# Modernizing MySQL High Availability: From Asynchronous Replication to InnoDB Cluster… in real life (almost…)!

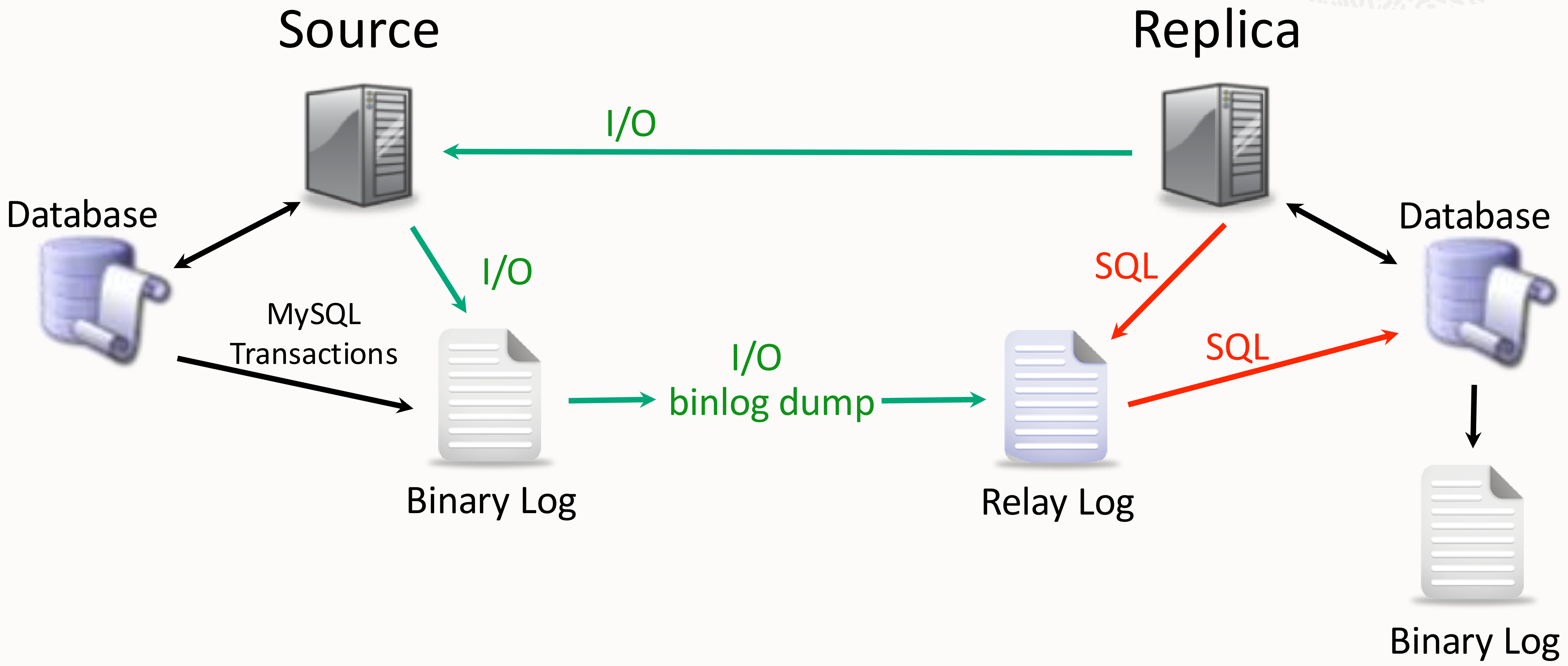
**Vittorio Cioe**
MySQL Solutions Engineer
vittorio.cioe@oracle.com

## Agenda

- Asynchronous Replication and MySQL HA Recap
- Moving from Replication to InnoDB Cluster
- Demo
- Wrap-up and Q&A

# Asynchronous Replication and MySQL HA Recap

## MySQL Asynchronous Replication

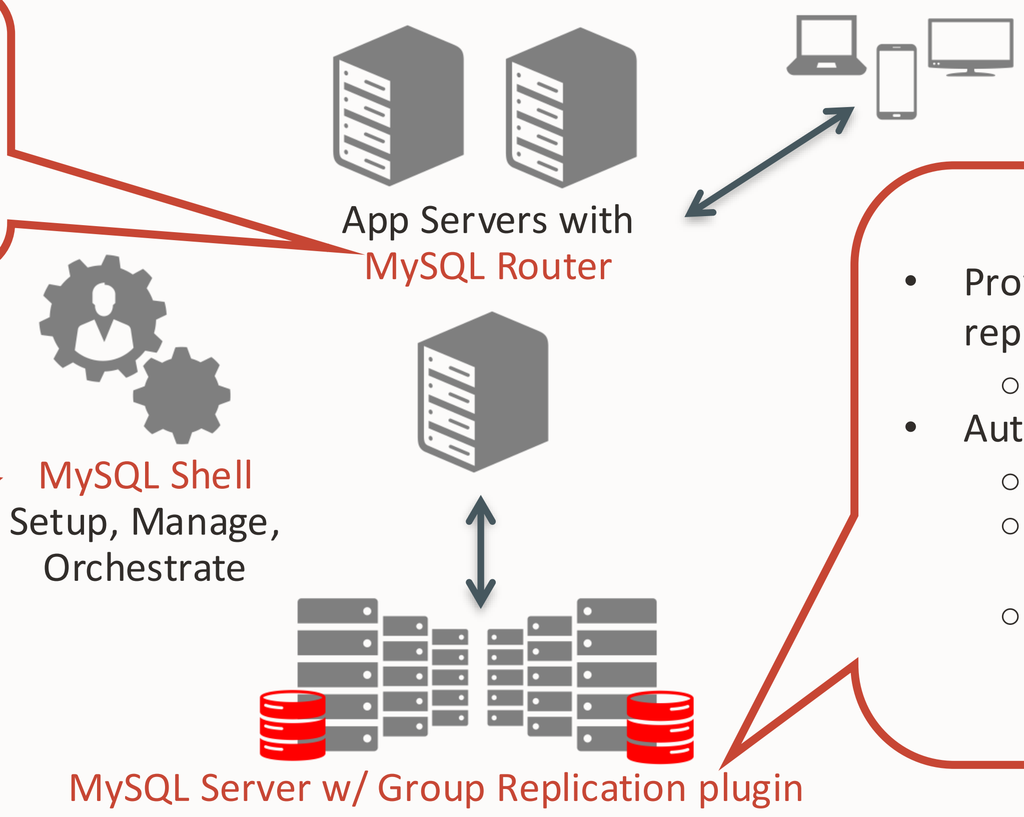


# MySQL InnoDB Cluster

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

- Route application connection to available nodes
- Automatic configuration
- REST API interface for monitoring
- Automatic R/W split

- Automate cluster creation and operation (clone)
- scriptable (JavaScript, Python, SQL)
- Document Store scripting


App Servers with
MySQL Router
MySQL Shell
Setup, Manage, Orchestrate
MySQL Server w/ Group Replication plugin

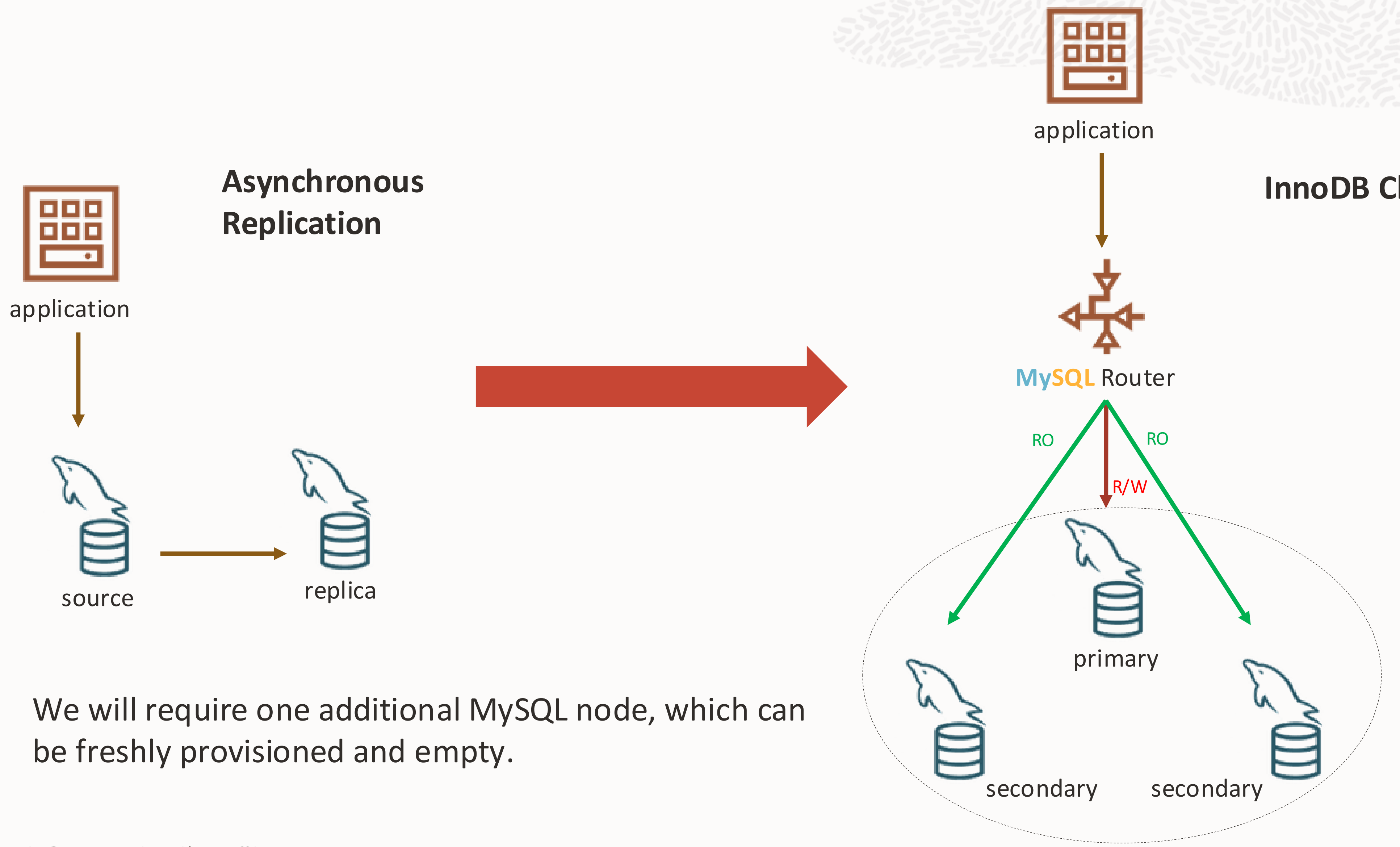
- Provides virtually synchronous replication
  - with Consistent Reads
- Automate operations
  - Conflict detection and resolution
  - Failure detection, fail-over, recovery
  - Group membership management and creation

Available on all MySQL supported platforms

## Moving from Replication to InnoDB Cluster

**…with minimal downtime!!**

## High level migration architecture



We will require one additional MySQL node, which can be freshly provisioned and empty.

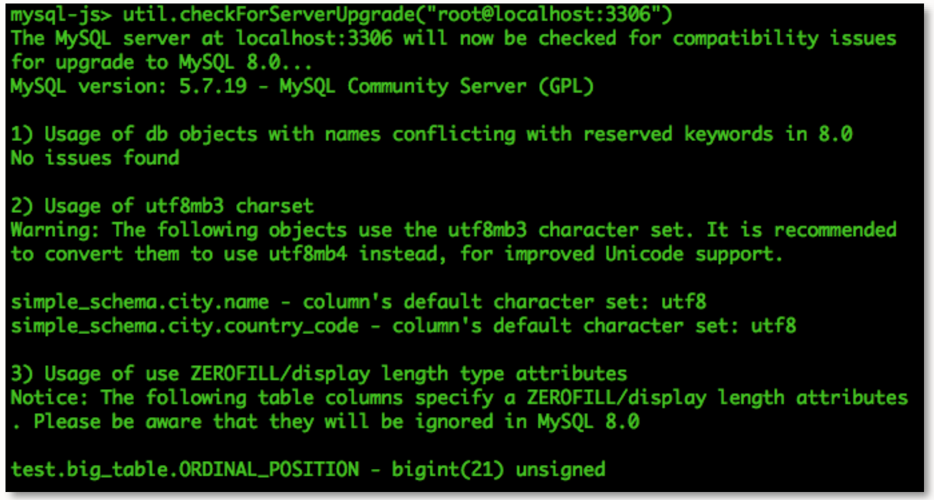
## InnoDB Cluster pre-requisites check list (aka common road blockers)

- Check that your MySQL is up to date
  - version 8.0 minimum, better if version 8.4
- You will need 3, 5, 7 or 9 MySQL nodes
- InnoDB only tables (beware of old MyISAM tables)
- Primary key is required on every table
- Binary Logs active (ROW)
- Global Transaction Identifiers (GTID) turned on
- Fast network between nodes (LAN speed)
- Full list here
  - https://dev.mysql.com/doc/refman/8.4/en/group-replication-requirements.html

## Need a version upgrade? Use MySQL Shell Upgrade Checker

- Quick and Easy MySQL Shell Utility
  - JavaScript or Python
  - MySQL 5.7 and onwards
  - New util class

    `util.checkForServerUpgade()`
- Identifies potential issues for upgrade
- Recommends Fixes:
  - Schema, configuration, tables, reserved keywords, use of deprecated or removed features or data- types and much more
- Recommends Fixes:
  - Upgrade the replica first and then the source



```
mysql-js> util.checkForServerUpgrade("root@localhost:3306")
The MySQL server at localhost:3306 will now be checked for compatibility issues
for upgrade to MySQL 8.0...
MySQL version: 5.7.19 - MySQL Community Server (GPL)

1) Usage of db objects with names conflicting with reserved keywords in 8.0
No issues found

2) Usage of utf8mb3 charset
Warning: The following objects use the utf8mb3 character set. It is recommended
to convert them to use utf8mb4 instead, for improved Unicode support.

simple_schema.city.name - column's default character set: utf8
simple_schema.city.country_code - column's default character set: utf8

3) Usage of use ZEROFILL/display length type attributes
Notice: The following table columns specify a ZEROFILL/display length attributes
. Please be aware that they will be ignored in MySQL 8.0

test.big_table.ORDINAL_POSITION - bigint(21) unsigned
```

http://lefred.be/content/how-to-safely-upgrade-to-mysql-8-0/

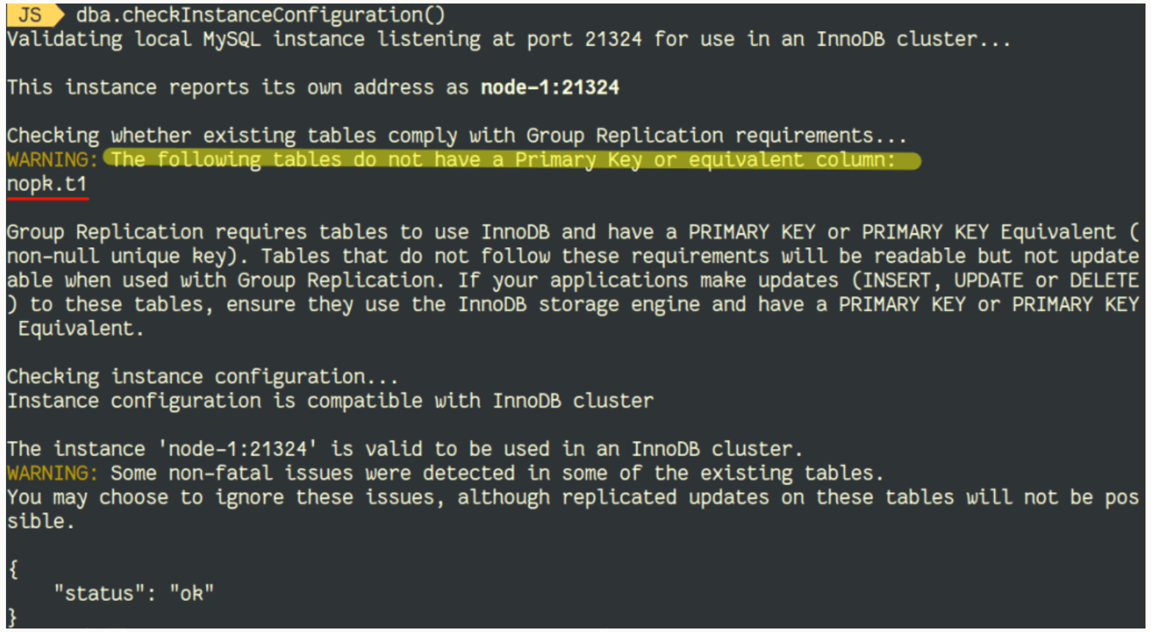
## Are your MySQL instances ready for InnoDB Cluster?

- MySQL Shell Command

  `dba.checkInstanceConfiguration()`

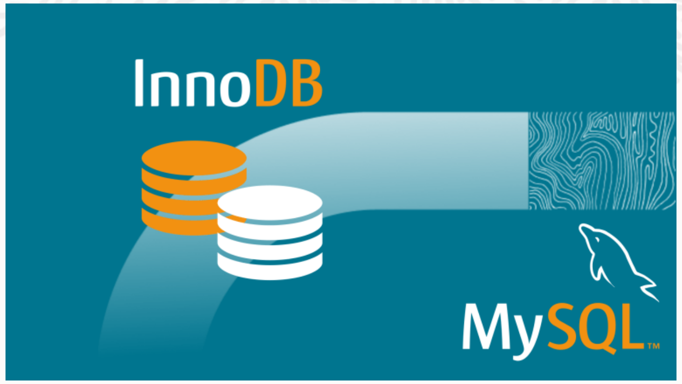
  will report incompatibilities with Group Replication. Most common ones:

  - Non-InnoDB tables
  - Missing primary keys
  - Missing GTID configuration
  - Configuration variables which need change
- It will suggest how to correct the action (if manually or automatically)
- Once you have completed the manual actions, run
  `dba.configureConfiguration()`
  to complete the instance configuration



```
JS > dba.checkInstanceConfiguration()
Validating local MySQL instance listening at port 21324 for use in an InnoDB cluster...

This instance reports its own address as node-1:21324

Checking whether existing tables comply with Group Replication requirements...
WARNING: The following tables do not have a Primary Key or equivalent column:
nopk.t1

Group Replication requires tables to use InnoDB and have a PRIMARY KEY or PRIMARY KEY Equivalent (
non-null unique key). Tables that do not follow these requirements will be readable but not update
able when used with Group Replication. If your applications make updates (INSERT, UPDATE or DELETE
) to these tables, ensure they use the InnoDB storage engine and have a PRIMARY KEY or PRIMARY KEY
 Equivalent.

Checking instance configuration...
Instance configuration is compatible with InnoDB cluster

The instance 'node-1:21324' is valid to be used in an InnoDB cluster.
WARNING: Some non-fatal issues were detected in some of the existing tables.
You may choose to ignore these issues, although replicated updates on these tables will not be pos
sible.

{
    "status": "ok"
}
```

## Still finding MyISAM tables ? Time to move to InnoDB!



- MyISAM drawbacks:
  - No transactions
  - No Foreign keys
  - No row-level locking
  - No crash recovery (redo logs)
- Get a list of all the tables NOT using InnoDB:

```
SELECT table_schema, table_name, engine, table_rows,
(index_length+data_length)/1024/1024 AS sizeMB
FROM information_schema.tables
WHERE engine != 'innodb' AND table_schema NOT IN ('information_schema',
'mysql', 'performance_schema');
```

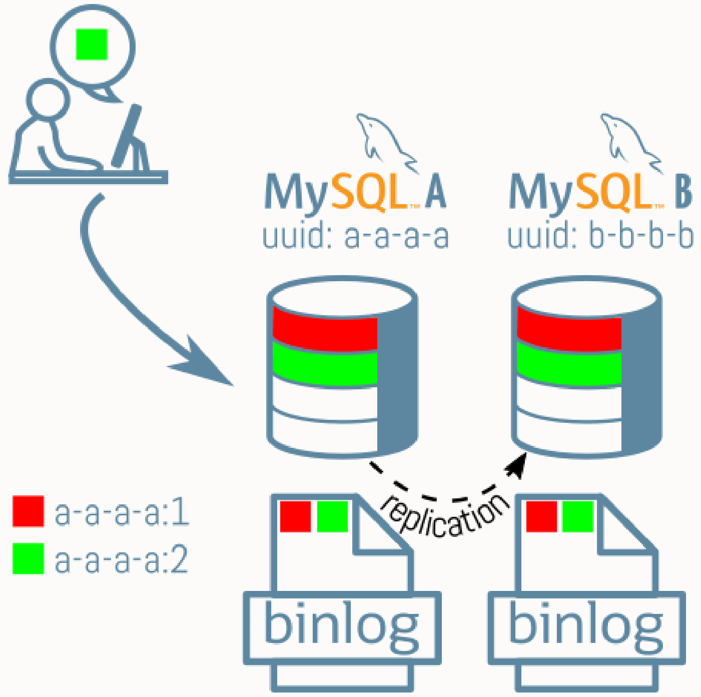
- Change the storage engine for a table:

```
ALTER TABLE table_name ENGINE=InnoDB;
```

- There is more to that, check the documentation:
  - https://dev.mysql.com/doc/refman/8.4/en/converting-tables-to-innodb.html

## Not using GTID on your replication setup? (Level: Easy)

- Applying this procedure to enable GTIDs to your existing replication setup will require a downtime.
- How to do it:
  1. Synchronize both servers by making them read-only
  2. Stop both servers
  3. On both server, add to my.cnf:
     ```
     gtid_mode=ON
     enforce-gtid-consistency=ON
     ```
  4. Restart both servers with --skip-replica-start
  5. Configure the replica to use GTID auto positioning:
     ```
     CHANGE REPLICATION SOURCE TO
     SOURCE_HOST = host,
     SOURCE_PORT = port,
     SOURCE_USER = user,
     SOURCE_PASSWORD = password,
     SOURCE_AUTO_POSITION = 1;
     ```
  6. Start the replica
  7. Take a new backup

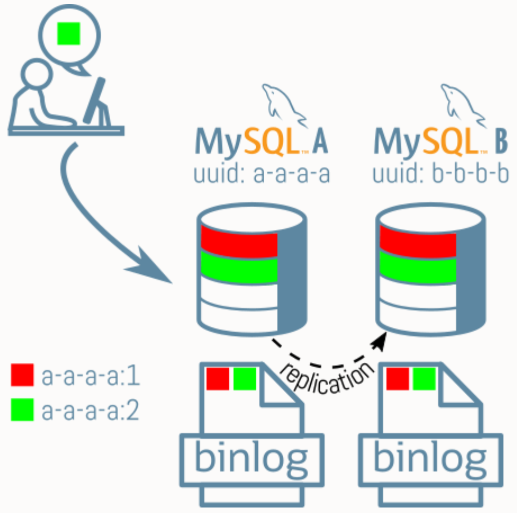


## Not using GTID on your replication setup? (Level: Pro)

Applying this procedure to enable GTIDs to existing replication setup will **NOT** require a downtime.
How to do it:

1. On each server, execute: `SET @@GLOBAL.ENFORCE_GTID_CONSISTENCY = WARN`, and set `enforce-gtid-consistency=WARN` in my.cnf
2. Let it run for a while. If there are any warnings , adjust your SQL to use GTID-compatible statements
3. On each server, execute: `SET @@GLOBAL.ENFORCE_GTID_CONSISTENCY = ON`, and set `enforce-gtid-consistency=ON` in my.cnf
4. On each server, execute: `SET @@GLOBAL.GTID_MODE = OFF_PERMISSIVE`, and set `gtid-mode=OFF_PERMISSIVE` in my.cnf
5. On each server, execute: `SET @@GLOBAL.GTID_MODE = ON_PERMISSIVE`, and set `gtid-mode=ON_PERMISSIVE` in my.cnf
6. On each server, wait until the status variable `ONGOING_ANONYMOUS_TRANSACTION_COUNT` is zero
7. Wait until all transactions that existed after step 6 are replicated to all servers
8. Your existing binlog may contain non-GTID transaction. So, wait for the binlog to expire
9. On each server, execute: `SET @@GLOBAL.GTID_MODE = ON`, and set `gtid-mode=ON` in my.cnf
10. On the replica, execute the following:

```
STOP SLAVE;
CHANGE MASTER TO MASTER_AUTO_POSITION = 1;
START SLAVE;
```



## Missing some Primary Keys?
## InnoDB GIPK (*Generate Invisible Primary Key*) mode

- As of MySQL 8.0.30, MySQL supports generated invisible primary keys when running in **GIPK mode**
- **GIPK** mode is controlled by the `sql_generate_invisible_primary_key` server system variable
  - When MySQL is running in **GIPK** mode, a primary key is added to a table by the server, the column and key name is always my_row_id
- PKs to existing tables can be add as invisible column

```
MySQL > SELECT @@sql_generate_invisible_primary_key;
+--------------------------------------+
| @@sql_generate_invisible_primary_key |
+--------------------------------------+
| 1 |
+--------------------------------------+
MySQL > CREATE TABLE devlive (name varchar(20),
beers int unsigned);
MySQL > INSERT INTO devlive VALUES ('kenny', 0),
('lefred',1);
MySQL > SELECT * FROM devlive;
+--------+-------+
| name   | beers |
+--------+-------+
| kenny  | 0     |
| lefred | 1     |
+--------+-------+
```

```
MySQL > SHOW CREATE TABLE devlive\G
********************* 1. row *********************
Table: devlive
Create Table: CREATE TABLE `devlive` (
`my_row_id` bigint unsigned NOT NULL AUTO_INCREMENT
/*!80023 INVISIBLE */,
`name` varchar(20) DEFAULT NULL,
`beers` int unsigned DEFAULT NULL,
PRIMARY KEY (`my_row_id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT
CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

## Preparation for Cluster deployment on the Replica

1. Using MySQL Shell in SQL mode, run the following commands:

```
STOP REPLICA;
RESET REPLICA ALL;
```

2. You are now running without replication (but still with no downtime).
3. Using MySQL Shell in JS mode, run the following commands:

```
dba.checkInstanceConfiguration("root@mysql2:3306")
dba.configureInstance("root@mysql2”, {'clusterAdmin': 'icadmin’})
```

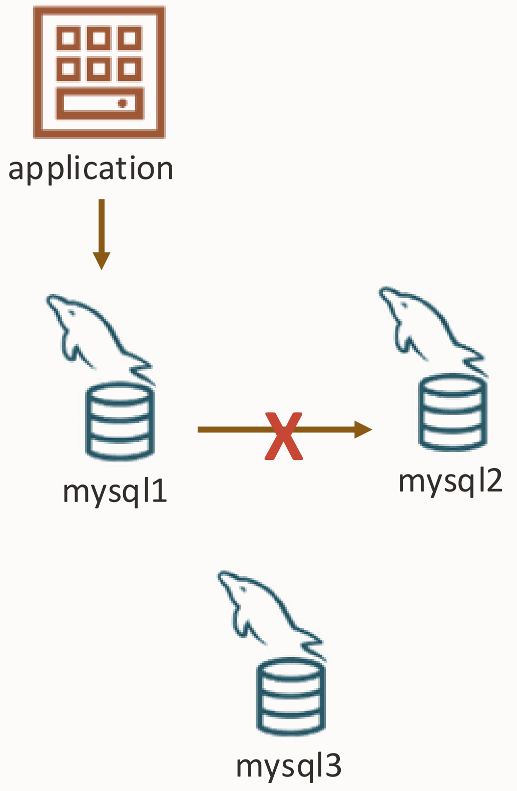
(it may require server restart)

4. Additionally, let’s check the configuration for the other nodes:

```
dba.checkInstanceConfiguration("root@mysql1")
dba.checkInstanceConfiguration("root@mysql3")
```

5. Proceed with configuration on the third node:

```
dba.configureInstance("root@mysql3", {'clusterAdmin': 'icadmin'})
```



## Cluster deployment

1. Using MySQL Shell in JS mode, run the following commands:

```
dba.configureInstance("root@mysql1", {'clusterAdmin': 'icadmin'})
```

(Beware! this command may require a restart)

2. Before creating the cluster, login with clusterAdmin user which you have created

```
\c icadmin@mysql1
```

3. Create the cluster

```
var cluster = dba.createCluster("testCluster")
```

4. Add the second node to the cluster:

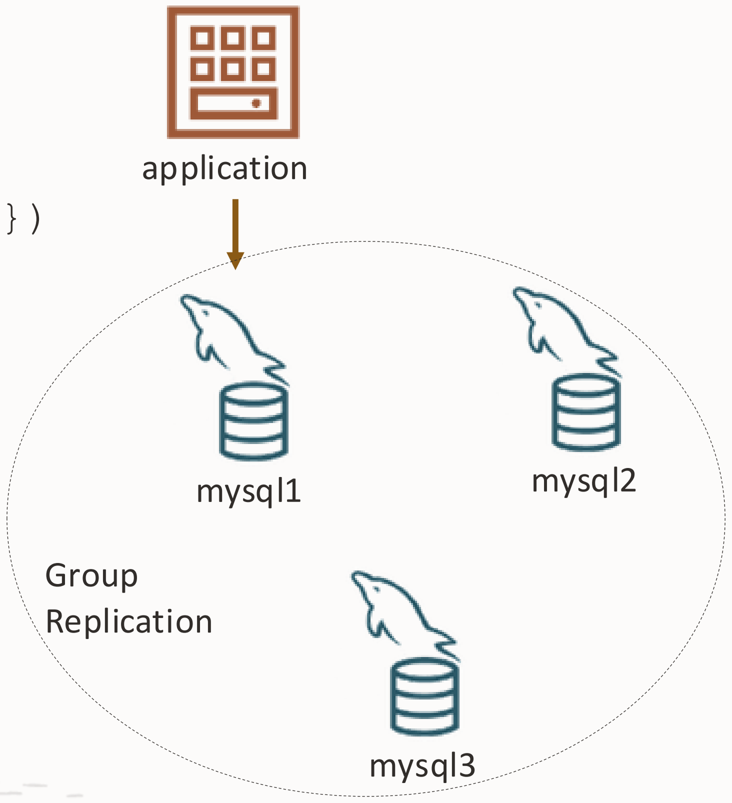
```
cluster.addInstance("icadmin@mysql2")
```

Since mysql2 contains a subset of the transactions contained in mysql1, the recovery method will be automatically incremental (using binary logs)

5. Finally, add the last node (mysql3) to the cluster:

```
cluster.addInstance("icadmin@mysql3")
```

(this one is the empty node: when prompted to choose for a recovery method, choose "C"- CLONE)



# MySQL Router deployment

This is the only phase which may require short application downtime

1. Install MySQL Router:

```
sudo yum localinstall mysql-router-commercial-9.3.0-
1.1.el9.aarch64.rpm
```

2. Bootstrap MySQL Router:

```
mysqlrouter --bootstrap root@mysql1:3306 --directory
/tmp/router –force
```

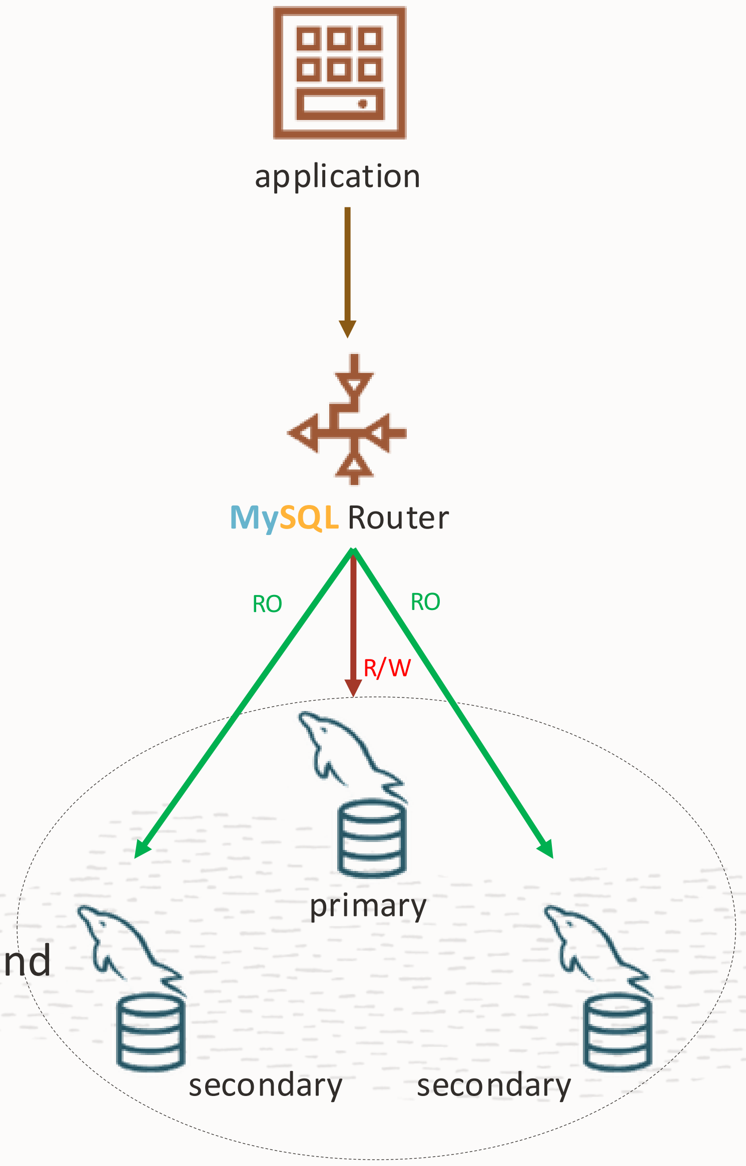
3. Check MySQL Router configuration:

```
sudo cat /tmp/router/mysqlrouter.conf
```

4. Start MySQL Router and check the status:

```
sudo systemctl start mysqlrouter.service
sudo systemctl status mysqlrouter.service
```

5. At last, configure the application to point at the MySQL Router host and to use one of the exposed ports.
   - For R/W split, use the port 6450



## Useful Resources

- **Convert tables from MyISAM to InnoDB:** https://dev.mysql.com/doc/refman/8.4/en/converting-tables-to-innodb.html
- **Enable GTIDs in Replication Offline:** https://dev.mysql.com/doc/refman/8.4/en/replication-gtids-howto.html
- **Enable GTIDs Online:** https://dev.mysql.com/doc/refman/8.4/en/replication-mode-change-online-enable-gtids.html
- **Generated Invisible Primary Keys:** https://dev.mysql.com/doc/refman/8.4/en/create-table-gipks.html
- **MySQL Shell Upgrade Checker Utility:** https://dev.mysql.com/doc/mysql-shell/9.3/en/mysql-shell-utilities-upgrade.html
- **Deploying a Production InnoDB Cluster:** https://dev.mysql.com/doc/mysql-shell/8.4/en/deploying-production-innodb-cluster.html
- **Deploying MySQL Router:** https://dev.mysql.com/doc/mysql-router/8.4/en/mysql-router-general-using-deploying.html

## Demo

## Wrap up and Q&A

## Wrap up

MySQL Everywhere, Solutions for Everyone

- MySQL is used everywhere and handles different workload patterns
- The toolset makes it remarkably easier to run it yourself
  - InnoDB ReplicaSet (Asynchronous)
  - InnoDB Cluster (HA, resilient, fault-tolerant)
  - InnoDB ClusterSet (Across clusters, Across regions)
- Same deployment and management for all MySQL platforms
- Let you achieve your RTO, RPO, SLA requirements

**MySQL Server**

- Durability
- Storage
- Clone
- Async Repl.

**MySQL Router**

- Application failover

**MySQL Shell**

- Automation
- Improved UX
- Standard recipes

**MySQL Group Replication**

- Automation
- Self-healing
- Distributed recovery
- Auto Membership
- Automated server failover

# Q&A

**Thank you!**

vittorio.cioe@oracle.com

ORACLE