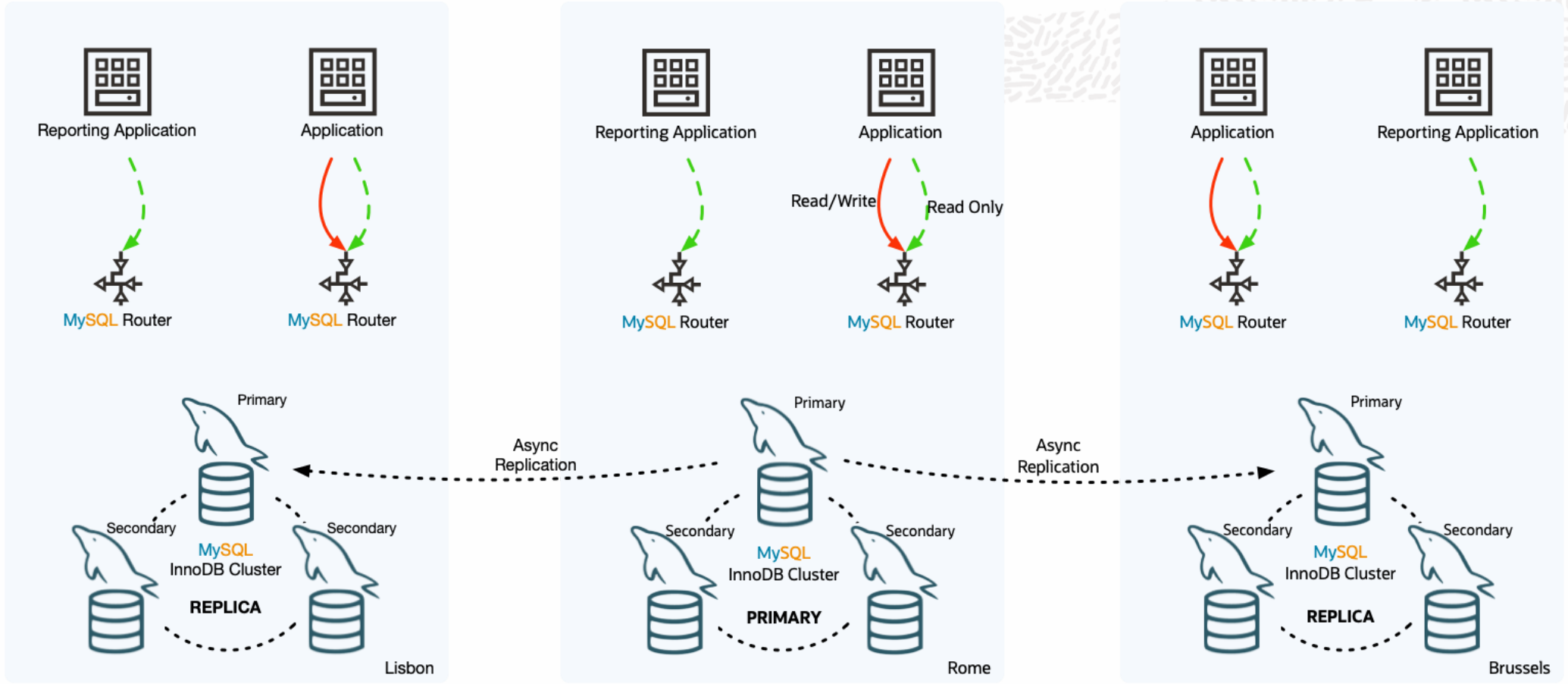
Disaster Recovery (Region Failure)

- RPO != 0
- RTO = minutes or more (manual failover)
- No write performance impact

Features

- Easy to use
- Familiar interface and usability
  `mysqlsh`, `CLONE`, ...
- Add/remove nodes/clusters online
- Router integration, no need to reconfigure application if the topology changes

MySQL InnoDB ClusterSet - 3 Datacenters



Business Requirements

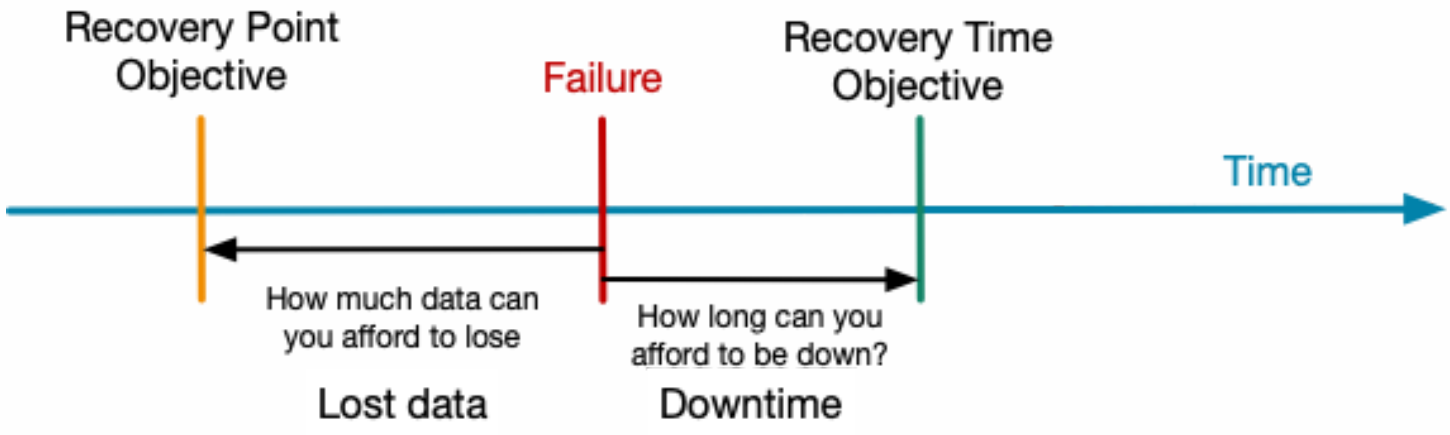
Business Requirements

Concepts - RTO & RPO

- RTO: Recovery Time Objective
  - How long does it take to recover from a single failure
- RPO: Recovery Point Objective
  - How much data can be lost when a failure occurs

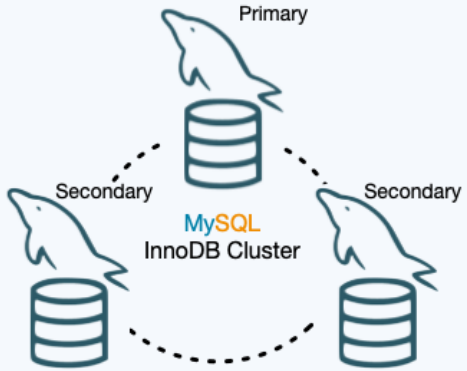
Types of Failure:

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



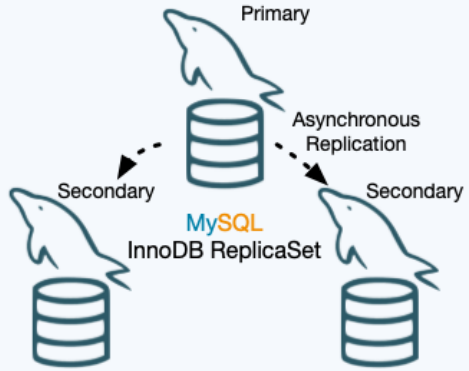
High Availability - Single Region

MySQL InnoDB Cluster



- RPO = 0
- RTO = Seconds

MySQL InnoDB ReplicaSet



- RPO != 0
- RTO = Minutes+ (manual failover)

👍 Best write performance

👎 Manual failover

Disaster Recovery - Multi Region

MySQL InnoDB Cluster

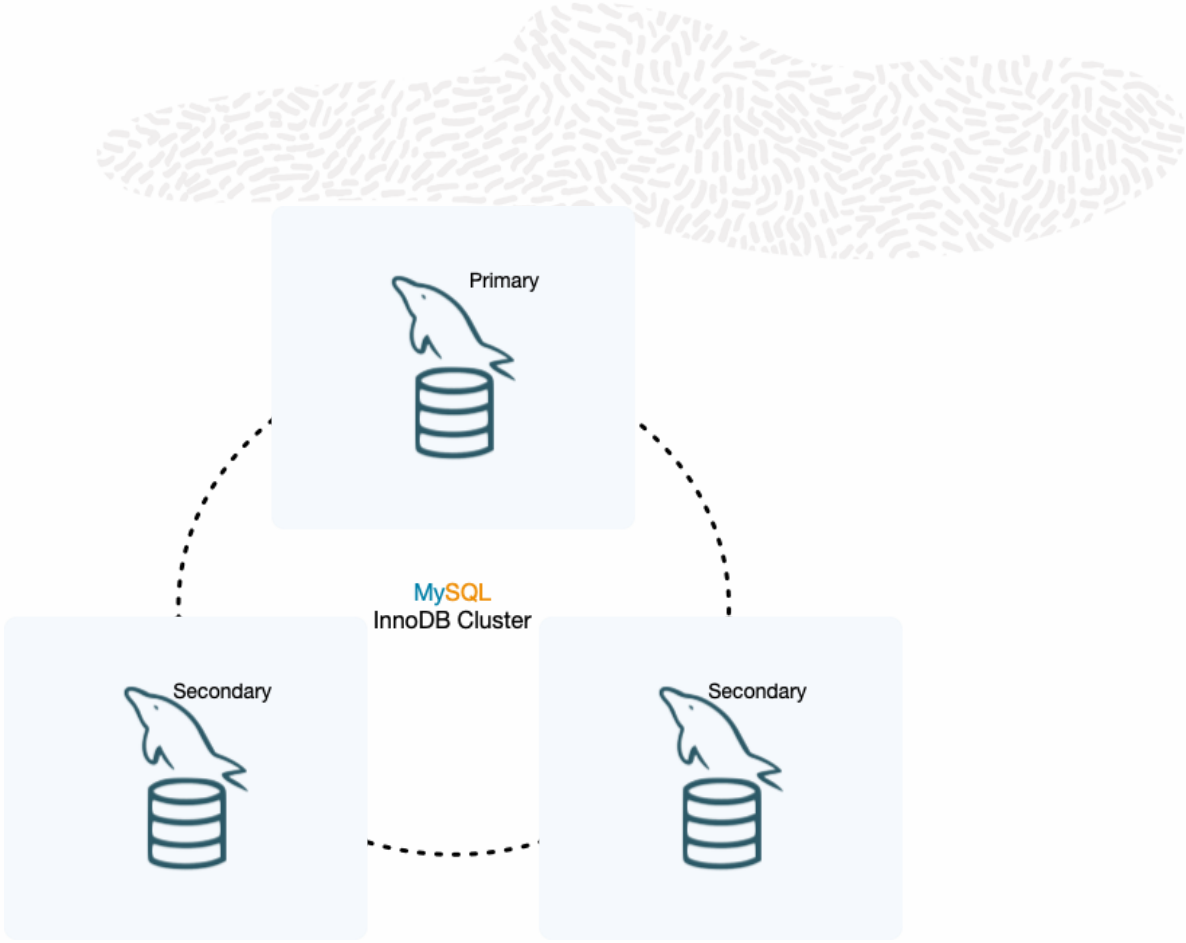
- RPO = 0
- RTO = Seconds

👍 Multi-Region Multi-Primary

👎 3 DC

👎 Requires very stable WAN

👎 Write performance affected by latency between dc's



Disaster Recovery - Multi Region

MySQL InnoDB ClusterSet

- RPO != 0
- RTO = Minutes+ (manual failover)

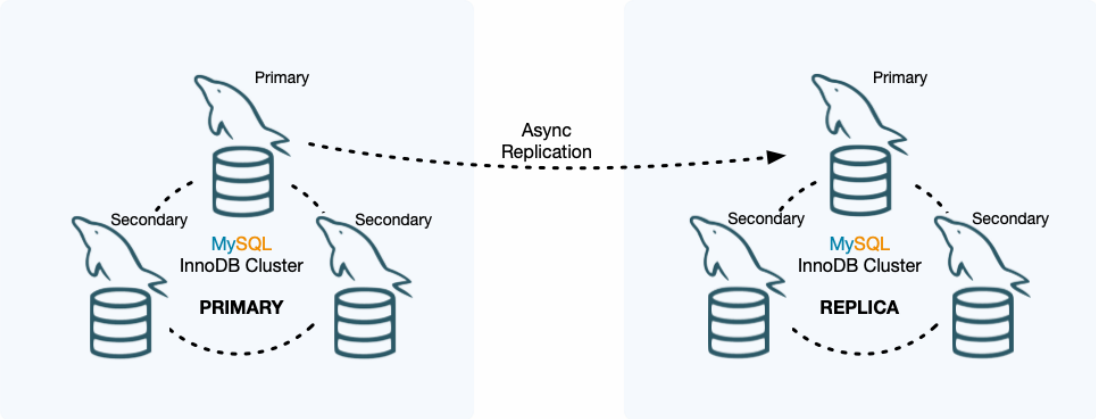
👍 RPO = 0 & RTO = seconds within Region (HA)

👍 Write performance (no sync to other region required)

👎 Higher RTO: Manual failover

👎 RPO != 0 when region fails

MySQL InnoDB ClusterSet



MySQL InnoDB ClusterSet
Demo

ClusterSet Demo

Environment, 3 regions, 3 mysql databases each, listening on different ports:

- ROM:
  - `rome:3331`
  - `rome:3332`
  - `rome:3333`
- BRU:
  - `brussels:4441`
  - `brussels:4442`
  - `brussels:4443`
- LIS:
  - `lisbon:5551`
  - `lisbon:5552`
  - `lisbon:5553`

Commands used in demo available on https://github.com/miguelaraujo/ClusterSet-Demo

Demo

Initial Setup

- Create MySQL InnoDB Cluster
- Create ClusterSet with 3 clusters
- ClusterSet Status
- Router Bootstrap

Change PRIMARYs

- Change `PRIMARY` member in `PRIMARY` cluster
- Change `PRIMARY` member in `REPLICA` cluster
- Change `PRIMARY` Cluster - `setPrimaryCluster()`

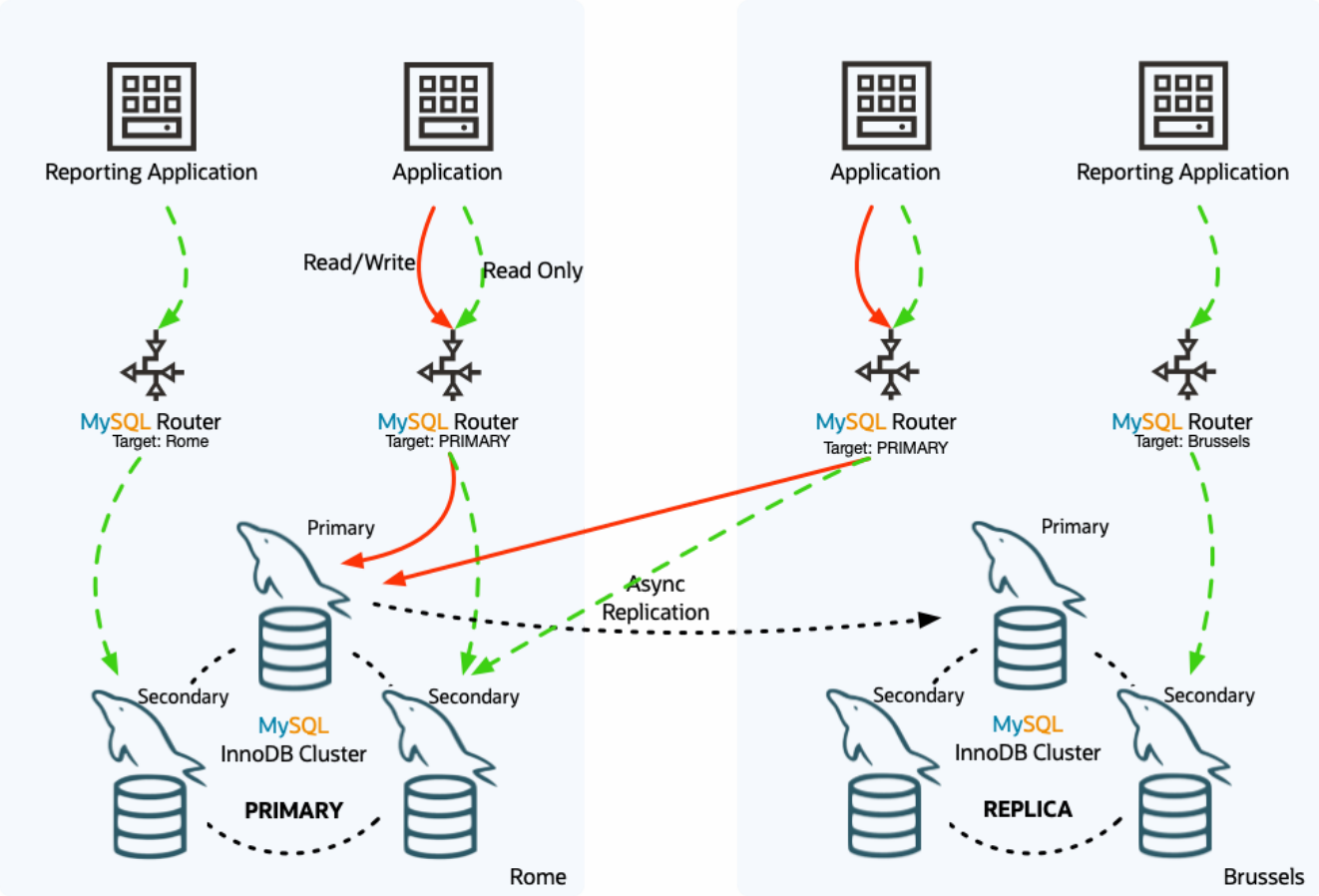
Router

- Changing Router Configuration Options
- Router Status with Cluster changes
- Router Logs

Failure Scenarios

- Automatic Handling of `PRIMARY` member in `PRIMARY` cluster
- Automatic Handling of `PRIMARY` member in `REPLICA` cluster
- Disaster - `PRIMARY` Cluster - Network Partition

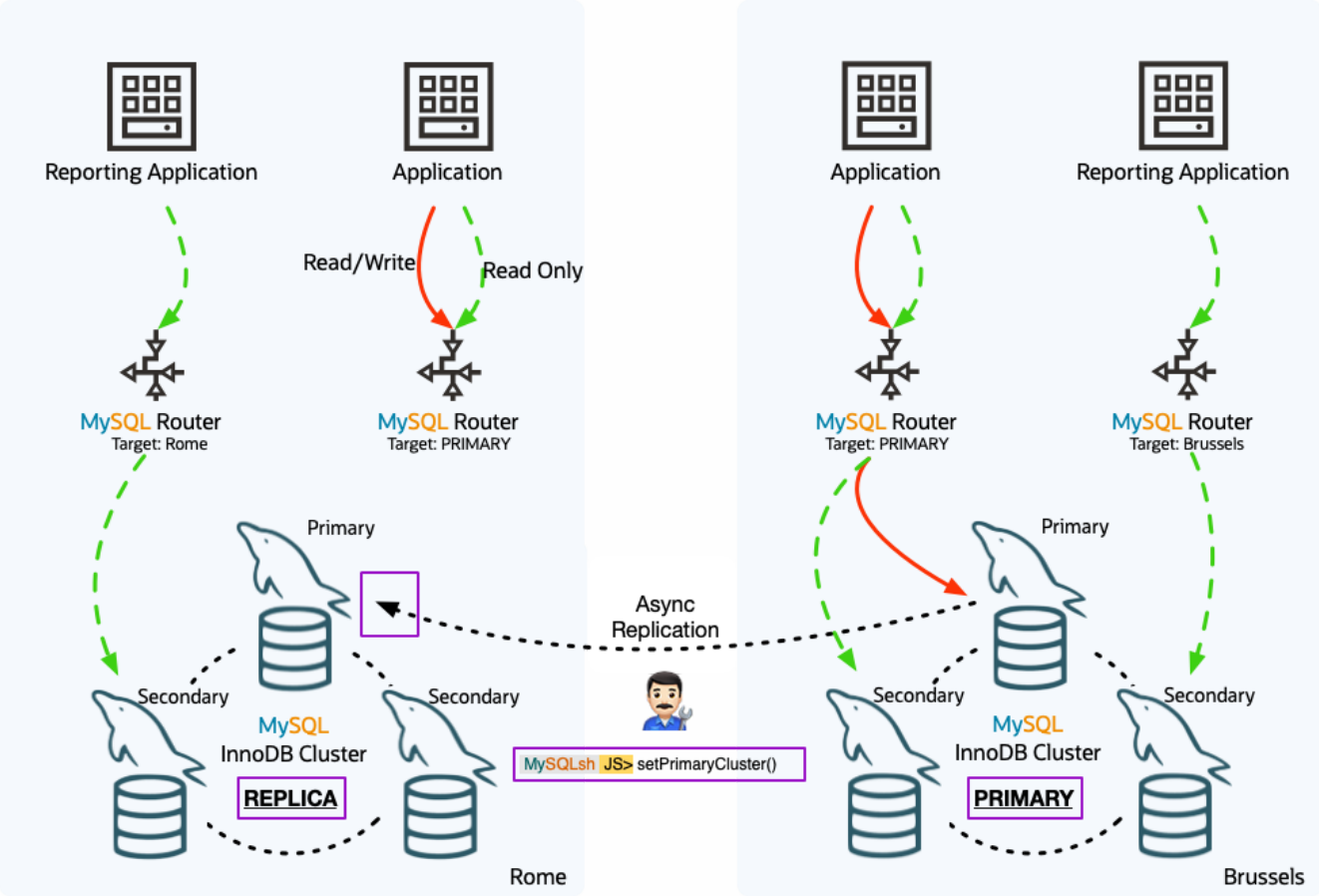
Changing Primary - Change Primary Cluster on Healthy System



Switchover

- one command that does it all: `setPrimaryCluster()`
- Asynchronous replication channels between clusters are automatically reconfigured
- Consistency guaranteed
- All routers will immediately redirect if needed (depending on target mode)

Changing Primary - `setPrimaryCluster()`



Switchover

- one command that does it all: `setPrimaryCluster()`
- Asynchronous replication channels between clusters are automatically reconfigured
- Consistency guaranteed
- All routers will immediately redirect if needed (depending on target mode)

Demo

Initial Setup

- Create MySQL InnoDB Cluster
- Create ClusterSet with 3 clusters
- ClusterSet Status
- Router Bootstrap

Change PRIMARYs

- Change `PRIMARY` member in `PRIMARY` cluster
- Change `PRIMARY` member in `REPLICA` cluster
- Change `PRIMARY` Cluster - `setPrimaryCluster()`

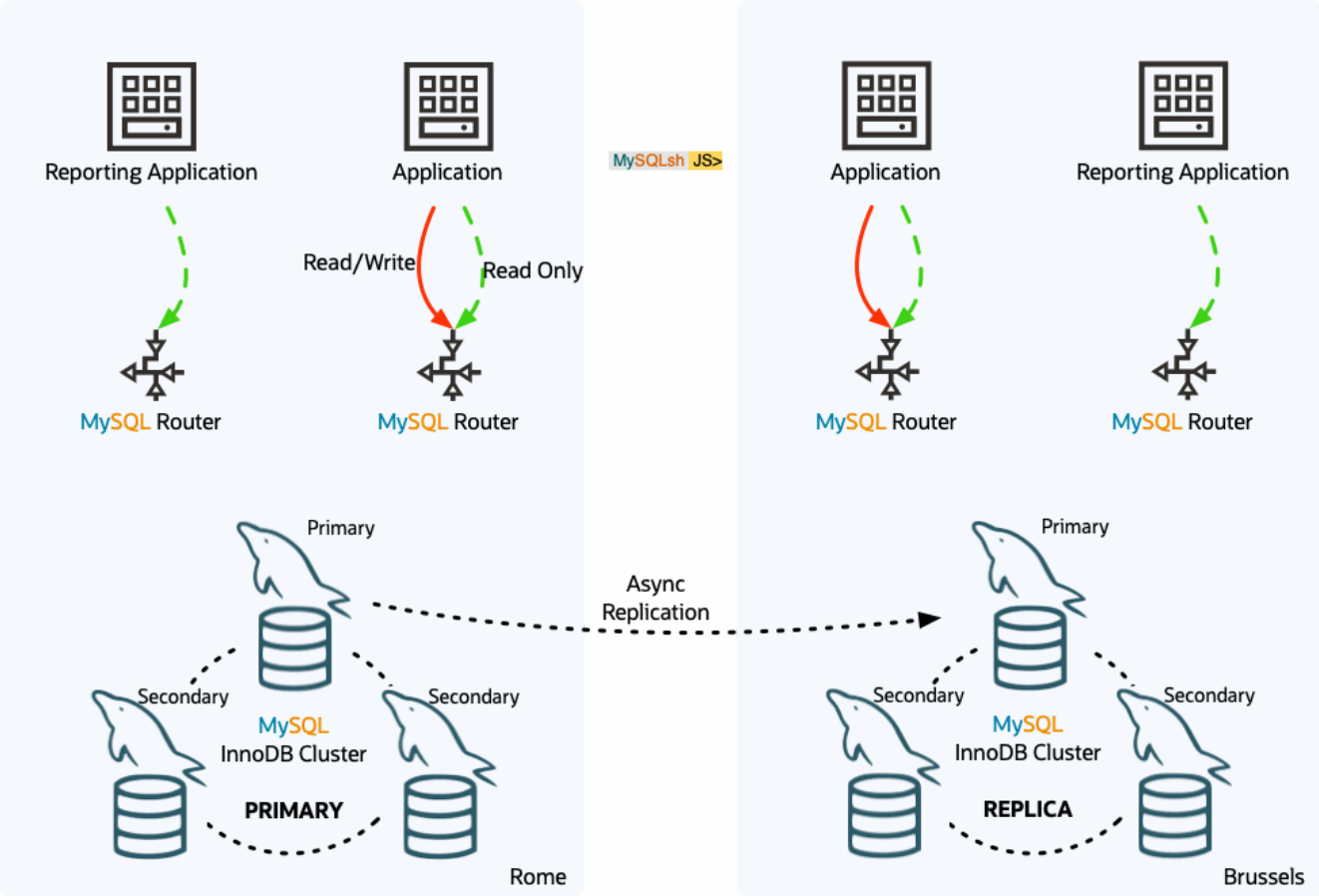
Router

- Changing Router Configuration Options
- Router Status with Cluster changes
- Router Logs

Failure Scenarios

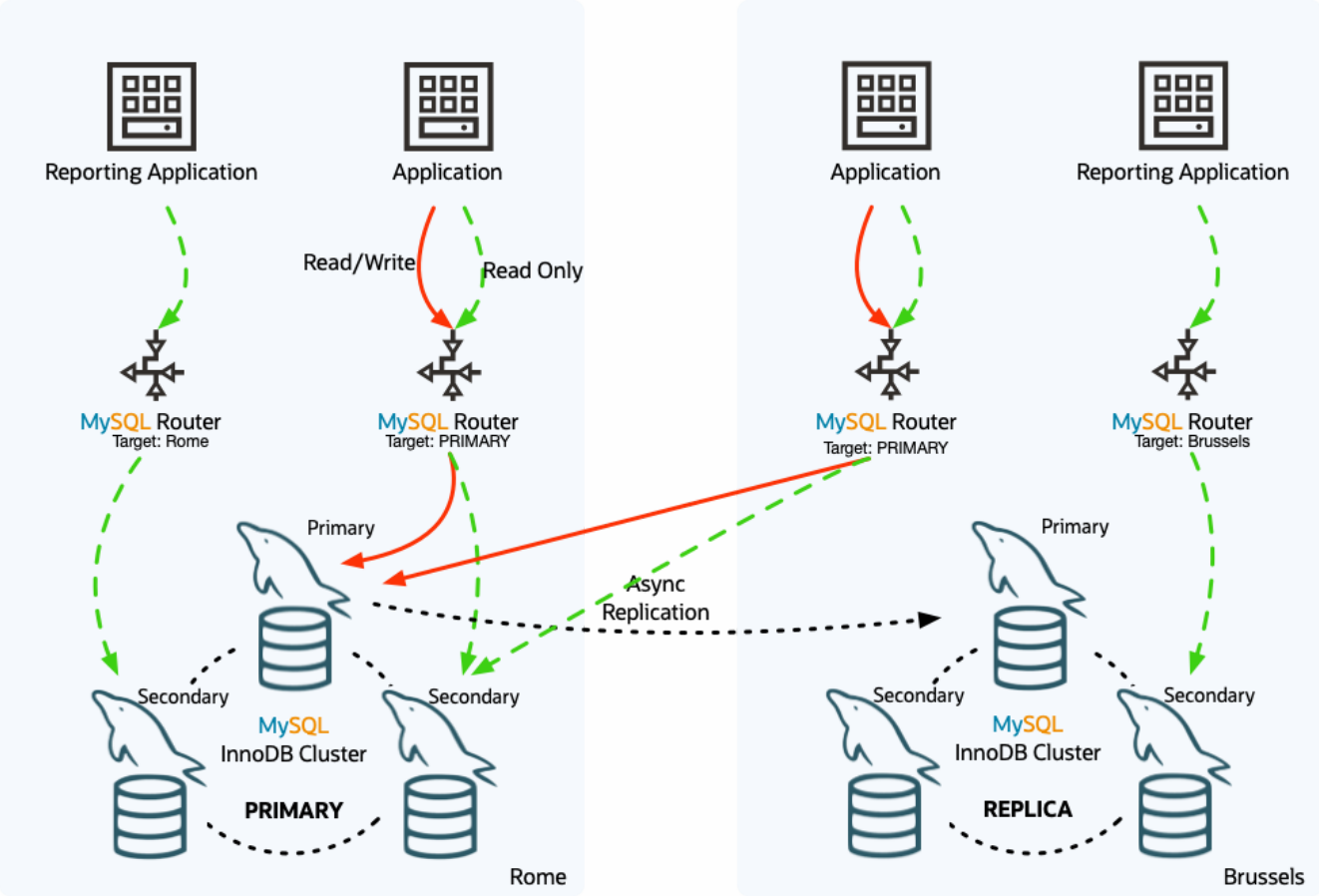
- Automatic Handling of `PRIMARY` member in `PRIMARY` cluster
- Automatic Handling of `PRIMARY` member in `REPLICA` cluster
- Disaster - `PRIMARY` Cluster - Network Partition

Router Integration



Configure your application to connect to a local MySQL Router to connect to the ClusterSet.

Router Integration



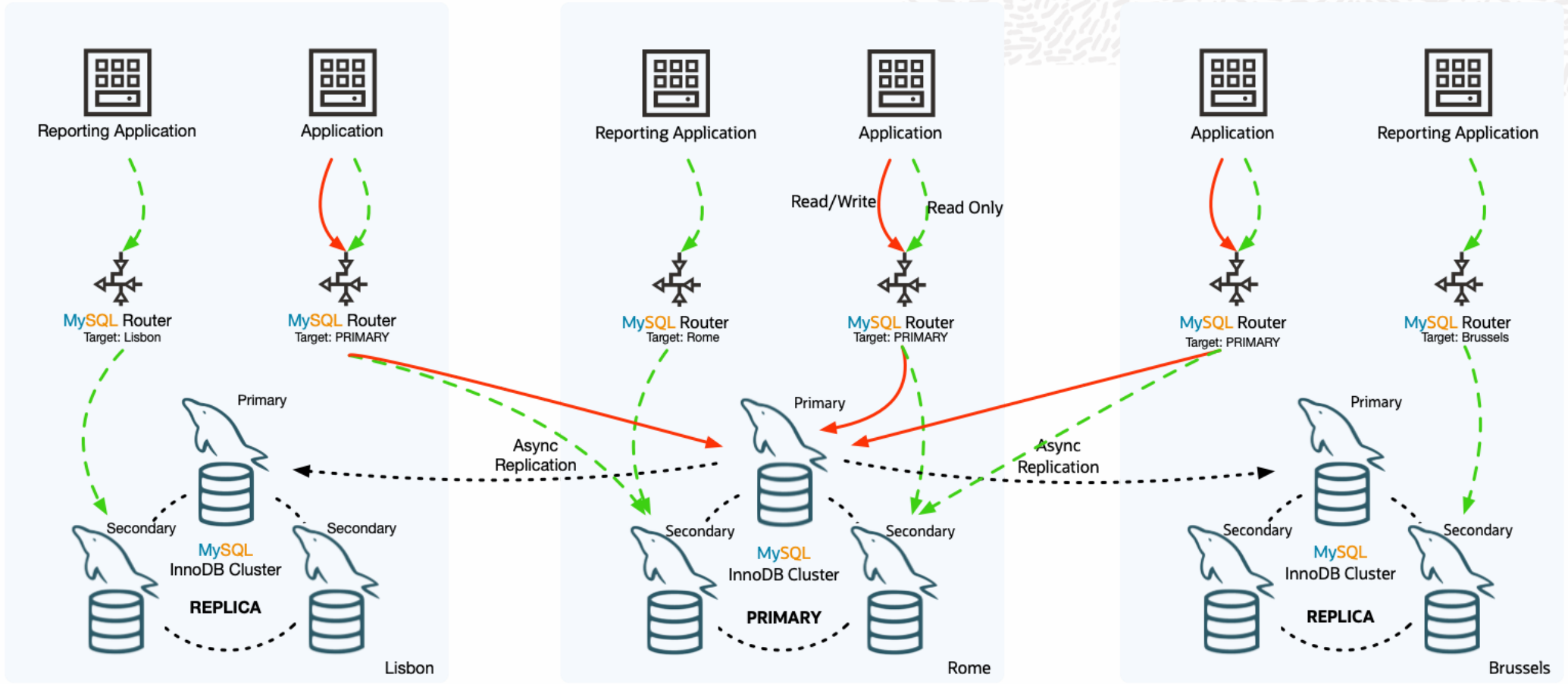
Router Target Modes:

- follow the `PRIMARY` cluster
  - Writes & Reads go to the `PRIMARY` Cluster
- connect to the configured target cluster
  - When target cluster is not `PRIMARY`:
    - only read traffic is open
    - writes will be denied
  - when target cluster is `PRIMARY`
    - write port opens

Features:

- Configurable per Router instance
- Configuration can be changed `ONLINE` in `mysqlsh`
- Deploy 2 types of routers:
  - target `PRIMARY` to send writes to `PRIMARY`
  - define target cluster to keep read traffic local
- `INVALIDATED` clusters can still be used for read traffic (configurable)

Router Integration - 3DC



Demo

Initial Setup

- Create MySQL InnoDB Cluster
- Create ClusterSet with 3 clusters
- ClusterSet Status
- Router Bootstrap

Change PRIMARYs

- Change `PRIMARY` member in `PRIMARY` cluster
- Change `PRIMARY` member in `REPLICA` cluster
- Change `PRIMARY` Cluster - `setPrimaryCluster()`

Router

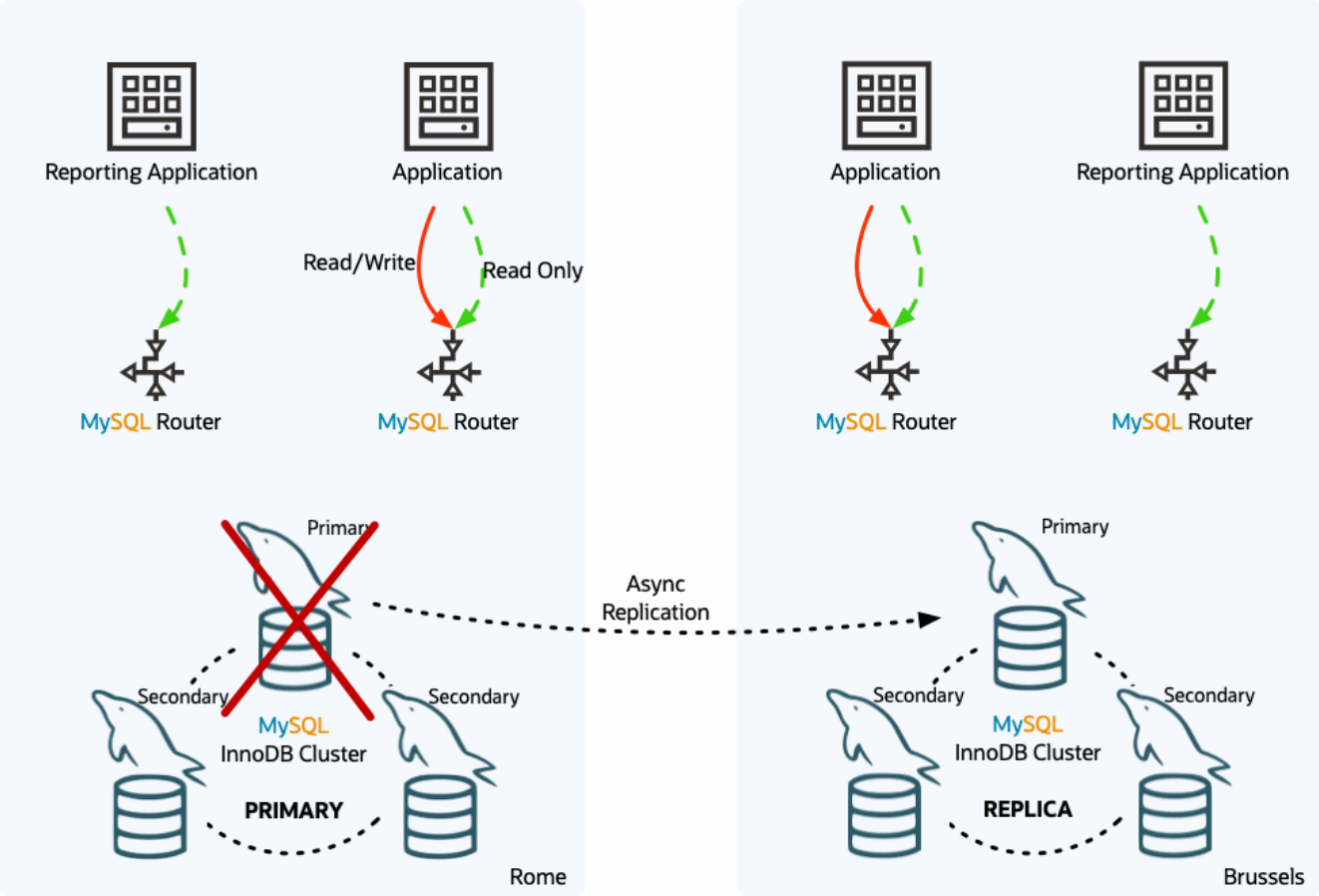
- Changing Router Configuration Options
- Router Status with Cluster changes
- Router Logs

Failure Scenarios

- Automatic Handling of `PRIMARY` member in `PRIMARY` cluster
- Automatic Handling of `PRIMARY` member in `REPLICA` cluster
- Disaster - `PRIMARY` Cluster - Network Partition

ClusterSet Scenarios

PRIMARY Cluster PRIMARY member Crash/Partition

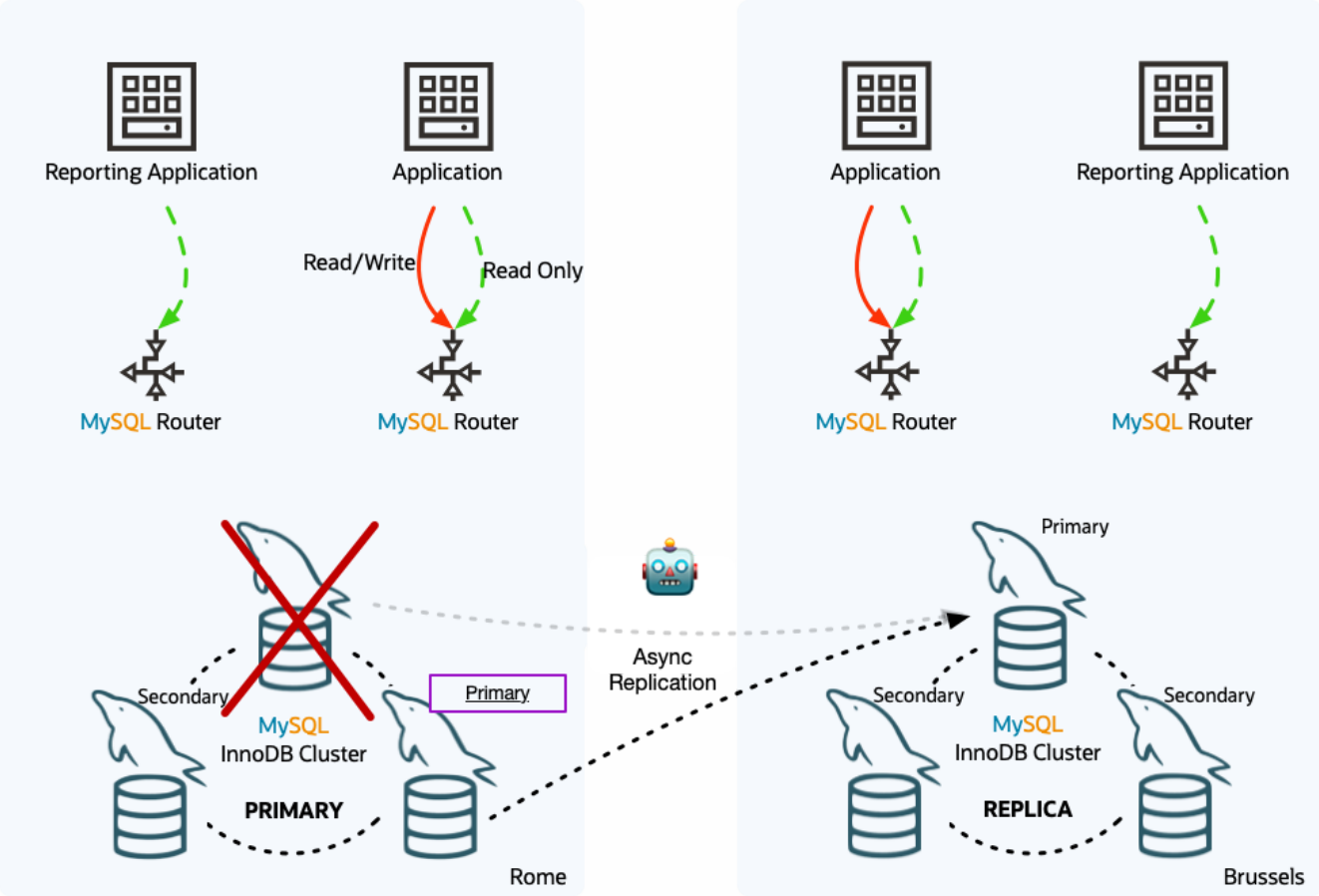


- When there is newly elected PRIMARY member in a cluster
- Works on failures in PRIMARY and REPLICA clusters

Automatic Handling of InnoDB Cluster state changes

- Asynchronous replication is automatically reconfigured after primary change

PRIMARY Cluster PRIMARY member Crash/Partition - Automatic!

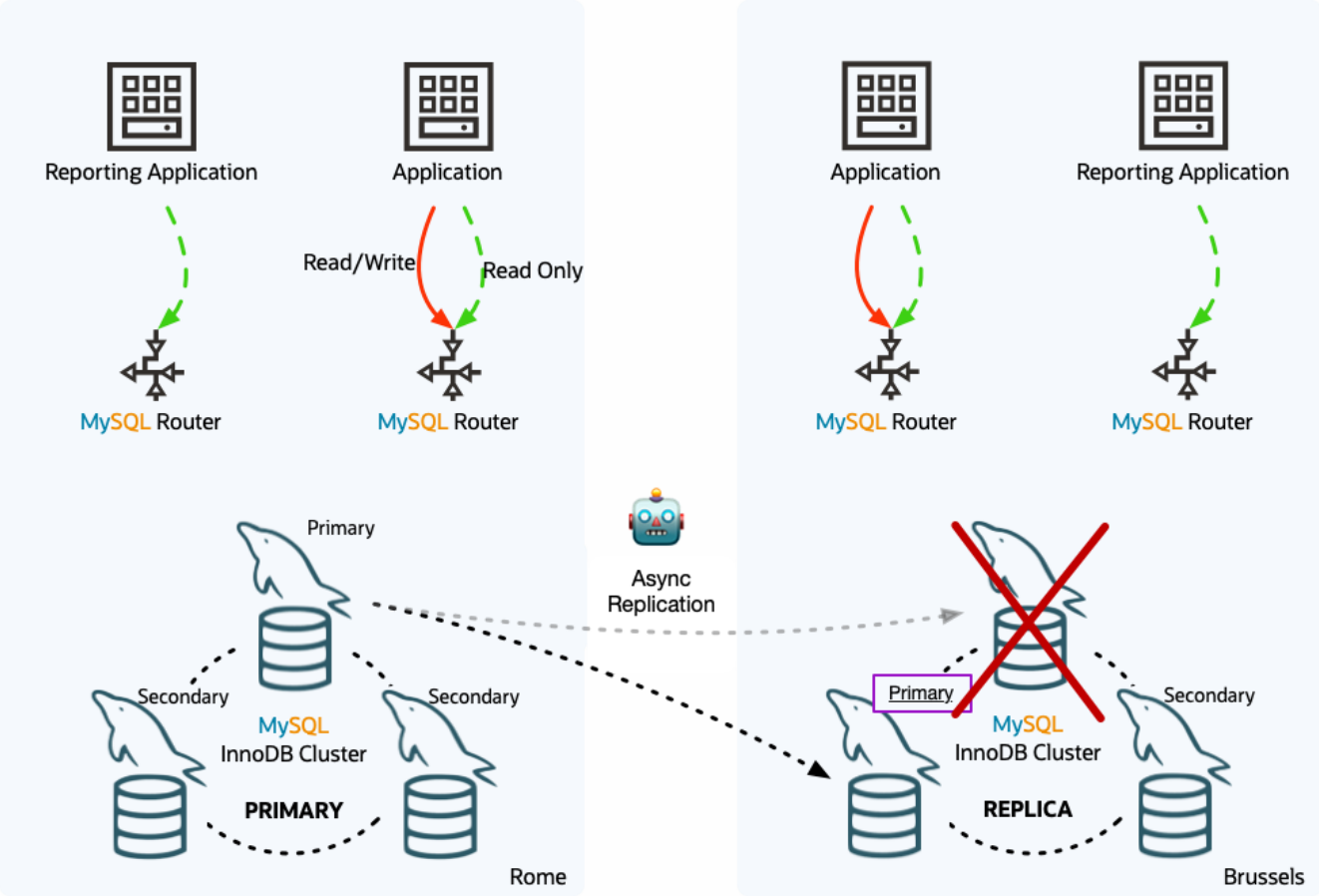


- When there is newly elected PRIMARY member in a cluster
- Works on failures in PRIMARY and REPLICA clusters

Automatic Handling of InnoDB Cluster state changes

- Asynchronous replication is automatically reconfigured after primary change

REPLICA Cluster PRIMARY member Crash/Partition - Automatic!

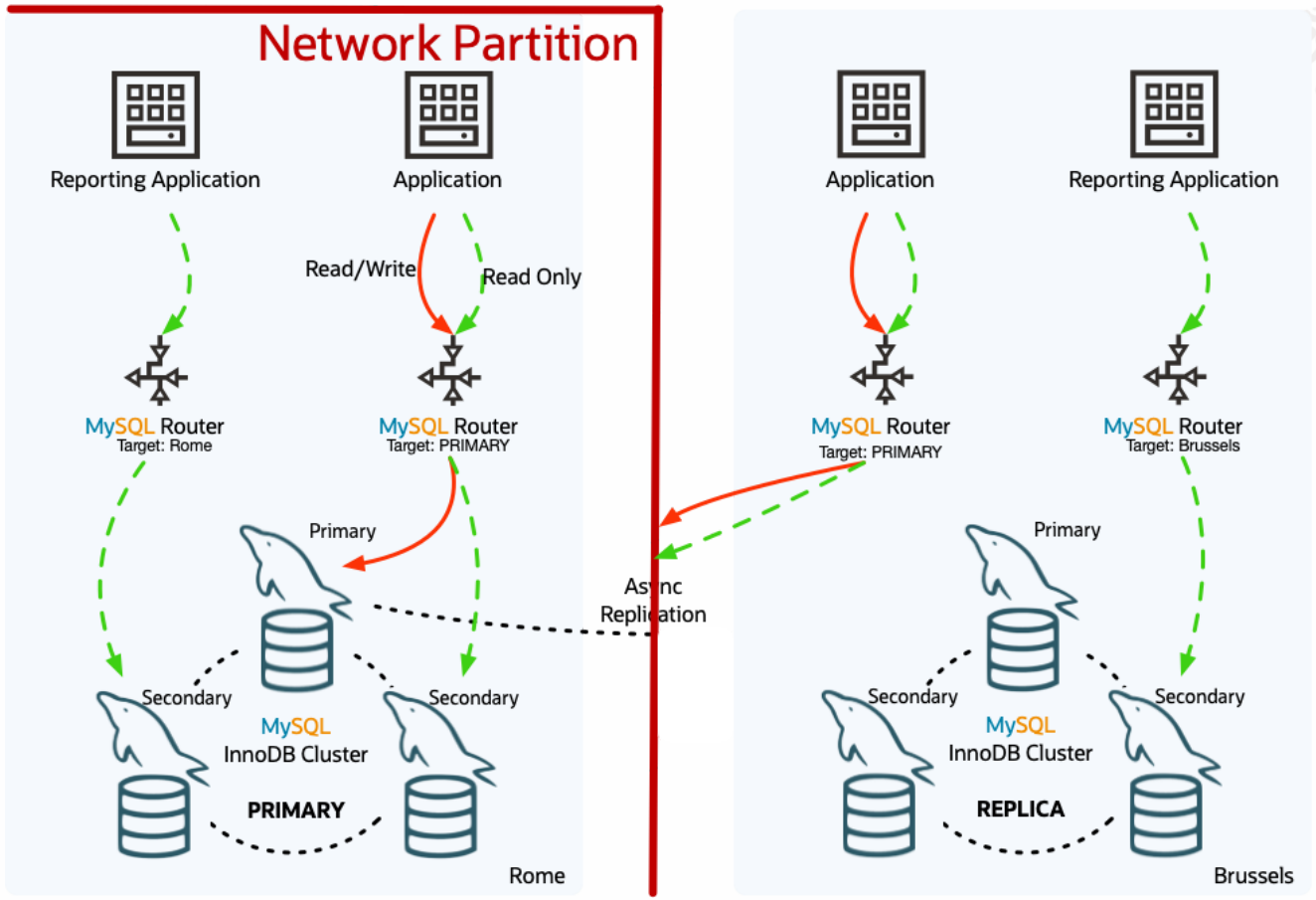


- When there is newly elected PRIMARY member in a cluster
- Works on failures in PRIMARY and REPLICA clusters

Automatic Handling of InnoDB Cluster state changes

- Asynchronous replication is automatically reconfigured after primary change

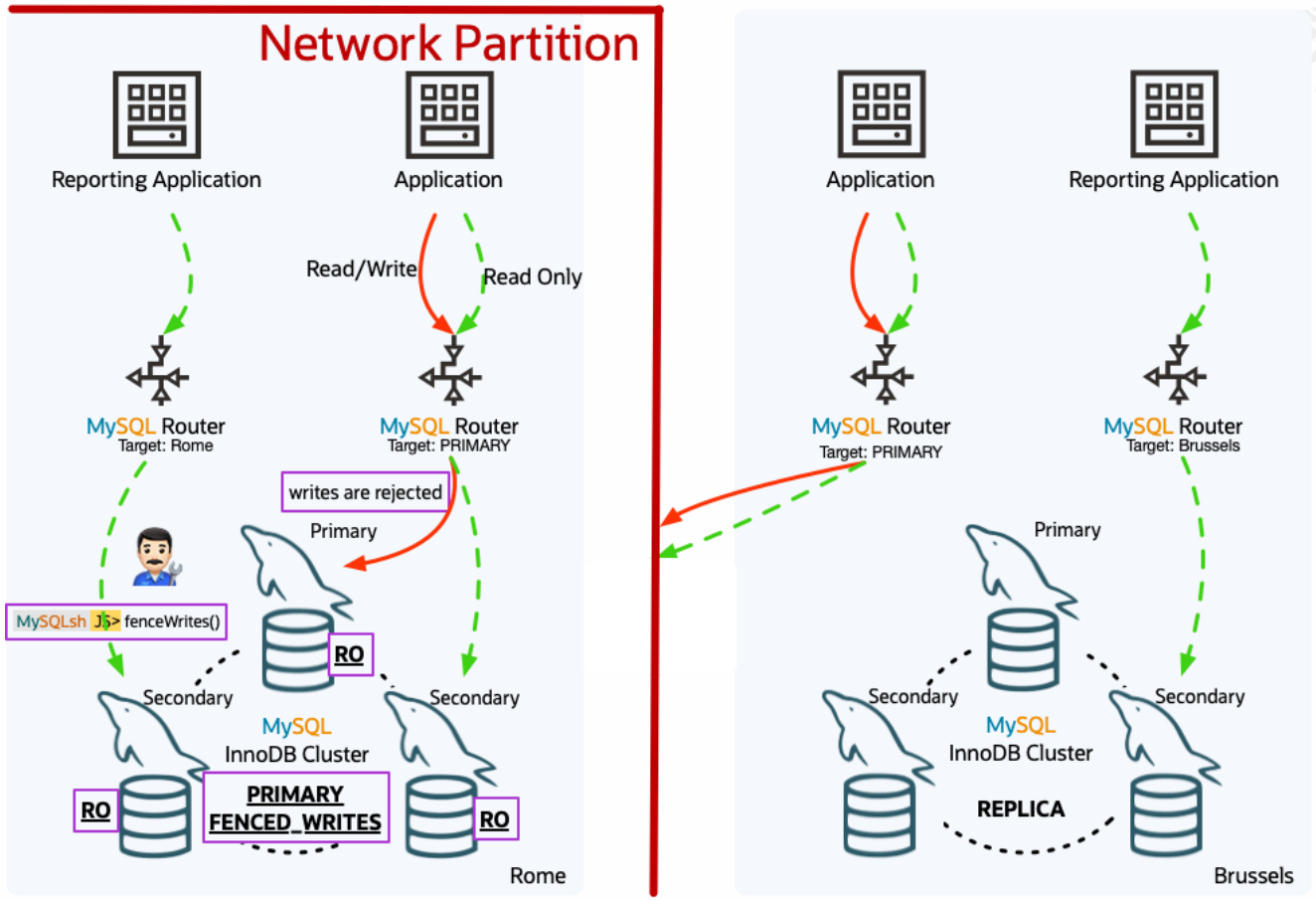
Disaster - PRIMARY Cluster - Network Partition



Network Partition

- PRIMARY Cluster is network partitioned and cannot longer serve traffic
- REPLICA Cluster is ready to be promoted

Disaster - PRIMARY Cluster - Network Partition - fenceWrites()

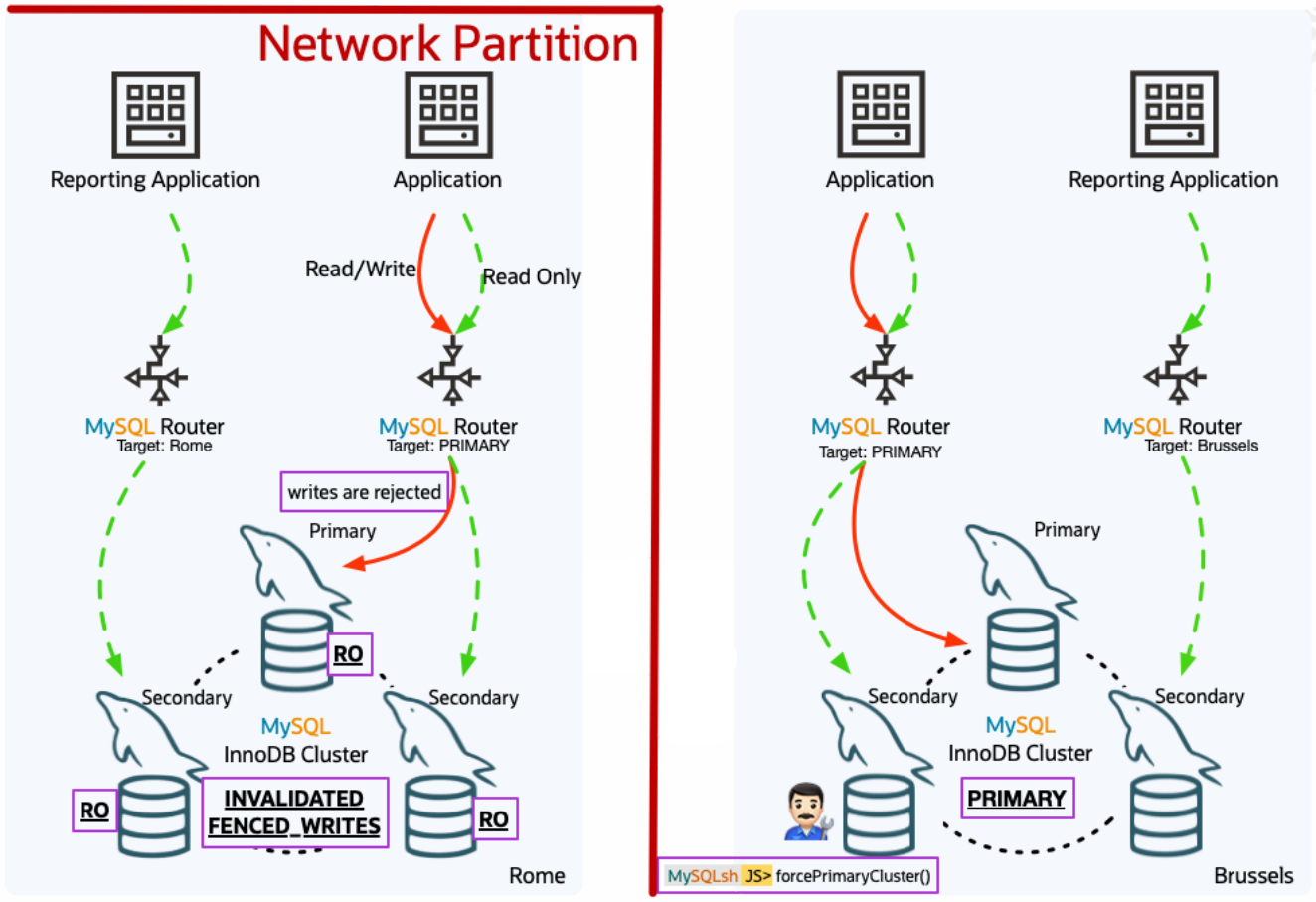


New feature in MySQL 8.0.28!

Fencing PRIMARY Cluster

- Reducing split brain
- Single command for operator to halt traffic and give time to investigate the problem
- puts all members & cluster in `super_read_only=1`
- 2 options:
  - `fenceWrites()`
    - Router does not accept writes
    - Router still accepts reads traffic
  - `fenceAllTraffic()`
    - Stops Group Replication on all members
    - Router does not accept Reads & Writes.
- Next action, is to choose to:
  - Failover to a `REPLICA` Cluster (see next slide)
  - Resolve Network partition & `unfenceWrites()` or reboot Cluster.

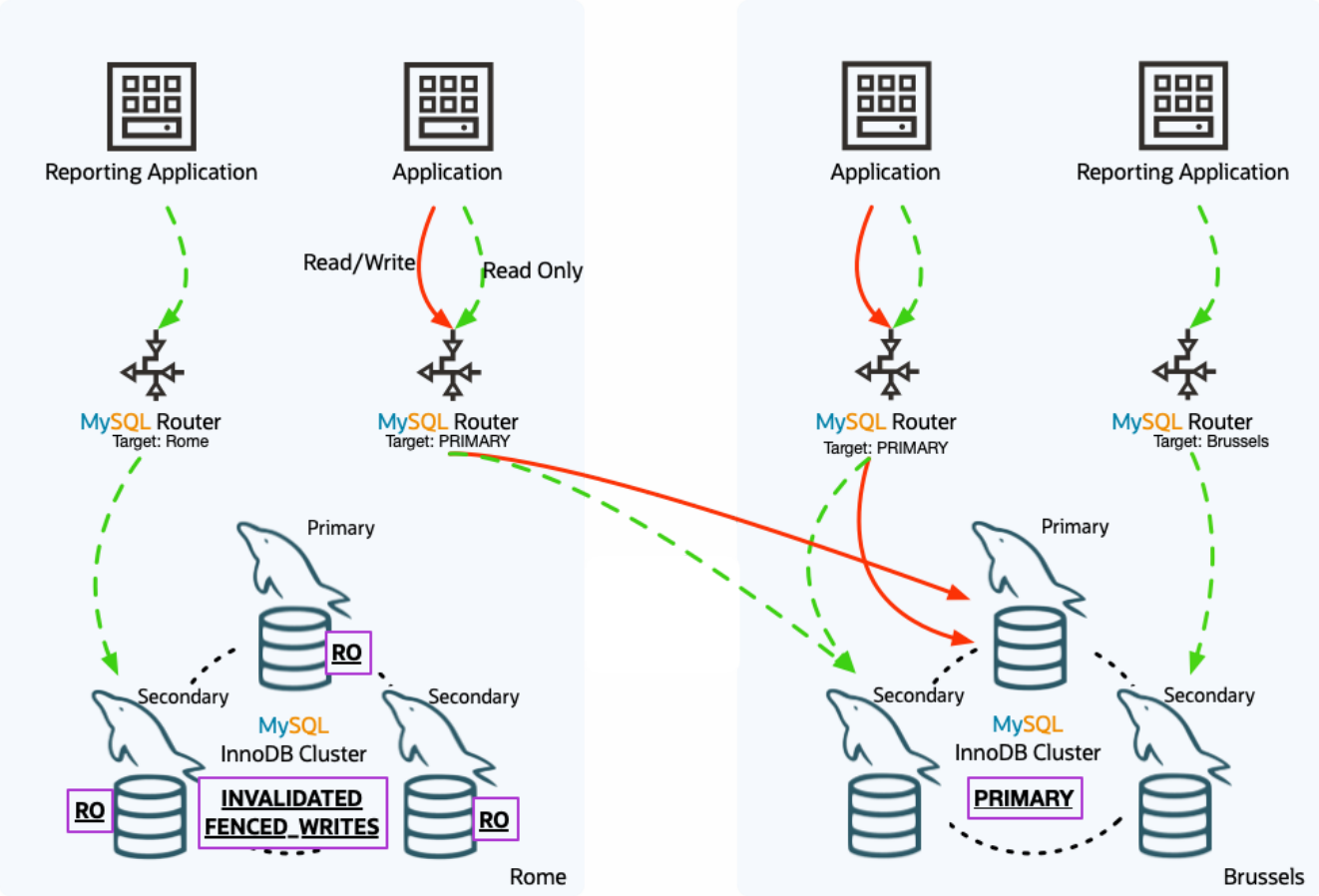
Disaster - PRIMARY Cluster - Network Partition - forcePrimaryCluster()



Failover to another Cluster

- one command to invalidate the PRIMARY cluster and promote a new PRIMARY cluster: `forcePrimaryCluster()`
- other REPLICA clusters replication will be reconfigured

Disaster - PRIMARY Cluster - Network Partition Resolved Partition



Router Integration

- Routers will learn about new topology and redirect traffic
- Routers can connect to the new PRIMARY Cluster will learn about new topology and abandon the old (fenced) cluster automatically

Demo

Initial Setup

- Create MySQL InnoDB Cluster
- Create ClusterSet with 3 clusters
- ClusterSet Status
- Router Bootstrap

Change PRIMARYs

- Change `PRIMARY` member in `PRIMARY` cluster
- Change `PRIMARY` member in `REPLICA` cluster
- Change `PRIMARY` Cluster - `setPrimaryCluster()`

Router

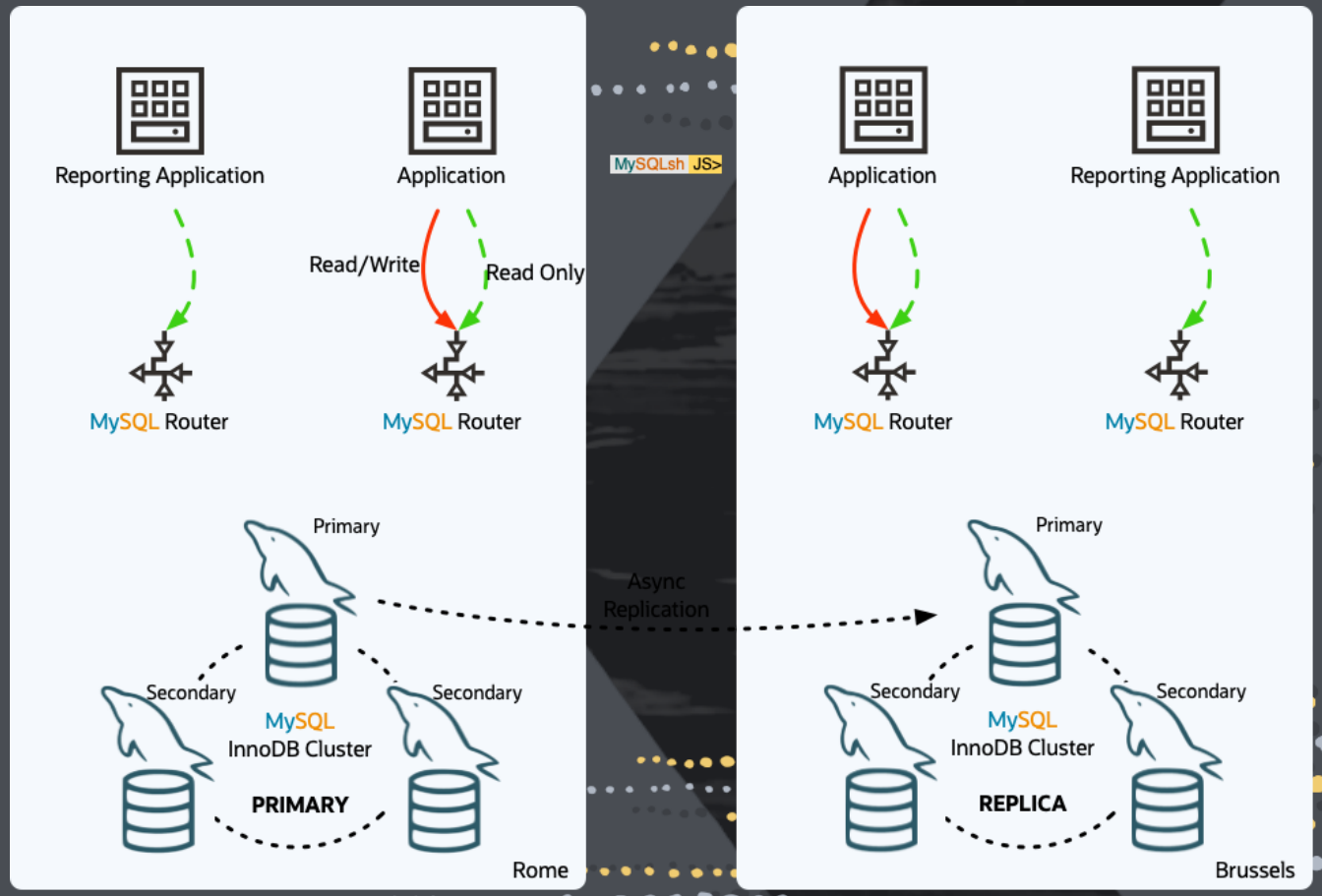
- Changing Router Configuration Options
- Router Status with Cluster changes
- Router Logs

Failure Scenarios

- Automatic Handling of `PRIMARY` member in `PRIMARY` cluster
- Automatic Handling of `PRIMARY` member in `REPLICA` cluster
- Disaster - `PRIMARY` Cluster - Network Partition

ORACLE

MySQL



MySQL InnoDB ClusterSet