Advanced Replication & High Availability

8.1 InnoDB Cluster Architecture

What is InnoDB Cluster?

InnoDB Cluster is MySQL's built-in high-availability solution combining:

- **Group Replication**: Automatic replication with data consistency
- **MySQL Router**: Connection routing and load balancing
- **MySQL Shell**: Administration and automation

InnoDB Cluster Components

Group Replication:

- Distributed, fault-tolerant replication
- Multi-primary or single-primary mode
- Automatic failover on node failure
- Data consistency guaranteed

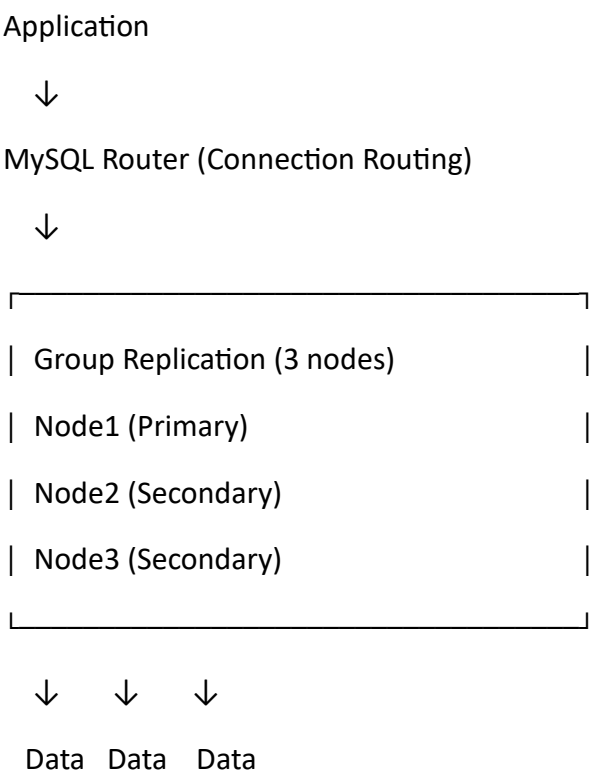
MySQL Router:

- Routes client connections to healthy nodes
- Handles failover automatically
- Read/write split for load balancing
- Session persistence

MySQL Shell:

- Advanced command-line interface
- Cluster administration commands
- Admin API for automation
- JavaScript/Python scripting support

Cluster Topology



8.2 Setting Up 3-Node InnoDB Cluster

Prerequisites

- Three MySQL 8.0.21+ servers
- Unique server_id for each
- InnoDB storage engine
- MySQL Shell installed
- Network connectivity between nodes

Initial Configuration

For each node (modify server_id):

[mysqld]

 # Unique server identifier

server_id = 1 *# Change to 2, 3 for other nodes*

 # Binary logging

log_bin = /var/log/mysql/mysql-bin

binlog_format = ROW

server_uuid = auto *# Let MySQL generate*

 # Group Replication

transaction_write_set_extraction = XXHASH64

loose_group_replication_group_name = "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee"

loose_group_replication_start_on_boot = OFF

loose_group_replication_local_address = "192.168.1.100:33061" *# Unique per node*

loose_group_replication_group_seeds =
"192.168.1.100:33061,192.168.1.101:33061,192.168.1.102:33061"

loose_group_replication_single_primary_mode = ON *# Single primary mode*

 # InnoDB

default_table_encryption = OFF

Create InnoDB Cluster

Step 1: Connect to Primary Node

mysqlsh

In MySQL Shell:

// Connect to primary node

shell> \connect icroot@192.168.1.100:3306

// Create cluster

shell> var cluster = dba.createCluster("myCluster")

// Cluster created successfully!

// {

```
//     "clusterName": "myCluster",
//     "defaultReplicaSet": {
//         "name": "default",
//         "primary": "192.168.1.100:3306",
//         "status": "OK",
//         "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
//         "topology": {
//             "192.168.1.100:3306": {
//                 "address": "192.168.1.100:3306",
//                 "label": "localhost",
//                 "role": "HA",
//                 "status": "ONLINE"
//             }
//         }
//     }
// }
```

Step 2: Add Secondary Nodes

```
// Add second node
shell> cluster.addInstance('icroot@192.168.1.101:3306')

// Add third node
shell> cluster.addInstance('icroot@192.168.1.102:3306')

// Check cluster status
shell> cluster.status()
```

MySQL Router Configuration

 # Bootstrap router (automatically configures)

sudo mysqlrouter --bootstrap icroot@192.168.1.100:3306 \

--directory /etc/mysqlrouter

 # Start MySQL Router

sudo systemctl start mysqlrouter

sudo systemctl enable mysqlrouter

 # Check status

sudo systemctl status mysqlrouter

Connect Through MySQL Router

 # Read-write port (6446) - connects to primary

mysql -h 127.0.0.1 -P 6446 -u icroot -p

 # Read-only port (6447) - connects to secondaries

mysql -h 127.0.0.1 -P 6447 -u icroot -p

 # X Protocol port (6448) - for advanced connections

mysql -h 127.0.0.1 -P 6448 -u icroot -p

8.3 Testing Automatic Failover

Create Test Database

CREATE DATABASE cluster_test;

USE cluster_test;

```
CREATE TABLE data (
    id INT PRIMARY KEY AUTO_INCREMENT,
    value VARCHAR(255),
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO data (value) VALUES ('Initial data 1');
INSERT INTO data (value) VALUES ('Initial data 2');

SELECT * FROM data;
```

Simulate Primary Failure

Terminal 1: Monitor cluster

```
mysqlsh
shell> \connect icroot@192.168.1.100:3306
shell> var cluster = dba.getCluster()
shell> cluster.status()
```

Terminal 2: Stop primary node

```
# On primary (192.168.1.100)
sudo systemctl stop mysql
```

Monitor automatic failover in Terminal 1

```
// Failover should occur automatically
shell> cluster.status()

// Should see:
// "primary": "192.168.1.101:3306"  // New primary
```

// "status": "OK_NO_TOLERANCE" // Can't tolerate more failures

Verify Application Continuity

 # Applications should auto-failover to new primary

mysql -h 127.0.0.1 -P 6446 -u icroot -p \

  -e "SELECT * FROM cluster_test.data;"

 # New data should still be accessible

mysql -h 127.0.0.1 -P 6446 -u icroot -p \

  -e "INSERT INTO cluster_test.data (value) VALUES ('After failover');"

Restart Failed Node

 # Restart stopped node

sudo systemctl start mysql

 # In MySQL Shell, cluster should automatically detect

shell> cluster.status()

// Should show node coming back online

8.4 Replication Troubleshooting

Check Replication Status

-- On any node

SHOW SLAVE STATUS\G

SHOW REPLICA STATUS\G *-- MySQL 8.0.22+ syntax*

-- Key fields:

-- Slave_IO_Running: Yes

-- Slave_SQL_Running: Yes

-- Seconds_Behind_Master: 0

Monitor Group Replication

-- View group members

SELECT * FROM performance_schema.replication_group_members;

-- View group replication applier status

SELECT * FROM performance_schema.replication_applier_status\G

-- View group replication connection status

SELECT * FROM performance_schema.replication_connection_status\G

Common Replication Issues

Issue 1: Node Not Joining Cluster

-- Check error log

grep -i "group replication" /var/log/mysql/error.log

-- Common causes:

-- 1. Network connectivity

-- 2. Configuration mismatch

-- 3. Data conflicts

-- Solution: Check group_replication_group_seeds

Issue 2: High Replication Lag

```
-- Monitor applier queue
SELECT * FROM performance_schema.replication_applier_status_by_worker\G

-- Check for slow queries
SELECT * FROM performance_schema.events_statements_summary_by_digest
WHERE DIGEST_TEXT LIKE '%SELECT%'
ORDER BY SUM_TIMER_WAIT DESC
LIMIT 10;
```

Issue 3: Data Inconsistency

```
# Use Percona Toolkit to check consistency
pt-table-checksum \
  --host=192.168.1.100 \
  --user=repl_user \
  --password=ReplPass123!

# Sync differences
pt-table-sync --execute \
  h=192.168.1.100,u=repl_user,p=ReplPass123! \
  h=192.168.1.101,u=repl_user,p=ReplPass123!
```

8.5 Cluster Administration

Add New Node to Running Cluster

```
mysqlsh
shell> \connect icroot@192.168.1.100:3306
shell> var cluster = dba.getCluster()
shell> cluster.addInstance('icroot@192.168.1.104:3306')
```

Remove Node from Cluster

shell> cluster.removeInstance('icroot@192.168.1.104:3306')

Change Primary Node

shell> cluster.setPrimaryInstance('icroot@192.168.1.102:3306')

// Waits for secondaries to catch up before switching

Upgrade Cluster

// Upgrade secondaries first, then primary

// Each node upgraded in rolling fashion

// No downtime

shell> cluster.upgradeMetadataAndGroupReplication()

Monitor Cluster Health

// Regular health check

shell> cluster.status()

// Detailed diagnostics

shell> cluster.describe()

// Check node status

shell> dba.getCluster().members

8.6 Summary: Key Takeaways

1. **InnoDB Cluster**: Three-component solution (Group Replication, MySQL Router, Shell)
2. **Automatic Failover**: Detects failures and promotes secondary automatically

3. **Setup**: 3 nodes with group replication, bootstrap router, connect via router
4. **Testing**: Simulate failures and verify automatic failover
5. **Troubleshooting**: Monitor group replication status, check data consistency
6. **Administration**: Add/remove nodes, change primary, rolling upgrades
7. **High Availability**: Reduces RTO and RPO for production databases