Tuesday May 28th 9am-5pm

Hill Country A

PERCONA LIVE

# MySQL InnoDB Cluster 8.0 in a Nutshell

## Full day tutorial !!

Kenny Gryp

Frédéric Descamps

MySQL™


ORACLE®

# MySQL InnoDB Cluster in a Nutshell

## Hands-On Tutorial

Percona Live USA - Austin, TX 2019


PERCONA LIVE


- **Frédéric Descamps** - MySQL Community Manager - Oracle
- **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.

# Who are we ?

# Frédéric Descamps

- @lefred
- MySQL Evangelist
- Managing MySQL since 3.23
- devops believer
- http://about.me/lefred

MySQL

ORACLE

# Kenny Gryp

- @gryp
- MySQL Product Manager

  High Availability & InnoDB

how will the session proceed?

# Agenda

# Agenda

- Prepare your workstation

# Agenda

- Prepare your workstation
- What is MySQL InnoDB Cluster & Group Replication ?

# Agenda

- Prepare your workstation
- What is MySQL InnoDB Cluster & Group Replication ?
- Migrating from Asynchronous Replication to Group Replication

# Agenda

- Prepare your workstation
- What is MySQL InnoDB Cluster & Group Replication ?
- Migrating from Asynchronous Replication to Group Replication
- Monitoring

# Agenda

- Prepare your workstation
- What is MySQL InnoDB Cluster & Group Replication ?
- Migrating from Asynchronous Replication to Group Replication
- Monitoring
- Connecting to the cluster

# Agenda

- Prepare your workstation
- What is MySQL InnoDB Cluster & Group Replication ?
- Migrating from Asynchronous Replication to Group Replication
- Monitoring
- Connecting to the cluster
- Various Configuration Options

# Agenda

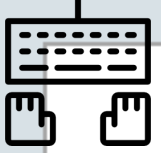
- Prepare your workstation
- What is MySQL InnoDB Cluster & Group Replication ?
- Migrating from Asynchronous Replication to Group Replication
- Monitoring
- Connecting to the cluster
- Various Configuration Options
- Application Behavior & Consistency Guarantees

# Agenda

- Prepare your workstation
- What is MySQL InnoDB Cluster & Group Replication ?
- Migrating from Asynchronous Replication to Group Replication
- Monitoring
- Connecting to the cluster
- Various Configuration Options
- Application Behavior & Consistency Guarantees
- Failure Scenarios - Labs

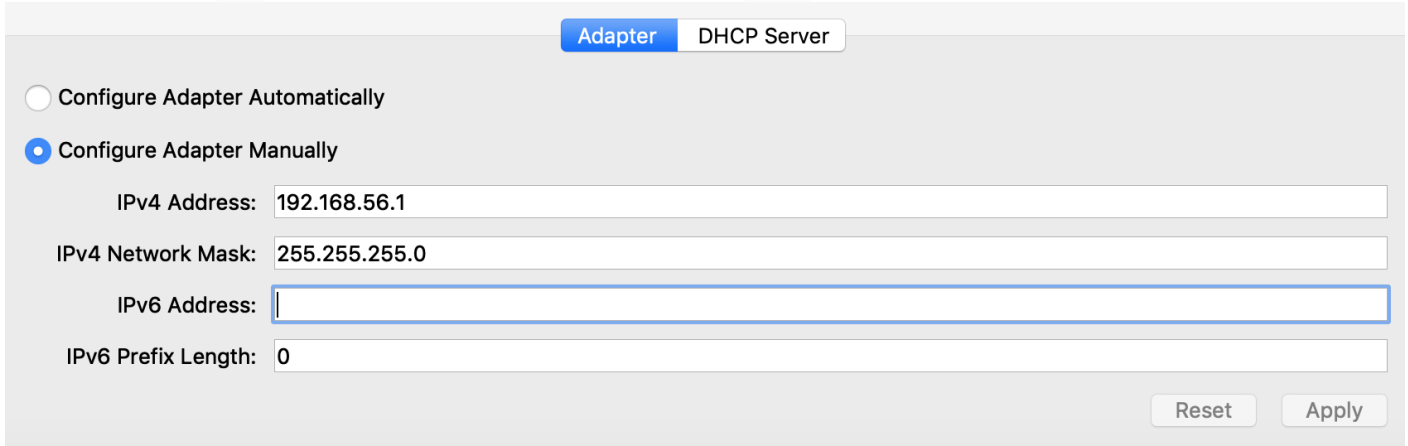
VirtualBox

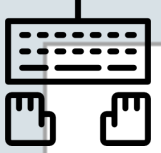
# Setup your workstation



# Setup your workstation

- Install VirtualBox 6.0
- On the USB key, **COPY** `MySQLInnoDBCluster_PLUS19.ova` and click on it
- Ensure you have `vboxnet0` network interface
  - Older VirtualBox: VirtualBox Pref. -> Network -> Host-Only Networks -> +
  - New VirtualBox: Global Tools -> Host Network Manager -> +

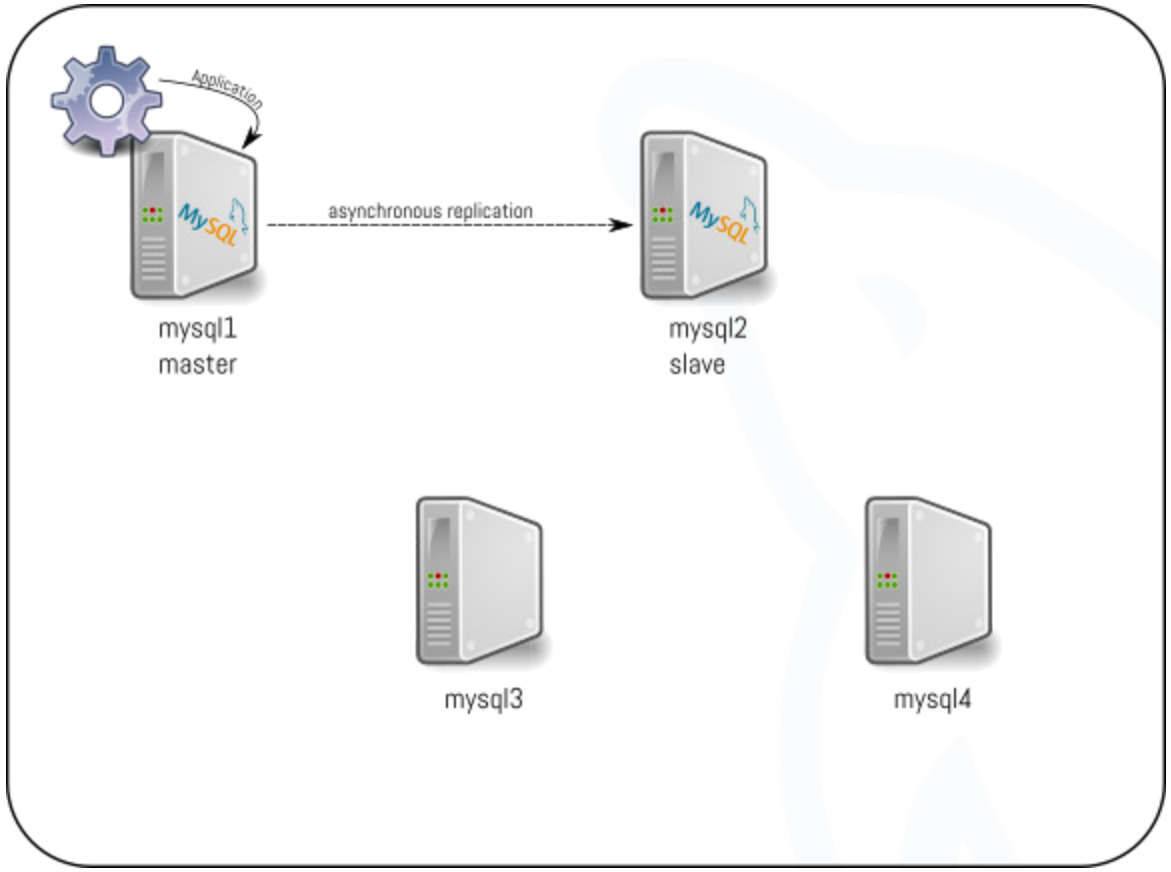




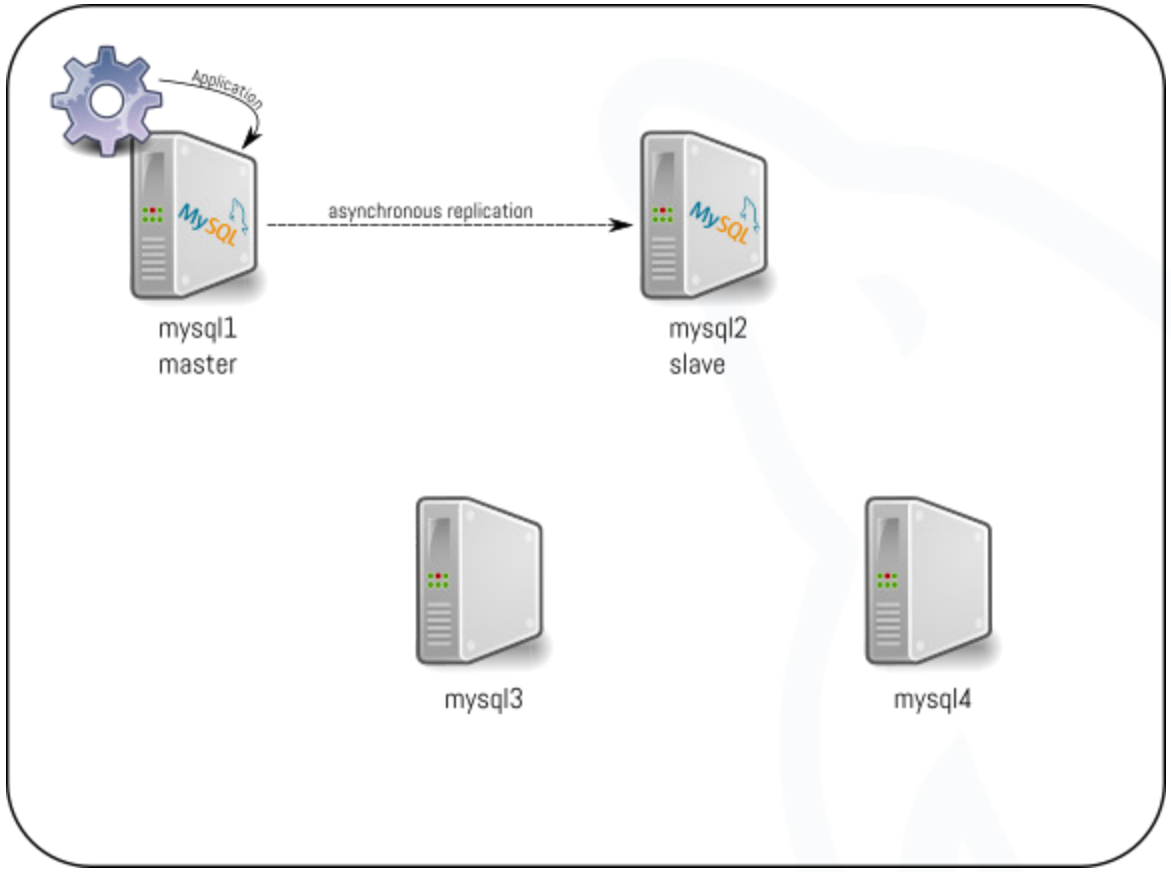
# Login on your workstation

- Start all virtual machines (`mysql1`, `mysql2`, `mysql3` & `mysql4`)
- Try to connect to all VM´s from your terminal or putty *(root password is* ***X****)*:
  - `ssh root@192.168.56.11` *mysql1*
  - `ssh root@192.168.56.12` *mysql2*
  - `ssh root@192.168.56.13` *mysql3*
  - `ssh root@192.168.56.14` *mysql4*

# LAB1: Current situation

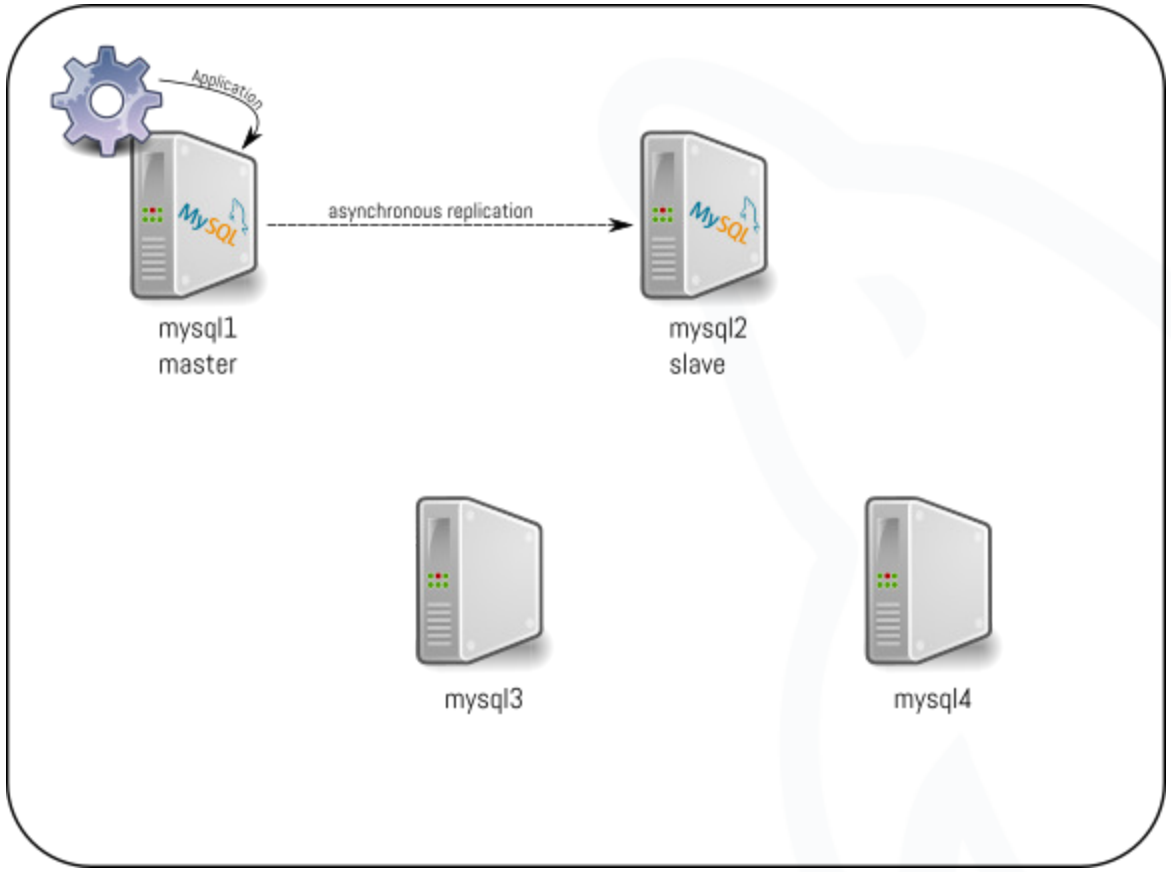


# LAB1: Current situation




all running database servers (`mysql1` & `mysql2`) are using MySQL 5.7.24

# LAB1: Current situation




- launch `run_app.sh` on `mysql1` into a **`screen`** session
- verify that `mysql2` is a running replica

# Summary

| | ROLE | INTERNAL IP |
|---|---|---|
| mysql1 | master | 192.168.56.11 |
| mysql2 | replica | 192.168.56.12 |
| mysql3 | n/a | 192.168.56.13 |
| mysql4 | n/a | 192.168.56.14 |

# Summary

| | ROLE | INTERNAL IP |
|---|---|---|
| mysql1 | master | 192.168.56.11 |
| mysql2 | replica | 192.168.56.12 |
| mysql3 | n/a | 192.168.56.13 |
| mysql4 | n/a | 192.168.56.14 |

write this down somewhere !

Easy High Availability

# MySQL InnoDB Cluster

## Ease-of-Use

Built-in HA

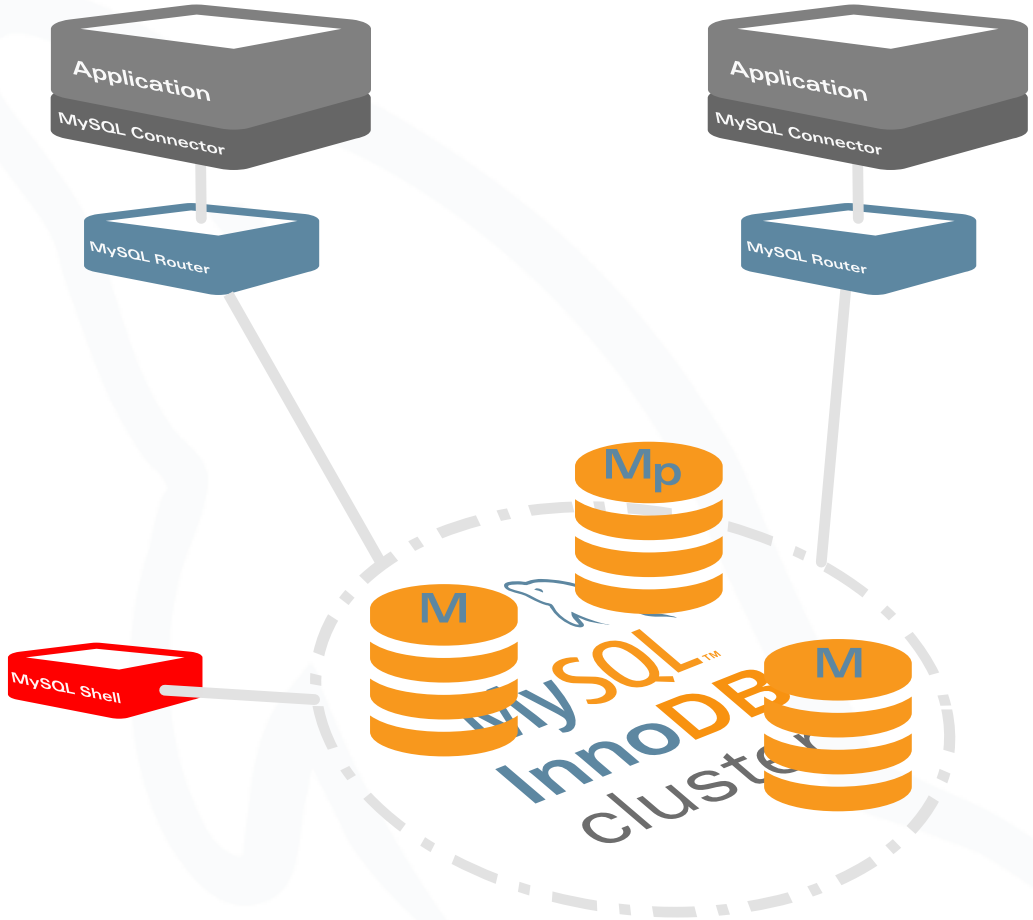

## Out-of-Box Solution

Everything Integrated

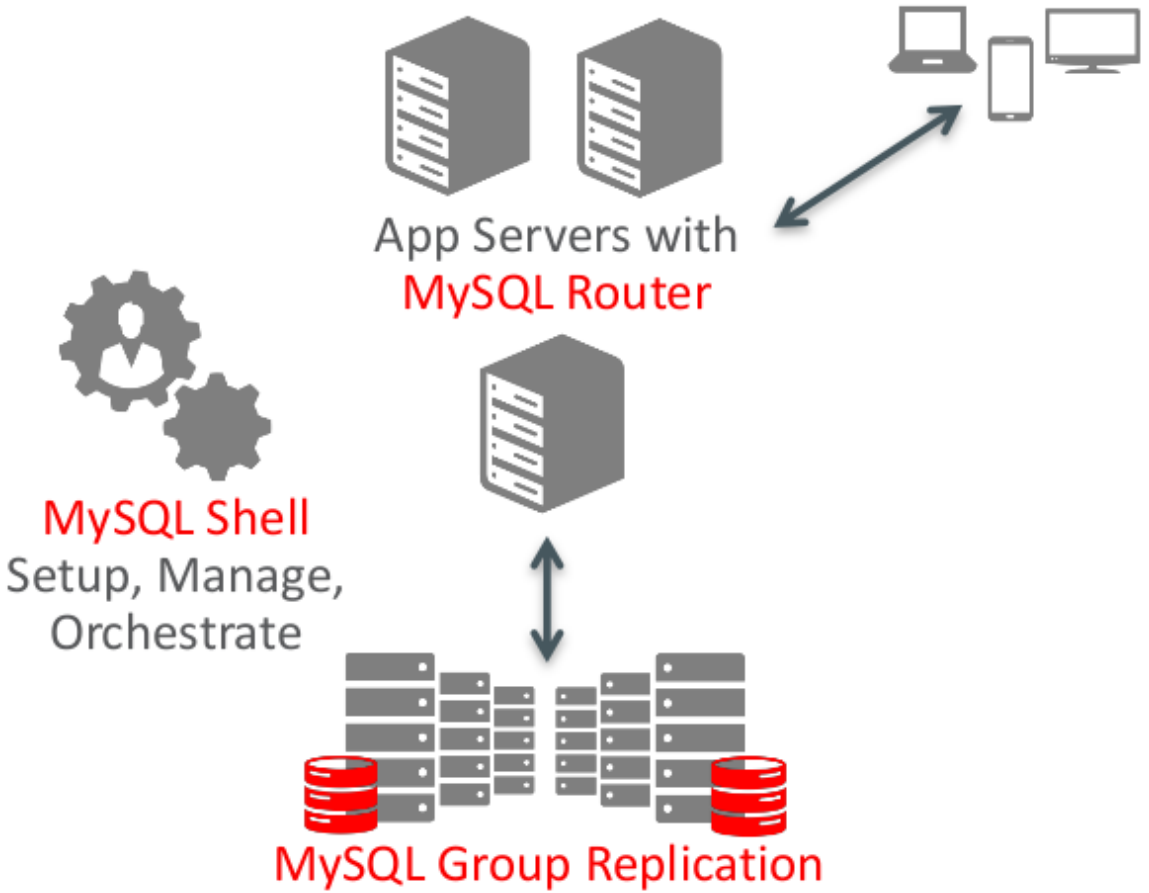
## Extreme Scale-Out

High Performance

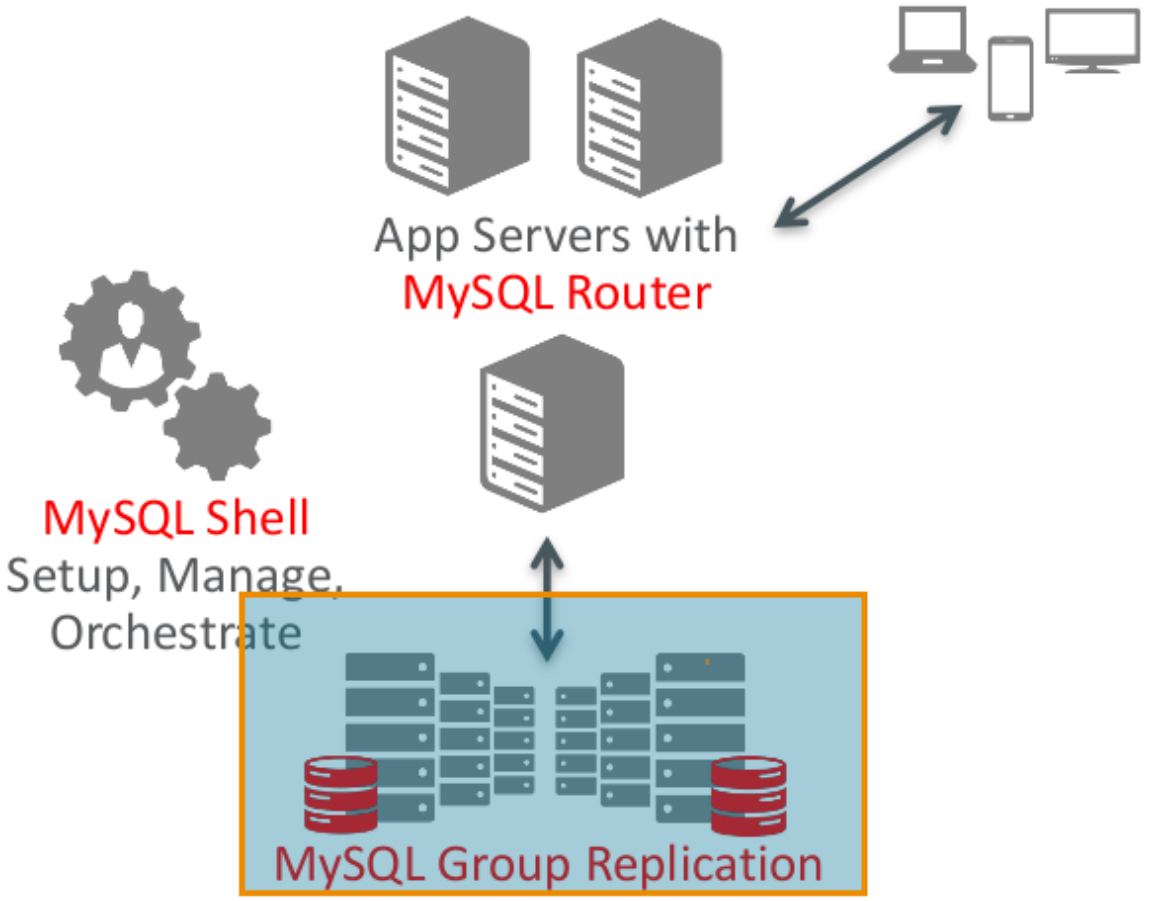
# InnoDB Cluster's Architecture



# Group Replication: heart of MySQL InnoDB Cluster

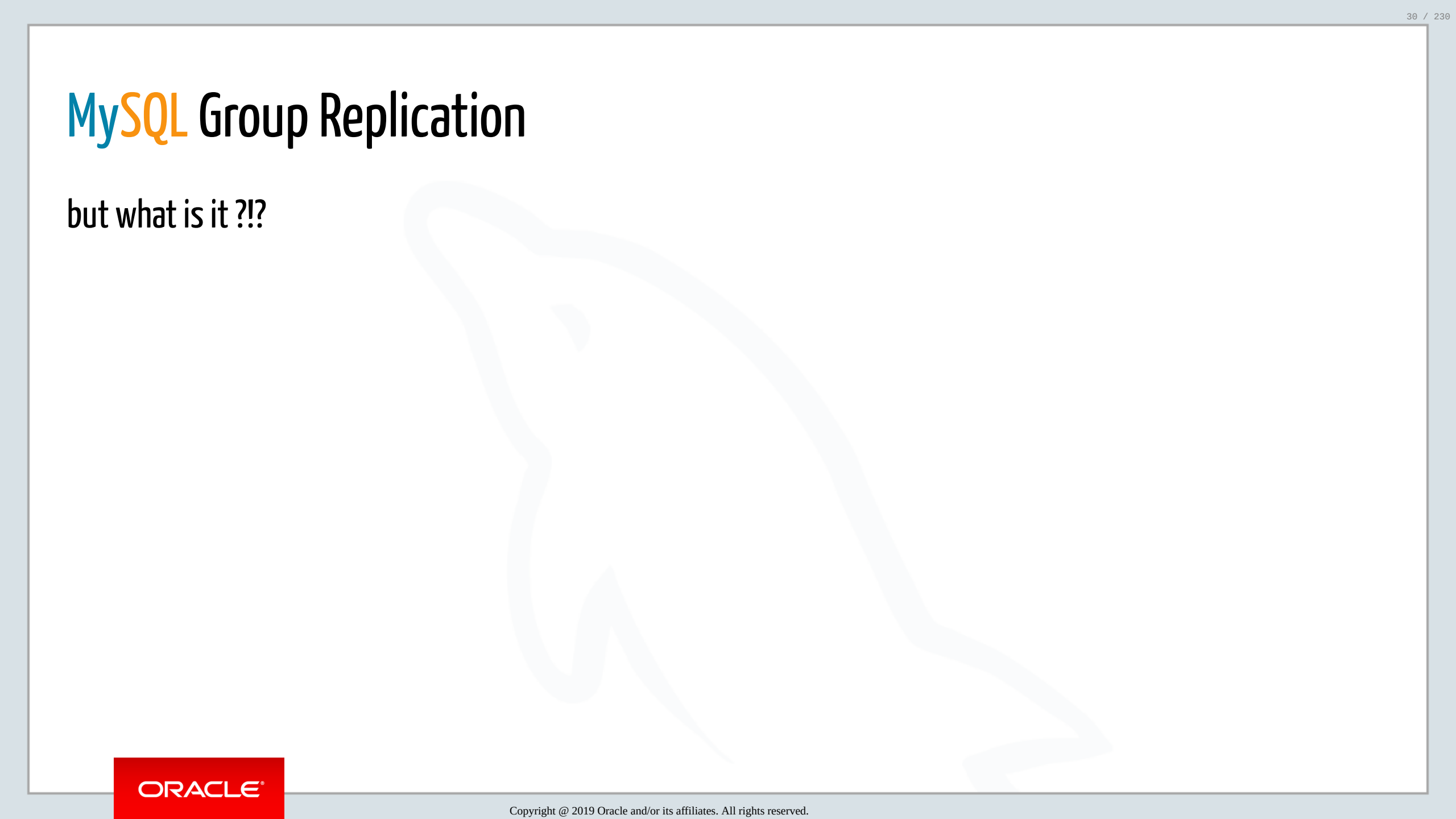


# Group Replication: heart of MySQL InnoDB Cluster



# MySQL Group Replication

but what is it ?!?



# MySQL Group Replication

but what is it ?!?

- GR is a plugin for MySQL, made by MySQL and packaged with MySQL

# MySQL Group Replication

but what is it ?!?

- GR is a plugin for MySQL, made by MySQL and packaged with MySQL
- GR is an implementation of Replicated Database State Machine theory

# MySQL Group Replication

but what is it ?!?

- GR is a plugin for MySQL, made by MySQL and packaged with MySQL
- GR is an implementation of Replicated Database State Machine theory
- Paxos based protocol

# MySQL Group Replication

but what is it ?!?

- GR is a plugin for MySQL, made by MySQL and packaged with MySQL
- GR is an implementation of Replicated Database State Machine theory
- Paxos based protocol
- GR allows to write on all Group Members (cluster nodes) simultaneously while retaining consistency

# MySQL Group Replication

## but what is it ?!?

- GR is a plugin for MySQL, made by MySQL and packaged with MySQL
- GR is an implementation of Replicated Database State Machine theory
- Paxos based protocol
- GR allows to write on all Group Members (cluster nodes) simultaneously while retaining consistency
- GR implements conflict detection and resolution

# MySQL Group Replication

## but what is it ?!?

- GR is a plugin for MySQL, made by MySQL and packaged with MySQL
- GR is an implementation of Replicated Database State Machine theory
- Paxos based protocol
- GR allows to write on all Group Members (cluster nodes) simultaneously while retaining consistency
- GR implements conflict detection and resolution
- GR allows automatic distributed recovery

# MySQL Group Replication

but what is it ?!?

- GR is a plugin for MySQL, made by MySQL and packaged with MySQL
- GR is an implementation of Replicated Database State Machine theory
- Paxos based protocol
- GR allows to write on all Group Members (cluster nodes) simultaneously while retaining consistency
- GR implements conflict detection and resolution
- GR allows automatic distributed recovery
- Supported on **all MySQL platforms** !!
  - Linux, Windows, Solaris, OSX, FreeBSD

# And for users ?

# And for users ?

- not longer necessary to handle server fail-over manually or with a complicated script

# And for users ?

- not longer necessary to handle server fail-over manually or with a complicated script
- GR provides fault tolerance

# And for users ?

- not longer necessary to handle server fail-over manually or with a complicated script
- GR provides fault tolerance
- GR enables update-everywhere setups

# And for users ?

- not longer necessary to handle server fail-over manually or with a complicated script
- GR provides fault tolerance
- GR enables update-everywhere setups
- GR handles crashes, failures, re-connects automatically

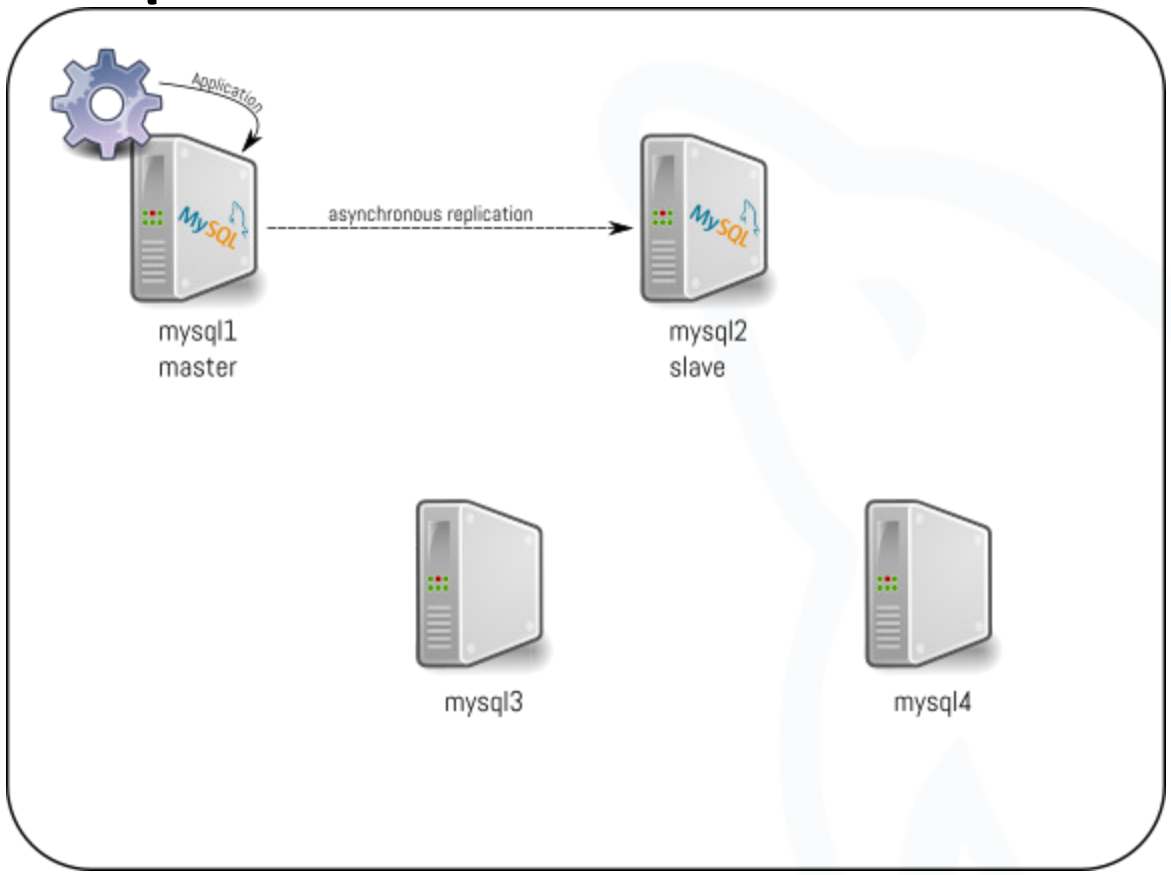
# And for users ?

- not longer necessary to handle server fail-over manually or with a complicated script
- GR provides fault tolerance
- GR enables update-everywhere setups
- GR handles crashes, failures, re-connects automatically
- **Allows an easy setup of a highly available MySQL service!**

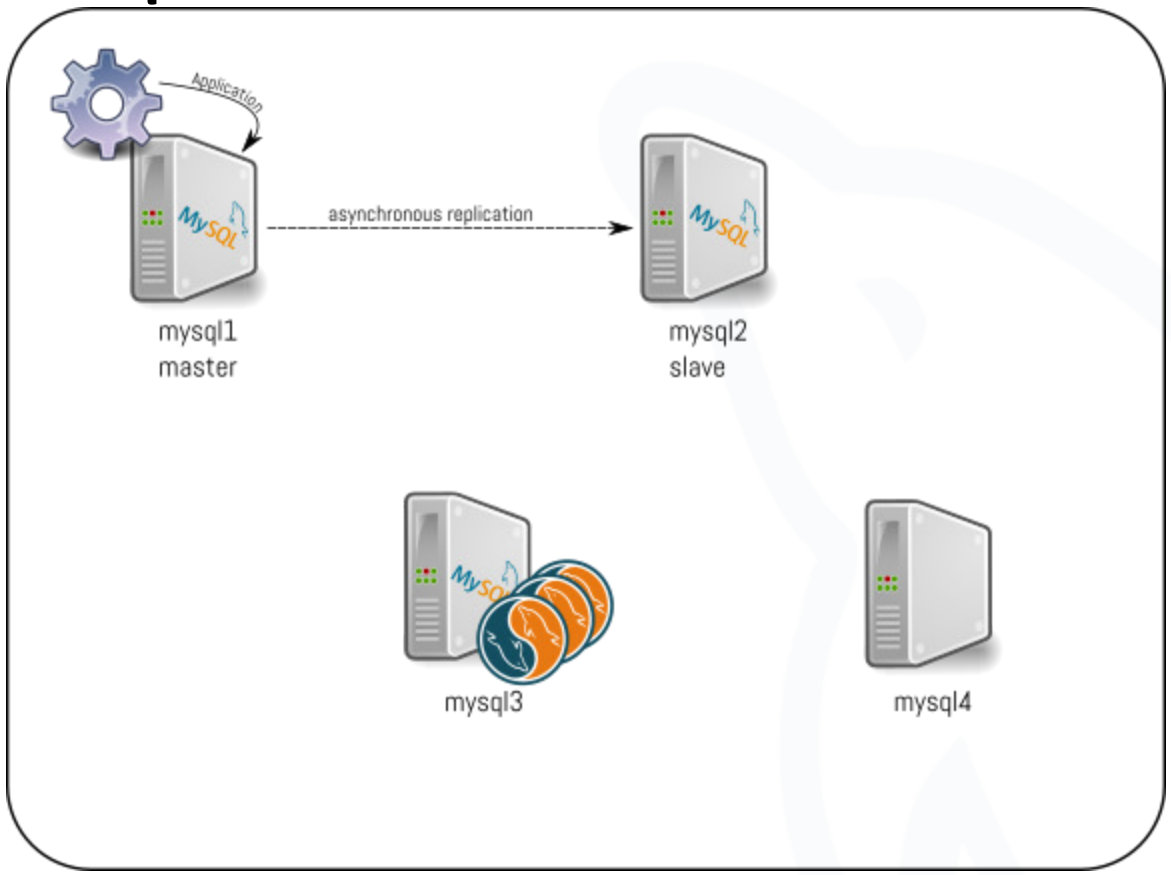
Migrating

# from Asynchronous Replication to Group Replication

# The plan



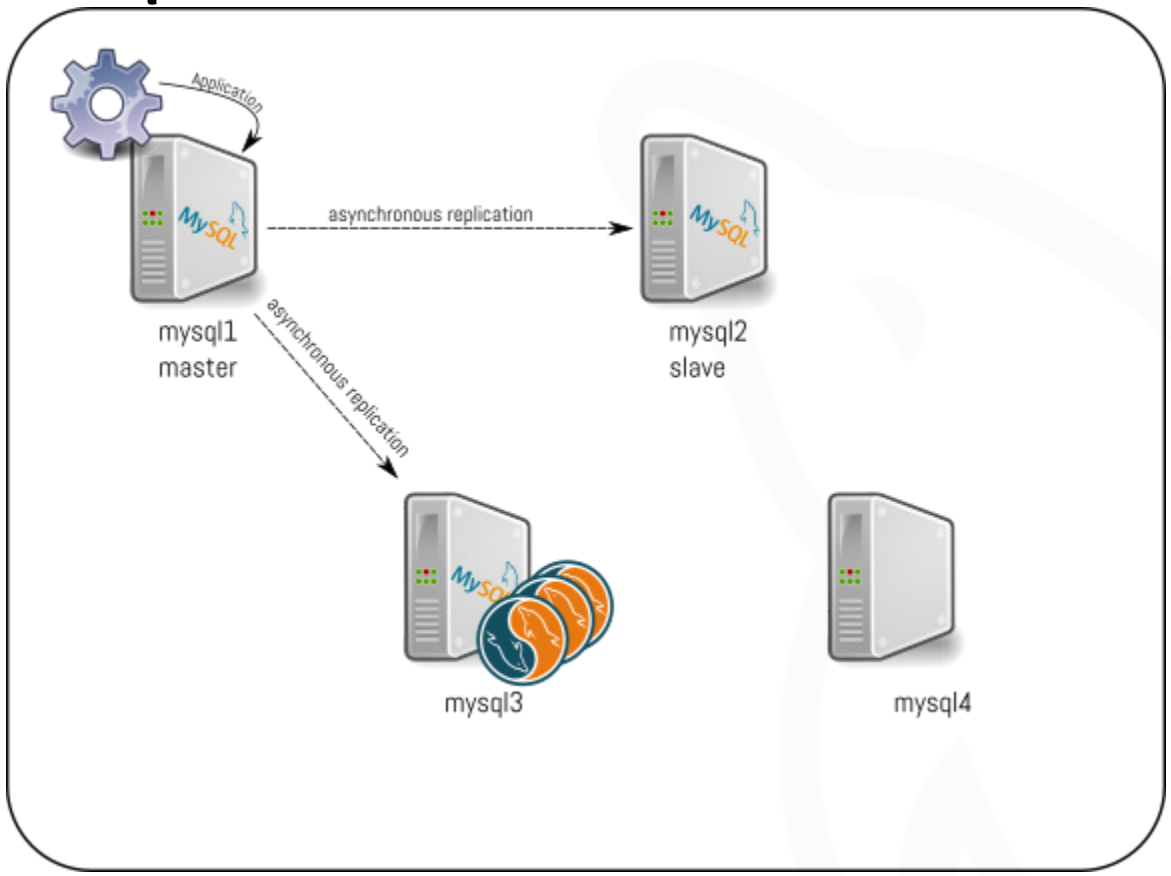
# The plan




These are the 10 steps for the migration:

1) We install and setup MySQL InnoDB Cluster on one of the new servers

# The plan

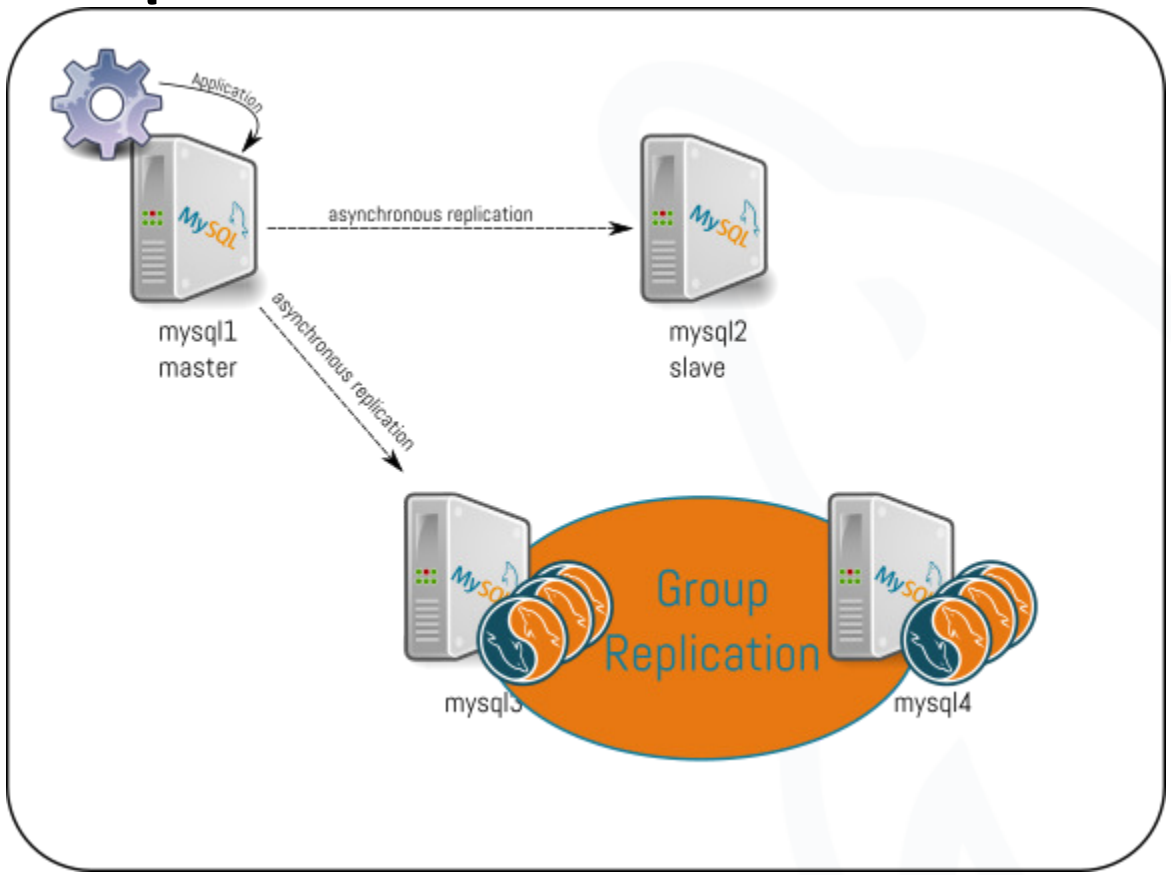



2) we restore a backup

3) we update to MySQL 8.0

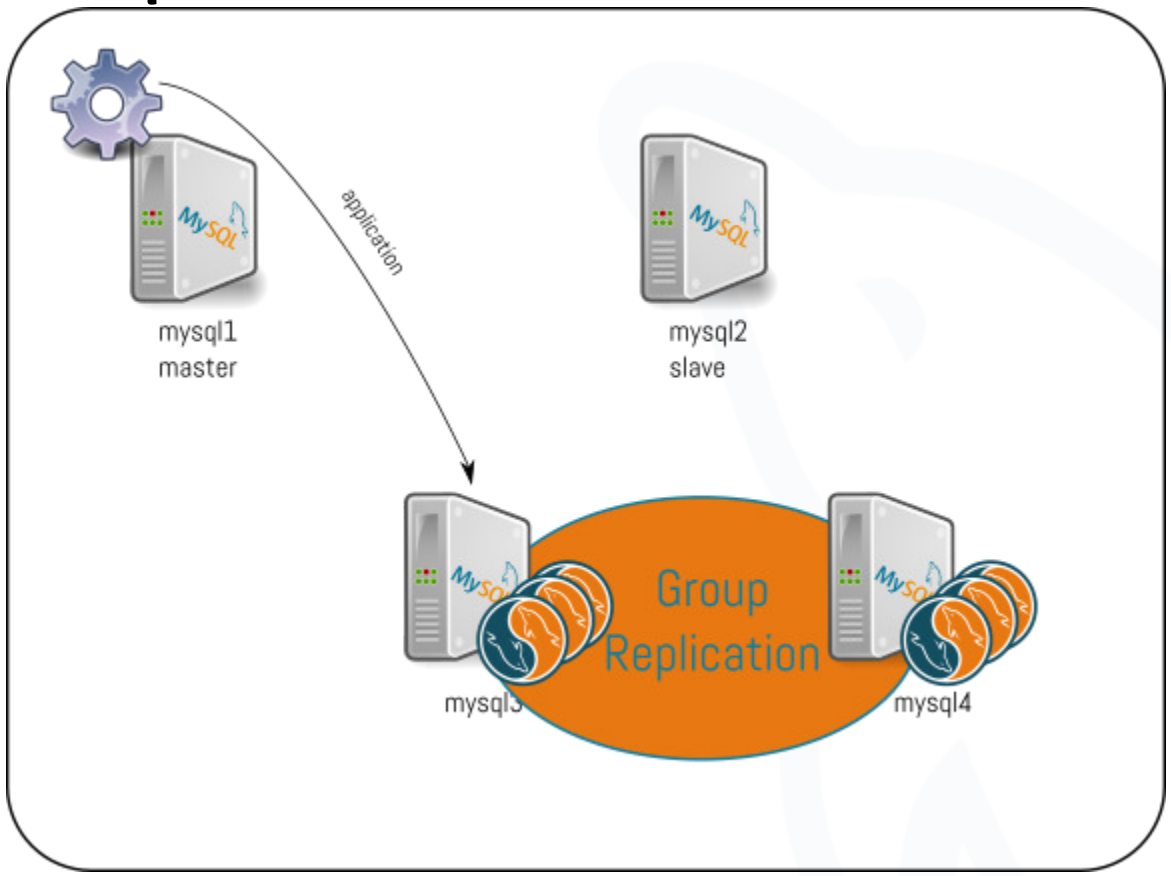
4) setup asynchronous replication on the new server.

# The plan




5) we add a new instance to our group

# The plan

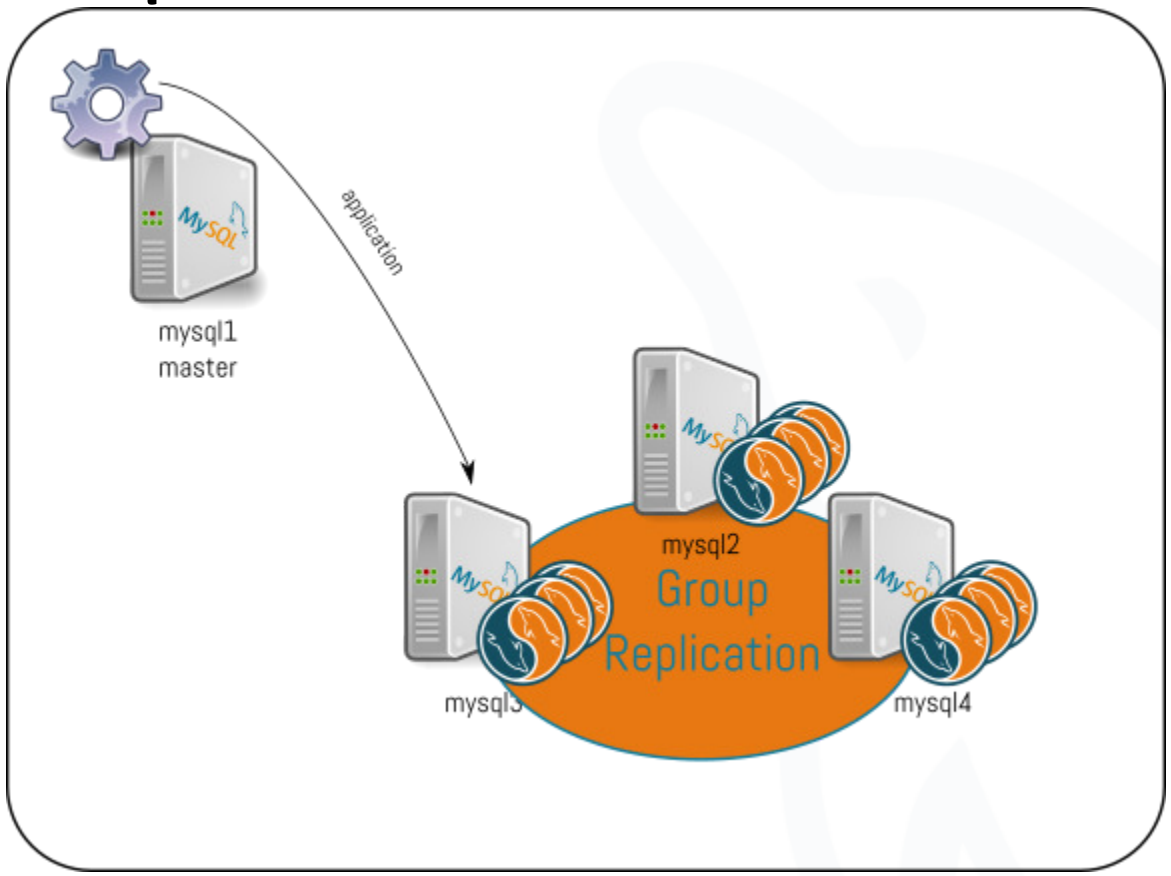



6) we point the application to one of our new nodes.

7) we wait and check that asynchronous replication is caught up

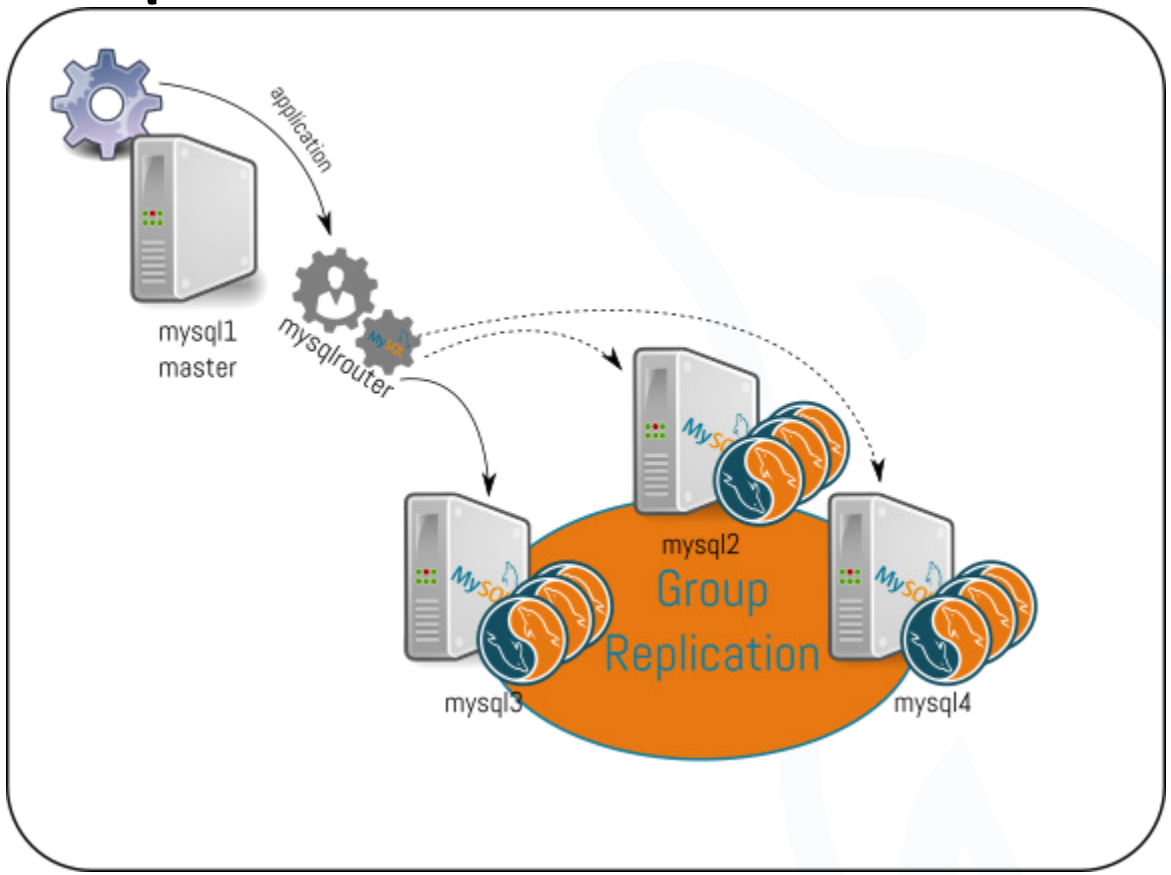
8) we stop those asynchronous slaves

# The plan

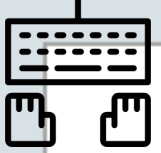



9) we upgrade and attach `mysql2` (previous slave) to the group

# The plan



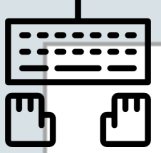

10) use MySQL Router for directing traffic



# LAB2: Verify if your environment can be upgraded to MySQL 8.0

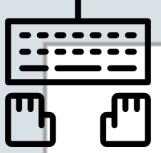
We will first create an admin account on `mysql1` that we will use throughout the tutorial:

```
mysql1> CREATE USER clusteradmin@'%' identified by 'mysql';
mysql1> GRAND ALL PRIVILEGES ON *.* TO  clusteradmin@'%'
        WITH GRANT OPTION;
```



# LAB2: Verify if your environment can be upgraded to MySQL 8.0 - util

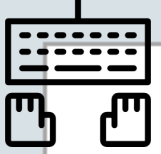
Now, we use MySQL Shell to check if the settings on `mysql1` can be upgraded to MySQL 8.0 without problem:



# LAB2: Verify if your environment can be upgraded to MySQL 8.0 - util

Now, we use MySQL Shell to check if the settings on `mysql1` can be upgraded to MySQL 8.0 without problem:

```
#  mysqlsh -- util check-for-server-upgrade clusteradmin@mysql1
```

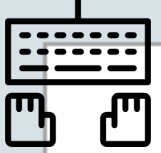


# LAB2: Verify if your environment can be upgraded to MySQL 8.0 - util

Now, we use MySQL Shell to check if the settings on `mysql1` can be upgraded to MySQL 8.0 without problem:

```
#  mysqlsh -- util check-for-server-upgrade clusteradmin@mysql1
```

Anything special in the output ?



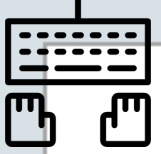
# LAB2: Verify if your environment can be upgraded to MySQL 8.0 - util

Now, we use MySQL Shell to check if the settings on `mysql1` can be upgraded to MySQL 8.0 without problem:

```
#  mysqlsh -- util check-for-server-upgrade clusteradmin@mysql1
```

Anything special in the output ?

```
# mysqlsh -- util check-for-server-upgrade clusteradmin@mysql1 \
  --configPath=/etc/my.cnf
```



# LAB2: Beautiful MySQL Shell

You can setup a better shell environment :

```
# mysqlsh
mysql-js> shell.options.setPersist('history.autoSave', 1)
mysql-js> shell.options.setPersist('history.maxSize', 5000)
```

```
# cp /usr/share/mysqlsh/prompt/prompt_256pl+aw.json ~/.mysqlsh/prompt.json
# mysqlsh clusteradmin@mysql2
```

```
MySQL  localhost:33060+  JS  \s
MySQL Shell version 8.0.11

Session type:                X
Connection Id:               8
```

# LAB2: Prepare mysql3 - *Asynchronous slave*

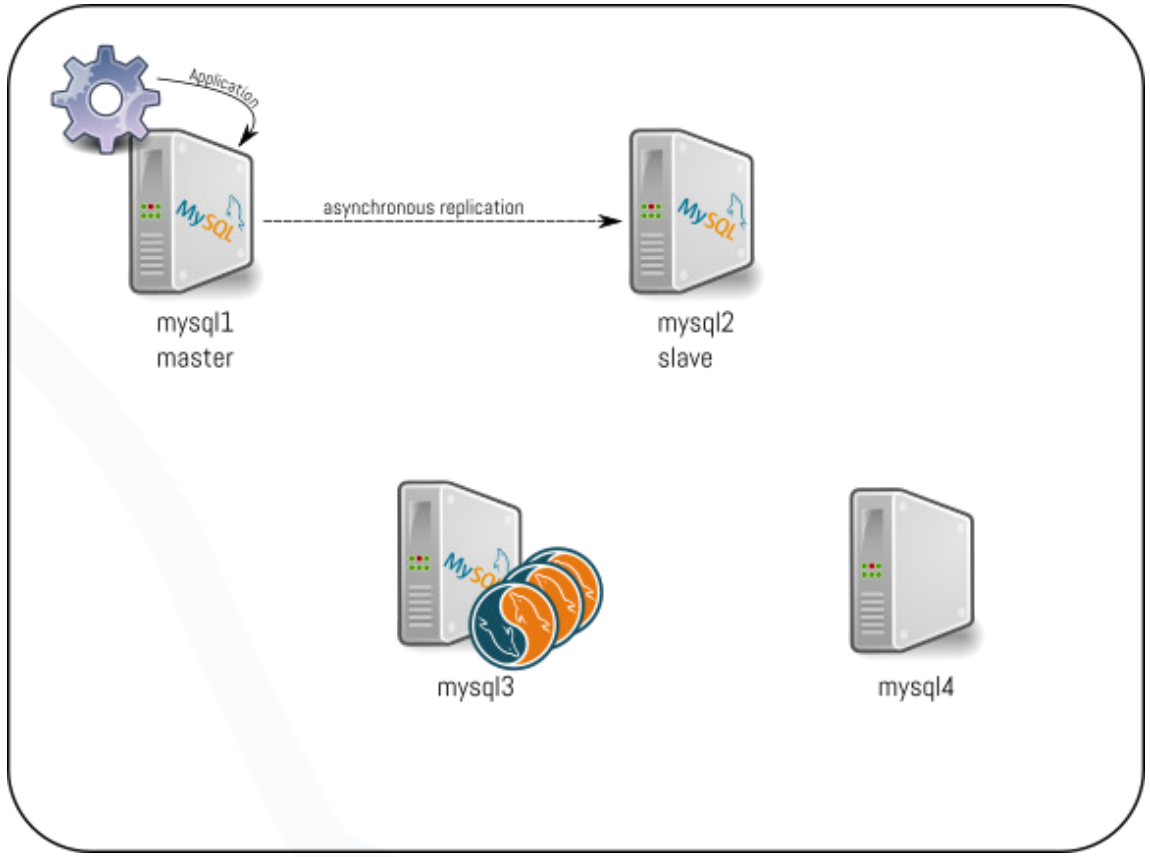
MySQL 8.0.16 is already installed on `mysql3`.

Let´s take a backup on `mysql1`:

```
[mysql1 ~]#  mysqlpump    \
--set-gtid-purged=ON      \
--exclude-databases=mysql \
--include-users=repl%,clusteradmin%,app% \
> /tmp/backup.sql
```

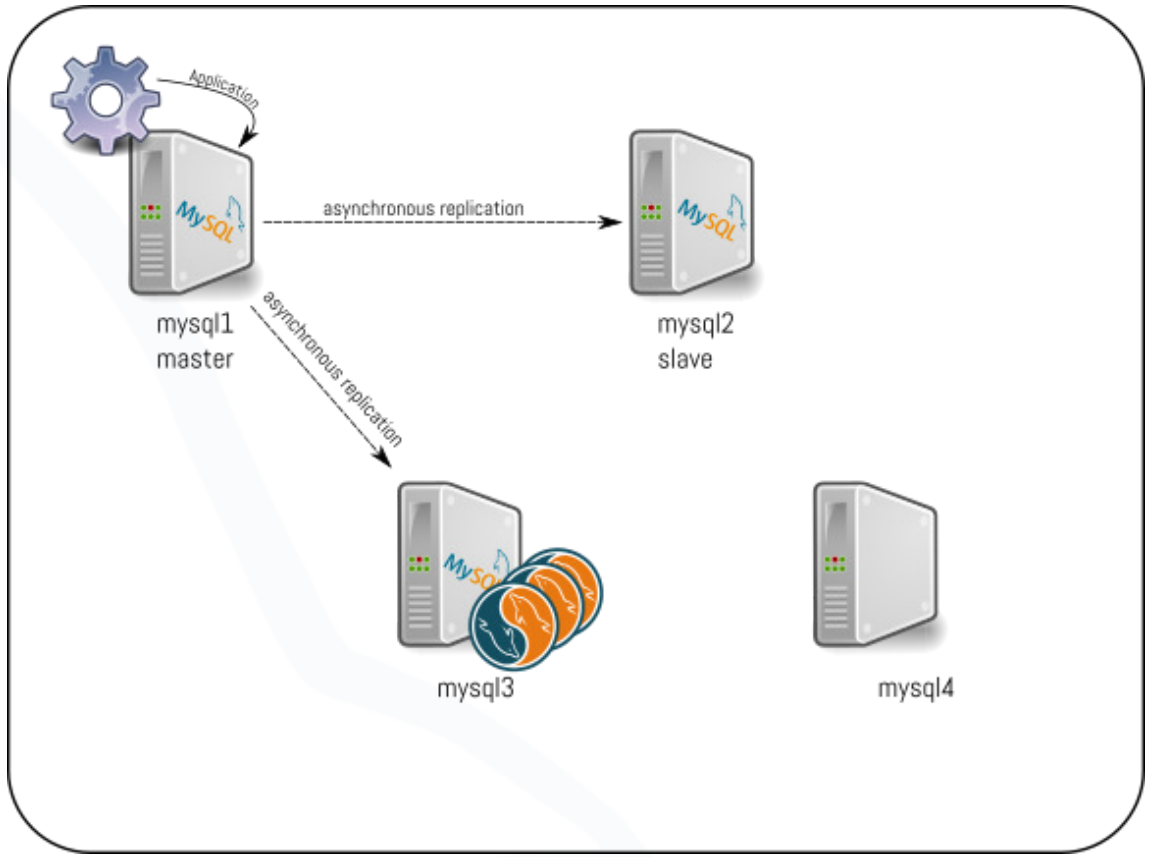
Copy the backup from `mysql1` to `mysql3`:

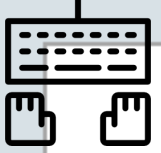
```
[mysql1 ~]# scp -r /tmp/backup.sql mysql3:/tmp
```




# LAB3: mysql3 as asynchronous slave (1)

- restore the dump
- change MASTER
- start replication





# LAB3: mysql3 as asynchronous slave (2)

Restore backup:

```
[mysql3 ~]# mysql < /tmp/backup.sql
```

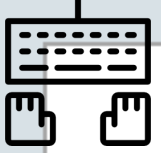
Connect to `mysql3` and setup replication:

```
mysql3> CHANGE MASTER TO MASTER_HOST="mysql1",
        MASTER_USER="repl", MASTER_PASSWORD='X',
        MASTER_AUTO_POSITION=1;

mysql3> START SLAVE;
```

Check that you receive the application´s traffic

Administration made easy and more...

# MySQL InnoDB Cluster



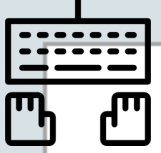
# LAB4: MySQL InnoDB Cluster

*Create a single instance cluster (mysql3)*

```
# mysqlsh
```

Let´s verify if our server is ready to become a member of a new cluster:

```
mysql-js> dba.checkInstanceConfiguration('clusteradmin@mysql3')
```



# LAB4: MySQL InnoDB Cluster

*Create a single instance cluster (mysql3)*

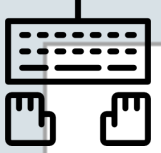
```
# mysqlsh
```

Let´s verify if our server is ready to become a member of a new cluster:

```
mysql-js> dba.checkInstanceConfiguration('clusteradmin@mysql3')
```

Change the configuration !

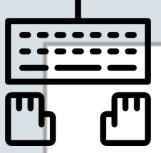
```
mysql-js> dba.configureInstance('clusteradmin@mysql3')
```



# LAB4: MySQL InnoDB Cluster (2)

*Wait just a bit if a restart is required (which is not the case here) and create a single instance cluster*

```
mysql-js> dba.checkInstanceConfiguration('clusteradmin@mysql3')

mysql-js> \c clusteradmin@mysql3

mysql-js> cluster = dba.createCluster('perconalive')
```



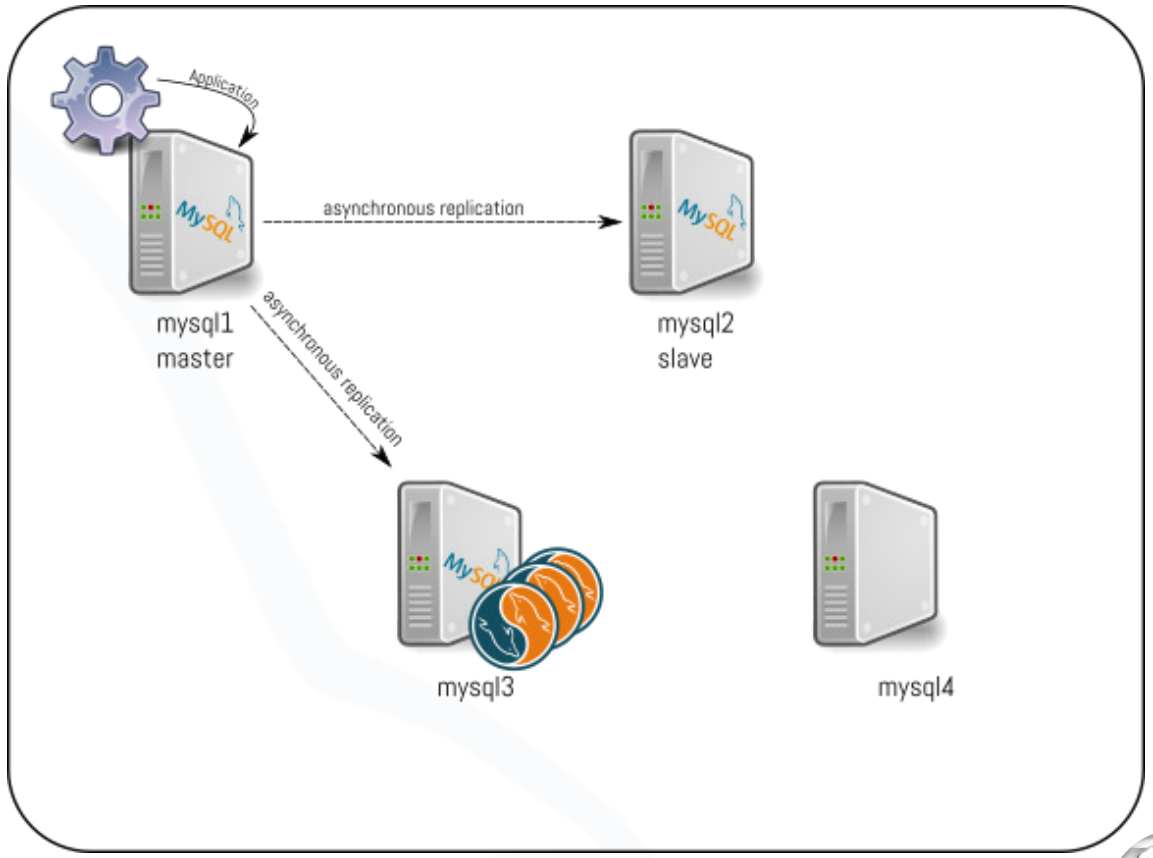
# LAB4: Cluster Status

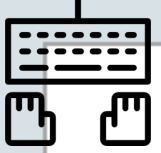
```
mysql-js> cluster.status()
{
    "clusterName": "perconalive",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "mysql3:3306",
        "ssl": "REQUIRED",
        "status": "OK_NO_TOLERANCE",
        "statusText": "Cluster is NOT tolerant to any failures.",
        "topology": {
            "mysql3:3306": {
                "address": "mysql3:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.16"
            }
        },
        "topologyMode": "Single-Primary"
    },
    "groupInformationSourceMember": "mysql3:3306"
}
```

# LAB5: add mysql4 to the cluster (1)

Add `mysql4` to the Group:

- restore the backup
- use MySQL Shell





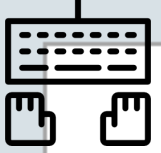
# LAB5: add mysql4 to the cluster (2)

Copy the backup from `mysql1` to `mysql4`:

```
[mysql1 ~]# scp -r /tmp/backup.sql mysql4:/tmp
```

Restore the backup:

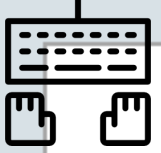
```
[mysql4 ~]# mysql < /tmp/backup.sql
```



# LAB5: MySQL Shell to add an instance (3)

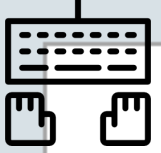
Let´s add `mysql4` (check configuration, check instance and add to cluster all in one command that can be done from the opened shell in `mysql3`):

```
mysql-js> dba.configureInstance('clusteradmin@mysql4')
mysql-js> cluster.addInstance('clusteradmin@mysql4')

mysql-js> cluster.status()
```



# Cluster Status

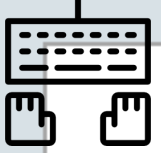
```
mysql-js> cluster.status()
{
    "clusterName": "perconalive",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "mysql3:3306",
        "ssl": "REQUIRED",
        "status": "OK_NO_TOLERANCE",
        "statusText": "Cluster is NOT tolerant to any failures. 1 member is no
        "topology": {
            "mysql3:3306": {
                "address": "mysql3:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "mysql4:3306": {
                "address": "mysql4:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "RECOVERING"
```



# Recovering progress

On standard MySQL, monitor the `group_replication_recovery` channel to see the progress:

```
mysql4> show slave status for channel 'group_replication_recovery'\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: mysql3
                  Master_User: mysql_innodb_cluster_rpl_user
                             ...
              Slave_IO_Running: Yes
             Slave_SQL_Running: Yes
                             ...
           Retrieved_Gtid_Set: 6e7d7848-860f-11e6-92e4-08002718d305:1-6,
7c1f0c2d-860d-11e6-9df7-08002718d305:1-15,
b346474c-8601-11e6-9b39-08002718d305:1964-77177,
e8c524df-860d-11e6-9df7-08002718d305:1-2
            Executed_Gtid_Set: 7c1f0c2d-860d-11e6-9df7-08002718d305:1-7,
b346474c-8601-11e6-9b39-08002718d305:1-45408,
e8c524df-860d-11e6-9df7-08002718d305:1-2
                             ...
```



# Cluster Status


```
{
    "clusterName": "perconalive",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "mysql3:3306",
        "ssl": "REQUIRED",
        "status": "OK_NO_TOLERANCE",
        "statusText": "Cluster is NOT tolerant to any failures.",
        "topology": {
            "mysql3:3306": {
                "address": "mysql3:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "mysql4:3306": {
                "address": "mysql4:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            }
```

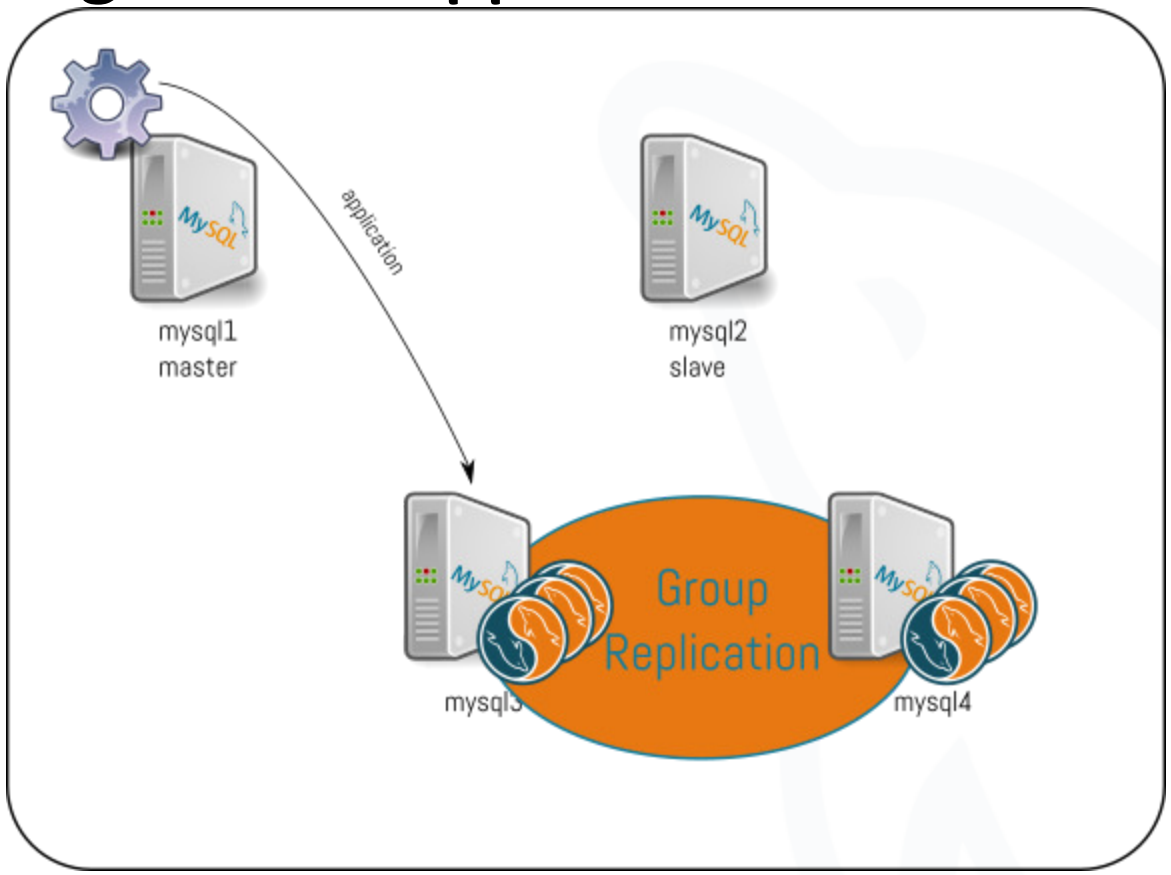
# Asynchronous Replication

When a group is created, the asynchronous **SQL Thread** is stopped and needs to be restarted.

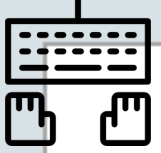
Verify that replication from `mysql1` is still running on `mysql3`, and restart the SQL Thread:

```
mysql-sql> \c clusteradmin@mysql3
mysql-sql> SHOW SLAVE STATUS\G
mysql-sql> START SLAVE SQL_THREAD;
```

# Migrate the application

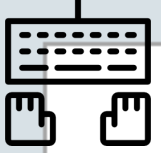



point the application to the cluster



# LAB6: Migrate the application

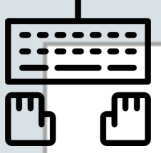
Now we need to point the application to `mysql3`, this is the only downtime !



# LAB6: Migrate the application

Now we need to point the application to `mysql3`, this is the only downtime !

- Make sure the master and replica is in sync

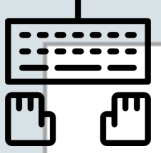


# LAB6: Migrate the application

Now we need to point the application to `mysql3`, this is the only downtime !

- Make sure the master and replica is in sync
- Start application, connecting to the Group Replication master (`mysql3`)

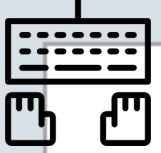
```
...
[  21257s] threads: 4, tps: 12.00, reads: 167.94, writes: 47.98, response t
[  21258s] threads: 4, tps: 6.00,  reads: 83.96, writes:  23.99, response t
[  21259s] threads: 4, tps: 7.00,  reads: 98.05, writes:  28.01, response t
[  31250s] threads: 4, tps: 8.00,  reads: 111.95, writes: 31.99, response t
[  31251s] threads: 4, tps: 11.00, reads: 154.01, writes: 44.00, response t
[  31252s] threads: 4, tps: 11.00, reads: 153.94, writes: 43.98, response t
[  31253s] threads: 4, tps: 10.01, reads: 140.07, writes: 40.02, response t
^C
[mysql1 ~]# run_app.sh mysql3
```



# LAB6: Migrate the application

Make sure `gtid_executed` range on `mysql2` is lower or equal than on `mysql3`

```
mysql[2-3]> SHOW GLOBAL VARIABLES LIKE 'gtid_executed'\G
```



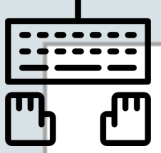
# LAB6: Migrate the application

Make sure `gtid_executed` range on `mysql2` is lower or equal than on `mysql3`

```
mysql[2-3]> SHOW GLOBAL VARIABLES LIKE 'gtid_executed'\G
```

**When they are OK**, stop asynchronous replication on `mysql2` and `mysql3`:

```
mysql2> STOP SLAVE;
mysql3> STOP SLAVE;
```



# LAB6: Migrate the application

Make sure `gtid_executed` range on `mysql2` is lower or equal than on `mysql3`

```
mysql[2-3]> SHOW GLOBAL VARIABLES LIKE 'gtid_executed'\G
```

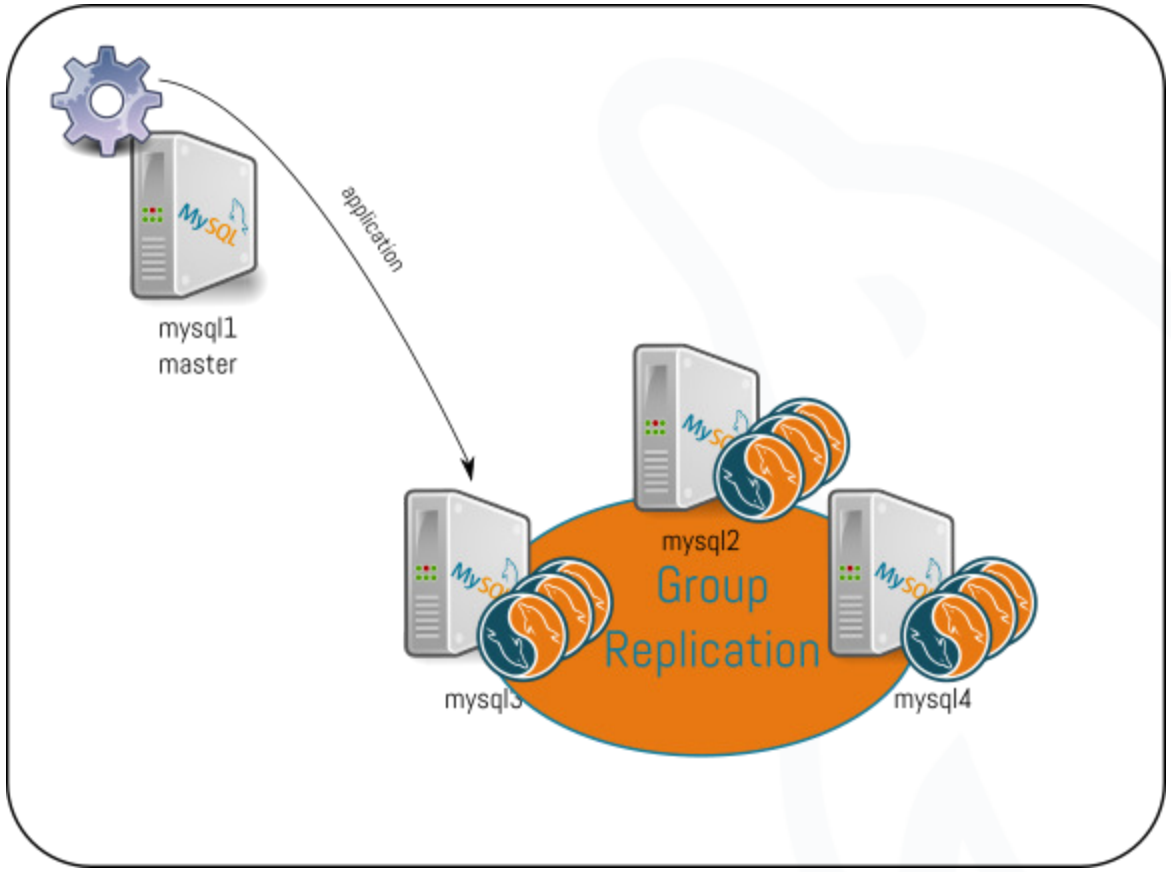
**When they are OK**, stop asynchronous replication on `mysql2` and `mysql3`:

```
mysql2> STOP SLAVE;
mysql3> STOP SLAVE;
```

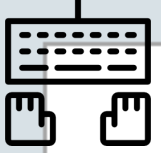
Now they can forget about `mysql1`:

```
mysql[2-3]> RESET SLAVE ALL FOR CHANNEL '';
```

# Add a third instance

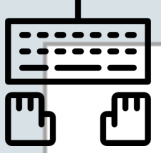



previous slave (`mysql2`) can now be part of the cluster



# LAB7: Add mysql2 to the group

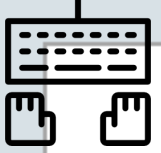
We first upgrade to MySQL 8.0.16 :



# LAB7: Add mysql2 to the group

We first upgrade to MySQL 8.0.16 :

```
[mysql2 ~]# systemctl stop mysqld
[mysql2 ~]# rpm -Uvh /root/rpms/*rpm --force
[mysql2 ~]# systemctl start mysqld
```



# LAB7: Add mysql2 to the group

We first upgrade to MySQL 8.0.16 :

```
[mysql2 ~]# systemctl stop mysqld
[mysql2 ~]# rpm -Uvh /root/rpms/*rpm --force
[mysql2 ~]# systemctl start mysqld
```

No need to run `mysql_updgrade` as we are using MySQL 8.0.16 ! 😁

# LAB7: Add mysql2 to the group (2)

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

# LAB7: Add mysql2 to the group (2)

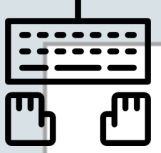
```
mysql-js> dba.configureInstance('clusteradmin@mysql2')
```

In case you don´t have the `cluster` object in your shell anymore, you can still retrieve it:

```
mysql-js> \c clusteradmin@mysql3
mysql-js> cluster = dba.getCluster()
```

Add `mysql2` to the cluster:

```
mysql-js> cluster.checkInstanceState('clusteradmin@mysql2')
mysql-js> cluster.addInstance("clusteradmin@mysql2")
mysql-js> cluster.status()
```

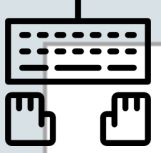


# LAB7: Add mysql2 to the group (3)


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

get more info

# Monitoring



# Performance Schema

## Group Replication uses `Performance_Schema` to expose status

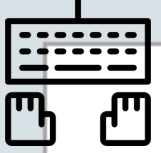
```
 mysql3> SELECT * FROM performance_schema.replication_group_members\G
 *************************** 1. row ***************************
  CHANNEL_NAME: group_replication_applier
     MEMBER_ID: ade14d5c-9e1e-11e7-b034-08002718d305
   MEMBER_HOST: mysql4
   MEMBER_PORT: 3306
  MEMBER_STATE: ONLINE
   MEMBER_ROLE: SECONDARY
MEMBER_VERSION: 8.0.16
 *************************** 2. row ***************************
  CHANNEL_NAME: group_replication_applier
     MEMBER_ID: b9d01593-9dfb-11e7-8ca6-08002718d305
   MEMBER_HOST: mysql3
   MEMBER_PORT: 3306
  MEMBER_STATE: ONLINE
   MEMBER_ROLE: PRIMARY
MEMBER_VERSION: 8.0.16
```

```
mysql3> SELECT * FROM performance_schema.replication_connection_status\G
*************************** 1. row ***************************
                                 CHANNEL_NAME: group_replication_applier
                                   GROUP_NAME: 8fc848d7-9e1c-11e7-9407...
                                  SOURCE_UUID: 8fc848d7-9e1c-11e7-9407...
                                    THREAD_ID: NULL
                                SERVICE_STATE: ON
                          COUNT_RECEIVED_HEARTBEATS: 0
                           LAST_HEARTBEAT_TIMESTAMP: 0000-00-00 00:00:00
                           RECEIVED_TRANSACTION_SET: 8fc848d7-9e1c-11e7-9407...
b9d01593-9dfb-11e7-8ca6-08002718d305:1-21,
da2f0910-8767-11e6-b82d-08002718d305:1-164741
                                LAST_ERROR_NUMBER: 0
                               LAST_ERROR_MESSAGE:
                             LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
                          LAST_QUEUED_TRANSACTION: 8fc848d7-9e1c-11e7-9407...
 LAST_QUEUED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP: 0000-00-00 00:00:00
LAST_QUEUED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP: 0000-00-00 00:00:00
     LAST_QUEUED_TRANSACTION_START_QUEUE_TIMESTAMP: 2017-09-20 16:22:36.486...
       LAST_QUEUED_TRANSACTION_END_QUEUE_TIMESTAMP: 2017-09-20 16:22:36.486...
                              QUEUEING_TRANSACTION:
    QUEUEING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP: 0000-00-00 00:00:00
   QUEUEING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP: 0000-00-00 00:00:00
        QUEUEING_TRANSACTION_START_QUEUE_TIMESTAMP: 0000-00-00 00:00:00
```

# Member State

These are the different possible state for a node member:

- `ONLINE`
- `OFFLINE`
- `RECOVERING`
- `ERROR`: when a node is leaving but the plugin was not instructed to stop
- `UNREACHABLE`



# Status information & metrics

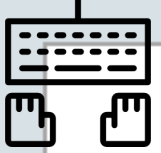
## Members

```
mysql> select member_host, member_state, member_role,
          member_version
          from performance_schema.replication_group_members;
+-------------+--------------+-------------+----------------+
| member_host | member_state | member_role | member_version |
+-------------+--------------+-------------+----------------+
| mysql3      | ONLINE       | PRIMARY     | 8.0.16         |
| mysql4      | ONLINE       | SECONDARY   | 8.0.16         |
| mysql2      | ONLINE       | SECONDARY   | 8.0.16         |
+-------------+--------------+-------------+----------------+
```

# Status information & metrics - connections

To see `group_replication_applier` and `group_replication_recovery`:

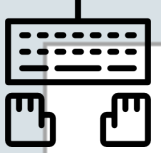
```
mysql> SELECT * FROM performance_schema.replication_connection_status\G
```



# Status information & metrics

## Previously there were only local node statistics, now they are exposed all over the Group

```
mysql> select * from performance_schema.replication_group_member_stats\G
```



# Status information & metrics

## Previously there were only local node statistics, now they are exposed all over the Group

```
mysql> select * from performance_schema.replication_group_member_stats\G
```

```
*************************** 1. row ***************************
                           CHANNEL_NAME: group_replication_applier
                                VIEW_ID: 15059231192196925:2
                              MEMBER_ID: ade14d5c-9e1e-11e7-b034-08002...
            COUNT_TRANSACTIONS_IN_QUEUE: 0
             COUNT_TRANSACTIONS_CHECKED: 27992
               COUNT_CONFLICTS_DETECTED: 0
      COUNT_TRANSACTIONS_ROWS_VALIDATING: 0
      TRANSACTIONS_COMMITTED_ALL_MEMBERS: 8fc848d7-9e1c-11e7-9407-08002...
b9d01593-9dfb-11e7-8ca6-08002718d305:1-21,
da2f0910-8767-11e6-b82d-08002718d305:1-164741
          LAST_CONFLICT_FREE_TRANSACTION: 8fc848d7-9e1c-11e7-9407-08002...
COUNT_TRANSACTIONS_REMOTE_IN_APPLIER_QUEUE: 0
         COUNT_TRANSACTIONS_REMOTE_APPLIED: 27992
         COUNT_TRANSACTIONS_LOCAL_PROPOSED: 0
```

```
*************************** 2. row ***************************
                            CHANNEL_NAME: group_replication_applier
                                 VIEW_ID: 15059231192196925:2
                               MEMBER_ID: b9d01593-9dfb-11e7-8ca6-08002...
             COUNT_TRANSACTIONS_IN_QUEUE: 0
              COUNT_TRANSACTIONS_CHECKED: 28000
                COUNT_CONFLICTS_DETECTED: 0
       COUNT_TRANSACTIONS_ROWS_VALIDATING: 0
       TRANSACTIONS_COMMITTED_ALL_MEMBERS: 8fc848d7-9e1c-11e7-9407-08002...
b9d01593-9dfb-11e7-8ca6-08002718d305:1-21,
da2f0910-8767-11e6-b82d-08002718d305:1-164741
           LAST_CONFLICT_FREE_TRANSACTION: 8fc848d7-9e1c-11e7-9407-08002...
COUNT_TRANSACTIONS_REMOTE_IN_APPLIER_QUEUE: 0
         COUNT_TRANSACTIONS_REMOTE_APPLIED: 1
         COUNT_TRANSACTIONS_LOCAL_PROPOSED: 28000
         COUNT_TRANSACTIONS_LOCAL_ROLLBACK: 0
```

# Performance_Schema

You can find GR information in the following **Performance_Schema** tables:

- `replication_applier_configuration`
- `replication_applier_status`
- `replication_applier_status_by_worker`
- `replication_connection_configuration`
- `replication_connection_status`
- `replication_group_member_stats`
- `replication_group_members`

# Get the info from the MySQL Shell

It´s also possible to get more info using the Shell:

```
mysql-js> cluster.status({extended: true})
```

# Get the info from the MySQL Shell

It´s also possible to get more info using the Shell:

```
mysql-js> cluster.status({extended: true})
```

In the next release you will also be able to see info from each members:

```
mysql-js> cluster.status({queryMembers: true})
```

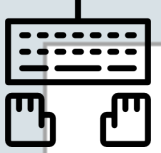
# Status during recovery

```
mysql> SHOW SLAVE STATUS FOR CHANNEL 'group_replication_recovery'\G
```

# Status during recovery

```
mysql> SHOW SLAVE STATUS FOR CHANNEL 'group_replication_recovery'\G
```

```
*************************** 1. row ***************************
          Slave_IO_State:
             Master_Host: <NULL>
             Master_User: gr_repl
             Master_Port: 0
                  ...
          Relay_Log_File: mysql4-relay-bin-group_replication_recovery.000001
                  ...
         Slave_IO_Running: No
        Slave_SQL_Running: No
                  ...
        Executed_Gtid_Set: 5de4400b-3dd7-11e6-8a71-08002774c31b:1-814089,
afb80f36-2bff-11e6-84e0-0800277dd3bf:1-5718
                  ...
              Channel_Name: group_replication_recovery
```



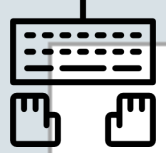
# Sys Schema

The easiest way to detect if a node is a member of the primary component (when there are partitioning of your nodes due to network issues for example) and therefore a valid candidate for routing queries to it, is to use the sys table.

Additional information for `sys` can be found at https://goo.gl/XFp3bt

On the primary node:

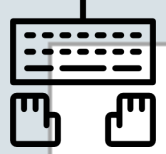
```
[mysql3 ~]# mysql < /root/addition_to_sys_8.0.2.sql
```



# Sys Schema

## Is this node part of PRIMARY Partition:

```
mysql3> SELECT sys.gr_member_in_primary_partition();
+------------------------------------+
| sys.gr_node_in_primary_partition() |
+------------------------------------+
| YES                                |
+------------------------------------+
```



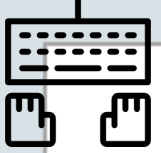
# Sys Schema

## Is this node part of PRIMARY Partition:

```
mysql3> SELECT sys.gr_member_in_primary_partition();
+------------------------------------+
| sys.gr_node_in_primary_partition() |
+------------------------------------+
| YES                                |
+------------------------------------+
```

## To use as healthcheck:

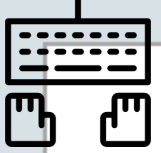
```
mysql3> SELECT * FROM sys.gr_member_routing_candidate_status;
+------------------+-----------+---------------------+----------------------+
| viable_candidate | read_only | transactions_behind | transactions_to_cert |
+------------------+-----------+---------------------+----------------------+
| YES              | YES       | 0                   | 0                    |
+------------------+-----------+---------------------+----------------------+
```



# LAB8: Sys Schema - Health Check

On one of the non Primary nodes, run the following command:

```
mysql-sql> flush tables with read lock;
```



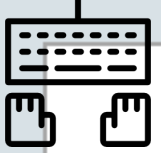
# LAB8: Sys Schema - Health Check

On one of the non Primary nodes, run the following command:

```
mysql-sql> flush tables with read lock;
```

Now you can verify what the healthcheck exposes to you:

```
mysql-sql> SELECT * FROM sys.gr_member_routing_candidate_status;
+------------------+-----------+---------------------+----------------------+
| viable_candidate | read_only | transactions_behind | transactions_to_cert |
+------------------+-----------+---------------------+----------------------+
| YES              | YES       |                 950 |                    0 |
+------------------+-----------+---------------------+----------------------+
```



# LAB8: Sys Schema - Health Check

On one of the non Primary nodes, run the following command:

```
mysql-sql> flush tables with read lock;
```

Now you can verify what the healthcheck exposes to you:

```
mysql-sql> SELECT * FROM sys.gr_member_routing_candidate_status;
+------------------+-----------+---------------------+----------------------+
| viable_candidate | read_only | transactions_behind | transactions_to_cert |
+------------------+-----------+---------------------+----------------------+
| YES              | YES       |                 950 |                    0 |
+------------------+-----------+---------------------+----------------------+
```

```
mysql-sql> UNLOCK TABLES;
```

# LAB9: Reporting in MySQL Shell 8.0.16

Since 8.0.16, it´s possible in MySQL Shell to create User Defined Reports.

We will create a report to monitor the asynchronous replication used in the recovery channel and see the progress of it.

# LAB9: Reporting in MySQL Shell 8.0.16

Since 8.0.16, it´s possible in MySQL Shell to create User Defined Reports.

We will create a report to monitor the asynchronous replication used in the recovery channel and see the progress of it.

Let´s start by stopping group replication on `mysql2`:

```
mysql2> stop group_replication;
```

# LAB9: Reporting in MySQL Shell 8.0.16 (2)

Now it´s time to create the folder where the User Defined Reports can be installed:

```
# mkdir ~/.mysqlsh/init.d
# cd ~/.mysqlsh/init.d
```

And we can download the report:

```
wget https://tinyurl.com/gr-recovery -O gr_recovery_progress.py
```

# LAB9: Reporting in MySQL Shell 8.0.16 (2)

Now it´s time to create the folder where the User Defined Reports can be installed:

```
# mkdir ~/.mysqlsh/init.d
# cd ~/.mysqlsh/init.d
```

And we can download the report:

```
wget https://tinyurl.com/gr-recovery -O gr_recovery_progress.py
```

```
mysqlsh  clusteradmin@mysql2
mysql-js> \show
Available reports: gr_recovery_progress, query.
```

# LAB9: Reporting in MySQL Shell 8.0.16 (3)

Now we can put back `mysql2` in the group and use the new report query:

```
mysql-js> \sql start group_replication
mysql-js> \show gr_recovery_progress
+-----------------+
| trx_to_recover |
+-----------------+
| 8673           |
+-----------------+
```

You can also try `\watch gr_recovery_progress`

# Connecting to the cluster

## MySQL Router

# MySQL Router

MySQL Router is lightweight middleware that provides transparent routing between your application and backend MySQL Servers. It can be used for a wide variety of use cases, such as providing high availability and scalability by effectively routing database traffic to appropriate backend MySQL Servers.

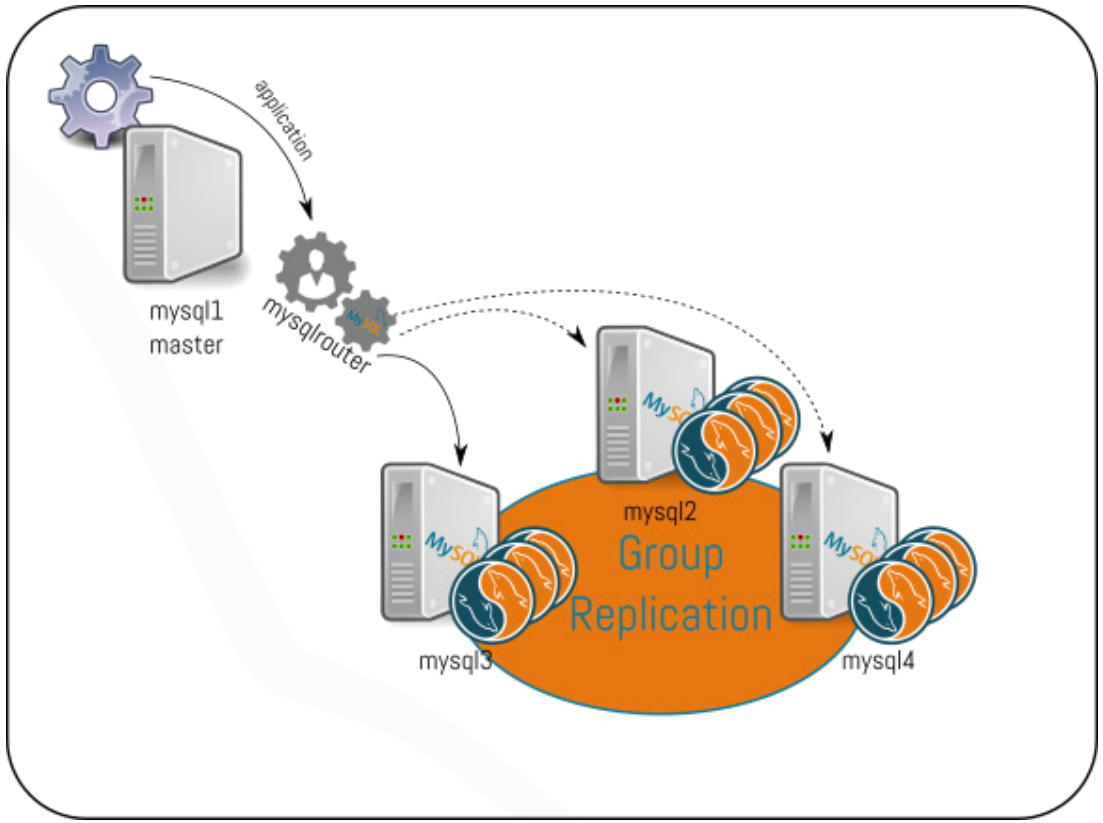
# MySQL Router

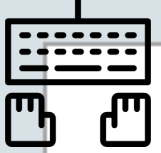
MySQL Router is lightweight middleware that provides transparent routing between your application and backend MySQL Servers. It can be used for a wide variety of use cases, such as providing high availability and scalability by effectively routing database traffic to appropriate backend MySQL Servers.

MySQL Router doesn´t require any specific configuration. It configures itself automatically (bootstrap) using MySQL InnoDB Cluster ´s metadata.

# LAB10: MySQL Router

We will now use `mysqlrouter` between our application and the cluster.

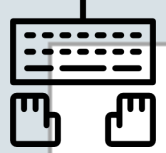




# LAB10: MySQL Router (2)

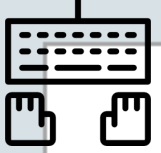
Configure MySQL Router that will run on the app server (`mysql1`). We bootstrap it using the **Primary-Master**:

```
[root@mysql1 ~]# mysqlrouter --bootstrap clusteradmin@mysql3:3306 \
                             --user mysqlrouter
Please enter MySQL password for clusteradmin:

Bootstrapping system MySQL Router instance...
Checking for old Router accounts
Creating account mysql_router1_rvbecwmlc6uf@'%'
MySQL Router  has now been configured for the InnoDB cluster 'perconalive'.

The following connection information can be used to connect to the cluster
after MySQL Router has been started with generated configuration..

Classic MySQL protocol connections to cluster 'perconalive':
- Read/Write Connections: localhost:6446
- Read/Only Connections: localhost:6447
X protocol connections to cluster 'perconalive':
- Read/Write Connections: localhost:64460
- Read/Only Connections: localhost:64470
```



# LAB10: MySQL Router (3)

Now let´s modify the configuration file to listen on port 3306:

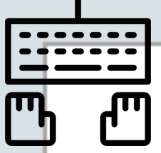


# LAB10: MySQL Router (3)

Now let´s modify the configuration file to listen on port 3306:

in `/etc/mysqlrouter/mysqlrouter.conf`:

```
[routing:perconalive_default_rw]
-bind_port=6446
+bind_port=3306
```



# LAB10: MySQL Router (3)

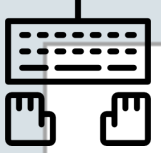
Now let´s modify the configuration file to listen on port 3306:

in `/etc/mysqlrouter/mysqlrouter.conf`:

```
[routing:perconalive_default_rw]
-bind_port=6446
+bind_port=3306
```

We can stop `mysqld` on `mysql1` and start `mysqlrouter`:

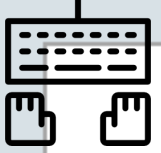
```
[mysql1 ~]# systemctl stop mysqld
[mysql1 ~]# systemctl start mysqlrouter
```



# LAB10: MySQL Router (5)

Now we can point the application to the router (back to mysql1):

```
[mysql1 ~]# run_app.sh
```



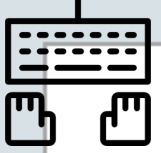
# LAB10: MySQL Router (5)

Now we can point the application to the router (back to mysql1):

```
[mysql1 ~]# run_app.sh
```

Check app and kill `mysqld` on **mysql3** (the Primary Master R/W node) !

```
[mysql3 ~]# failure-scenario.sh kill-node
```



# LAB10: MySQL Router (5)

Now we can point the application to the router (back to mysql1):

```
[mysql1 ~]# run_app.sh
```

Check app and kill `mysqld` on **mysql3** (the Primary Master R/W node) !

```
[mysql3 ~]# failure-scenario.sh kill-node
```

```
mysql2> select member_host as `primary`
       from performance_schema.replication_group_members
       where member_role='PRIMARY';
+----------+
| primary  |
+----------+
| mysql4   |
+----------+
```

ProxySQL / HA Proxy / F5 / ...

# 3rd party router/proxy

# 3rd party router/proxy

MySQL InnoDB Cluster can also work with third party router / proxy.

# 3rd party router/proxy

MySQL **InnoDB Cluster** can also work with third party router / proxy.

If you need some specific features that are not yet available in MySQL **Router**, like transparent R/W splitting, then you can use your software of choice.

# 3rd party router/proxy

**MySQL InnoDB Cluster** can also work with third party router / proxy.

If you need some specific features that are not yet available in **MySQL Router**, like transparent R/W splitting, then you can use your software of choice.

The important part of such implementation is to use a good health check to verify if the MySQL server you plan to route the traffic is in a valid state.

# 3rd party router/proxy

MySQL **InnoDB Cluster** can also work with third party router / proxy.

If you need some specific features that are not yet available in MySQL **Router**, like transparent R/W splitting, then you can use your software of choice.

The important part of such implementation is to use a good health check to verify if the MySQL server you plan to route the traffic is in a valid state.

MySQL **Router** implements that natively, and it´s very easy to deploy.

# 3rd party router/proxy

**MySQL InnoDB Cluster** can also work with third party router / proxy.

If you need some specific features that are not yet available in **MySQL Router**, like transparent R/W splitting, then you can use your software of choice.

The important part of such implementation is to use a good health check to verify if the MySQL server you plan to route the traffic is in a valid state.

**MySQL Router** implements that natively, and it´s very easy to deploy.

**ProxySQL** also has native support for Group Replication which makes it maybe the best choice for advanced users.

operational tasks

# Recovering Node

# Recovering Nodes/Members

- The old master (`mysql3`) got killed.
- Start it again:

```
mysql3# systemctl start mysqld
```

- The node will automatically join the group again

You know... There are ways...

# Various Configuration Options

# Various Configuration Options

- Single & Multiple Primary Mode
- Network Partitions
  - What to do?
  - when will a node leave?
  - when will a node be expelled?
  - what should a node do when it´s gone?

# Default = Single Primary Mode

By default, MySQL InnoDB Cluster enables `Single Primary Mode`.

# Default = Single Primary Mode

By default, MySQL InnoDB Cluster enables `Single Primary Mode`.

```
mysql> show global variables like 'group_replication_single_primary_mode';
+----------------------------------------+-------+
| Variable_name                          | Value |
+----------------------------------------+-------+
| group_replication_single_primary_mode | ON    |
+----------------------------------------+-------+
```

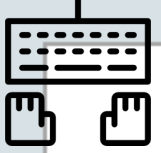
# Default = Single Primary Mode

By default, MySQL InnoDB Cluster enables `Single Primary Mode`.

```
mysql> show global variables like 'group_replication_single_primary_mode';
+----------------------------------------+-------+
| Variable_name                          | Value |
+----------------------------------------+-------+
| group_replication_single_primary_mode | ON    |
+----------------------------------------+-------+
```

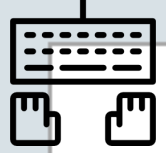
In Single Primary Mode, a single member acts as the writable master (`PRIMARY`) and the rest of the members act as hot-standbys (`SECONDARY`).

The group itself coordinates and configures itself automatically to determine which member will act as the `PRIMARY`, through a leader election mechanism.



# Who´s the Primary Master ?

You can already see it in the output of `cluster.status()`



# Who´s the Primary Master ?

You can already see it in the output of `cluster.status()`

But you can also check in `performance_schema`:

```
mysql> select member_host
       from performance_schema.replication_group_members
       where member_role='PRIMARY';
+-------------+
| member_host |
+-------------+
| mysql3      |
+-------------+
```

# Create a Multi-Primary Cluster:

It´s also possible to create a Multi-Primary Cluster using the Shell:

```
mysql-js> cluster=dba.createCluster('perconalive',{multiMaster: true})
```

# Create a Multi-Primary Cluster:

It´s also possible to create a Multi-Primary Cluster using the Shell:

```
mysql-js> cluster=dba.createCluster('perconalive',{multiMaster: true})
```

```
A new InnoDB cluster will be created on instance 'clusteradmin@mysql3:3306'.

The MySQL InnoDB cluster is going to be setup in advanced Multi-Master Mode.
Before continuing you have to confirm that you understand the requirements and
limitations of Multi-Master Mode. Please read the manual before proceeding.


I have read the MySQL InnoDB cluster manual and I understand the requirements
and limitations of advanced Multi-Master Mode.
Confirm [y|N]:
```

# Create a Multi-Primary Cluster:

It´s also possible to create a Multi-Primary Cluster using the Shell:

```
mysql-js> cluster=dba.createCluster('perconalive',{multiMaster: true})
```

```
A new InnoDB cluster will be created on instance 'clusteradmin@mysql3:3306'.

The MySQL InnoDB cluster is going to be setup in advanced Multi-Master Mode.
Before continuing you have to confirm that you understand the requirements and
limitations of Multi-Master Mode. Please read the manual before proceeding.


I have read the MySQL InnoDB cluster manual and I understand the requirements
and limitations of advanced Multi-Master Mode.
Confirm [y|N]:
```

Or you can force it to avoid interaction (for automation) :

# Create a Multi-Primary Cluster:

It´s also possible to create a Multi-Primary Cluster using the Shell:

```
mysql-js> cluster=dba.createCluster('perconalive',{multiMaster: true})
```

```
A new InnoDB cluster will be created on instance 'clusteradmin@mysql3:3306'.

The MySQL InnoDB cluster is going to be setup in advanced Multi-Master Mode.
Before continuing you have to confirm that you understand the requirements and
limitations of Multi-Master Mode. Please read the manual before proceeding.


I have read the MySQL InnoDB cluster manual and I understand the requirements
and limitations of advanced Multi-Master Mode.
Confirm [y|N]:
```

Or you can force it to avoid interaction (for automation) :

```
> cluster=dba.createCluster('perconalive',{multiMaster: true, force: true})
```

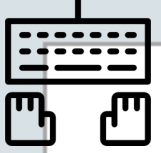
# Dynamically change to Multi Primary Mode (UDFs)

- Change to/from single primary mode online:

```
mysql3> SELECT group_replication_switch_to_multi_primary_mode();
mysql3> SELECT group_replication_switch_to_single_primary_mode();
```

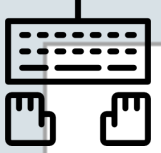
- Choose primary node

```
mysql3> SELECT group_replication_set_as_primary(server_uuid);
```



# Dynamically change to Multi Primary Mode (UDFs)

```
mysql3> SELECT group_replication_switch_to_multi_primary_mode();
+-----------------------------------------------------+
| group_replication_switch_to_multi_primary_mode() |
+-----------------------------------------------------+
| Mode switched to multi-primary successfully.     |
+-----------------------------------------------------+

mysql3> select member_host
        from performance_schema.replication_group_members
        where member_role='PRIMARY';
+-------------+
| member_host |
+-------------+
| mysql2      |
| mysql3      |
| mysql4      |
+-------------+
```

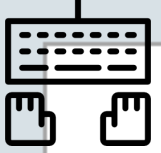


# Dynamically change Mode: UDFs VS Metadata

However, this should be done from the MySQL Shell to avoid metadata inconsistencies.

Try to get the cluster status:

```
JS> cluster.status()
```



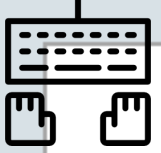
# Dynamically change Mode: UDFs VS Metadata

However, this should be done from the MySQL Shell to avoid metadata inconsistencies.

Try to get the cluster status:

```
JS> cluster.status()
```

What do you see ?



# Dynamically change Mode: UDFs VS Metadata

However, this should be done from the MySQL Shell to avoid metadata inconsistencies.

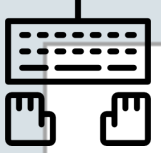
Try to get the cluster status:

```
JS> cluster.status()
```

What do you see ?

Fix it:

```
JS> cluster.rescan()
JS> cluster.status()
```



# Dynamically change to Single Primary Mode - Shell

Switch back to Single Primary Mode:

```
JS> cluster.switchToSinglePrimaryMode()
JS> cluster.status()
```

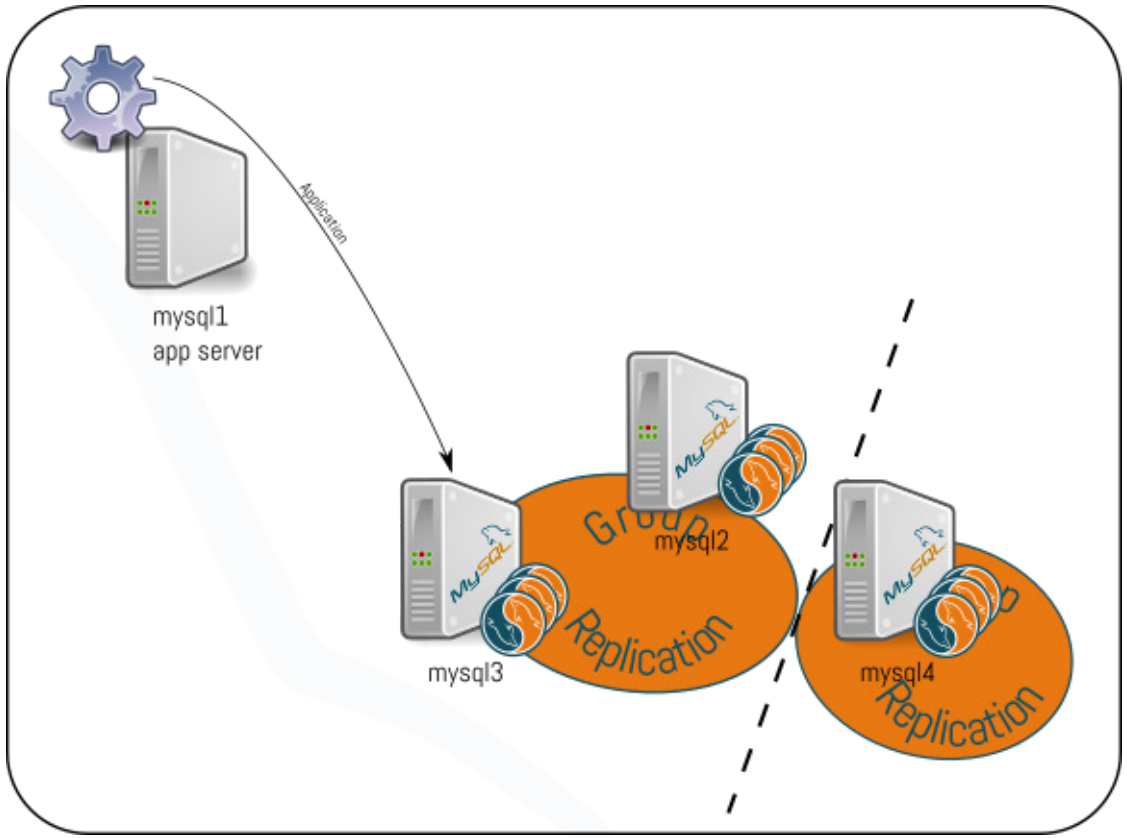
Other MySQL Shell syntax:

(do not execute !)

```
JS> cluster.setPrimaryInstance('mysql1:3306')
JS> cluster.switchToMultiPrimaryMode()
```

# Network Partitions

What happens when `mysql4` is being network partitioned?



# Network Partitions - Quorum Calculations

- 3 notes, each 1 vote.
  - Majority is when a partition sees +50%
  - Minority is when a partition sees <=50%

# Network Partitions - Quorum Calculations

- 3 notes, each 1 vote.
  - Majority is when a partition sees +50%
  - Minority is when a partition sees <=50%
- `mysql2` and `mysql3` can still connect to eachother, 2 votes out of 3
  = **66%** = **MAJORITY**
- `mysql4` can only see itself, 1 vote out of 3
  = **33%** = **MINORITY**

Majority network partition is `mysql2` and `mysql3`

# Network Partitions - Quorum Calculations

With even amount of nodes:

- it is possible to have 50% for each partition
- both will have minority

# Network Partitions - Actions by Majority

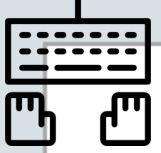
`mysql2` and `mysql3` actions:

1. wait 5 seconds, then suspect `mysql4` is gone
2. expel `mysql4` after `group_replication_member_expel_timeout` seconds (default `0`, immediately)
   do not put the variable too high, between suspect and expel
   - members cannot be added/removed
   - no new leader can be elected

# Network Partitions - Actions by Minority

`mysql4` actions:

1. wait 5 seconds before suspecting itself to lost connection to the majority
2. wait `group_replication_unreachable_majority_timeout` (default `0`, forever)
3. when timeout: `group_replication_autorejoin_tries` (default `0` times)
   - retries every 5 minutes
4. Only then `group_replication_exit_state_action`
   - `READ_ONLY`: allow stale reads to be served (latest default)
   - `ABORT_SERVER`: Shutdown MySQL



# LAB11: Network Partitions - Default

Let´s network partition `mysql4`!

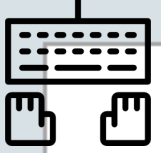
Check logs (we have `log_error_verbosity=3`)

```
mysql3# tail -f /var/log/mysqld.log &
```

```
mysql4# tail -f /var/log/mysqld.log &
```

On `mysql4`, run:

```
mysql4# failure-scenario.sh network-partition begin
```



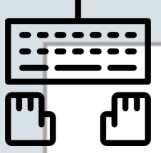
# LAB11: Network Partitions - Default

## mysql3

```
'[GCS] Installed site start={ed88d526 1339 0} boot_key={ed88d526 1328 0}
       event_horizon=10 node 1'
'[GCS] Group is able to support up to communication protocol version 8.0.16'
'Members removed from the group: mysql4:3306'
'Group membership changed to mysql3:3306, mysql2:3306 on view 1557809996177
'[GCS] Failure reading from fd=-1 n=18446744073709551615'
```

## mysql4

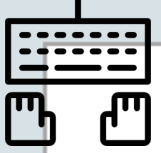
```
'Member with address mysql2:3306 has become unreachable.'
'Member with address mysql3:3306 has become unreachable.'
'This server is not able to reach a majority of members in the group.
   This server will now block all updates. The server will remain blocked
   until contact with the majority is restored. It is possible to use
   group_replication_force_members to force a new group membership.'
```



# LAB11: Network Partition - Default

```
mysql4# failure-scenario.sh network-partition end
```

Does the node come back?



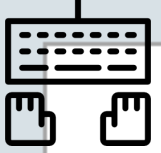
# LAB11: Network Partition - Default

```
mysql4# failure-scenario.sh network-partition end
```

Does the node come back?

**NO**: it was expelled:

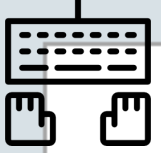
```
'Member with address mysql2:3306 is reachable again.'
'Member with address mysql3:3306 is reachable again.'
'The member has resumed contact with a majority of the members in the group.
      Regular operation is restored and transactions are unblocked.'
'[GCS] Re-using server node 0 host mysql2'
'[GCS] Re-using server node 1 host mysql3'
'[GCS] Installed site start={ed88d526 1339 0} boot_key={ed88d526 1328 0}
       event_horizon=10 node 4294967295'
'[GCS] Node has already been removed: mysql4:33061 15578108329721490.'
'Member was expelled from the group due to network failures,
       changing member status to ERROR.'
```



# LAB11: Network Partition - Default

## Fix with shell:

```
JS>cluster.rejoinInstance('clusteradmin@mysql4')
Rejoining the instance to the InnoDB cluster. Depending on the original
problem that made the instance unavailable, the rejoin operation might not be
successful and further manual steps will be needed to fix the underlying
problem.

Please monitor the output of the rejoin operation and take necessary action if
the instance cannot rejoin.

Rejoining instance to the cluster ...

The instance 'mysql4' was successfully rejoined on the cluster.
```



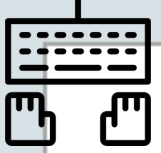
# LAB12: Network Partitions - With Retries

Let´s try again, but let´s configure retries:

```
mysql4> SET GLOBAL group_replication_autorejoin_tries=2;       # default is 0
```

```
mysql4# failure-scenario.sh network-partition begin && sleep 10 && \
        failure-scenario.sh network-partition end
```

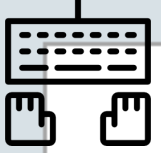
Did `mysql4` join the group again?



# LAB12: Network Partitions - With Retries

## Did mysql4 join the group again? **YES!**

```
'Member with address mysql2:3306 is reachable again.'
'The member has resumed contact with a majority of the members in the group.
       Regular operation is restored and transactions are unblocked.'
'Member with address mysql3:3306 is reachable again.'
'[GCS] Re-using server node 0 host mysql2'
'[GCS] Re-using server node 1 host mysql3'
'[GCS] Installed site start={ed88d526 2445 0} boot_key={ed88d526 2434 0}
       event_horizon=10 node 4294967295'
'[GCS] Node has already been removed: mysql4:33061 15578118987619030.'
'Member was expelled from the group due to network failures,
     changing member status to ERROR.'
'Started auto-rejoin procedure attempt 1 of 2'
...
'This server was declared online within the replication group.'
```



# LAB12: Network Partitions - monitoring auto-rejoin

It´s possible to verify if there is an auto-rejoin:

```
SELECT COUNT(*) FROM performance_schema.events_stages_current WHERE EVENT_NAME LIKE '%auto-rejoin%';
```

View the amount of retries so far:

```
SELECT WORK_COMPLETED FROM performance_schema.events_stages_current WHERE EVENT_NAME LIKE '%auto-rejoin%';
```

Estimated time remaining until next retry:

```
SELECT (300.0 - ((TIMER_WAIT*10e-12) - 300.0 * num_retries)) AS time_remaining FROM
(SELECT COUNT(*) - 1 AS num_retries FROM
performance_schema.events_stages_current WHERE EVENT_NAME LIKE '%auto-rejoin%') AS T,
performance_schema.events_stages_current WHERE EVENT_NAME LIKE '%auto-rejoin%';
```

be aware of the behavior change & take control

# Application Behavior & Consistency Guarantees

# Application Behavior & Consistency Guarantees

- Group Conflicts
- Configurable Consistency Guarantees

# Group Conflicts

Local vs. Group:

- Local Conflicts are handled using InnoDB's row locking.
- Conflicts between different nodes are handled with "**first committer wins!**"

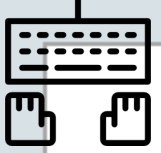
# Group Conflicts

Local vs. Group:

- Local Conflicts are handled using InnoDB's row locking.
- Conflicts between different nodes are handled with "**first committer wins!**"

Group Conflicts:

- happen only in Multi-Primary Mode
- happen if 2 transactions are updating/deleting the same records on two different members of the group,
- will be detected only during group certification (after at least one has committed) (Optimistic Locking)

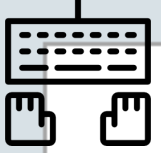


# LAB13: Group Conflicts

Let´s see this in action !

First switch to Multi-Primary Mode:

```
mysql-js> cluster.switchToMultiPrimaryMode()
```



# LAB13: Group Conflicts

Let´s see this in action !

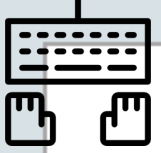
First switch to Multi-Primary Mode:

```
mysql-js> cluster.switchToMultiPrimaryMode()
```

Now on `mysql2` start the following transaction:

```
mysql2> start transaction;
mysql2> update app.sbtest1 set k=k+50000 where id = 2000;
```

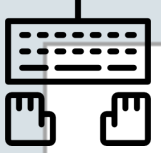
*Note that we don't commit the transaction yet*



# LAB13: Group Conflicts (2)

Now on `mysql4` we also create a new transaction:

```
mysql4> start transaction;
mysql4> update app.sbtest1 set k=k+200 where id > 1999 and id < 2010;
```



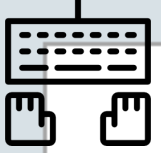
# LAB13: Group Conflicts (2)

Now on `mysql4` we also create a new transaction:

```
mysql4> start transaction;
mysql4> update app.sbtest1 set k=k+200 where id > 1999 and id < 2010;
```

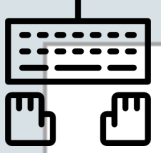
Now commit the one in `mysql4`:

```
mysql4> commit;
```



# LAB13: Group Conflicts (3)

What do you see on `mysql2`´s session ?



# LAB13: Group Conflicts (3)

What do you see on `mysql2`´s session ?

If you try any statement you will see:

```
ERROR: 1213: Deadlock found when trying to get lock; try restarting transaction
```

If you try to `commit` you will see:

```
ERROR: 1180: Got error 149 - 'Lock deadlock; Retry transaction' during COMMIT
```

This happens because a conflict was detected during certification !

# Group Conflicts (4)

2 types of Group Conflicts:

- Conflicts detected by InnoDB high priority transactions (previous lab)
- Conflicts detected by Group Replication:

Commit 2 transactions at almost the same time:

```
ERROR 3101 (40000): Plugin instructed the server
to rollback the current transaction.
```

# Group Conflicts (5)

Check these in `Performance_Schema` (or using `cluster.status({extended: true})`):

```
SELECT MEMBER_ID,COUNT_CONFLICTS_DETECTED,COUNT_TRANSACTIONS_LOCAL_ROLLBACK
FROM performance_schema.replication_group_member_stats\G
*************************** 1. row ***************************
                  MEMBER_ID: 130942a7-75c4-11e9-b75d-08002718d305
        COUNT_CONFLICTS_DETECTED: 1
COUNT_TRANSACTIONS_LOCAL_ROLLBACK: 0
*************************** 2. row ***************************
                  MEMBER_ID: 265d96f7-7333-11e9-b4c9-08002718d305
        COUNT_CONFLICTS_DETECTED: 1
COUNT_TRANSACTIONS_LOCAL_ROLLBACK: 1
*************************** 3. row ***************************
                  MEMBER_ID: 6361b4e3-75c2-11e9-851b-08002718d305
        COUNT_CONFLICTS_DETECTED: 1
COUNT_TRANSACTIONS_LOCAL_ROLLBACK: 0
```

# Configurable Consistency Guarantees

Default consistency guarantee is `EVENTUAL`

- if you write some data on `nodeA`,
- if you immediately read it on `nodeB`,
- it might be there... or not yet.

Since MySQL 8.0.14, you can configure various transaction consistency guarantees

```
mysql> show variables like 'group_replication_consistency';
+--------------------------------+----------+
| Variable_name                  | Value    |
+--------------------------------+----------+
| group_replication_consistency | EVENTUAL |
+--------------------------------+----------+
```

# Consistency Levels

The consistency levels are defined in `group_replication_consistency`.

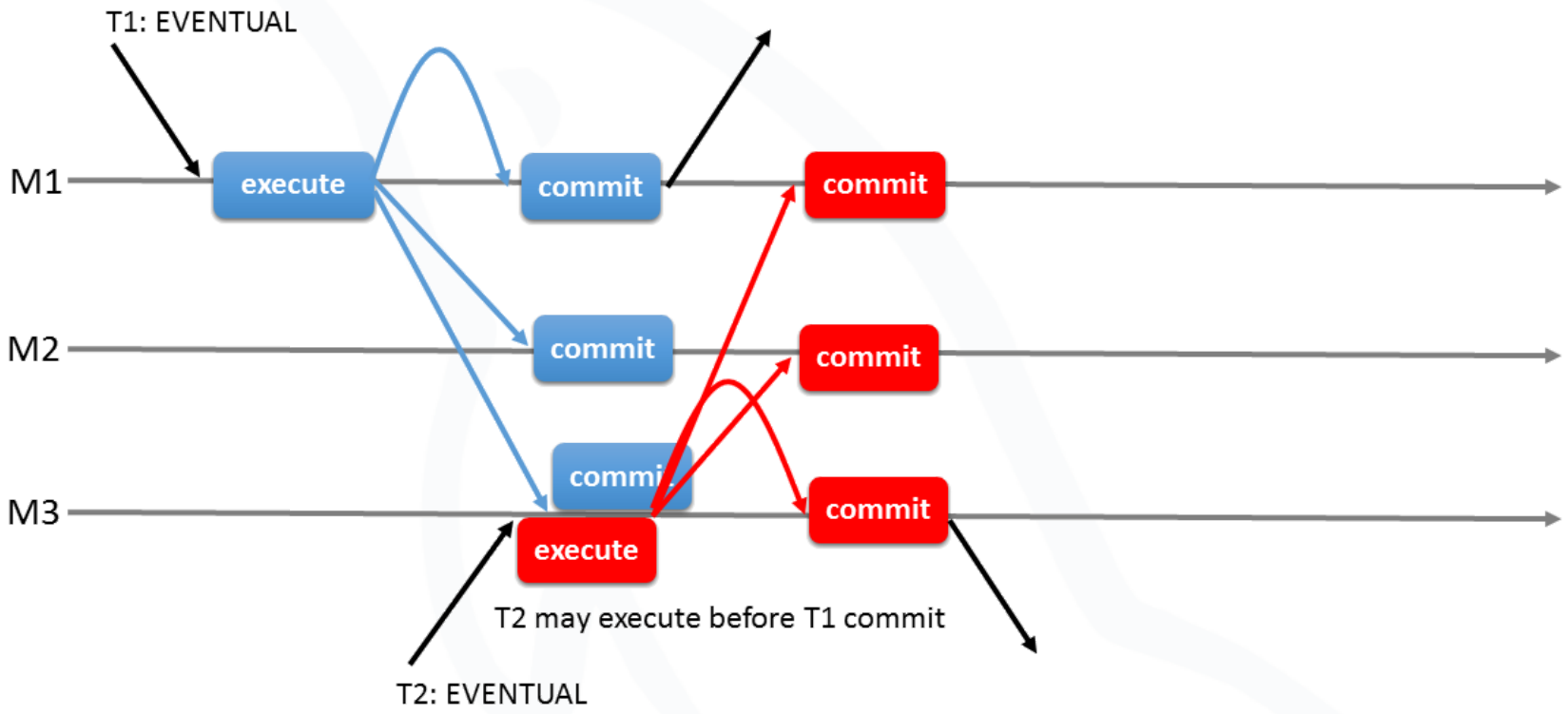
The scope can be `SESSION` or `GLOBAL`.

The possible values are:

- `EVENTUAL` (default)
- `BEFORE_ON_PRIMARY_FAILOVER`
- `BEFORE`
- `AFTER`
- `BEFORE_AND_AFTER`

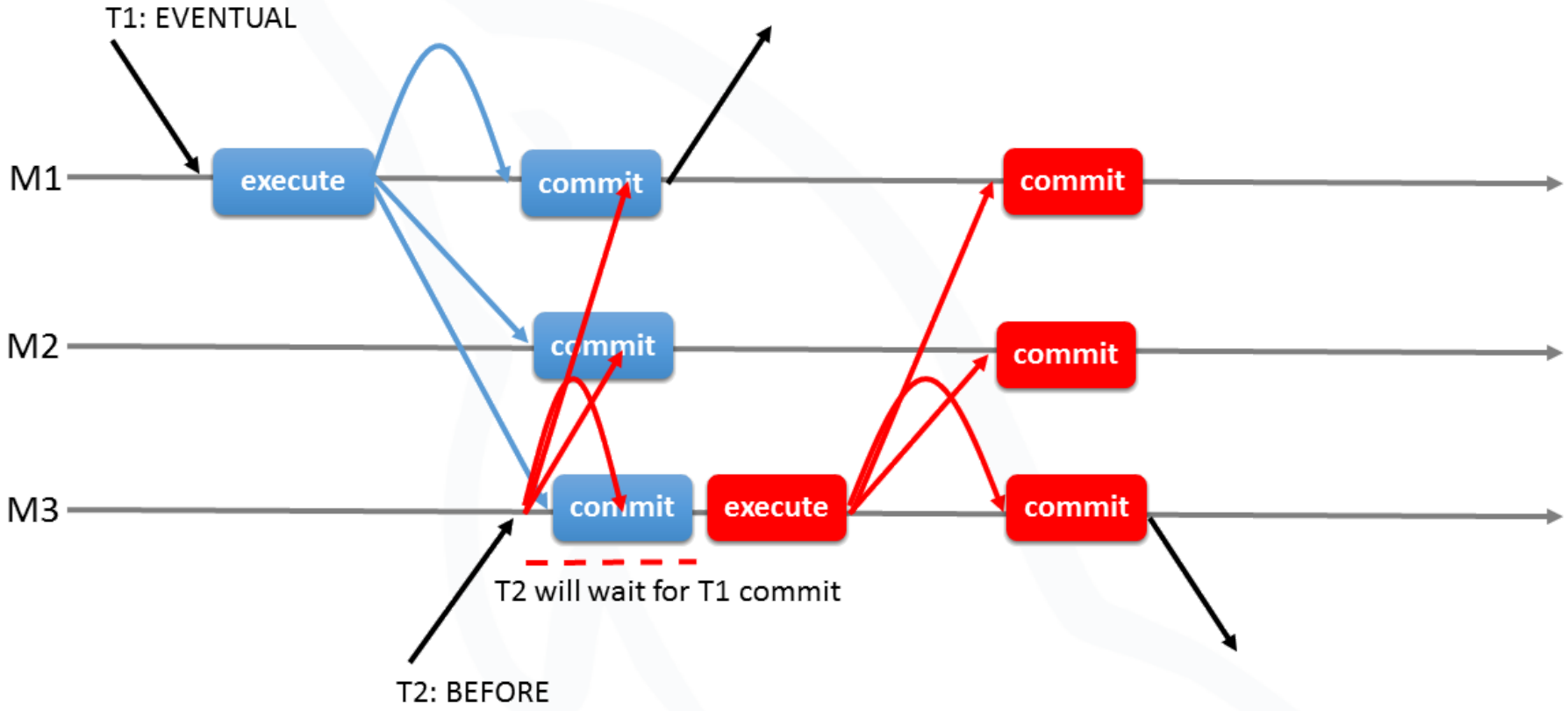
# Consistency Levels - EVENTUAL

EVENTUAL: do not wait



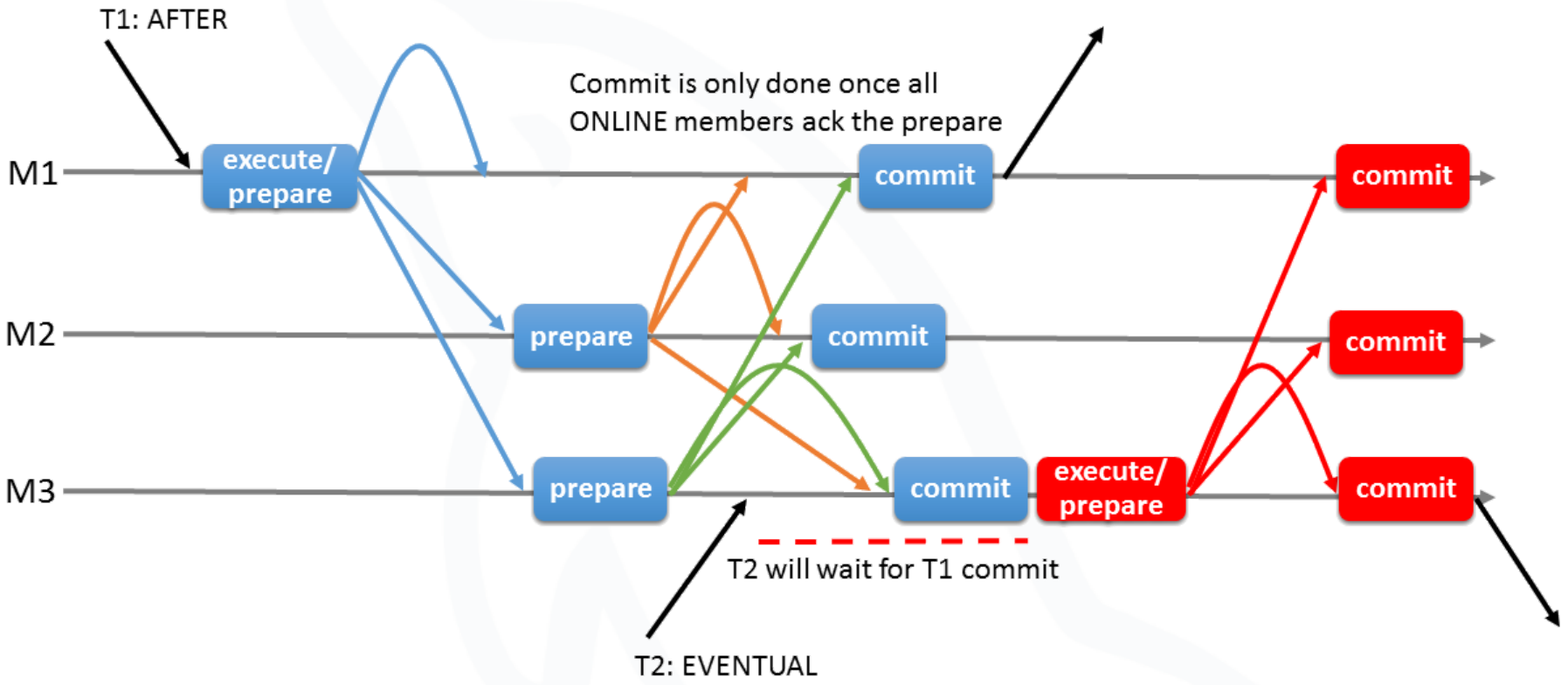
# Consistency Levels - BEFORE

BEFORE: reads & writes wait on other trx BEFORE they start



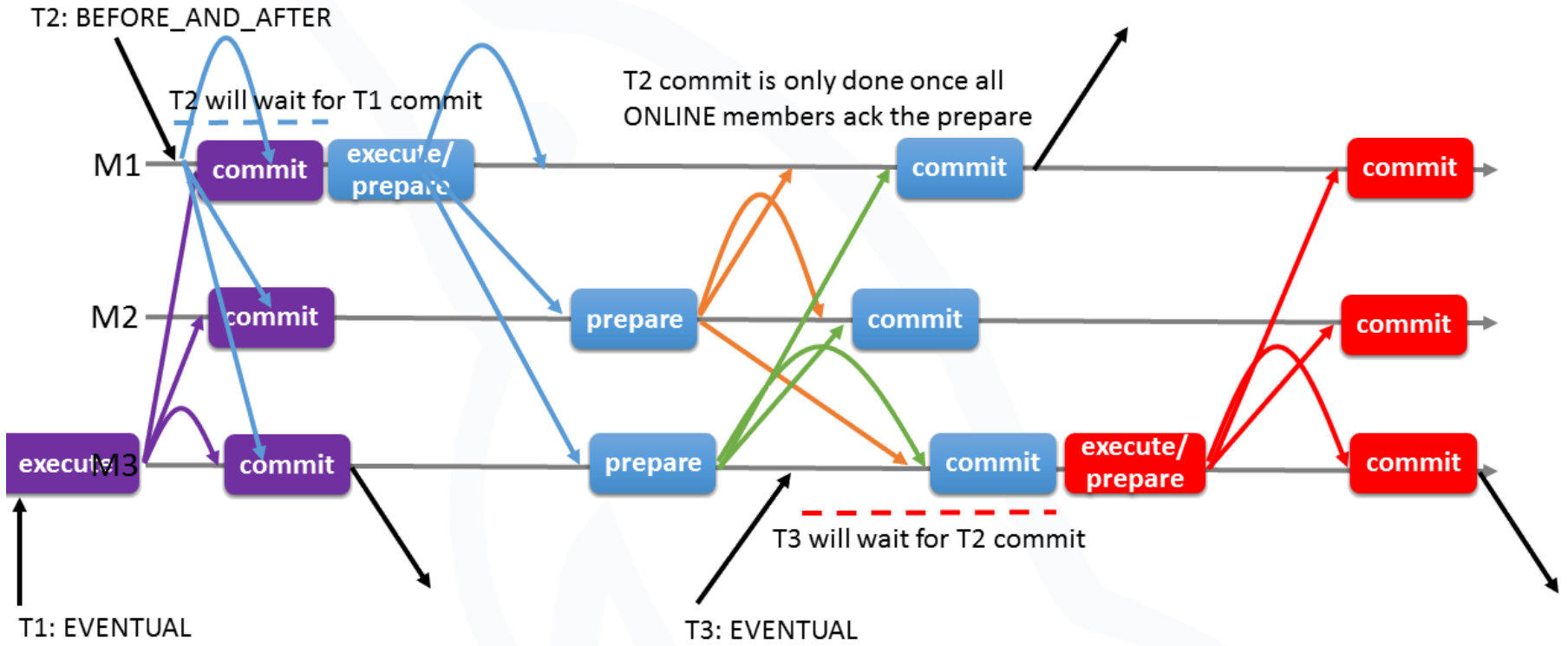
# Consistency Levels - AFTER

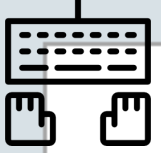
AFTER: writes wait that other nodes have prepared before committing



# Consistency Levels - BEFORE_AND_AFTER

BEFORE_AND_AFTER: reads & writes wait before start, writes wait until other nodes prepared before commit





# LAB14: Consistency Levels: EVENTUAL

On `mysql4` lock a table and write to that table on `mysql2`:

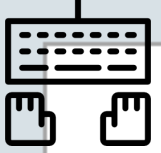
```
mysql4> lock tables app.sbtest1 read;
```

```
mysql2> update app.sbtest1 set k=k+123 where id = 2000;
```

You can now select that row on all members:

```
mysql[2-3-4]> select id, k from app.sbtest1 where id=2000;
```

What do you see ?



# LAB14: Consistency Levels: EVENTUAL

On `mysql4` lock a table and write to that table on `mysql2`:

```
mysql4> lock tables app.sbtest1 read;
```

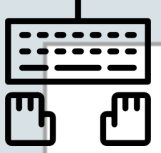
```
mysql2> update app.sbtest1 set k=k+123 where id = 2000;
```

You can now select that row on all members:

```
mysql[2-3-4]> select id, k from app.sbtest1 where id=2000;
```

What do you see ?

```
mysql4> UNLOCK TABLES;
```



# LAB14: Consistency Levels: AFTER

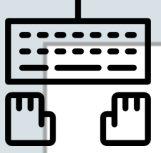
- set the session consistency level to AFTER (WRITE) on `mysql2`
- and lock the table on `mysql4`:

```
mysql2> SET @@SESSION.group_replication_consistency='AFTER';
```

```
mysql4> lock tables app.sbtest1 read;
```

Now let´s modify one record on `mysql2`:

```
mysql2> update app.sbtest1 set k=k+456 where id = 2000;
```



# LAB14: Consistency Levels: AFTER

- set the session consistency level to AFTER (WRITE) on `mysql2`
- and lock the table on `mysql4`:

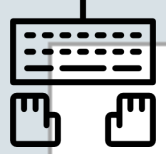
```
mysql2> SET @@SESSION.group_replication_consistency='AFTER';
```

```
mysql4> lock tables app.sbtest1 read;
```

Now let´s modify one record on `mysql2`:

```
mysql2> update app.sbtest1 set k=k+456 where id = 2000;
```

What do you see?



# LAB14: Consistency Levels: AFTER

- set the session consistency level to AFTER (WRITE) on mysql2
- and lock the table on mysql4:

```
mysql2> SET @@SESSION.group_replication_consistency='AFTER';
```

```
mysql4> lock tables app.sbtest1 read;
```

Now let´s modify one record on mysql2:

```
mysql2> update app.sbtest1 set k=k+456 where id = 2000;
```

What do you see?

```
mysql4> UNLOCK TABLES;
```

# LAB14: Consistency Levels: observability

On the server writing (mysql2), you can run this to see the query waiting for the AFTER synchronization:

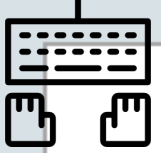
```
mysql2>SELECT *, TIME_TO_SEC(TIMEDIFF(now(),trx_started)) lock_time_sec
     FROM information_schema.innodb_trx JOIN information_schema.processlist
     ON processlist.ID=innodb_trx.trx_mysql_thread_id
     WHERE state='waiting for handler commit' ORDER BY trx_started\G
*************************** 1. row ***************************
                trx_id: 49234
             trx_state: RUNNING
           trx_started: 2019-05-14 22:34:31
     trx_mysql_thread_id: 14
             trx_query: update app.sbtest1 set k=456 where id = 2000
                        ...
                    ID: 14
                  USER: clusteradmin
                  HOST: mysql3:33484
                 STATE: waiting for handler commit
                  INFO: PLUGIN: update app.sbtest1 set k=k+456 where id=2000
          lock_time_sec: 52
```

# More about Consistency Levels

By default the wait time (timeout) for these operations is very long (8 hours) and can be modified to a shorter period by setting `wait_timeout` to a shorter delay.

More examples and observability queries at

https://lefred.be/content/mysql-innodb-cluster-consistency-levels/



# Back to Single Primary Mode

Let´s put the cluster back in single primary mode and make sure `mysql3` is primary

```
mysql-js> cluster.switchToSinglePrimaryMode('mysql3:3306')
```

Let´s break it!

# Failure Scenarios

# Failure Scenarios - Labs

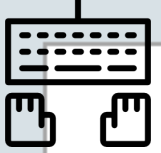
- Stopping a node, bringing it back
- Full power loss
- No more quorum
- The Impact of Network Latency & Packet Loss
- Degraded Performance On A Node

# Stopping A Node

- As seen in previous labs: nodes can be removed/expelled from cluster
- But nodes can be gracefully removed as well (for maintenance...)

2 distinct terms are used:

- Group: The current set of members of an active Group Replication view.
- Cluster: All mysql nodes defined in MySQL InnoDB Cluster, regardless if they are active or not.



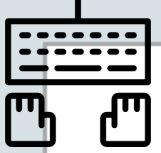
# LAB15: Stopping A Node

On `mysql4`, check the error log:

```
mysql4# tail -f /var/log/mysqld.log
```

Cleanly stop MySQL on `mysql2`:

```
mysql2# systemctl stop mysqld
```



# LAB15: Stopping A Node

On `mysql4`, check the error log:

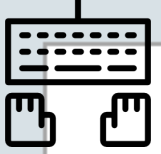
```
mysql4# tail -f /var/log/mysqld.log
```

Cleanly stop MySQL on `mysql2`:

```
mysql2# systemctl stop mysqld
```

We can see:

```
[MY-011499] [Repl] Plugin group_replication reported:
           'Members removed from the group: mysql2:3306'
```



# LAB15: Stopping A Node

On `mysql4`, check the error log:

```
mysql4# tail -f /var/log/mysqld.log
```

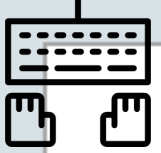
Cleanly stop MySQL on `mysql2`:

```
mysql2# systemctl stop mysqld
```

We can see:

```
[MY-011499] [Repl] Plugin group_replication reported:
           'Members removed from the group: mysql2:3306'
```

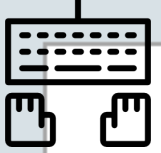
when performing a graceful shutdown, quorum is not taken in consideration



# LAB15: Stopping A Node : Member VS Node

```
mysql> SELECT MEMBER_HOST, MEMBER_PORT, MEMBER_STATE, MEMBER_ROLE,
       MEMBER_VERSION FROM performance_schema.replication_group_members;
+-------------+-------------+--------------+-------------+----------------+
| MEMBER_HOST | MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION |
+-------------+-------------+--------------+-------------+----------------+
| mysql4      |        3306 | ONLINE       | PRIMARY     | 8.0.16         |
| mysql3      |        3306 | ONLINE       | PRIMARY     | 8.0.16         |
+-------------+-------------+--------------+-------------+----------------+
```

```
mysql-js> cluster.status()
{
    ...
            "mysql2:3306": {
                "address": "mysql2:3306",
                "mode": "n/a",
                "readReplicas": {},
                "role": "HA",
                "status": "(MISSING)"
            },
    ...
}
```



# LAB15: Rejoining A Node

As `group_replication_start_on_boot` is enabled by default, just restarting `mysql2` will allow it to join back the group.

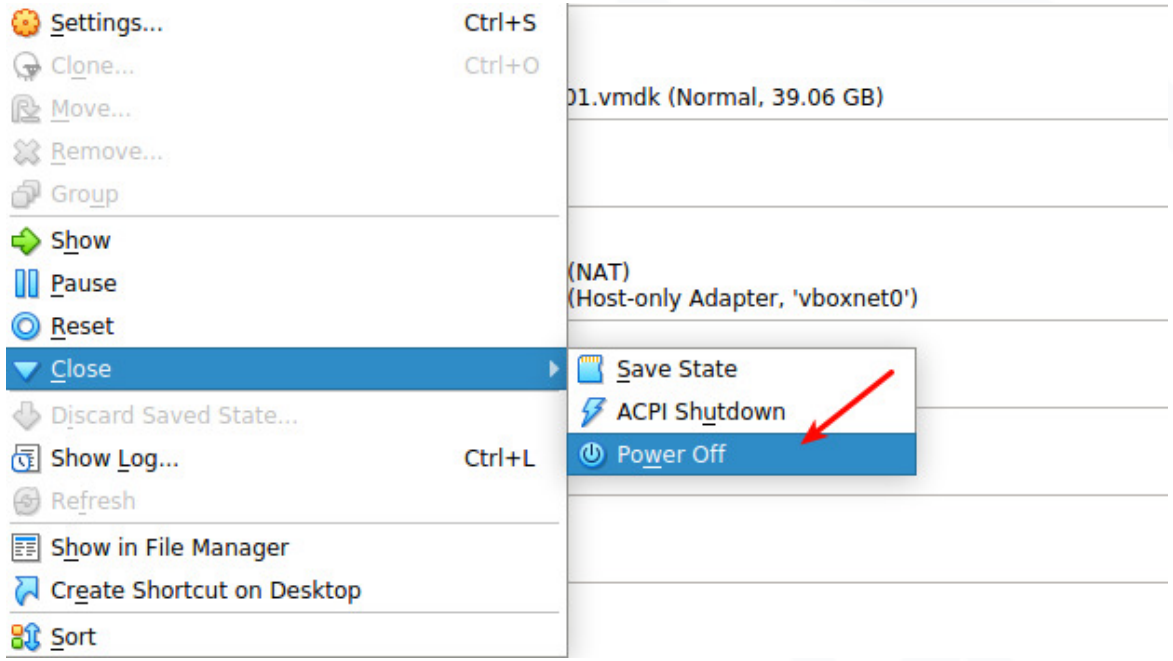
```
mysql2# systemctl start mysqld
```

If it was not the case, from the MyShell you could use the `rejoinInstance()` method:

```
mysql-js> cluster.rejoinInstance('clusteradmin@mysql2')
```

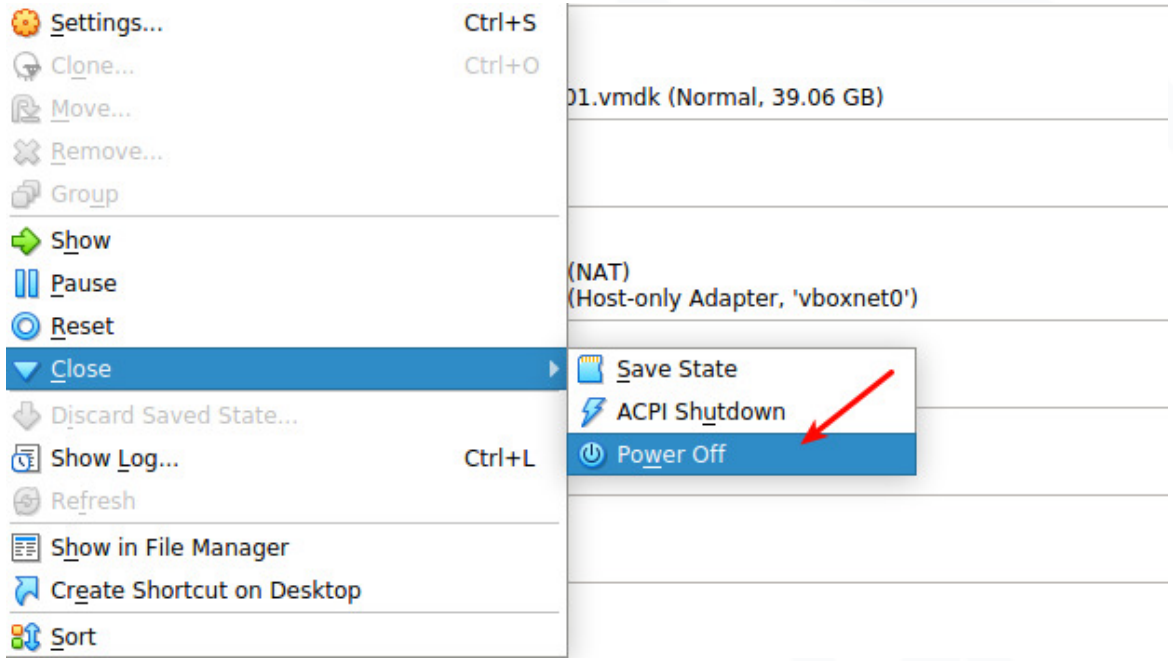
# LAB16: Full Power Loss

Now just power off all the 3 VMs part of the cluster (`mysql2`, `mysql3` and `mysql4`)

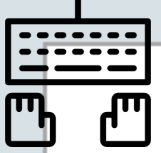


# LAB16: Full Power Loss

Now just power off all the 3 VMs part of the cluster (`mysql2`, `mysql3` and `mysql4`)




And restart them and check that `mysqld` is restarted too.



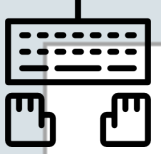
# LAB16: Full Power Loss (2)

Connect to one node and retrieve the cluster object, after a timeout, the following error will be seen:

```
mysql1# mysqlsh
mysql-js> \c clusteradmin@mysql2
mysql-js> cluster=dba.getCluster()
Dba.getCluster: This function is not available through a session
to a standalone instance (metadata exists, but GR is not active)
(RuntimeError)
```

Check the error log, no members where able to join and recreate a group:

```
'[GCS] Error connecting to all peers. Member join failed. Local port: 33061'
'[GCS] The member was unable to join the group. Local port: 33061'
'Timeout on wait for view after joining group'
```



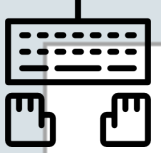
# LAB16: Full Power Loss - restart the cluster (3)

Use the `dba` object to 'reboot' the cluster:

```
mysql-js> cluster=dba.rebootClusterFromCompleteOutage()
```

Note:

```
mysql-js> cluster=dba.rebootClusterFromCompleteOutage()
Reconfiguring the default cluster from complete outage...

The instance 'mysql3:3306' was part of the cluster configuration.
Would you like to rejoin it to the cluster? [y/N]: y

The instance 'mysql4:3306' was part of the cluster configuration.
Would you like to rejoin it to the cluster? [y/N]: y

Dba.rebootClusterFromCompleteOutage: The active session instance isn't the most
updated in comparison with the ONLINE instances of the Cluster's metadata.
Please use the most up to date instance: 'mysql3:3306'. (RuntimeError)
```



# LAB16: Full Power Loss - restart the cluster (4)

If one of the previous members need to recover some transactions, the next node won´t join automatically (fixed in 8.0.17).

You need to use:

```
cluster.rejoinInstance('clusteradmin@mysqlX')
```

Join the members in `"status": "(MISSING)"`

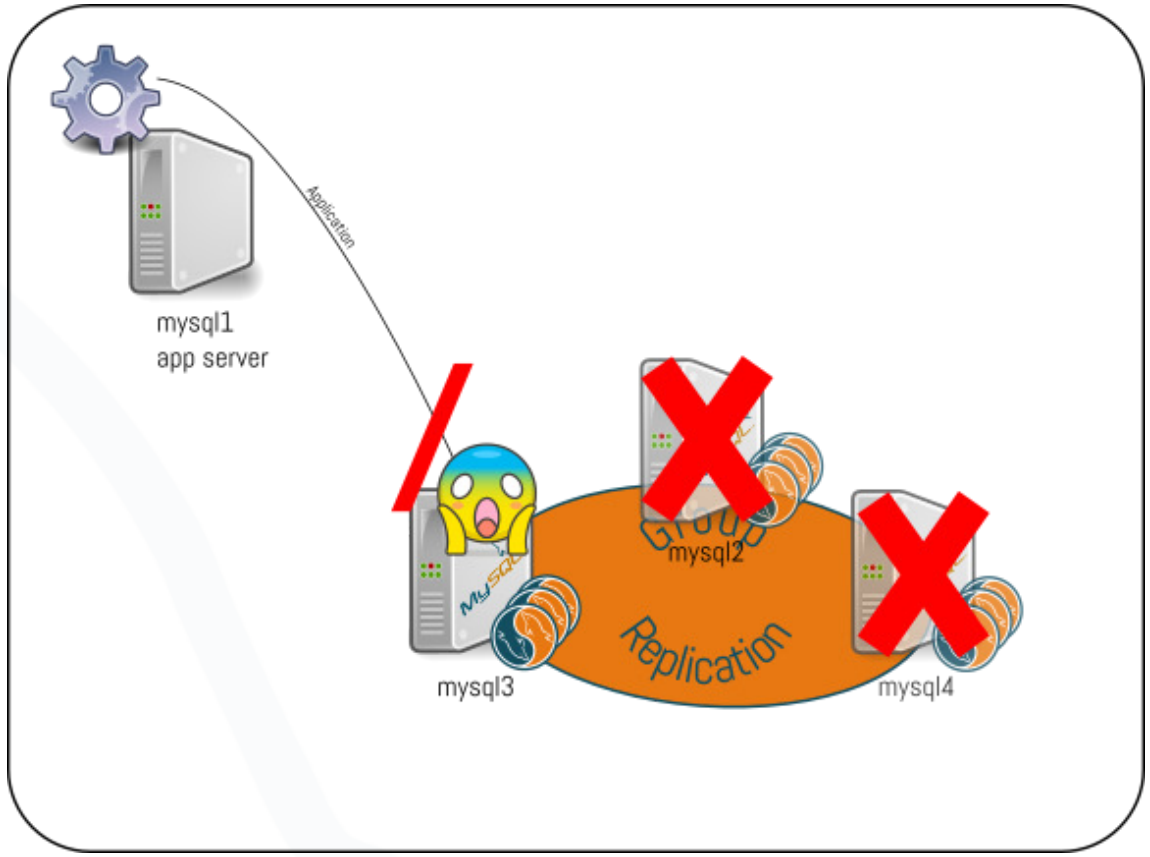
# No more Quorum

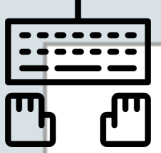
When the majority of nodes are 'lost', the remaining minority (not reaching quorum) will not accept any transactions to avoid split-brain situation.

How to bring back the service with the remaining ondes?

This does require human intervention

- first guarantee the lost majority of the nodes are actually gone
- then make the minority network partition primary

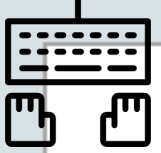




# LAB17: Run MySQL without quorum

For this lab, stop `mysqld` on `mysql2` and `mysql4`:

```
mysql3# tail -f /var/log/mysqld.log

mysql2# failure-scenario.sh kill-node

mysql4# failure-scenario.sh kill-node
```

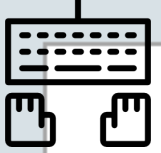


# LAB17: Run MySQL without quorum

For this lab, stop `mysqld` on `mysql2` and `mysql4`:

```
mysql3# tail -f /var/log/mysqld.log

mysql2# failure-scenario.sh kill-node

mysql4# failure-scenario.sh kill-node
```

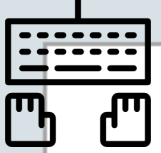
```
[WARNING]
      'Member with address mysql2:3306 has become unreachable.'
      'Member with address mysql4:3306 has become unreachable.'
[ERROR]
      'This server is not able to reach a majority of members in the group.
       This server will now block all updates. The server will remain blocked
       until contact with the majority is restored. It is possible to use
       group_replication_force_members to force a new group membership.'
```



# LAB17: Run MySQL without quorum (2)

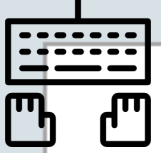
Run `cluster.status()` on `mysql3`:

```
{
    "clusterName": "perconalive",
    "defaultReplicaSet": {
        "name": "default",
        "ssl": "REQUIRED",
         "status": "NO_QUORUM",
         "statusText": "Cluster has no quorum as visible from
                       'mysql3:3306' and cannot process write
                       transactions. 2 members are not active",
        "topology": {
            "mysql2:3306": {
                "address": "mysql2:3306",
                "mode": "n/a",
                "readReplicas": {},
                "role": "HA",
                "status": "UNREACHABLE",
                "version": "8.0.16"
            },...
```



# LAB17: Run MySQL without quorum (3)

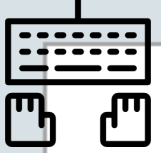
- assume `mysql2` and `mysql4` are gone forever
- there is no majority
- `mysql3` will not be primary
- make `mysql3` primary



# LAB17: Run MySQL without quorum (3)

- assume `mysql2` and `mysql4` are gone forever
- there is no majority
- `mysql3` will not be primary
- make `mysql3` primary

```
mysql-js> cluster.forceQuorumUsingPartitionOf('clusteradmin@mysql3')
Restoring replicaset 'default' from loss of quorum, by using the partition composed of [mysql3:3306]

Restoring the InnoDB cluster ...

The InnoDB cluster was successfully restored using the partition from the instance 'clusteradmin@mysql3'.
WARNING: To avoid a split-brain scenario, ensure that all other members of the replicaset are removed
or joined back to the group that was restored.
```

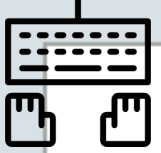


# LAB17: Run MySQL without quorum (3)

- assume `mysql2` and `mysql4` are gone forever
- there is no majority
- `mysql3` will not be primary
- make `mysql3` primary

```
mysql-js> cluster.forceQuorumUsingPartitionOf('clusteradmin@mysql3')
Restoring replicaset 'default' from loss of quorum, by using the partition composed of [mysql3:3306]

Restoring the InnoDB cluster ...

The InnoDB cluster was successfully restored using the partition from the instance 'clusteradmin@mysql3'.
WARNING: To avoid a split-brain scenario, ensure that all other members of the replicaset are removed
or joined back to the group that was restored.
```

As soon as this is done, the application is running again !



# LAB17: Run MySQL without quorum (4)

Restart MySQL on `mysql2` and `mysql4`:

```
mysql2# systemctl start mysqld

mysql4# systemctl start mysqld
```

Check `cluster.status()`, you may need to use `cluster.rejoinInstance()`.

## More info:

- https://lefred.be/content/mysql-innodb-cluster-how-to-manage-a-split-brain-situation/
- https://lefred.be/content/mysql-innodb-cluster-avoid-split-brain-while-forcing-quorum/

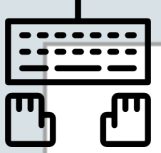
# The Impact of Network Latency & Packet Loss

How is the group impacted when there is higher network latency between nodes?

- Does it impact read & write performance?
- If only 1 node is in another datacenter, does it impact latency?
  - Given it´s a majority concensus protocol, does it mean there is no impact as majority can be achieved in 1 datacenter?

Let´s do some tests... change the app to only perform writes (on `mysql1`):

```
^C
# run_app.sh mysql1 write
```



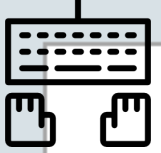
# LAB18: Distant Node Latency

On `mysql2` ping `mysql4`:

```
mysql2# ping mysql4
```

Add some latency on `mysql4`:

```
mysql4# failure-scenario.sh add-latency begin
```



# LAB18: Distant Node Latency

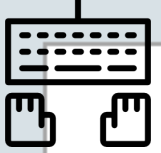
On `mysql2` ping `mysql4`:

```
mysql2# ping mysql4
```

Add some latency on `mysql4`:

```
mysql4# failure-scenario.sh add-latency begin
```

You can see that now it takes 200ms to ping



# LAB18: Distant Node Latency

On `mysql2` ping `mysql4`:

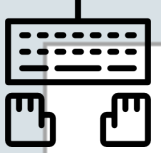
```
mysql2# ping mysql4
```

Add some latency on `mysql4`:

```
mysql4# failure-scenario.sh add-latency begin
```

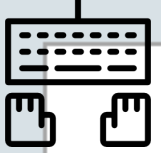
You can see that now it takes 200ms to ping

What do we see in the app ?



# LAB18: Distant Node Latency (2)

Time to time, we can see that the application slows down... why ?



# LAB18: Distant Node Latency (2)

Time to time, we can see that the application slows down... why ?

That´s because the system has to wait for the **noop** (single skip message) from the "distant" node.

Members can propose messages for their slots without having to wait for other members. But if they don't have anything to say, they need to tell it too and this is where we are affected by the latency.

## More Info:

- https://lefred.be/content/mysql-group-replication-about-ack-from-majority/

# LAB18: General Latency

## What happens if we add latency on all nodes ?

```
mysql2# failure-scenario.sh add-latency begin

mysql3# failure-scenario.sh add-latency begin
```

# LAB18: General Latency

What happens if we add latency on all nodes ?

```
mysql2# failure-scenario.sh add-latency begin

mysql3# failure-scenario.sh add-latency begin
```

this increases the overal latency much more than expected !

# LAB18: General Latency

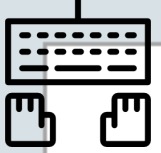
## What happens if we add latency on all nodes ?

```
mysql2# failure-scenario.sh add-latency begin

mysql3# failure-scenario.sh add-latency begin
```

this increases the overal latency much more than expected !

## Remove the latency:

```
mysql2# failure-scenario.sh add-latency end
mysql3# failure-scenario.sh add-latency end
mysql4# failure-scenario.sh add-latency end
```

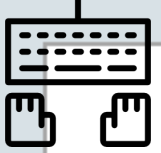


# LAB18 Unreliable Network (packet loss)

Put `mysql4` on a flaky network:

```
mysql4# failure-scenario.sh packet-loss begin
```

Check the error log on `mysql4`.



# LAB18 Unreliable Network (packet loss)

Put `mysql4` on a flaky network:

```
mysql4# failure-scenario.sh packet-loss begin
```

Check the error log on `mysql4`.

```
'Member with address mysql2:3306 has become unreachable.'
'Member with address mysql2:3306 is reachable again.'
'Member with address mysql2:3306 has become unreachable.'
'Member with address mysql3:3306 has become unreachable.'
'This server is not able to reach a majority of members in the group.
 This server will now block all updates. The server will remain blocked
 until contact with the majority is restored. It is possible to use
 group_replication_force_members to force a new group membership.'
'Member with address mysql3:3306 is reachable again.'
'Member with address mysql3:3306 has become unreachable.'
Member was expelled from the group due to network failures,
 changing member status to ERROR.'
```

# LAB18 Unreliable Network (packet loss)

## Remove packet loss:

```
mysql4# failure-scenario.sh packet-loss end
```

# LAB18 Unreliable Network (packet loss)

Remove packet loss:

```
mysql4# failure-scenario.sh packet-loss end
```

We can stop the specific application and run it now in a different mode to prepare the next lab:

```
^C
# run_app.sh mysql1 flowcontrol
```

# Degraded Performance On A Node

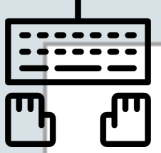
**Flow control** is implemented to prevent from falling behind:

- less hardware resources
- hardware/resource issue
- too high workload, ...

On the nodes that are receiving writes:

- as soon as another node in the group has a growing apply queue
- **Flow Control** will start rate limit

Default: `group_replication_flow_control_applier_threshold` is 25000



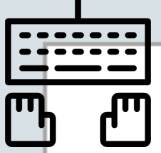
# LAB19: Degraded Performance On A Node

Set `group_replication_flow_control_applier_threshold` to 100 on all nodes:

```
mysql> SET GLOBAL group_replication_flow_control_applier_threshold = 100;
```

Now we simulate degraded performance on `mysql4` (make sure it is not `PRIMARY`):

```
mysql4# failure-scenario.sh disk-issue
```



# LAB19: Degraded Performance On A Node

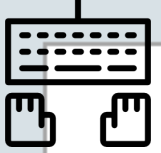
Set `group_replication_flow_control_applier_threshold` to 100 on all nodes:

```
mysql> SET GLOBAL group_replication_flow_control_applier_threshold = 100;
```

Now we simulate degraded performance on `mysql4` (make sure it is not `PRIMARY`):

```
mysql4# failure-scenario.sh disk-issue
```

```
SELECT COUNT_TRANSACTIONS_REMOTE_IN_APPLIER_QUEUE
FROM performance_schema.replication_group_member_stats;
+----------------------------------------------+
| COUNT_TRANSACTIONS_REMOTE_IN_APPLIER_QUEUE |
+----------------------------------------------+
|                                          697 |
|                                            0 |
|                                            0 |
+----------------------------------------------+
```



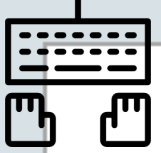
# LAB19: Degraded Performance On A Node (2)

When 100 is reached:

- writes reduced slowly
- as queue grows: more slowdown
- never goes to 0

It´s possible to ignore the health of `mysql4` by telling to the PRIMARY to ignore the FC statistics received:

```
mysql[2-3]> set global group_replication_flow_control_mode='DISABLED';
```



# LAB19: Degraded Performance On A Node (2)

When 100 is reached:

- writes reduced slowly
- as queue grows: more slowdown
- never goes to 0

It´s possible to ignore the health of `mysql4` by telling to the PRIMARY to ignore the FC statistics received:

```
mysql[2-3]> set global group_replication_flow_control_mode='DISABLED';
```

As soon as it´s disabled, you can see the application going back at full speed!

other talks

# More during the Conference

Wednesday May 29th 11:55am-12:45pm

Texas 1

PERCONA LIVE

# New Features in MySQL 8.0 Replication

Luís Soares

MySQL™

ORACLE

Thursday May 30th 2:55pm-3:45pm

Texas 1

PERCONA LIVE

# MySQL Group Replication

## The Magic Explained v2

Frédéric Descamps
@lefred

MySQL™

ORACLE

# Thank you !

## Any Questions ?